

GeneXproTools Help



[GeneXproTools Editions](#) | [About GeneXproTools 4.0 Help File](#) | [Contact Us](#)



Introduction

GeneXproTools is an extremely flexible modeling tool designed for **Function Finding, Classification, Time Series Prediction, and Logic Synthesis**.

There are four different editions of GeneXproTools: **Standard, Advanced, Professional, and Enterprise**. In addition, **Academic Versions** are available for both the Professional and Enterprise editions. The Academic versions have all the features of the respective Professional and Enterprise editions and can only be used for academic purposes. See a feature comparison table of the four editions [here](#).

GeneXproTools is very easy to use and as soon as you learn how to load your data, you will be able to generate a model immediately as GeneXproTools comes with a series of templates that only require a mouse click for you to design a model.

With GeneXproTools you can fully understand and analyze the evolved models as GeneXproTools automatically translates the models it generates using its native Karva code into a wide set of programming languages (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL) through the use of **built-in grammars**. Furthermore, GeneXproTools 4.0 also allows you to create your own grammars for translating the native Karva code, and, therefore, with GeneXproTools 4.0 you can automatically translate all the evolved models into virtually any programming language through the use of **User Defined Grammars**. And for Logic

Synthesis problems, each built-in grammar comes with six different sub-grammars (All Gates, Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) for designing logic circuits with specific types of gates.

You can then paste the evolved models into your IDE or code files and use them to create highly sophisticated software applications such as software for predicting unknown behavior, classifying previously unclassified samples, forecasting future events based on past ones, designing parsimonious circuits, and so on.

Notwithstanding, with GeneXproTools 4.0 you don't have to leave the GeneXproTools environment for making predictions about the future or predicting unknown behavior with the evolved models, as GeneXproTools 4.0 comes equipped with a **scoring engine** that allows you to immediately apply the evolved models to an unlimited number of records. Indeed, as soon as a model is created by GeneXproTools, it can be immediately used for making predictions or extrapolations exactly as it is, with the advantage that one doesn't have to know how to program to do that.

The modeling algorithms of GeneXproTools are based on **Gene Expression Programming** (GEP), an extremely fast and powerful learning algorithm. GeneXproTools can be used without understanding Gene Expression Programming, but if you do you will be able to explore more thoroughly all of its features. In this Help file, you'll find all the necessary information, both theoretical and practical, for exploring all GeneXproTools features successfully. For complementary information, though, we recommend the seminal GEP paper published in the journal *Complex Systems* in 2001:

Ferreira, C., 2001. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems, *Complex Systems*, 13 (2): 87-129,

which is available in pdf format at the [GEP Online Bibliography](#) and the book:

Ferreira, C., 2006. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, 2nd Edition, Springer-Verlag, Germany,

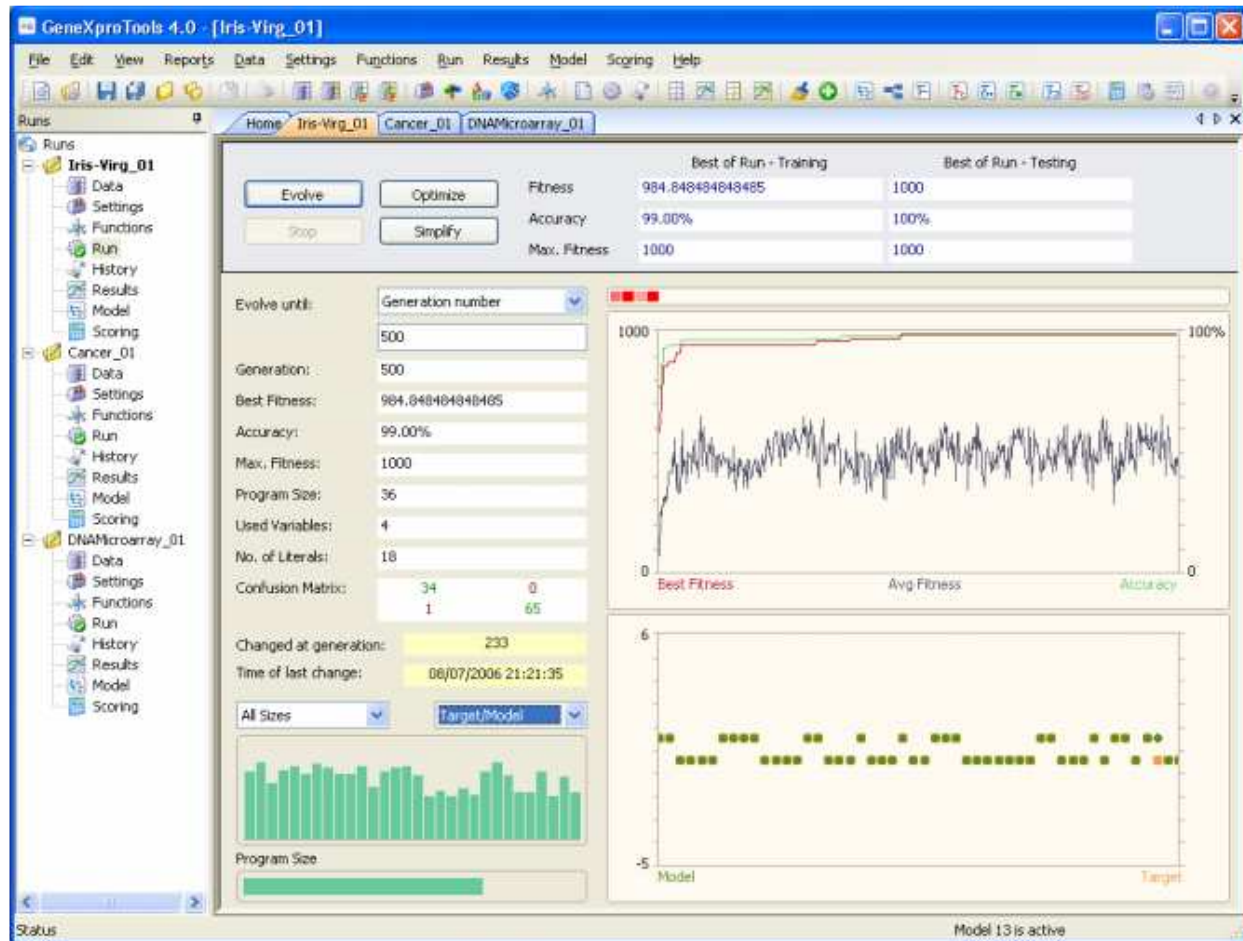
about which you can learn more at the [Gene Expression Programming website](#).

□ | □

GeneXproTools Environment

This chapter introduces you to the art of modeling with GeneXproTools and teaches you how to use the features and capabilities of the GeneXproTools 4.0 automatic programming system. In this chapter, you'll learn how to:

- [Get started with GeneXproTools.](#)
- [Explore GeneXproTools modeling tools.](#)
- [Navigate GeneXproTools panels.](#)
- [Create models with GeneXproTools.](#)
- [Apply an evolved model to make predictions or extrapolations.](#)

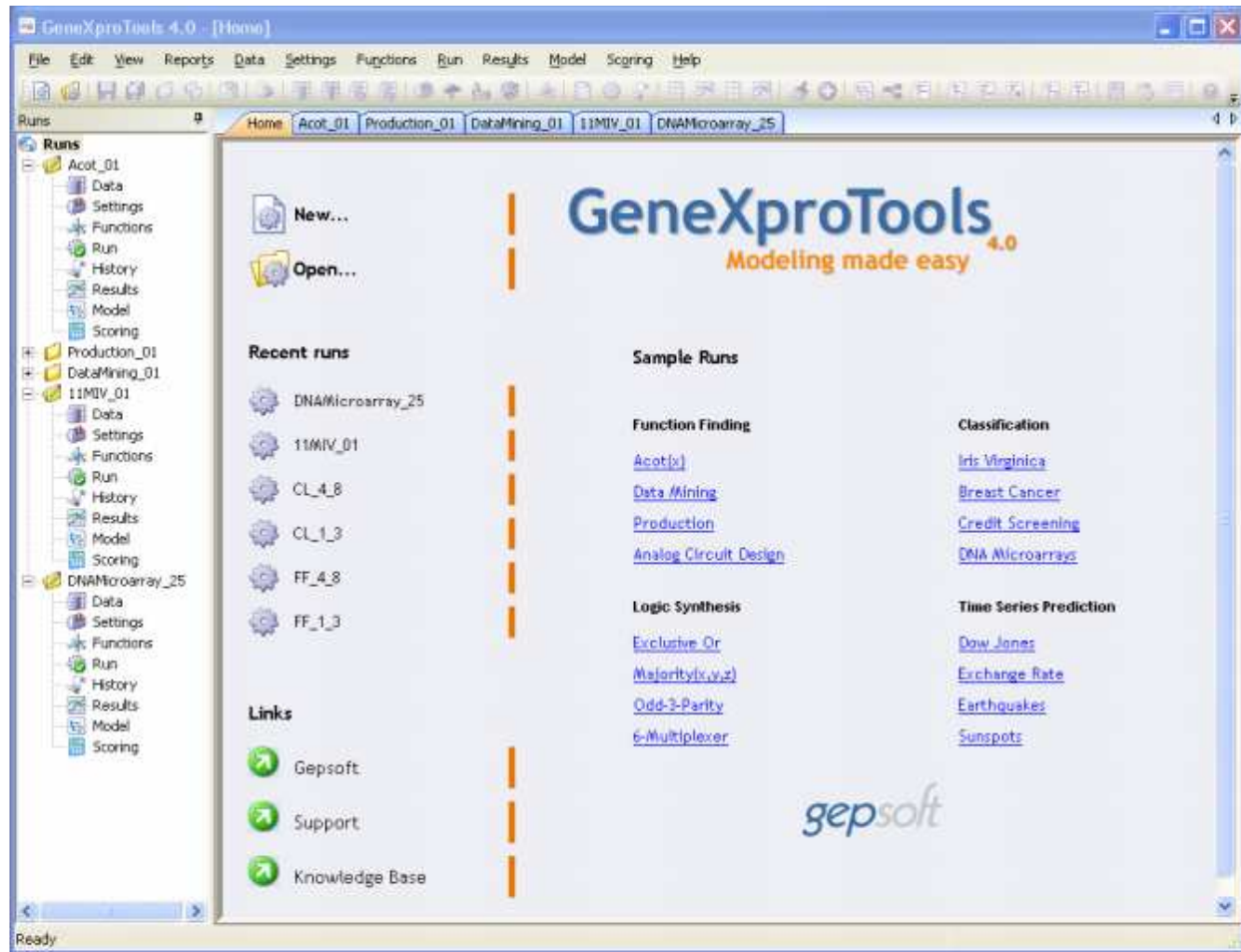


GeneXproTools Environment

Getting Started

When you get started with GeneXproTools 4.0 you'll learn how to:

- Start the GeneXproTools software.
- Load data.
- Evolve a model.
- Optimize a model evolved by GeneXproTools.
- Simplify a model evolved by GeneXproTools.
- Evaluate the accuracy of a model.
- Test the generalizing capabilities of a model.
- Generate code automatically.
- Visualize parse trees.
- Score a model.



GeneXproTools Environment

Starting GeneXproTools

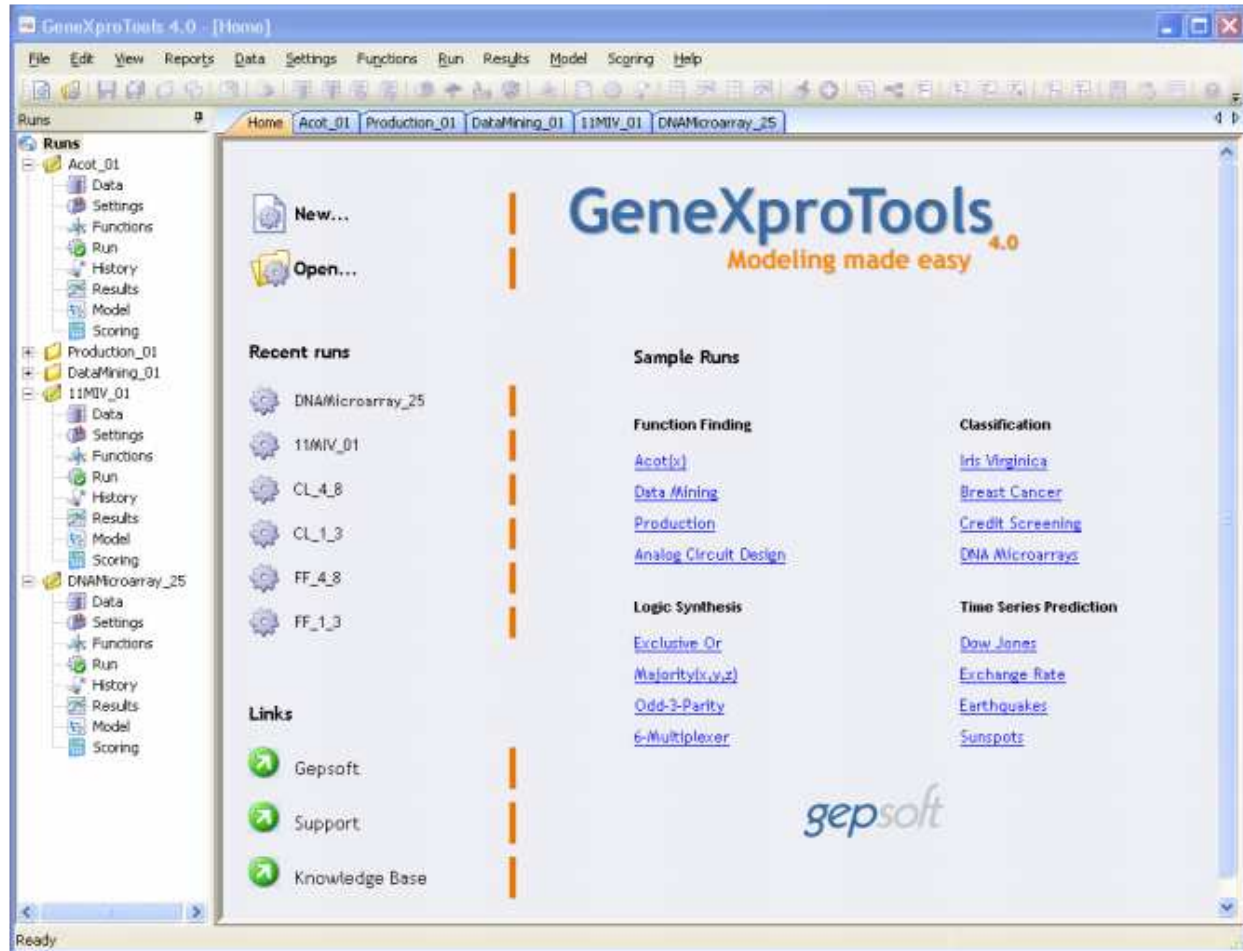
The first step in using GeneXproTools is launching it and opening existing files or creating new ones.

To Start GeneXproTools 4.0

1. In Windows, click Start, point to Programs, and point to the Gepsoft GeneXproTools 4.0 folder. The icons in the Gepsoft GeneXproTools 4.0 folder appear in a list.
2. Click the GeneXproTools 4.0 program icon. The Welcome Screen appears. This welcome screen presents you with several choices, including start a new run or open an existing run.
3. To start a new run, in the Welcome Screen choose the command New run or click the File menu and then choose New. The New Run Wizard appears. This wizard helps you gather the necessary information to start evolving a model.
4. To open an existing run, in the Welcome Screen choose the command Open run or click the File

menu and then choose Open.

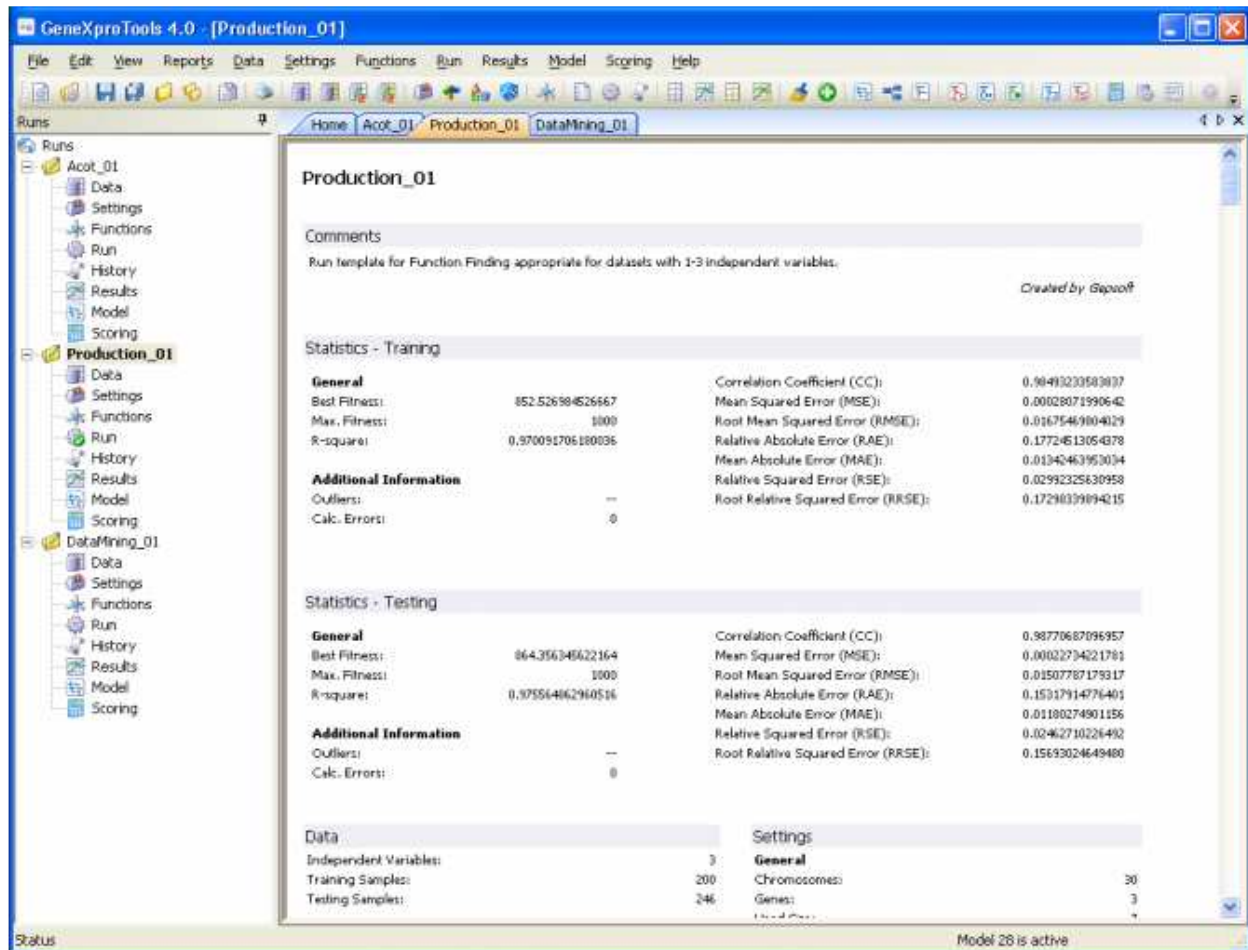
In the Report Panel you will find relevant information concerning this run, including all the settings used in the run, the statistical evaluation of the best-of-run or active model, and a detailed account of the evolutionary history of the run.



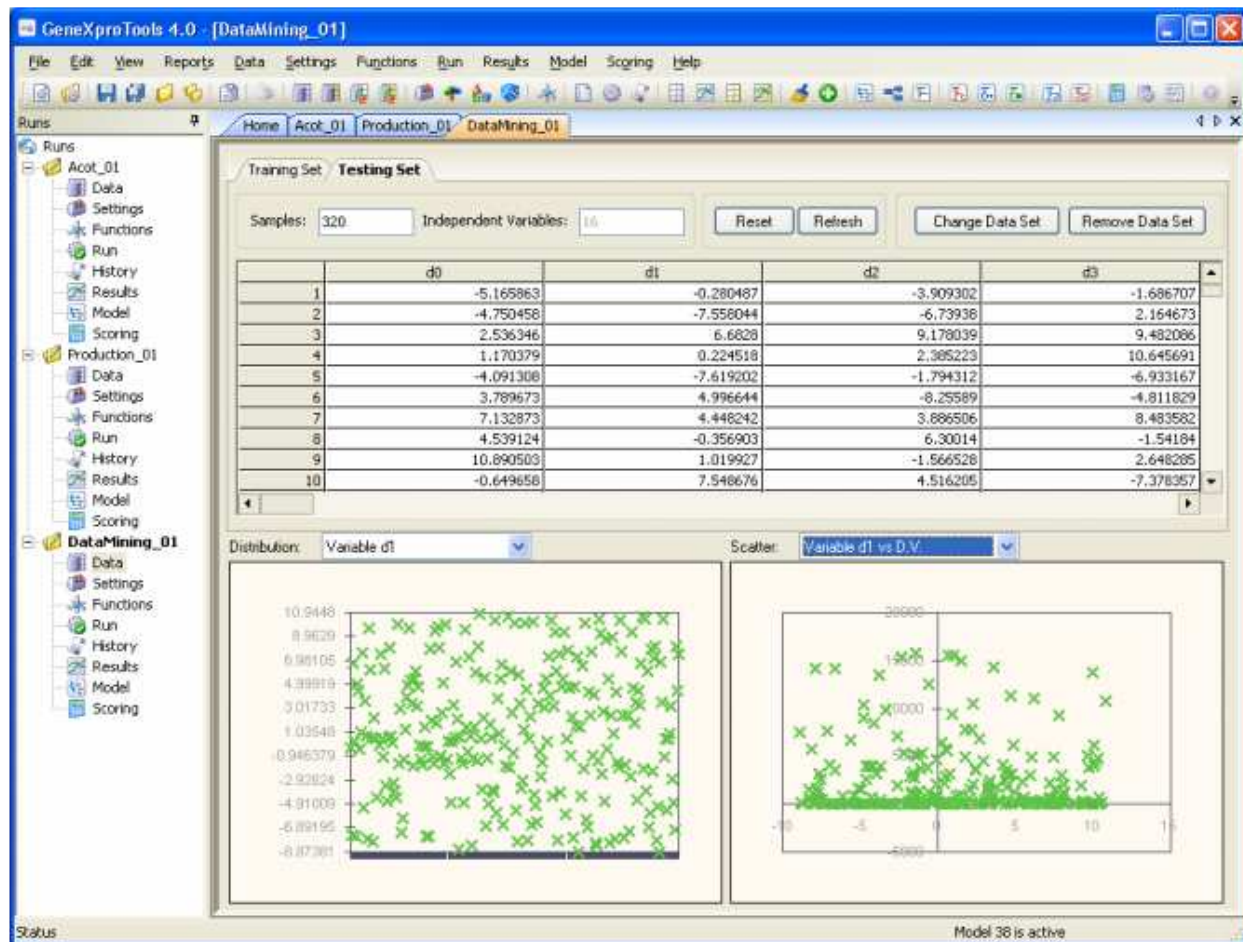
The GeneXproTools 4.0 modeling environment contains 10 basic panels:

- [Report Panel](#)
- [Data Panel](#)
- [Settings Panel](#)
- [Functions Panel](#)
- [Run Panel](#)
- [History Panel](#)
- [Results Panel](#)
- [Predictions Panel](#)
- [Model Panel](#)
- [Scoring Panel](#)

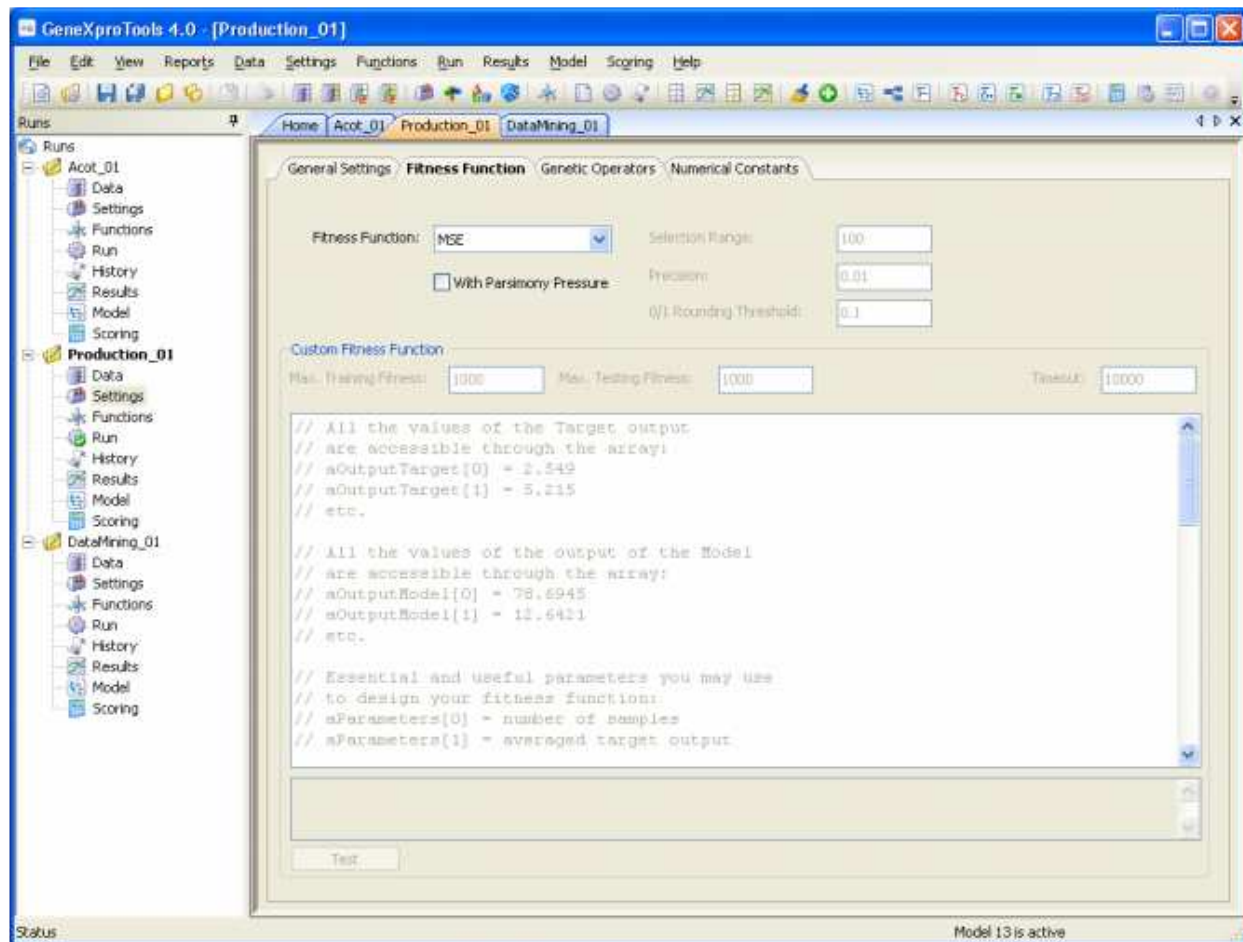
In the **Report Panel** you can find relevant information concerning the run, including all the settings used in the run, the statistical evaluation of the best-of-run or active model (highlighted in yellow in the History heading), and a detailed account of the evolutionary history of the run.



In the **Data Panel** you can visualize both the training and testing data, check easily the distribution of all variables and how each one of them relates to the dependent variable using the scatter plot, change a data set, or choose a subset of your data to use either for training or testing.



The **Settings Panel** gives you access to the most important settings of GeneXproTools, which are organized into four different tabs: General Settings, Fitness Function, Genetic Operators, and Numerical Constants.



In the **General Settings Tab** you can change the number of training and testing samples, the chromosome architecture, the linking function, the population size, and choose the parameters for the Complexity Increase Engine of GeneXproTools.

General Settings Fitness Function Genetic Operators Numerical Constants

Training Samples: 120 Reset

Records in Time Series: 140 Reset

Number of Chromosomes: 30 Embedding Dimension: 10

Head Size: 8 Delay Time: 1

Number of Genes: 5 Prediction Mode: Testing

Linking Function: Addition Testing Predictions: 10

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 8

☒ Save Run History

In the **Fitness Function Tab** you can choose one of the 36 GeneXproTools built-in fitness functions or design and test a fitness function of your own.

General Settings **Fitness Function** Genetic Operators Numerical Constants

Fitness Functions: BASE Selection Range: 100

☐ With Parsimony Pressure Precision: 0.01

B/I Rounding Threshold: 0.1

Custom Fitness Function

Max. Training Fitness: 1000 Max. Testing Fitness: 1000 Timeout: 10000

```
// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

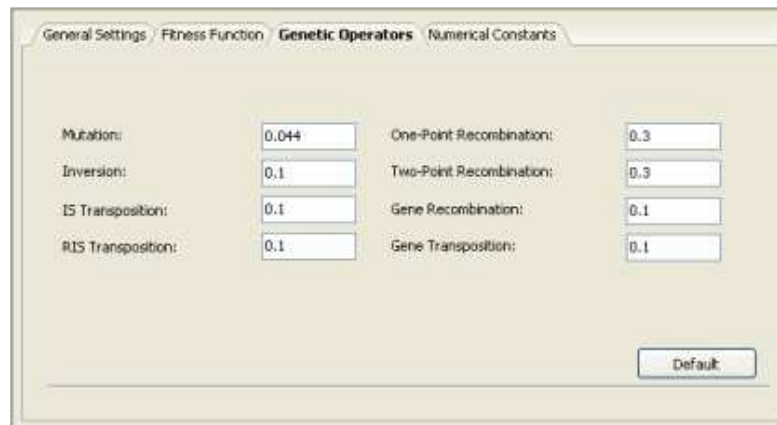
// Useful information about the evolving models:
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals

// Your custom fitness function must return a value, for example:
// return fitness;
```

Test

In the **Genetic Operators Tab** you can change the degree of genetic modification you want to use

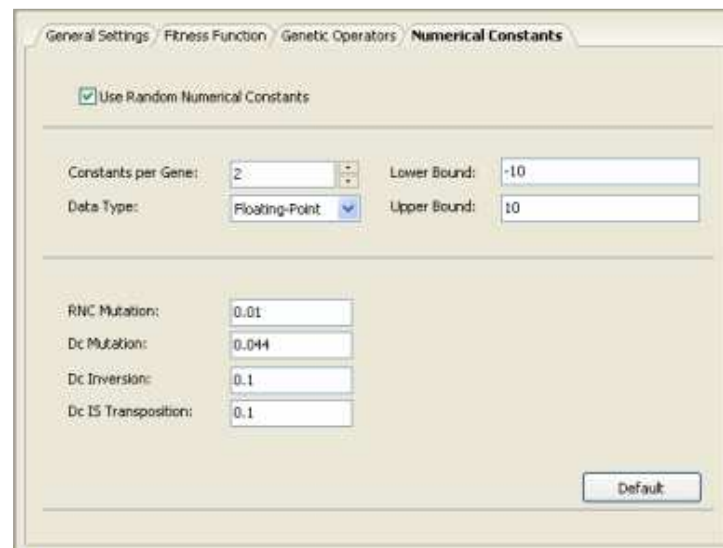
during evolution by choosing the kind of genetic operators and their corresponding rates.



The screenshot shows the 'Genetic Operators' tab in the GeneXproTools interface. It contains several input fields for setting the rates of different genetic operators. The 'Mutation' rate is set to 0.044, 'Inversion' to 0.1, 'IS Transposition' to 0.1, and 'RIS Transposition' to 0.1. On the right side, 'One-Point Recombination' is set to 0.3, 'Two-Point Recombination' to 0.3, 'Gene Recombination' to 0.1, and 'Gene Transposition' to 0.1. A 'Default' button is located at the bottom right of the tab.

Operator	Rate
Mutation	0.044
Inversion	0.1
IS Transposition	0.1
RIS Transposition	0.1
One-Point Recombination	0.3
Two-Point Recombination	0.3
Gene Recombination	0.1
Gene Transposition	0.1

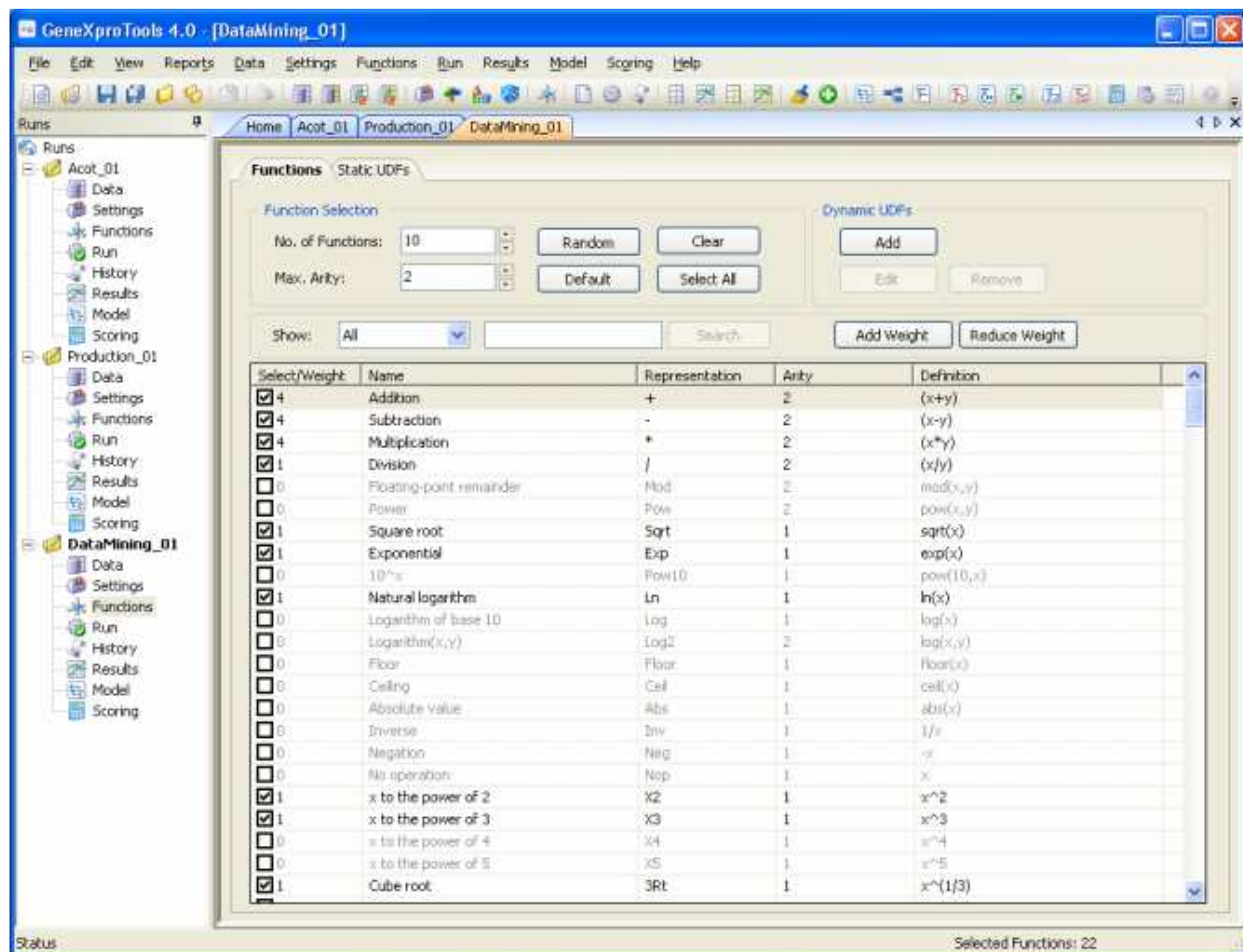
In the **Numerical Constants Tab** you can activate the algorithm that enables the use of random numerical constants and customize both the type and range of your constants, the number of constants per gene, and also choose the genetic operators especially designed for fine-tuning the random constants in your models.



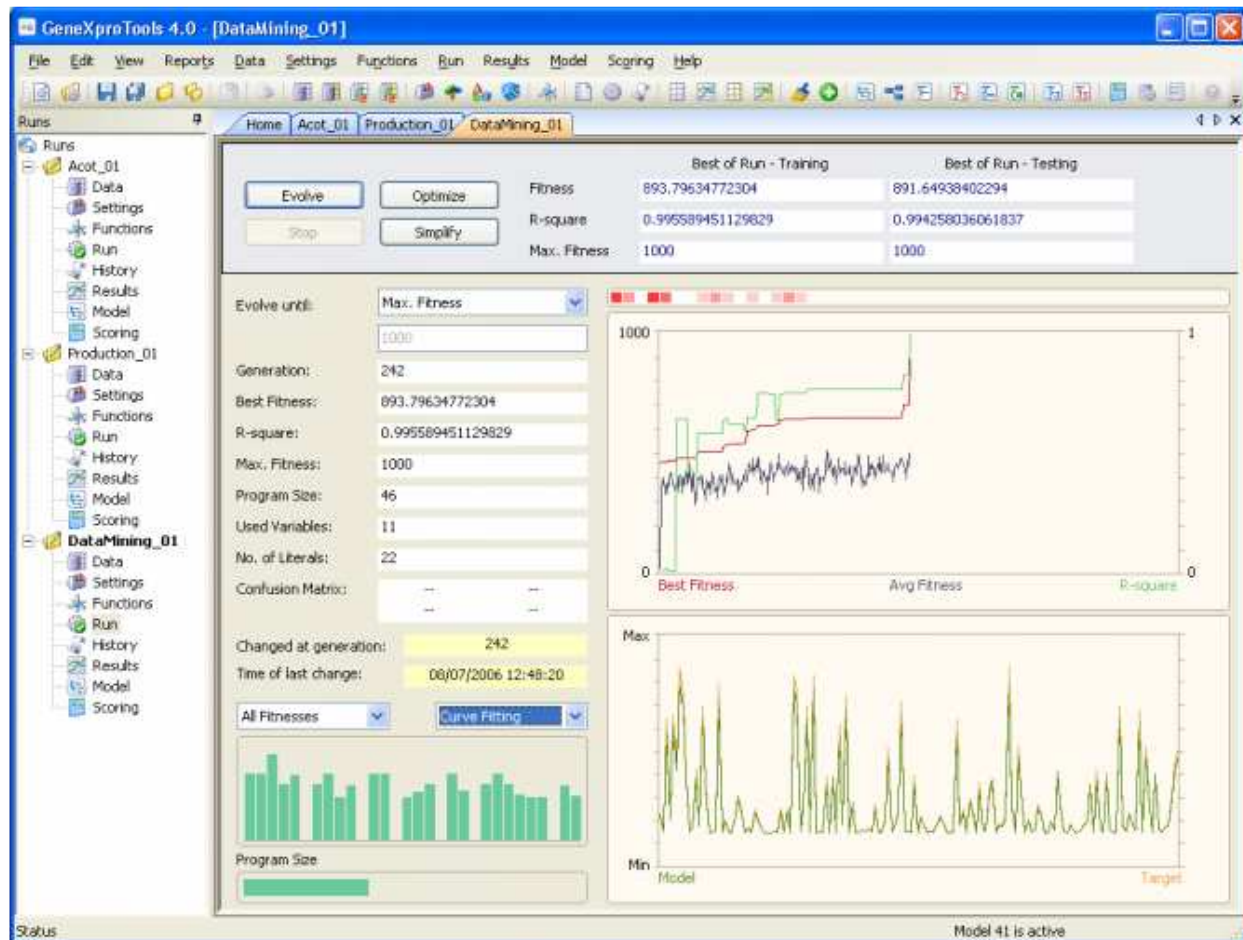
The screenshot shows the 'Numerical Constants' tab in the GeneXproTools interface. It features a checkbox labeled 'Use Random Numerical Constants' which is checked. Below this, there are input fields for 'Constants per Gene' (set to 2), 'Data Type' (set to 'Floating-Point'), 'Lower Bound' (set to -10), and 'Upper Bound' (set to 10). At the bottom, there are input fields for 'RNC Mutation' (0.01), 'Dc Mutation' (0.044), 'Dc Inversion' (0.1), and 'Dc IS Transposition' (0.1). A 'Default' button is located at the bottom right of the tab.

Parameter	Value
Use Random Numerical Constants	☒
Constants per Gene	2
Data Type	Floating-Point
Lower Bound	-10
Upper Bound	10
RNC Mutation	0.01
Dc Mutation	0.044
Dc Inversion	0.1
Dc IS Transposition	0.1

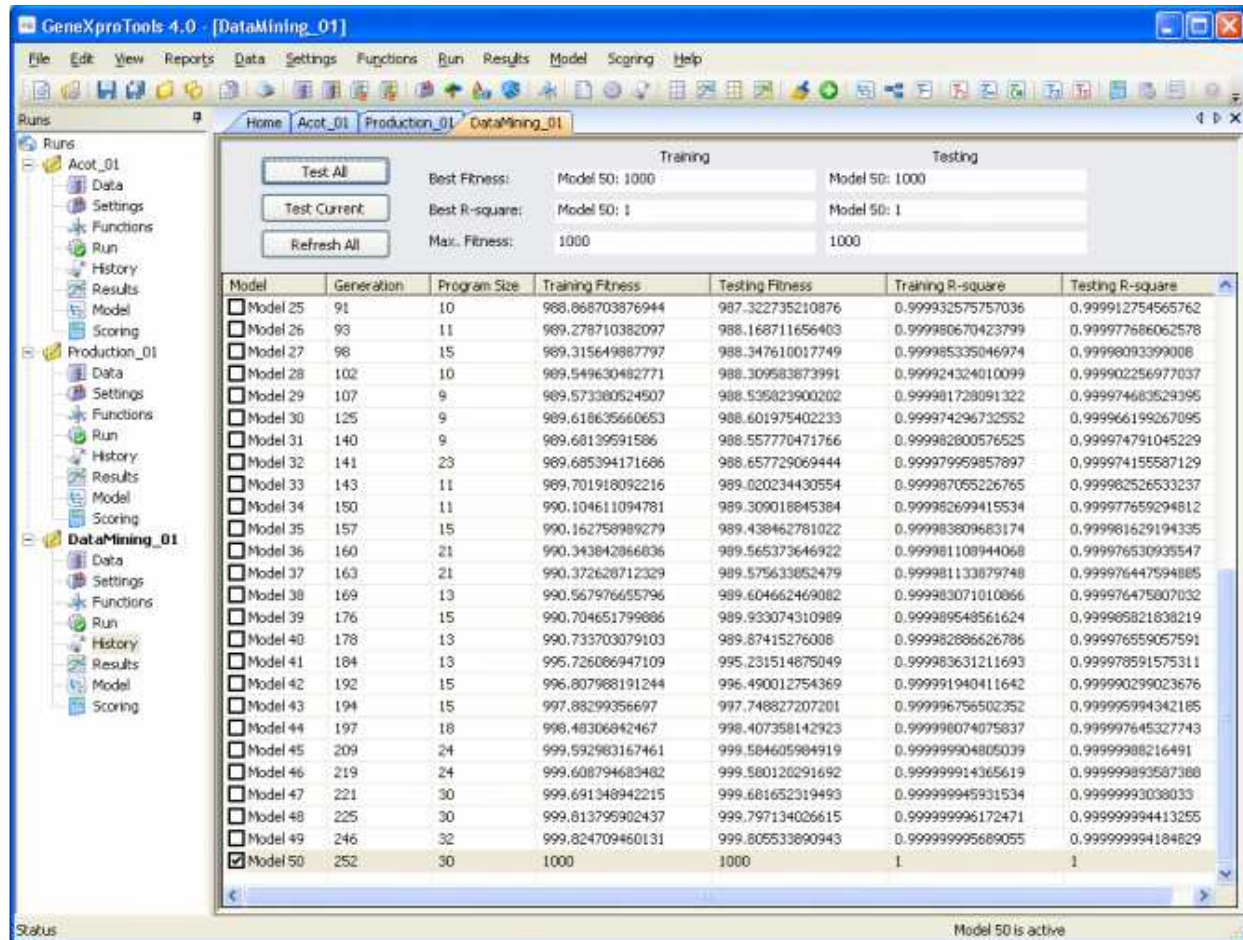
In the **Functions Panel** you choose the functions GeneXproTools will use to model your data. For mathematical problems, there are a total of 279 built-in functions and comparison rules, whereas for Boolean problems there are a total of 258 built-in logical functions.



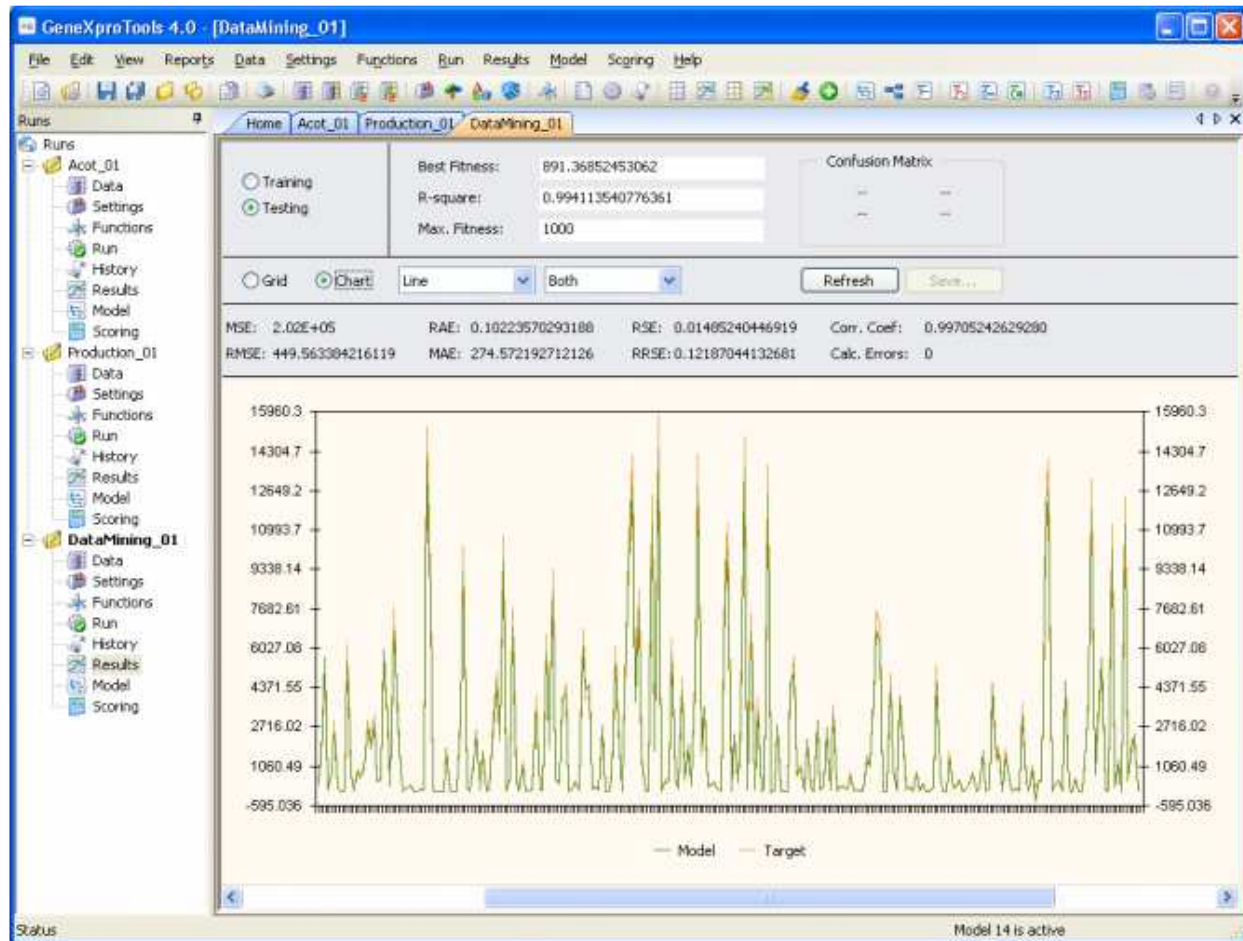
From the **Run Panel** you command and observe the creation, optimization, and simplification of your models through the use of eight different kinds of plots, including a Curve Fitting plot to show how well the model being designed is fitting your data. For Classification and Logic Synthesis problems a Target/Model comparison is shown instead during the evolutionary process.



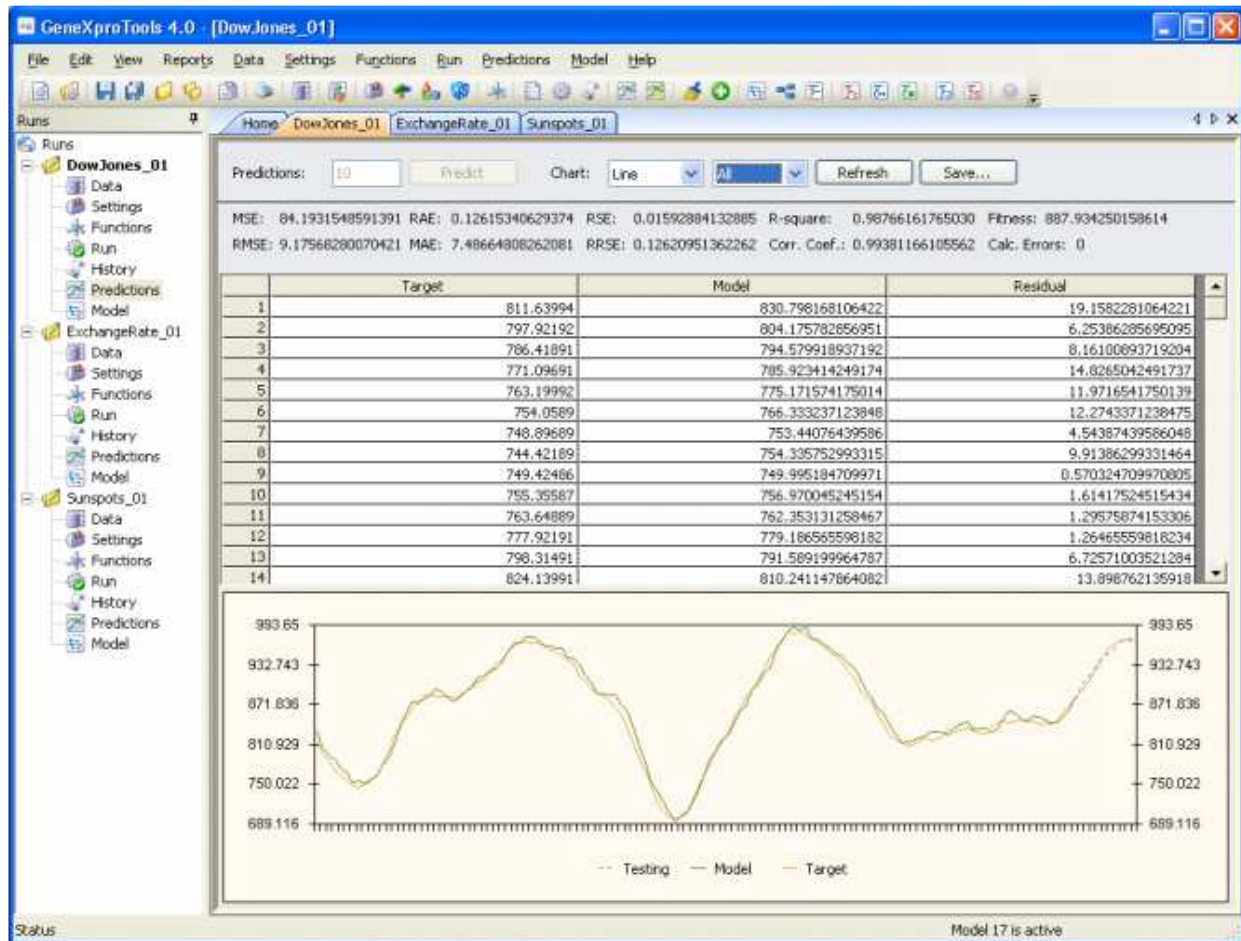
The **History Panel** allows you to explore the best ancestors of the best model as you can select the most intriguing ones and then check them up in more detail in the Results and Model Panels.



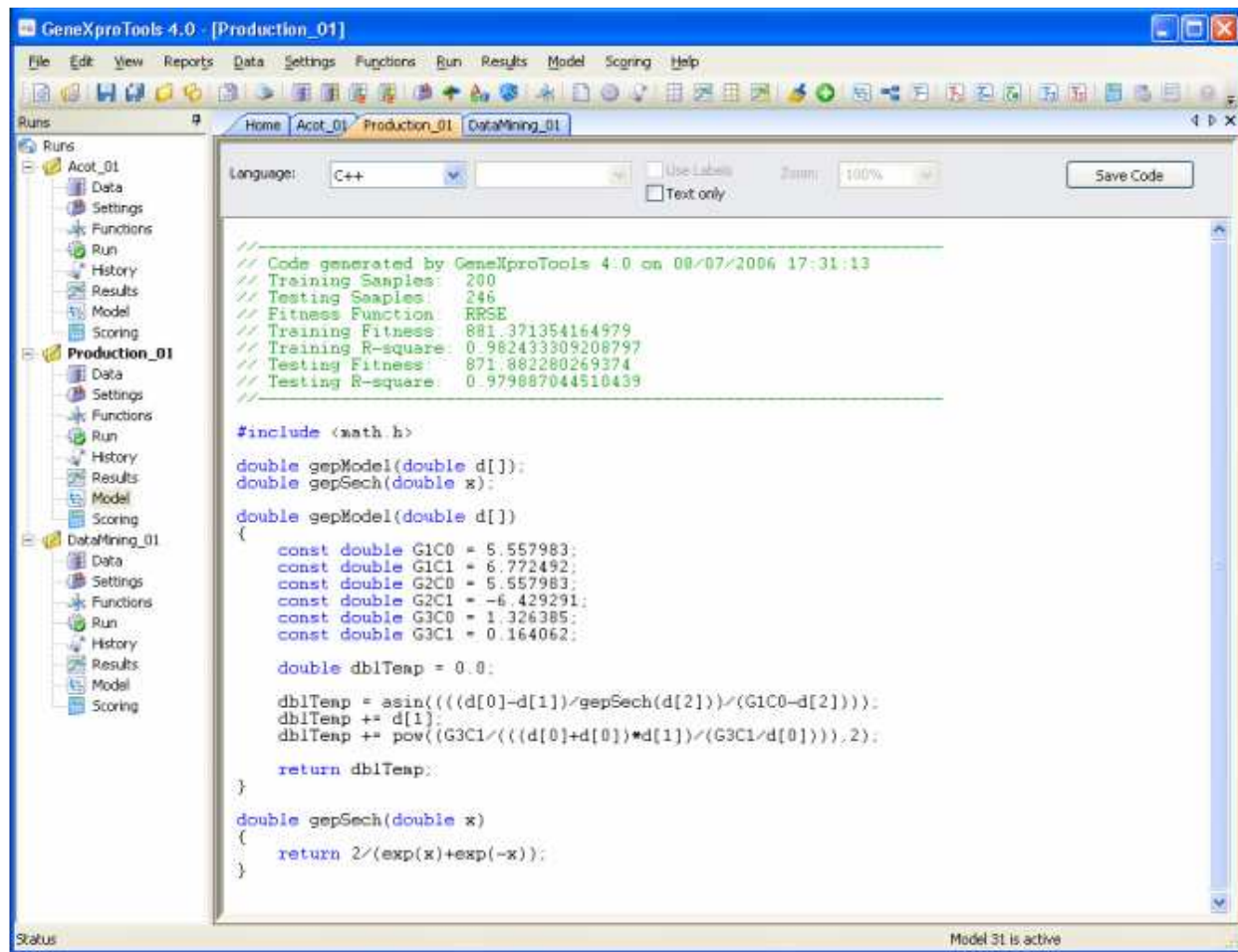
In the **Results Panel** a thorough statistical analysis of the evolved model is done on the fly both for the training and testing sets. Furthermore, the model output and the target are compared using both a table and a chart (curve fitting for Function Finding and Target/Model Comparison for Classification and Logic Synthesis).



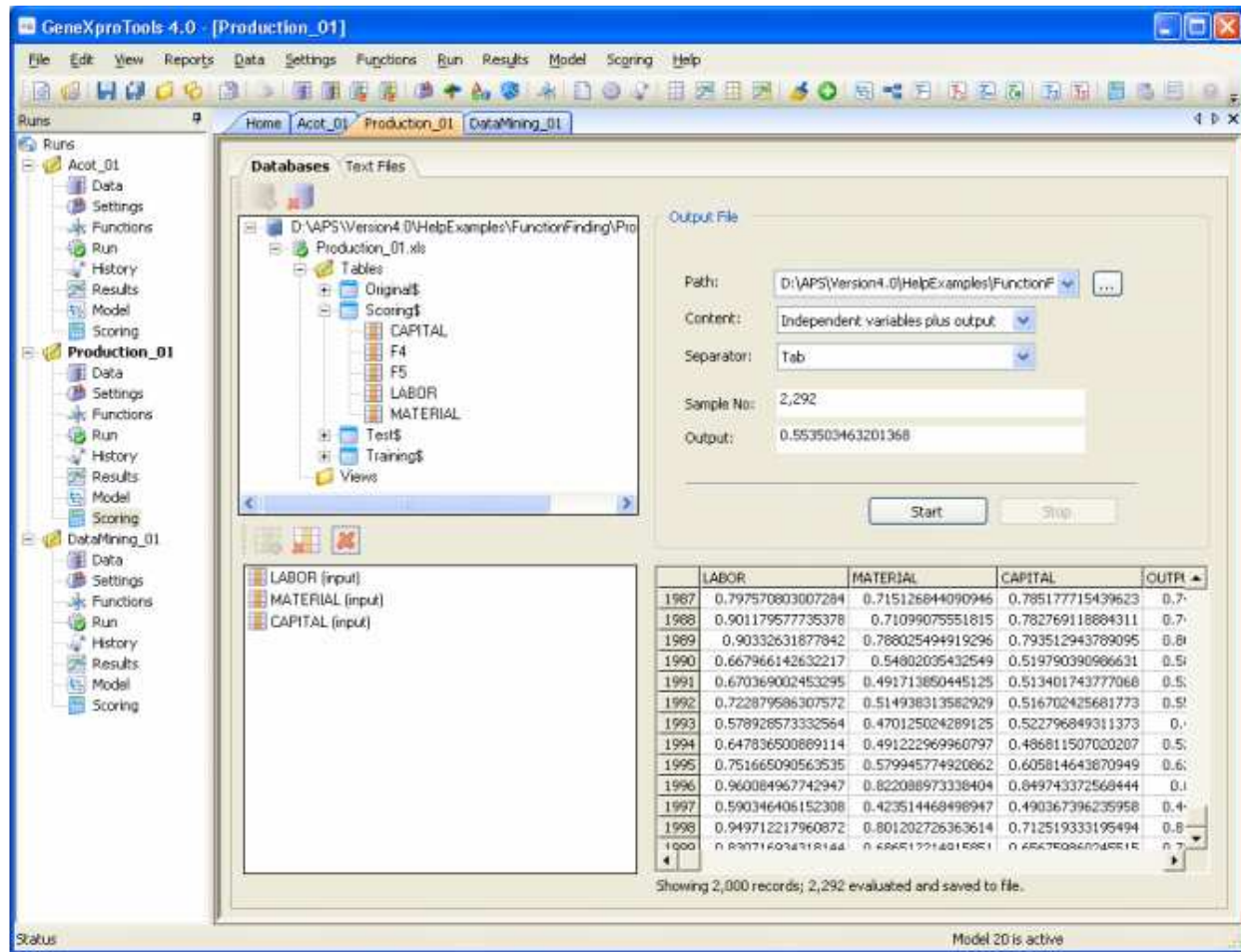
In the **Predictions Panel** is where you use the evolved model to make predictions about future events in Time Series Prediction. Here is also where a thorough statistical analysis of the evolved model takes place and, if in Testing Mode, here is also where you can test the predictive accuracy on the testing samples.



In the **Model Panel** you can see the mathematical/logical intricacies of the evolved model in the universal representation of Expression Trees or automatically translate the native Karva Code into several programming languages (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL).



In the **Scoring Panel** you can apply the evolved model immediately without leaving the GeneXproTools environment. The scoring engine of GeneXproTools uses the JavaScript code of your models to score your data. Both Excel/Databases and Text Files can be scored and the output saved to Excel (csv) or text files. In addition, for small datasets, the output can also be observed in its entirety from the Scoring Panel.



GeneXproTools Environment

Loading Data

Before evolving a model with GeneXproTools 4.0 you must first load the input data for the learning algorithm.

To Load Input Data for Modeling

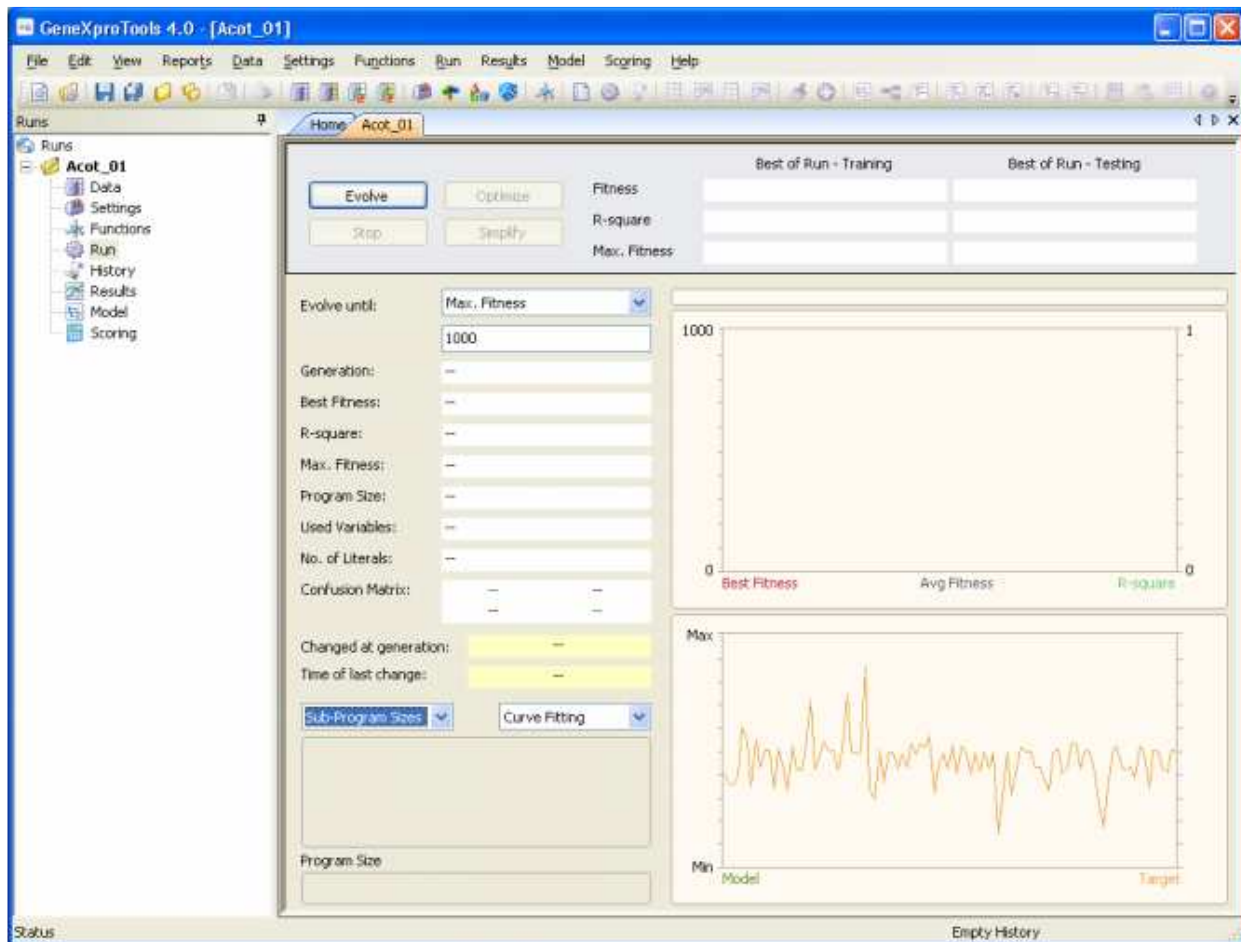
1. Click the File Menu and then choose New.
The New Run Wizard appears. You must give a name to your new run file (the default filename extension of GeneXproTools 4.0 run files is .gep) and then choose the problem category and the kind of source file. GeneXproTools 4.0 allows you to work either with Excel/databases or text files.
2. Then go to the Training Data window by clicking the Next button.
Choose the path for the training set by browsing the Open dialog box and choose the appropriate data matrix format. GeneXproTools accepts two different data formats: the standard Samples x Variables format where samples are in rows and variables in columns, with the dependent

variable or class occupying the rightmost position; and Gene Expression Matrixes commonly used in DNA microarrays studies where samples are in columns and variables in rows, with the dependent variable or class occupying the topmost position. You can visualize the loaded samples in the Run Wizard window. Note that GeneXproTools uses the Samples x Variables format and therefore all formats are automatically converted and shown in this format.

3. Then go to the Testing Data window by clicking the Next button.
Repeat the same steps of the previous point if you wish to use a testing set to evaluate the generalizing abilities of your model.

4. Click the Finish button to save your new run file.

The Save As dialog box appears and after choosing the directory where you want your new run file to be saved, the GeneXproTools 4.0 modeling environment appears. Then you just have to click the Evolve button to create a model as GeneXproTools 4.0 can automatically choose from a gallery of templates default settings that will enable you to evolve a model immediately.



GeneXproTools Environment

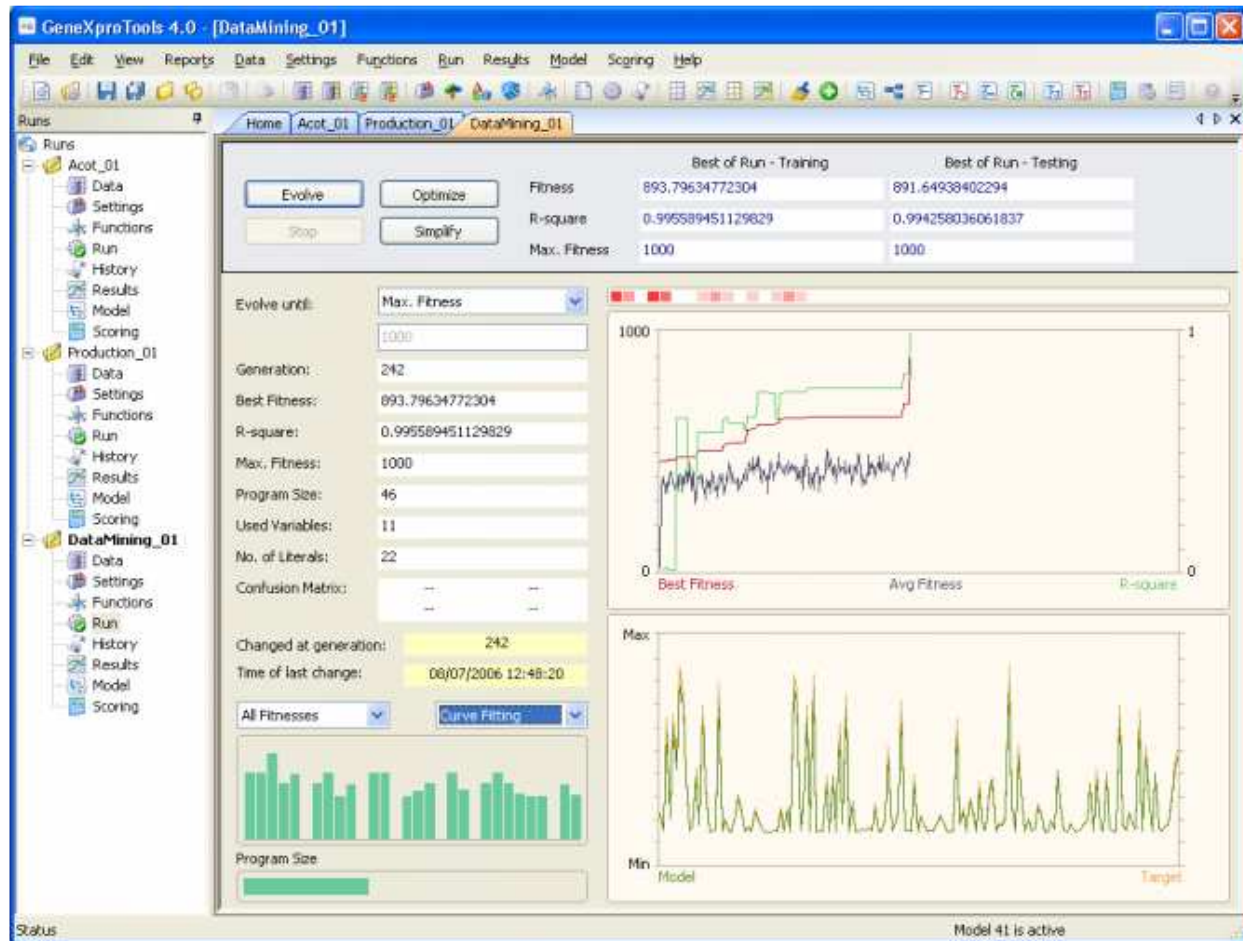
Evolving a Model

In order to evolve a model with GeneXproTools 4.0 you must first choose the problem category which is chosen while you are loading the input data for modeling. There are four different types of problem

category in GeneXproTools 4.0: [Function Finding](#), [Classification](#), [Time Series Prediction](#), and [Logic Synthesis](#), each using specific learning algorithms and requiring special features.

To Evolve a Model with GeneXproTools 4.0

1. In the Settings Panel, select the [General Settings Tab](#) and then choose the appropriate settings for your problem.
For all types of problems, the most important is the chromosome architecture, that is, the number of genes, the head size and the linking function.
2. In the [Functions Panel](#), choose the appropriate set of functions to model your data.
GeneXproTools 4.0 offers a great variety of [built-in functions](#) (a total of 279 for mathematical models and 258 for Boolean problems). Besides, GeneXproTools 4.0 also allows you to create both static and dynamic user-defined functions and evolve complex models with them. [Static UDFs](#) allow you to establish a permanent relationship between specific independent variables, for instance $UDF1 = do + d1 + d2 + d3$, and use this new UDF as an extra variable/terminal in your modeling kit. [Dynamic UDFs](#) are quite different from static UDFs as they behave exactly like the built-in functions of GeneXproTools and therefore can be used to model all kinds of relationships between variables or complex expressions. For instance, you can design DDF1 so that it will model the sum of four variables or expressions, that is, $DDF1 = (\text{expression 1}) + (\text{expression 2}) + (\text{expression 3}) + (\text{expression 4})$, where the value of each expression will depend on the context of DDF1 in the expression tree. A note of caution, though, although extremely interesting, DDFs decrease the speed of the algorithm somewhat and therefore we advise you to choose your functions from the wide set of GeneXproTools 4.0 built-in functions.
3. In the Settings Panel, select the [Fitness Function Tab](#) and choose the fitness function.
GeneXproTools 4.0 provides you with 36 built-in fitness functions for [Function Finding](#) and [Time Series Prediction](#) problems and 30 different fitness functions for [Classification](#) and [Logic Synthesis](#) problems. In addition, GeneXproTools 4.0 also allows you to create your own fitness functions and therefore allows you to explore even more solution landscapes.
4. In the Settings Panel, select the [Genetic Operators Tab](#) and choose the appropriate degree of genetic variation.
GeneXproTools 4.0 offers several genetic operators which allow you to introduce the ideal degree of genetic modification in order to allow an efficient evolution of your models. If you are overwhelmed by all these operators, please use the defaults as they work very well for all problems.
5. In the [Run Panel](#), click the Evolve button.
GeneXproTools 4.0 shows all the essential features of the best-of-generation model during the discovery process, including a Curve Fitting chart and a Heat Map of the variables that are being used by the best models. Whenever you are satisfied with the result, just press the Stop button. Your best-of-run model is ready either for analysis or scoring. And if you are still not happy with the results, you can still try to fine-tune the evolved model by pressing the Optimize button or design a simpler version of it by pressing the Simplify button.



GeneXproTools Environment

Optimizing a Model Evolved by GeneXproTools

In order to optimize a model with GeneXproTools 4.0, first you must either evolve one or enter an existing model through the [Change Seed](#) window (see the chapter [Designing New Models from Old Models](#)).

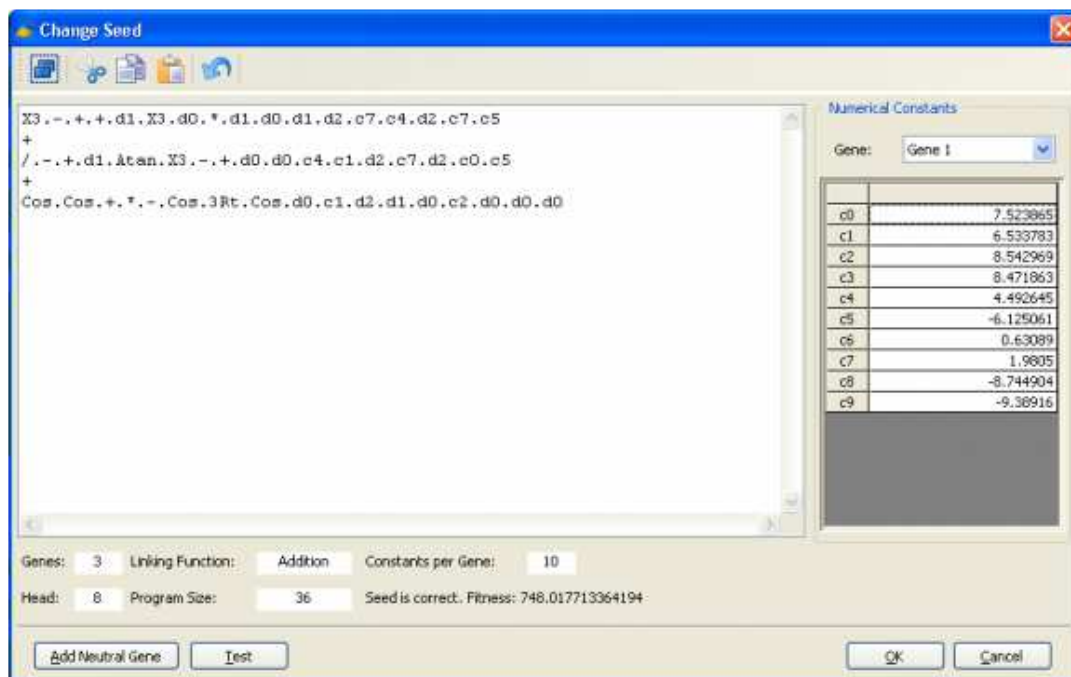
GeneXproTools 4.0 offers three different ways of optimizing an existing model. Follow the next steps to explore all of them.

To Optimize a Model Previously Evolved by GeneXproTools 4.0

1. In the [Run Panel](#), click the Optimize button.
This is the simplest way of optimizing a model using GeneXproTools 4.0. This can be done for as long as you wish or until the system seems to have stopped evolving.
2. Go to the Edit Menu and choose Change Seed.
The second simplest way of optimizing a model using GeneXproTools 4.0 is through the

introduction of an additional neutral gene. Neutral genes can also be introduced through the Edit Menu, Add Neutral Gene.

3. In the Change Seed dialog box, click Add Neutral Gene.
You will see a neutral gene being added to your model. Neutrality is important for an efficient evolution and, by introducing a neutral gene, you are giving the learning algorithm more room to play and, hopefully, a better, more complex program will evolve.
4. In the [Run Panel](#), click the Optimize button.
GeneXproTools 4.0 will hopefully transform the neutral gene into a significant term of your model. Again, you can repeat this strategy until the system stops improving.
5. For the third method, again go to the Edit Menu and choose Change Seed.
The last not so simple way of optimizing a model using GeneXproTools 4.0 is through the direct modification of the seed. For instance, you can replace any node (function or terminal) in the head by another and any terminal in the tail by another terminal; you can also fine-tune a numerical constant in your seed; and so forth. But if your modifications are worse than the original model, they will most probably be lost in the evolutionary process. By clicking the Test button you can verify that your changes didn't bring about a decrease in fitness.



GeneXproTools Environment

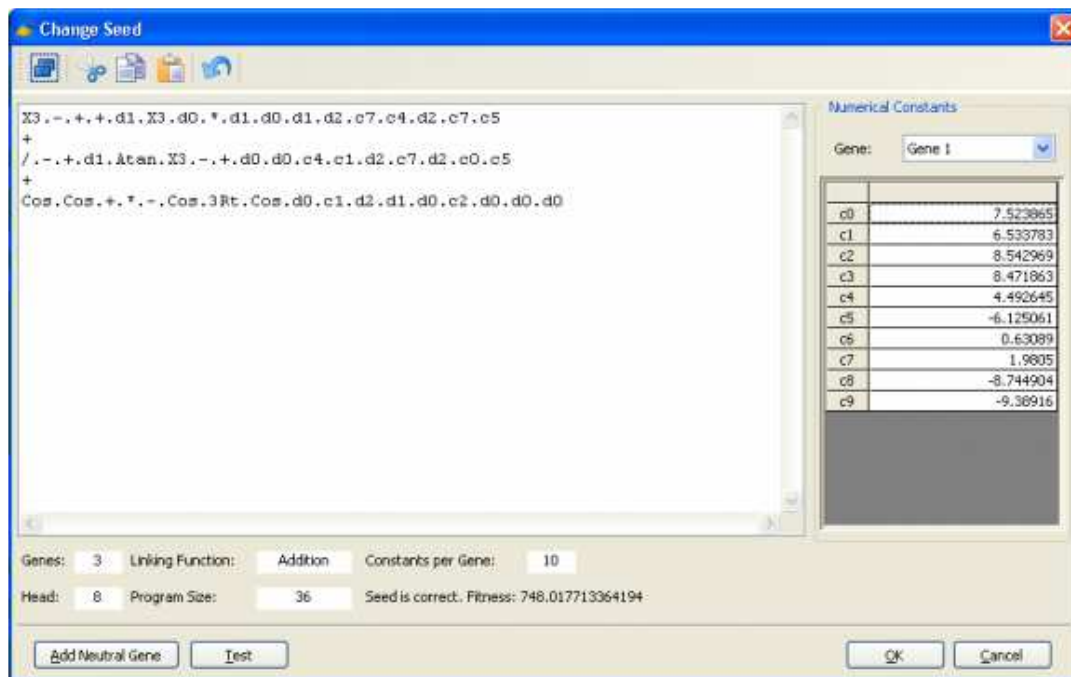
Simplifying a Model Evolved by GeneXproTools

In order to simplify a model with GeneXproTools 4.0, first you must either evolve one or enter an existing model through the [Change Seed](#) window (see the chapter [Designing New Models from Old Models](#)).

GeneXproTools 4.0 offers three different ways of simplifying an existing model. Follow the next steps to explore all of them.

To Simplify a Model Previously Evolved by GeneXproTools 4.0

1. In the [Run Panel](#), click the Simplify button.
This is the simplest way of simplifying a model using GeneXproTools 4.0. This can be done for as long as you wish or until you feel the program is as parsimonious as it will get. By monitoring the plots Avg/Best Size and Sub-Program Sizes as well as the Heat Map in the Run Panel, you will know exactly how the simplification process is progressing.
2. Go to the Settings Menu and choose Fitness Function.
By checking the box With Parsimony Pressure, GeneXproTools will select your models also according to their size.
3. Then in the [Run Panel](#), click the Optimize button.
This is very similar to the previous form of simplifying a model. The only difference is that you are always factoring the program size into the fitness, even if you are just trying to optimize a model. For most problems, we recommend you to use a fitness function without parsimony pressure in the initial stages of evolution and then apply parsimony pressure when you are happy with it by clicking the Simplify button.
4. Go to the Edit Menu and choose Change Seed.
The last not so simple way of simplifying a model using GeneXproTools 4.0 is doing it by hand through the direct modification of the seed. For instance, you can replace any node (function or terminal) in the head by another and any terminal in the tail by another terminal; you can also fine-tune a numerical constant in your seed; and so forth. Note, however, that if your modifications are worse than the original model, they will most probably be lost in the evolutionary process. By clicking the Test button you can verify that your changes didn't result in a decrease in fitness.



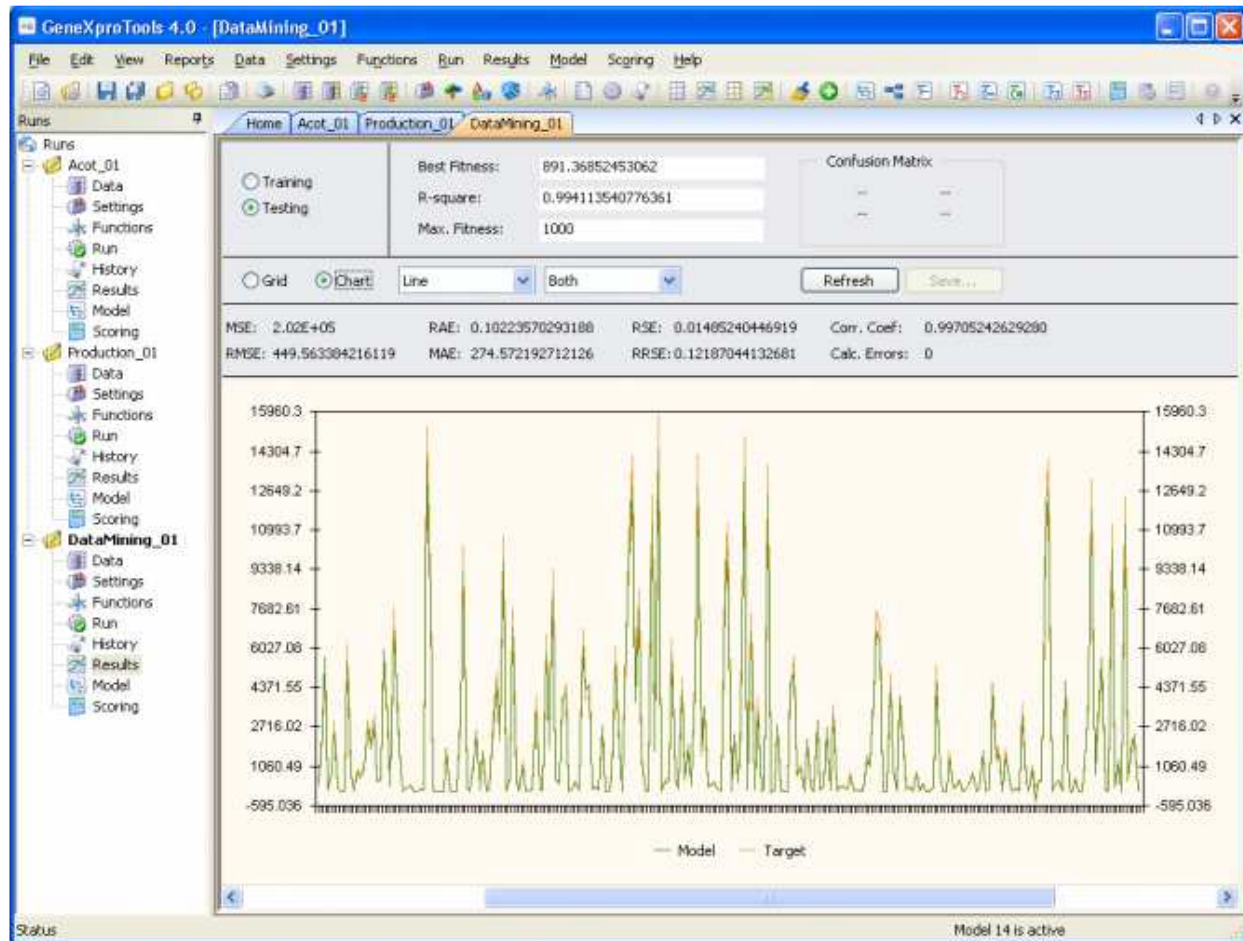
GeneXproTools Environment

Evaluating the Accuracy of a Model

GeneXproTools 4.0 offers an extremely complete package of statistical tools which you can use to evaluate the accuracy of your models without leaving the GeneXproTools modeling environment.

To Evaluate the Accuracy of a Model with GeneXproTools 4.0

1. In the [Run Panel](#), during a run, check the generation values of Best Fitness and R-square/Accuracy.
For Function Finding and Time Series Prediction both the fitness and R-square of the best model of each generation are shown and plotted in a chart during a run. For Classification and Logic Synthesis the classification accuracy is shown instead.
2. In the [Run Panel](#), after a run, check the best-of-run values of Fitness and R-square/Accuracy.
After a run, both the fitness and R-square/Accuracy of the best model discovered in the run are given both for the training and testing sets. Since the testing set is not used during training, the values of fitness and R-square/Accuracy on the testing set are a good indicator of the generalizing capabilities of your model.
3. In the [Results Panel](#), check the values of additional statistical functions.
Depending on the problem category, a varied set of statistical evaluations of your model are shown to you while you are analyzing your new model. These statistics, performed both for the training and testing sets, include: mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, correlation coefficient, and R-square for all types of problem category; and the classification error, classification accuracy, confusion matrix (true positives, true negatives, false positives, and false negatives), sensitivity, specificity, positive predictive value, and negative predictive value for Classification and Logic Synthesis.
4. In the [Report Panel](#), check the values of all the statistical functions performed by GeneXproTools in the Results Panel for your active model.
Depending on the problem category, a varied set of statistical evaluations of your model are shown in the Report Panel. These statistics, performed both for the training and testing sets, include: mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, correlation coefficient, and R-square for all types of problem category; and the classification error, classification accuracy, confusion matrix (true positives, true negatives, false positives, and false negatives), sensitivity, specificity, positive predictive value, and negative predictive value for Classification and Logic Synthesis.



GeneXproTools Environment

Testing the Generalizing Capabilities of a Model

In order to test the generalizing capabilities of your model, you must provide an additional set of sample data (testing set) which is only used after the learning process is complete.

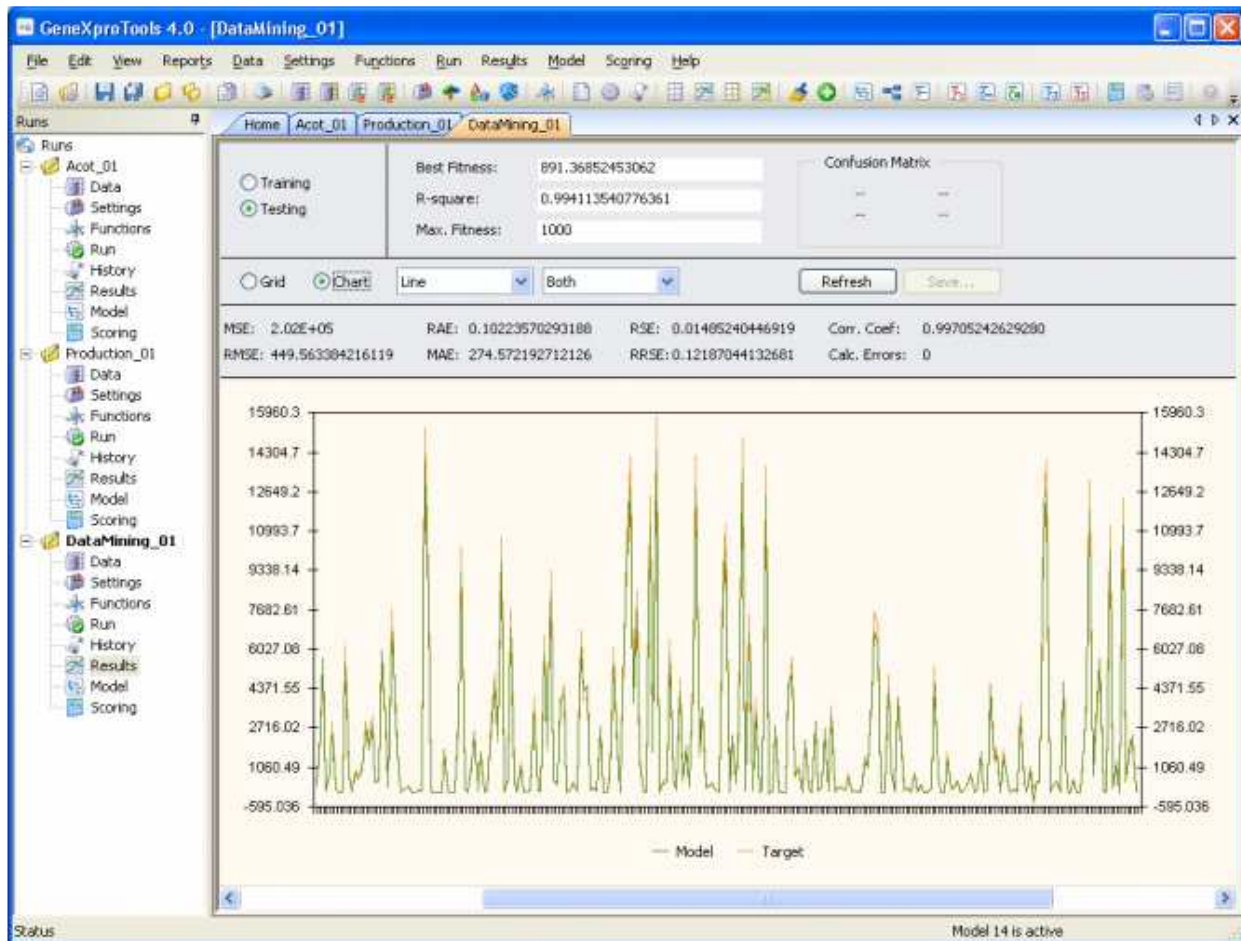
To Test the Generalizing Capabilities of a Model with GeneXproTools 4.0

1. On the Data Menu, select Add Testing Data.
The Change/Add Testing Data wizard appears. It asks for the type of data source and the path and format of your testing file. Click on Finish to load the testing set.
2. In the **Run Panel**, after a run, check the best-of-run values of Fitness and R-square/Accuracy obtained for the testing set.
After a run, GeneXproTools evaluates the performance of the evolved model on the testing set and shows both the fitness and the R-square/Accuracy obtained on the testing set.
3. In the **Results Panel**, select the Testing option.
Depending on the problem category, you can analyze a varied set of **statistical functions** of your

model on the testing set. These statistics include: mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, correlation coefficient, and R-square for all types of problem category; and the classification error, classification accuracy, confusion matrix (true positives, true negatives, false positives, and false negatives), sensitivity, specificity, positive predictive value, and negative predictive value for Classification and Logic Synthesis.

4. In the [Report Panel](#), check the entry Statistics - Testing.

The complete set of statistical functions evaluated by GeneXproTools in the Results Panel is shown in the Report Panel. These statistics include: mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, correlation coefficient, and R-square for all types of problem category; and the classification error, classification accuracy, confusion matrix (true positives, true negatives, false positives, and false negatives), sensitivity, specificity, positive predictive value, and negative predictive value for Classification and Logic Synthesis problems.



◀ | ▶

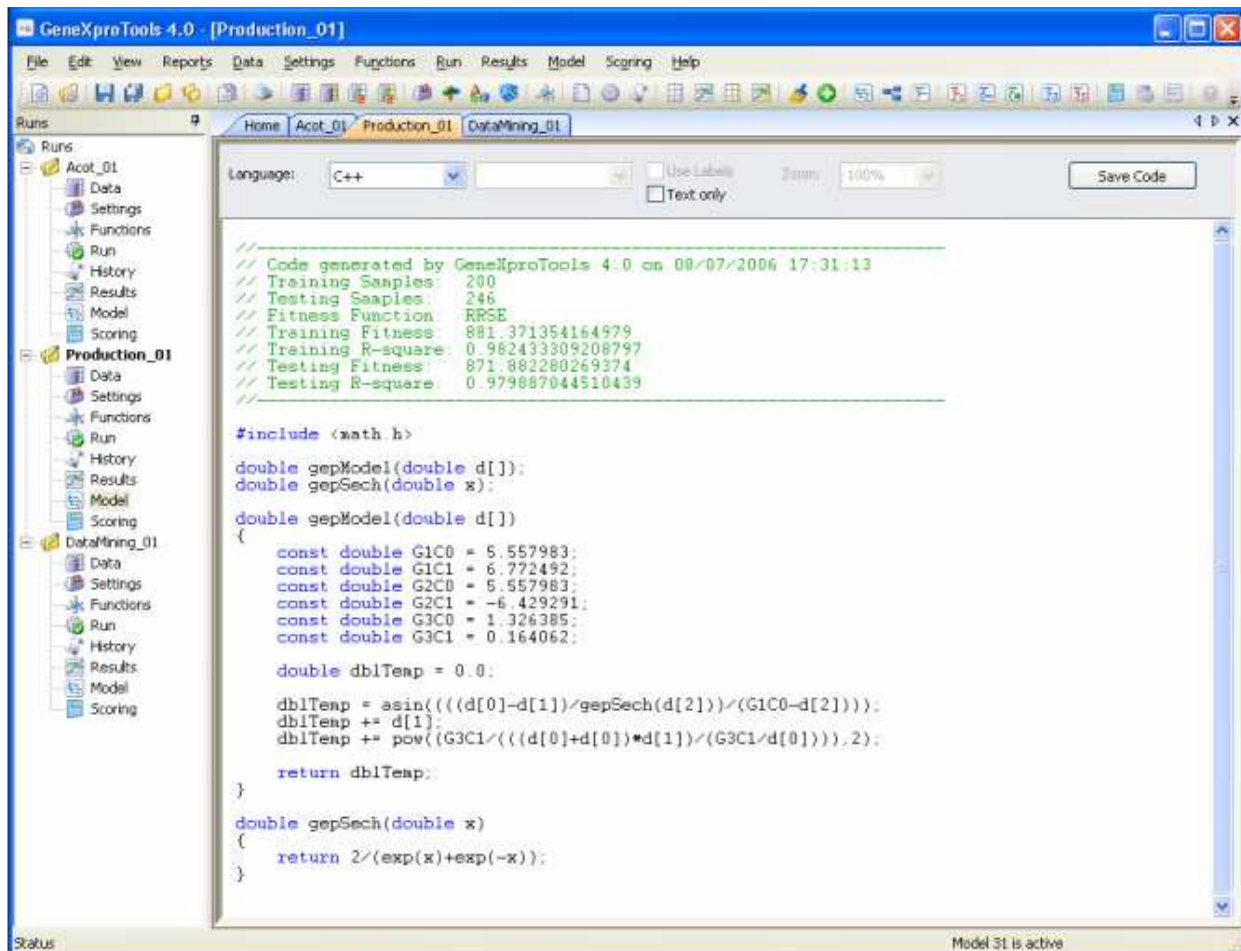
GeneXproTools Environment

Generating Code Automatically

In order to generate code automatically with GeneXproTools 4.0, first you must evolve a model. All the models evolved by GeneXproTools are in the native [Karva notation](#) which can be immediately translated into virtually any programming language through the use of grammars. GeneXproTools 4.0 provides already a total of 16 [built-in grammars](#) but others can be added by the user (see the section [User Defined Grammars](#)).

To Generate Code Automatically with GeneXproTools 4.0

1. On the GeneXproTools modeling environment, click the [Model Panel](#).
GeneXproTools 4.0 has the following [built-in grammars](#): Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL. Besides, GeneXproTools 4.0 has a built-in parser which allows the visualization of the evolved models as [expression trees](#).
2. On the Language list box, select Expression Tree.
A clear representation of your model appears in the code window. The expression tree can be copied to the clipboard or saved to disk.
3. On the Language list box, select the language of your choice.
The code for your model is automatically generated by GeneXproTools and appears in the code window. The code can be copied to the clipboard or saved to disk.
4. For Logic Synthesis models, on the Language list box, select the gate type.
With GeneXproTools 4.0 you can port any Boolean function into six different [Logical Systems](#) (All Gates, Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System). The code for your model is automatically generated by GeneXproTools and appears in the code window. The code can be copied to the clipboard or saved to disk.





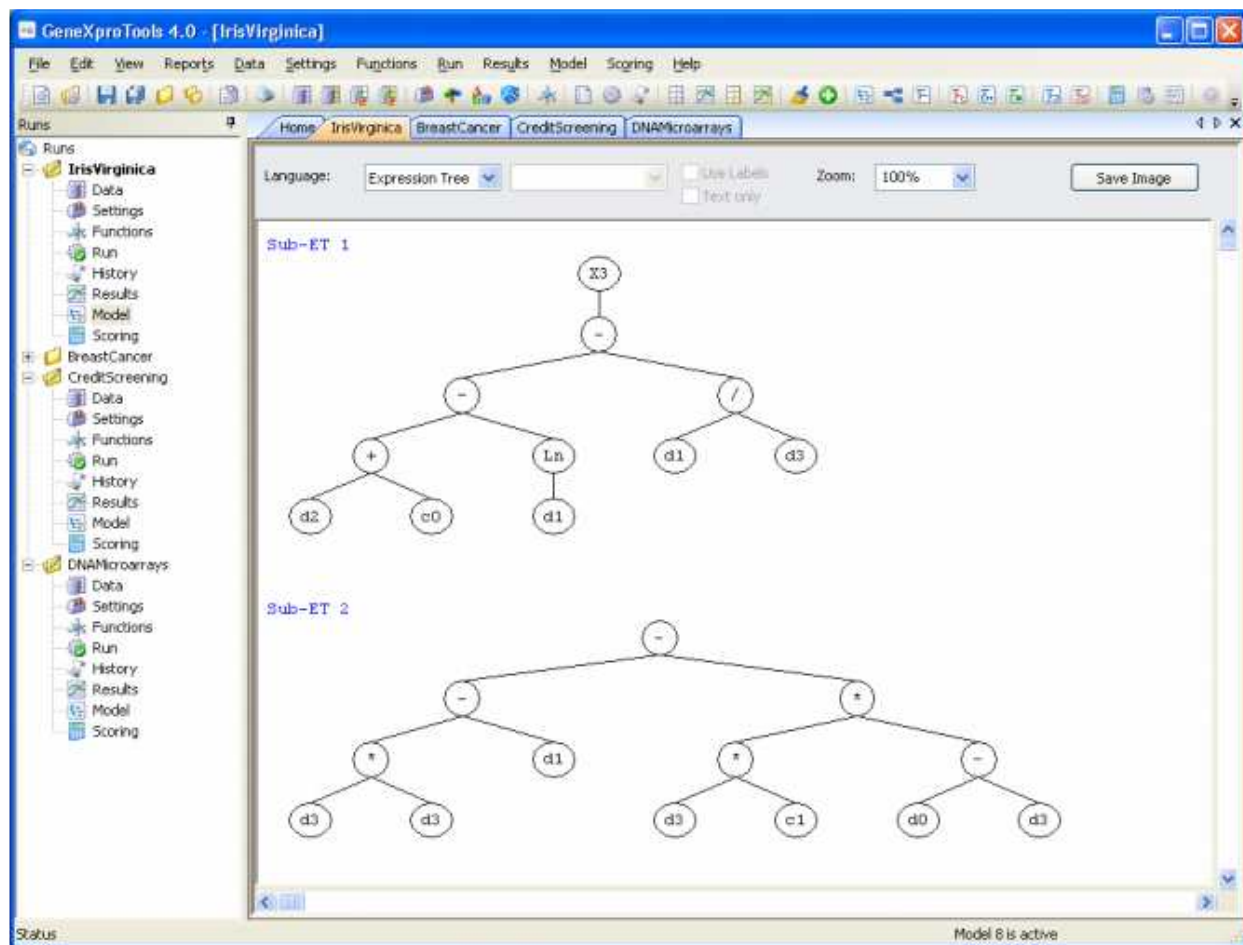
GeneXproTools Environment

Visualizing Parse Trees

GeneXproTools 4.0 comes with a built-in parser for visualizing clearly your model as a parse tree. After evolving a model, the native code of GeneXproTools can be used to draw visually appealing expression trees.

To Visualize your Model as a Parse Tree with GeneXproTools 4.0

1. On the GeneXproTools modeling environment, click the [Model Panel](#).
2. On the Language list box, select Expression Tree.
A clear diagram representation of your model appears in the code window. The expression tree can be copied to the clipboard or saved to disk.



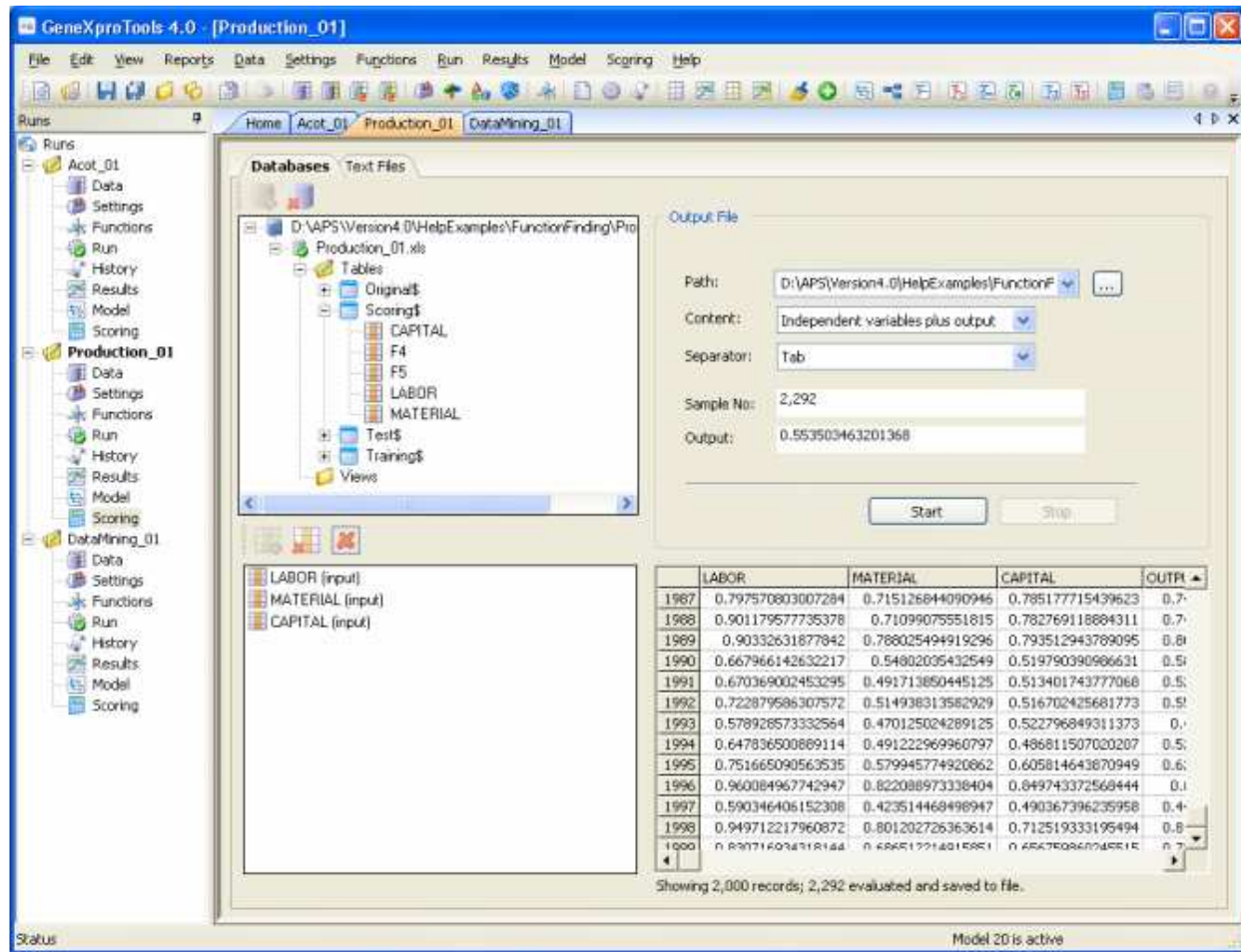
GeneXproTools Environment

Scoring a Model

GeneXproTools 4.0 comes equipped with a scoring engine that allows the immediate scoring of your databases/Excel or text files with the evolved model. The scoring engine of GeneXproTools uses the JavaScript code of your models to score your data. Usually a model is used multiple times to score different databases.

To Score a Database/Excel or a Text File with GeneXproTools 4.0

1. Open the GeneXproTools file with the model you want to use to make predictions.
All models evolved by GeneXproTools can be used to score a database. The scoring process can take place immediately after the creation of the model or from existing models.
2. On the Scoring Menu, select the type of file.
The scoring engine of GeneXproTools allows both the use of databases/Excel and text files as input for the scoring engine.
3. Prepare your data for scoring.
Make sure that the data is consistent with the training data that was used to create the model. This means that all the data fields that were used as inputs to create the model must be available for the model during the scoring process.
4. After filling all the relevant fields in the [Scoring Panel](#), click the Start button.
The scoring results are saved to Excel (csv) or text files and are also shown in the Scoring Panel. For large datasets, though, in the Scoring Panel only the outputs of the first 2000 records are shown.

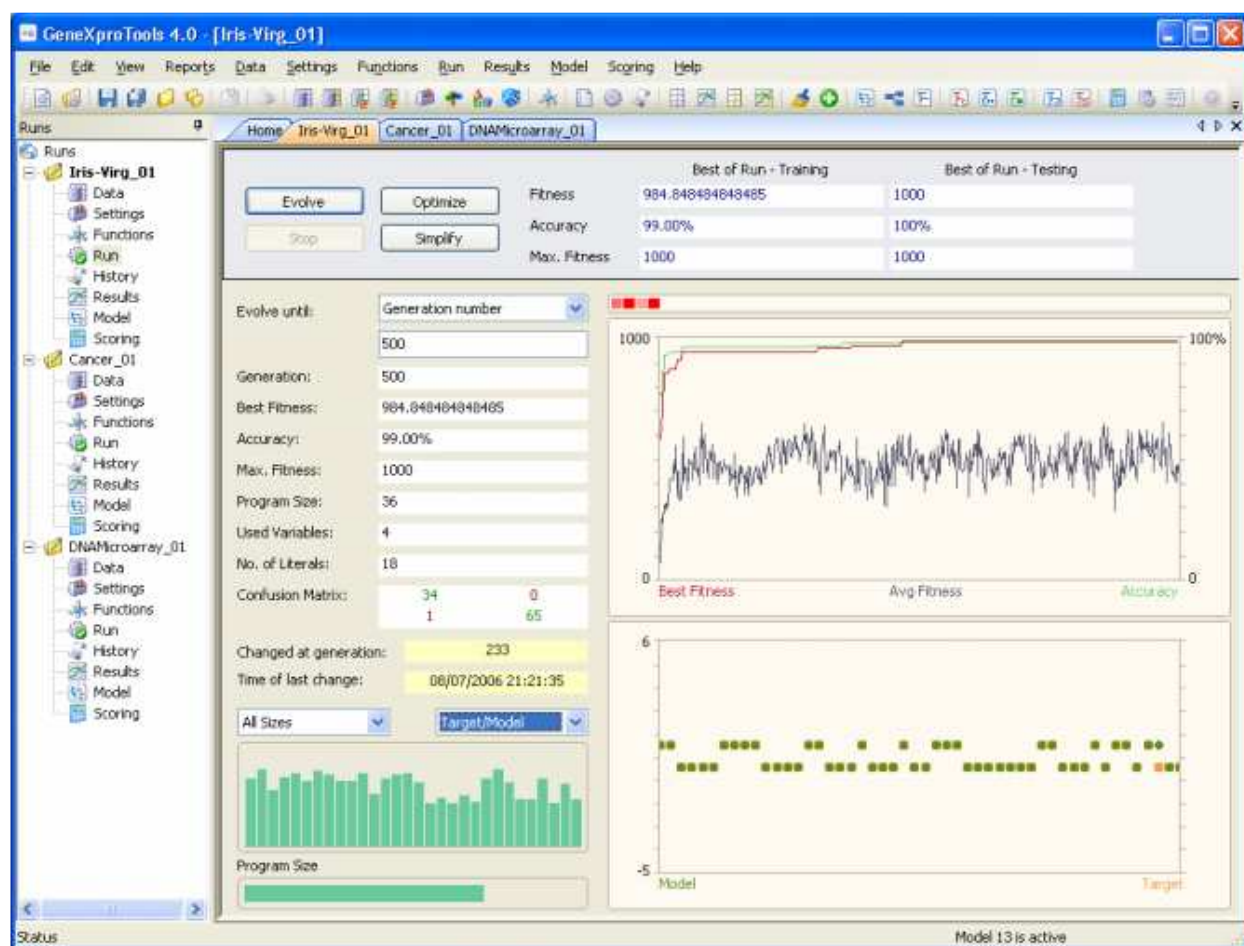


GeneXproTools Environment

Modeling Tools

The modeling environment of GeneXproTools provides a wide range of tools especially created for handling:

- Function Finding problems.
- Classification problems.
- Time Series Analysis.
- Logic Synthesis.



GeneXproTools Environment

Function Finding Tools

The successful creation of Function Finding models requires the use of very sharp modeling tools and the most important are the following:

- Data Screening Engine
- Data Visualization Tool
- Built-in Fitness Functions
- User Defined Fitness Functions
- Built-in Mathematical Functions
- Function Selection Tool
- Dynamic User Defined Functions
- Static User Defined Functions
- Learning Algorithms
- Monitoring the Modeling Process
- Analyzing Intermediate Models

- [Simplifying a Model](#)
- [Comparing Actual and Predicted Values](#)
- [Essential Statistics](#)
- [Modeling from Seed Models](#)
- [Adding a Neutral Gene](#)
- [Complexity Increase Engine](#)
- [Visualizing Models as Parse Trees](#)
- [Generating Code Automatically Using Built-in Grammars](#)
- [Generating Code Automatically Using User Defined Grammars](#)
- [Scoring Engine](#)

Data Screening Engine

GeneXproTools accepts two different data formats: the standard Samples x Variables format where samples are in rows and variables in columns, with the dependent variable occupying the rightmost position; and Gene Expression Matrixes commonly used in DNA microarray studies where samples are in columns and variables in rows, with the dependent variable occupying the topmost position. Note, however, that GeneXproTools uses the Samples x Variables format and therefore all formats are automatically converted and shown in this format.

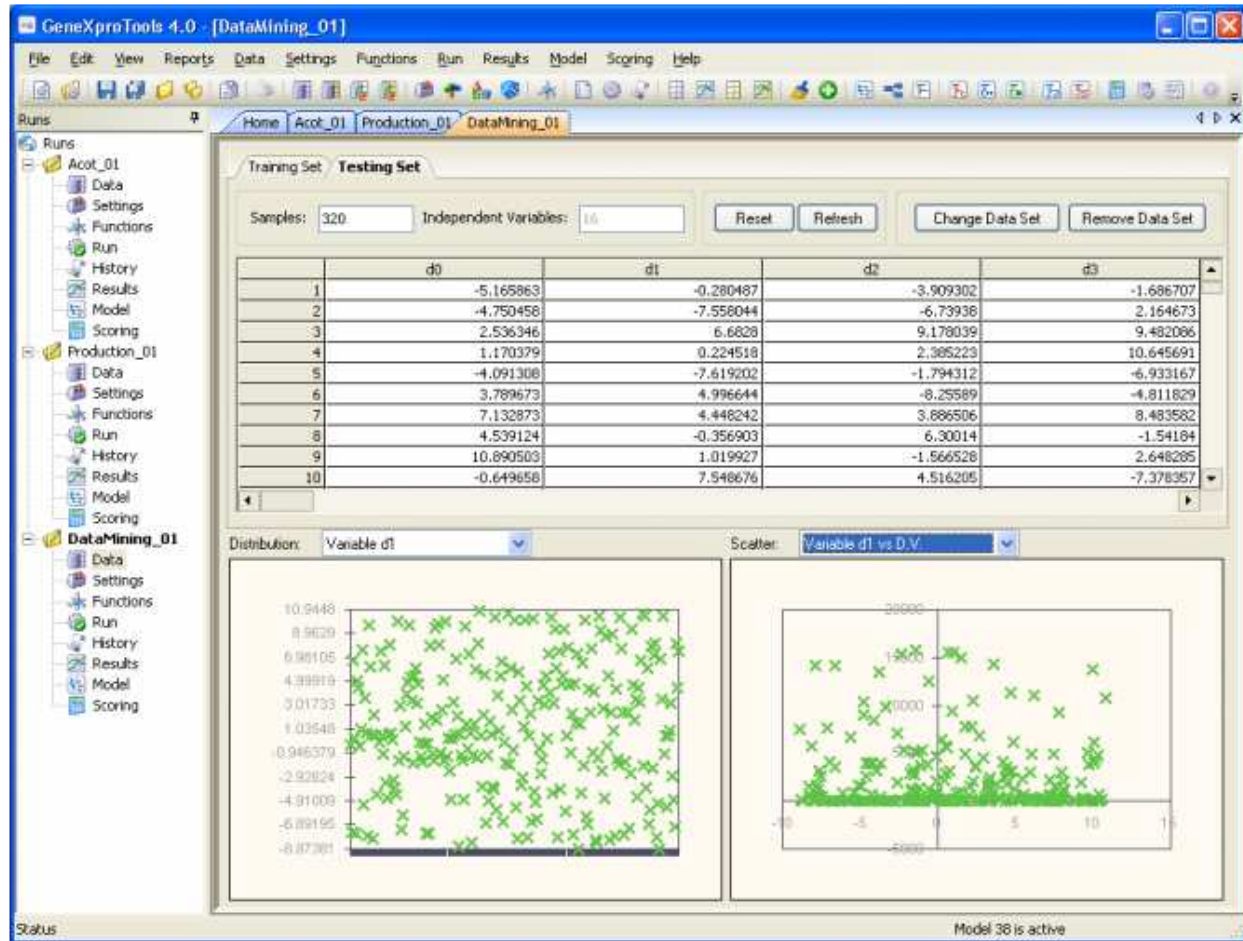
The data screening engine of GeneXproTools 4.0 checks the validity of all the data sets used both in the creation and testing of the model. The data screening engine is operative every time you load data sets either for training or testing.

Function finding problems require data files with numerical inputs of any kind, but it is common for the files to have missing or nominal values which render them useless for modeling or testing. GeneXproTools 4.0 data screening engine can help you identify these faulty samples and correct them.

Data Visualization Tool

The data visualization tool of GeneXproTools 4.0 enables you to analyze the distribution of all the inputs and to spot simple relationships between the independent variables and the dependent variable.

The visualization of both the distribution and relationships between variables is a valuable modeling tool as it can help you detect outliers or simple relationships between your variables that can be used to evolve more efficient models.



Built-in Fitness Functions

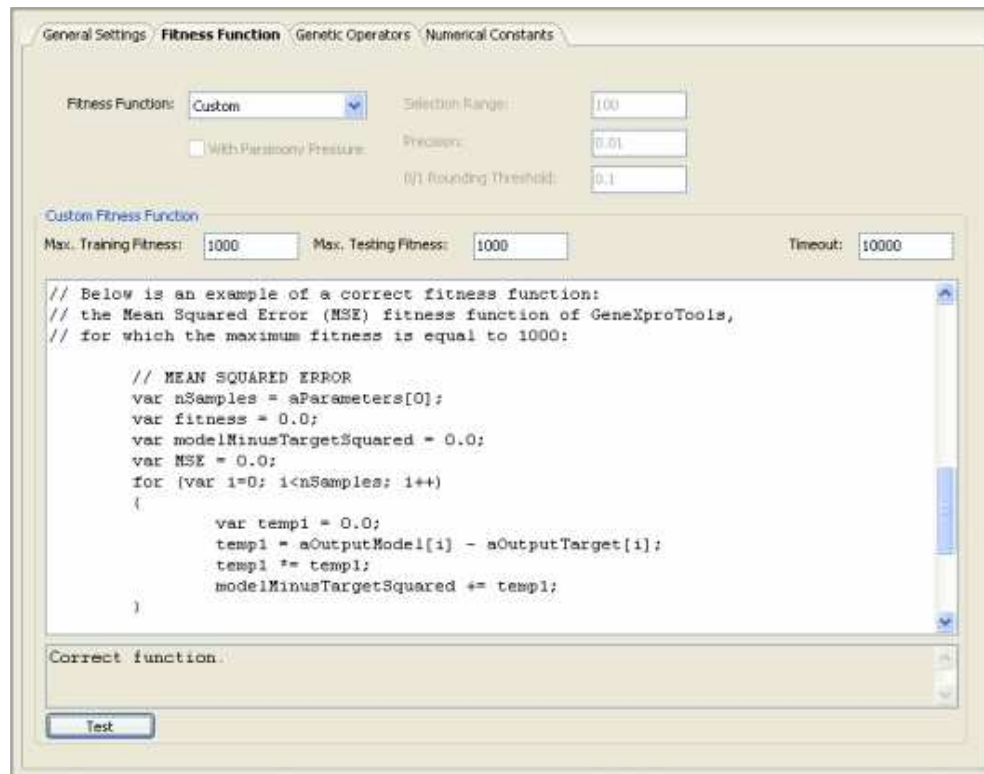
GeneXproTools 4.0 offers a total of 36 [built-in fitness functions](#) based on well-known statistical functions.

Fitness functions are fundamental modeling tools for they determine the nature of the search landscape. And different fitness functions open and explore different paths in the solution space. The fitness functions of GeneXproTools 4.0 are named as follows: [Absolute with SR](#), [Absolute/Hits](#), [MSE](#), [RMSE](#), [MAE](#), [RSE](#), [RRSE](#), [RAE](#), [Relative with SR](#), [Relative/Hits](#), [rMSE](#), [rRMSE](#), [rMAE](#), [rRSE](#), [rRRSE](#), [rRAE](#), [R-square](#), and [Correlation Coefficient](#). Furthermore, all these fitness functions have their own counterpart with [parsimony pressure](#).

User Defined Fitness Functions

Despite the wide set of GeneXproTools [built-in fitness functions](#), some users sometimes want to experiment with fitness functions of their own. GeneXproTools 4.0 gives the user the possibility of creating custom tailored fitness functions and evolve models with them.

The code for the [custom fitness function](#) must be in JavaScript and is written in the Custom Fitness Function window and can be tested before evolving a model with it.



Built-in Mathematical Functions

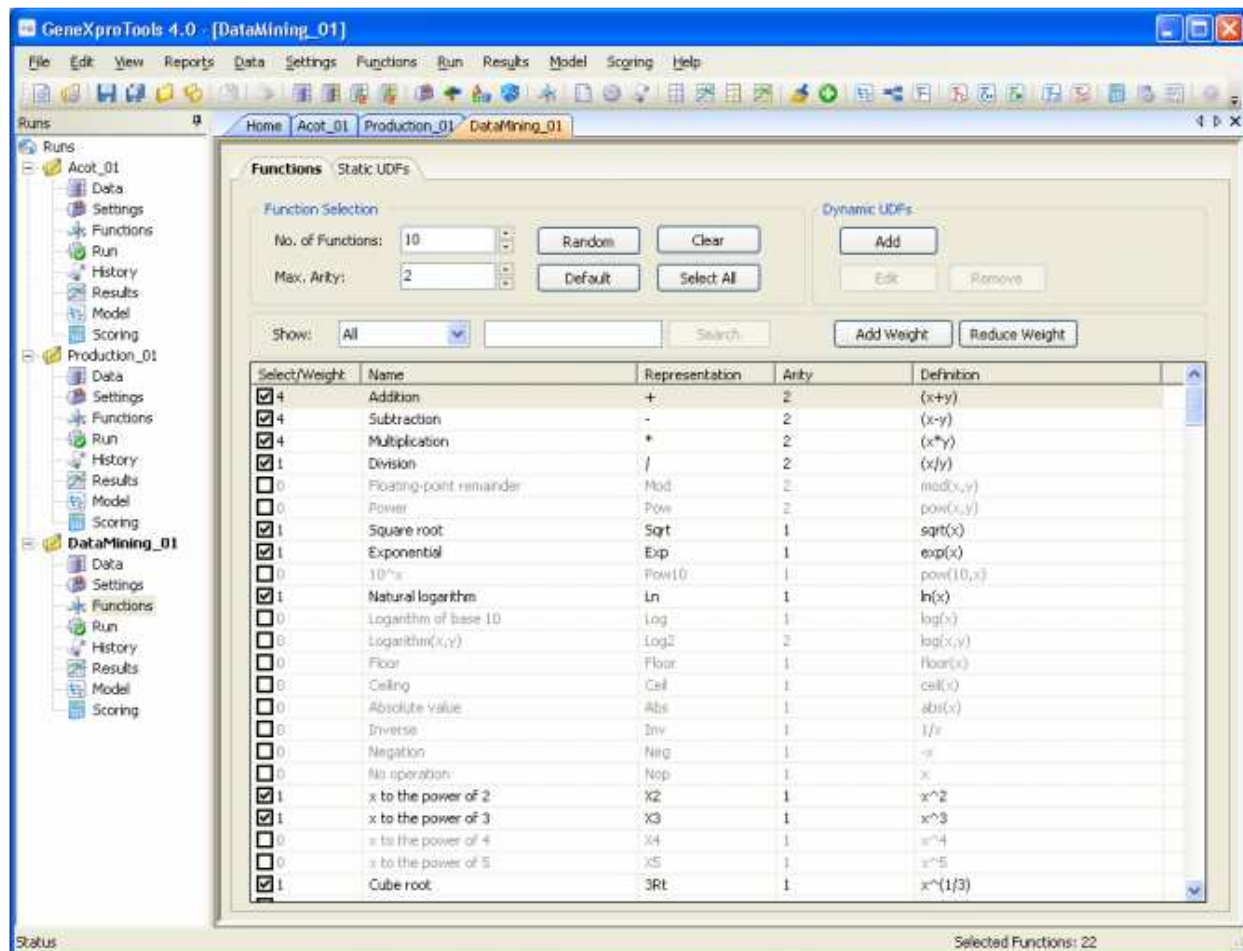
GeneXproTools 4.0 offers a total of 279 [built-in mathematical functions](#). This wide set of mathematical functions allows the evolution of complex and rigorous models quickly built with the most appropriate functions.

From the simple arithmetic operators to complex mathematical functions available in most programming languages to more complex mathematical functions commonly used by engineers and scientists, the modeling algorithms of GeneXproTools 4.0 enable you the integration of the most appropriate functions in your models.

Function Selection Tool

For some problems, the simple arithmetic operators are more than enough to create extremely accurate and elegant models. But for other problems more complex functions are necessary to create good models and when inside knowledge of the problem at hand is not producing the expected results, it might be interesting to experiment with different combinations of functions. The Function Selection Tool of GeneXproTools 4.0 helps you to experiment with different function sets very quickly either by pressing the Random button in the Functions Panel or by selecting certain sub-sets of functions.

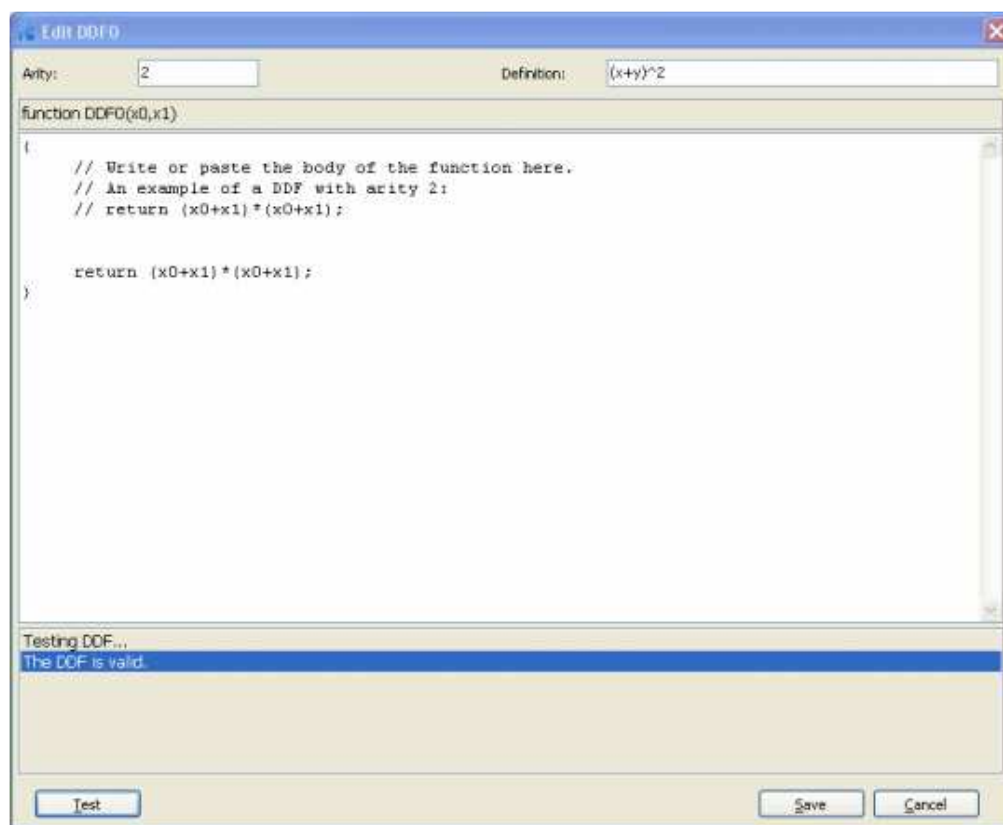
In addition, although you can always handcraft your function set so that it is well-balanced relatively to the number of independent variables in the problem at hand, in GeneXproTools 4.0 this is no longer essential as GeneXproTools balances automatically all function sets. This is particularly important for high-dimensional datasets, say above 100-1000 variables.



Dynamic User Defined Functions (DDFs)

Despite the wide set of GeneXproTools [built-in mathematical functions](#), some users sometimes want to model with different ones. GeneXproTools 4.0 gives the user the possibility of creating custom tailored functions and evolve models with them. A note of warning though: the use of dynamic UDFs slows considerably the evolutionary process and therefore should be used with moderation.

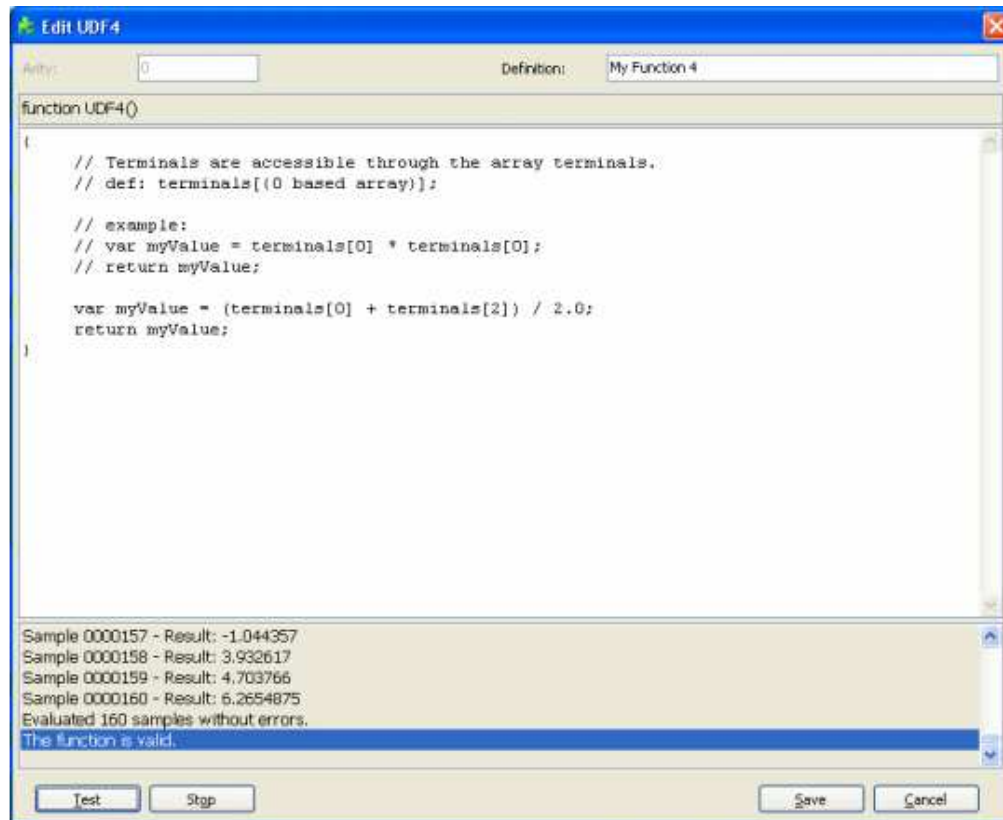
The code for the DDFs must be in JavaScript and is written in the Edit DDF window (in the Functions Panel, select the Functions Tab and then click Add in the Dynamic UDFs tool box).



Static User Defined Functions (UDFs)

Sometimes it is possible to spot relatively simple relationships between certain variables in your data, and GeneXproTools 4.0 allows you to use this information to build custom tailored functions (UDFs). These functions allow the discovery of more complex models composed of several simpler models.

The code for the UDFs must be in JavaScript and is written in the Edit UDF window (in the Functions Panel, select the Static UDFs Tab and then click Add in the Static UDFs tool box).



Learning Algorithms

GeneXproTools 4.0 is equipped with two different [learning algorithms](#) for evolving complex nonlinear regression models.

The first is the fastest and simplest of the two. The second algorithm is very similar to the first, with the difference that it gives the user the possibility of choosing the range and type of numerical constants that will be used during the learning process. And because numerical constants are crucial to a wide set of problems, the second algorithm is the default in GeneXproTools 4.0. Keep in mind, however, that for problems where numerical constants are not necessary it pays to use the first algorithm as the models are not only much simpler but also the evolutionary process is much quicker.

In order to evolve models using the second algorithm, select Numerical Constants in the Settings Panel and then check the box that activates this algorithm. You will notice that additional parameters become available, including a small set of genetic operators especially developed for handling numerical constants (if you are not familiar with these operators, please use the default values for they work very well in all cases). To use the first algorithm, just uncheck the Use Random Numerical Constants box.

Monitoring the Modeling Process

GeneXproTools 4.0 allows you to monitor the modeling process by plotting the essential parameters of a run during the discovery process, including average fitness of the population and the fitness and R-square of the best-of-generation model.

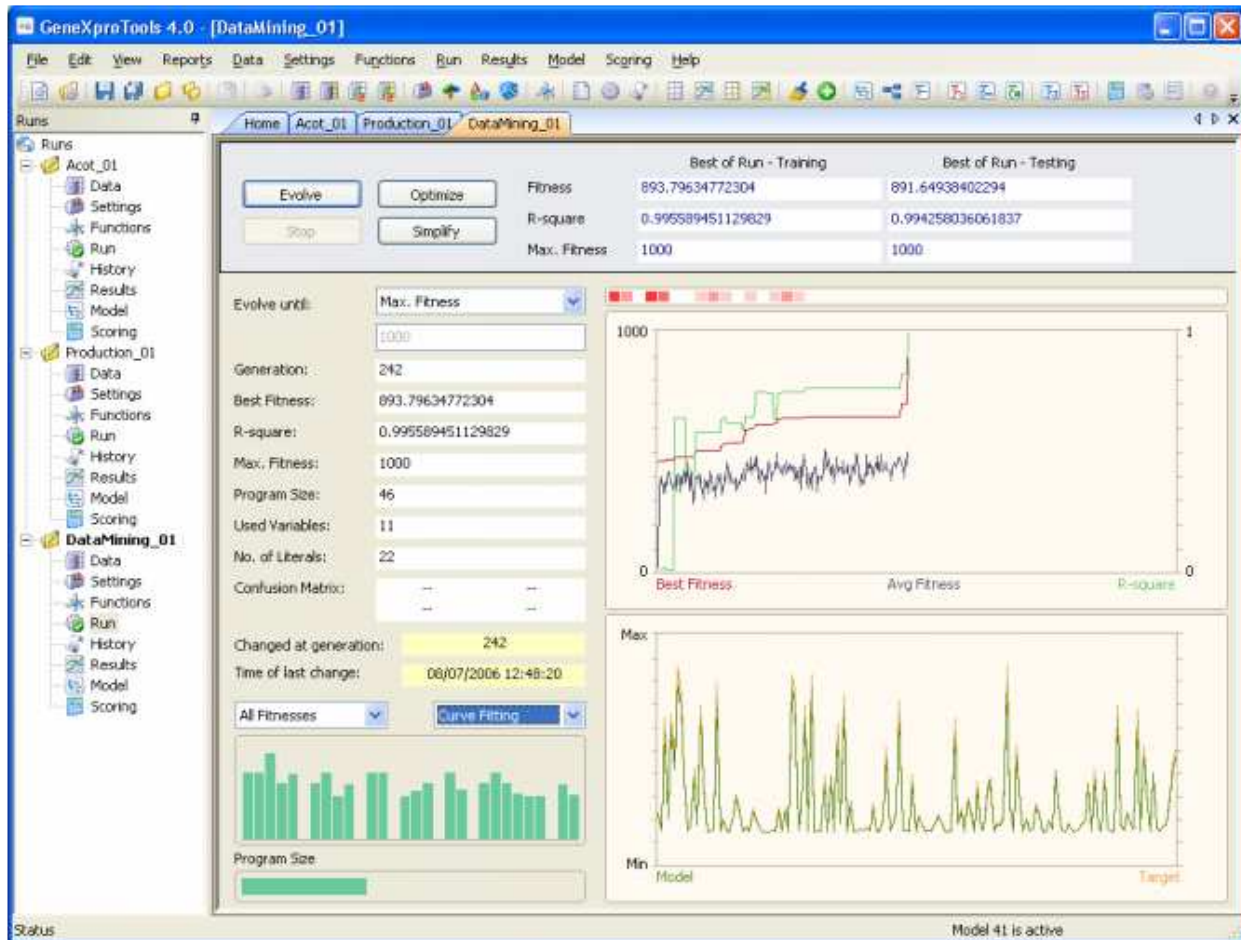
In addition, GeneXproTools allows you to visualize the modeling process by plotting the essential features of a run during the discovery process. In the [Run Panel](#), a total of eight different charts are shown:

1. The Variables Usage map.
This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square you can access the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic.
2. The Evolutionary Dynamics chart.
This chart shows the average fitness of the population plus the fitness and R-square of the best-of-generation model.
3. The Average/Best Size chart.
This chart shows the average program size of the population versus the program size of the best-of-generation model. This chart is especially useful during simplification and can be activated any time during evolution by selecting Avg/Best Size in the rightmost combo box in the bottom.
4. The Curve Fitting chart.
This chart plots the first 250 data points of the training set and shows how well the evolving models are fitting the target. The Curve Fitting chart can be activated any time during evolution by selecting Curve Fitting in the rightmost combo box in the bottom.
5. The Sub-Program Sizes chart.
This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be activated any time during evolution by selecting Sub-Program Sizes in the leftmost combo box in the bottom.
6. The All Sizes chart.
This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be activated any time during evolution by selecting All Sizes in the leftmost combo box in the bottom.
7. The All Fitnesses chart.
This chart shows the fitnesses of all the models in the population. Not only after a run but also

during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be activated any time during evolution by selecting All Fitnesses in the leftmost combo box in the bottom.

8. The Program Size chart.

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model.



The evolutionary process can be stopped whenever you are satisfied with the results by pressing the Stop button or you can use one of the stop criteria of GeneXproTools and have GeneXproTools stop automatically after a certain condition had been met.

When the evolutionary process stops, the best-of-run model is ready either for analysis or scoring. And if you are still not happy with the results, you can still try to fine-tune the evolved model by pressing the Optimize button and repeat this process until you are completely satisfied with your model. In addition, you might also wish to try and simplify your model and for that you just have to press the Simplify button.

Analyzing Intermediate Models

GeneXproTools 4.0 gives you the possibility of gaining some insights into the modeling process by analyzing all the best-of-generation intermediate models.

All the best-of-generation models discovered during a run are saved and you can pick them up for analysis at the [History Panel](#) by checking the model you are interested in. Each of these intermediate models can then be analyzed exactly as you do for the best-of-run model, that is, you can check their

performance in the testing set, check their vital statistics, automatically generate code with them, visualize their parse trees, use them as seed to create better models from them and so forth.

The screenshot shows the GeneXproTools 4.0 interface with the 'DataMining_01' project selected. The 'Runs' tree on the left shows the project structure. The main window displays a table of model statistics for models 25 through 50. Model 50 is highlighted as the active model.

Model	Generation	Program Size	Training Fitness	Testing Fitness	Training R-square	Testing R-square
Model 25	91	10	988.868703876944	987.322735210876	0.999932575757036	0.999912754565762
Model 26	93	11	989.278710382097	988.168711656403	0.999980670423799	0.99997686062578
Model 27	98	15	989.315649887797	988.347610017749	0.999985335046974	0.99998093399008
Model 28	102	10	989.549630482771	988.309583873991	0.999924324010099	0.999902256977037
Model 29	107	9	989.573380524507	988.535823900202	0.999981728091322	0.999974683529395
Model 30	125	9	989.618635660653	988.601975402233	0.999974296732552	0.999966199267095
Model 31	140	9	989.68139591586	988.557770471766	0.999982800576525	0.999974791045229
Model 32	141	23	989.685394171686	988.657729069444	0.999979959857897	0.999974155587129
Model 33	143	11	989.701918092216	989.020234430554	0.999987055226765	0.999982526533237
Model 34	150	11	990.104611094781	989.309018845384	0.999982699415534	0.999977659294812
Model 35	157	15	990.162758989279	989.438462781022	0.999983809683174	0.999981629194335
Model 36	160	21	990.343842866836	989.565373646922	0.999981108944068	0.999976530935547
Model 37	163	21	990.372628712329	989.575633852479	0.999981133879748	0.999976447594885
Model 38	169	13	990.567976655796	989.604662469082	0.999983071010866	0.999976475807032
Model 39	176	15	990.704651799886	989.933074310989	0.999989548561624	0.999985821838219
Model 40	178	13	990.733703079103	989.87415276008	0.999982886626786	0.999976559057591
Model 41	184	13	995.726086947109	995.231514875049	0.999983631211693	0.999978591575311
Model 42	192	15	996.807988191244	996.490012754369	0.999991940411642	0.999990299023876
Model 43	194	15	997.88299356697	997.748827207201	0.999996756502352	0.999995994342185
Model 44	197	18	998.48306842467	998.407358142923	0.999996074075837	0.999997645327743
Model 45	209	24	999.592983167461	999.584605984919	0.999999904805039	0.9999998216491
Model 46	219	24	999.608794683482	999.580120291692	0.999999914365619	0.999999893587388
Model 47	221	30	999.691348942215	999.681652319493	0.999999945931534	0.99999993038033
Model 48	225	30	999.813795902437	999.797134026615	0.999999996172471	0.999999994413255
Model 49	246	32	999.824709460131	999.805533890943	0.999999995689055	0.999999994184829
Model 50	252	30	1000	1000	1	1

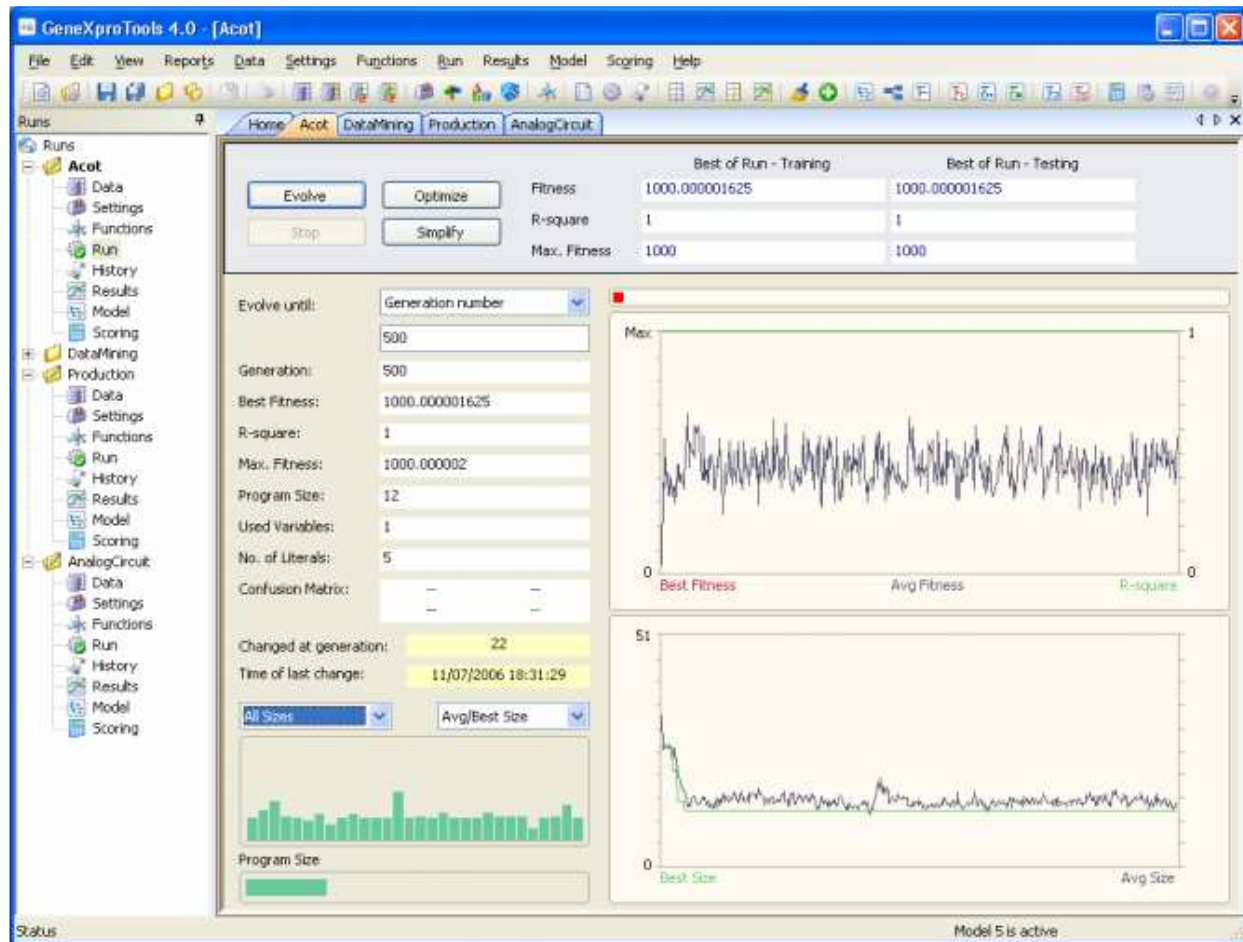
Simplifying a Model

GeneXproTools 4.0 allows you to simplify an existing model (which could have been either generated by GeneXproTools or by another modeling tool) either by pressing the Simplify button on the [Run Panel](#) or by choosing a fitness function with parsimony pressure and then press Optimize or Simplify on the Run Panel.

For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.

Then, in the [Run Panel](#), by clicking the Simplify button, an evolutionary process starts in which all the subsequent models will be descendants of the model you want to simplify. Keep in mind, however, that the simplification algorithm used by GeneXproTools is an evolutionary one and, despite parsimony pressure being applied, models continue to be selected preferentially by fitness and sometimes their complexity might even increase temporarily if the gain in fitness outweighs the loss in simplicity.

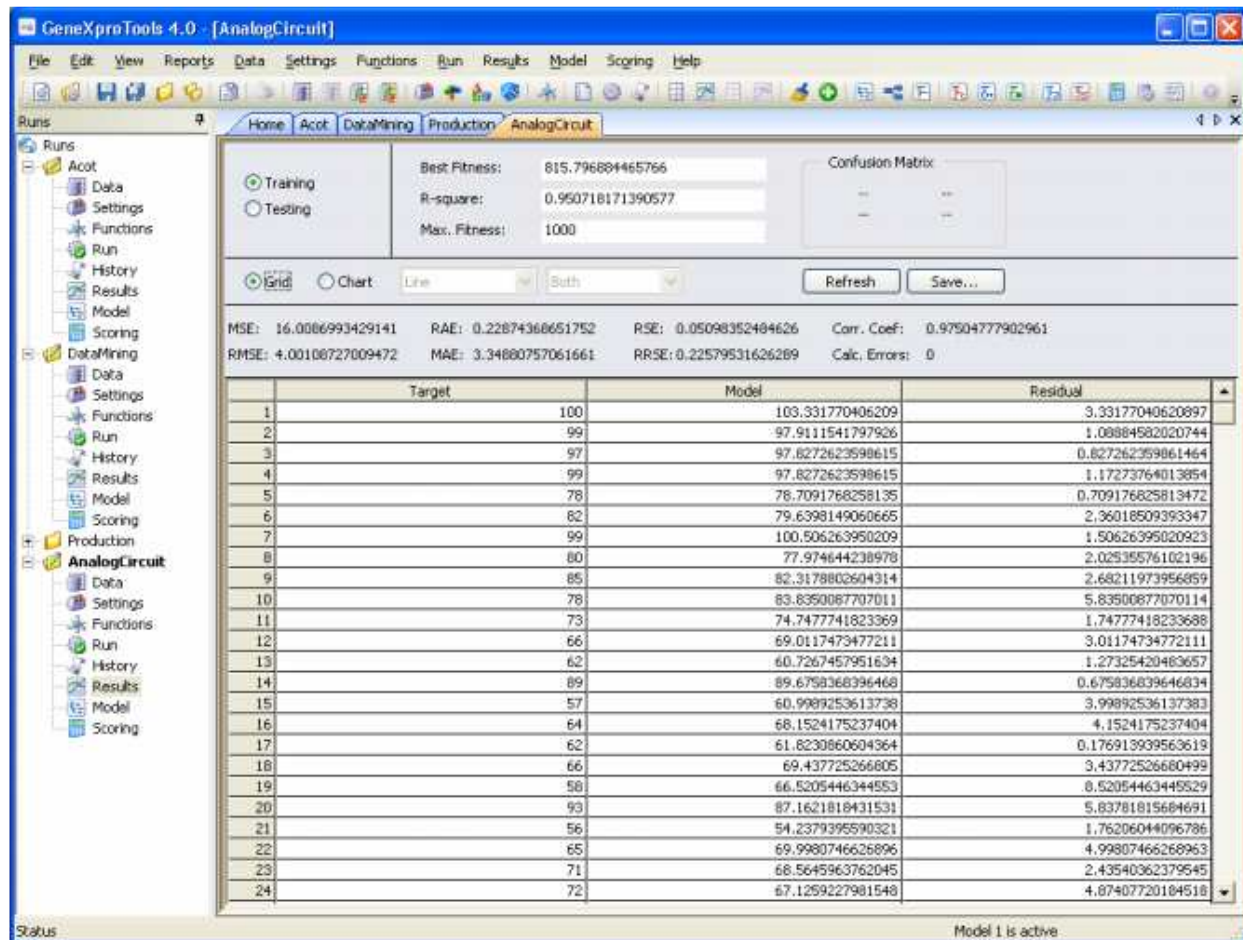
For models created in the GeneXproTools 4.0 environment, you just have to select the model you want to simplify (either the best-of-run or an intermediate model picked up in the [History Panel](#)) and then click the Simplify button and let the algorithm create better descendants not only in terms of fitness but also in terms of size.



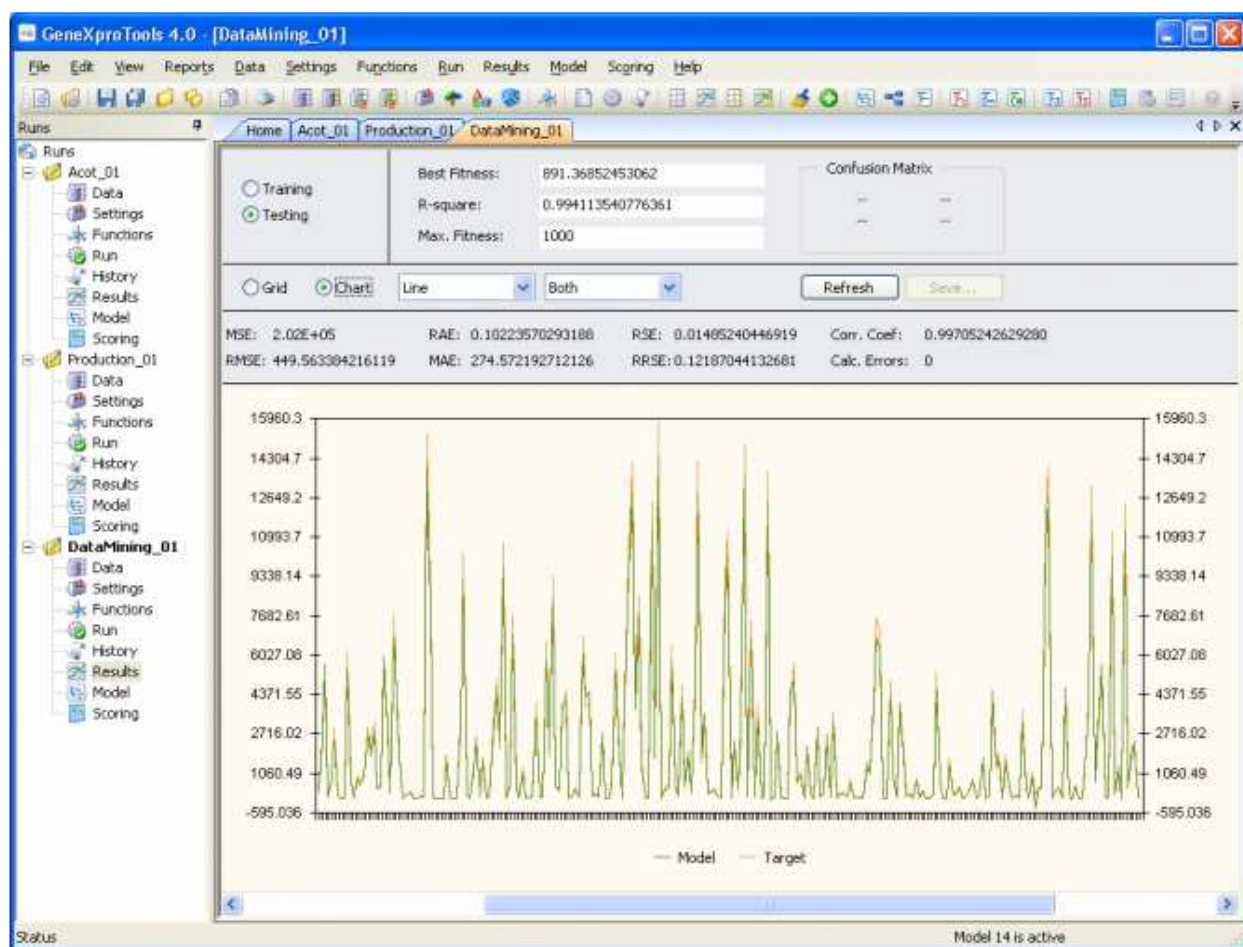
Comparing Actual and Predicted Values

GeneXproTools 4.0 offers two different ways of analyzing and comparing the output of your model with the actual or target values both for the training and testing sets.

In the first, the target or actual values are listed in a table side by side with the predicted or model values.



In the second, the target and predicted values are plotted in a Curve Fitting chart for easy visualization. By moving the scroll bar in the bottom, you can see the results for the entire dataset.



Essential Statistics

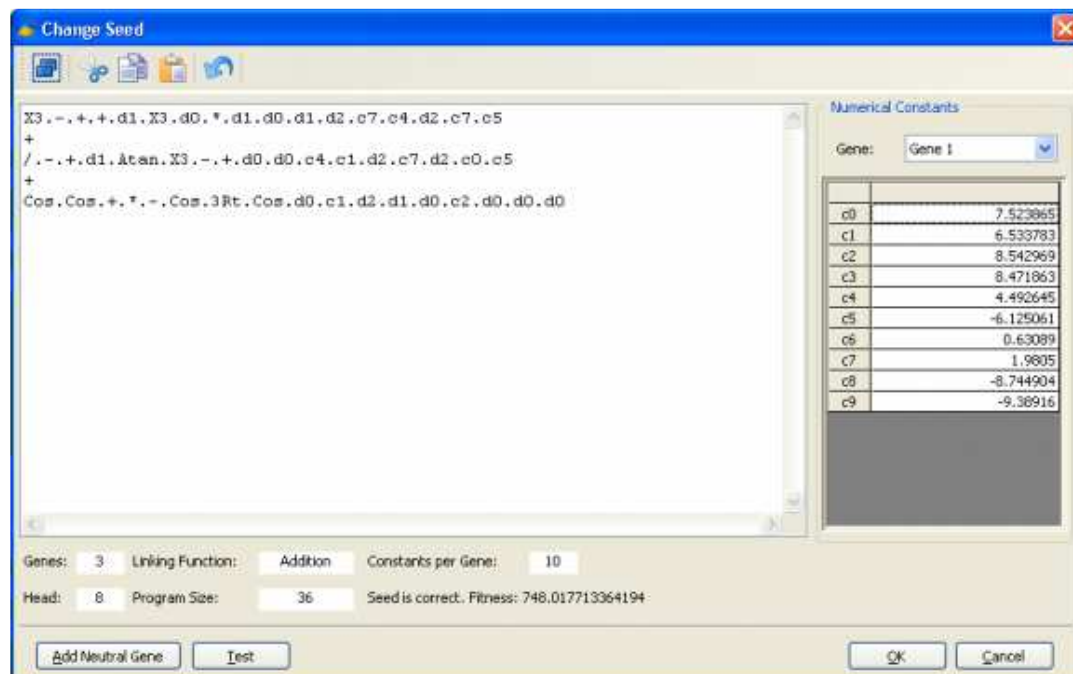
GeneXproTools 4.0 allows a quick and easy assessment of a wide set of [statistical functions](#). Most of them are immediately computed and shown every time you check the [Results Panel](#) (for instance, [fitness](#), [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [R-square](#), and [correlation coefficient](#)).



Modeling from Seed

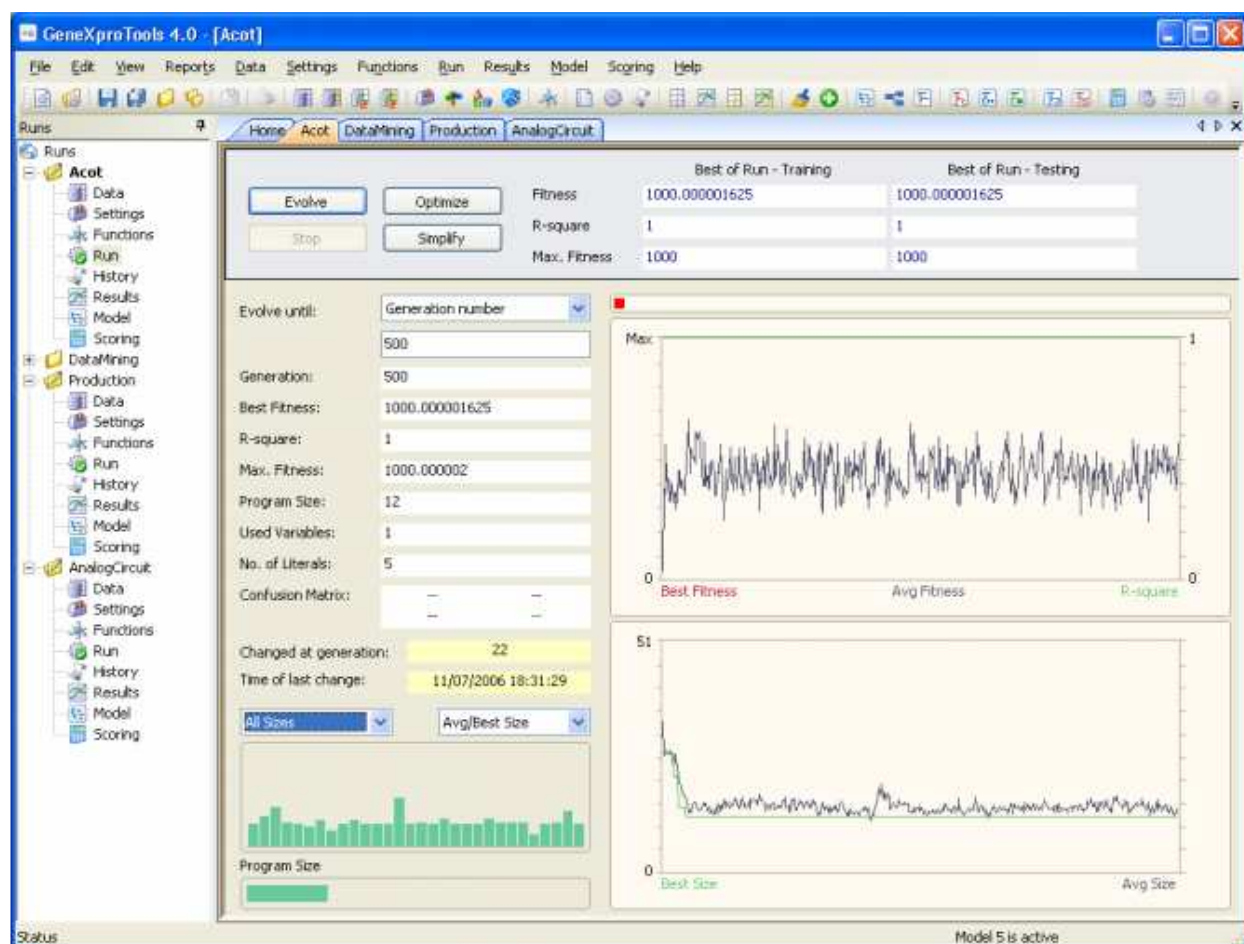
GeneXproTools 4.0 allows the use of an existing model (which could have been either generated by GeneXproTools or by another modeling tool) as the starting point of an evolutionary process so that more complex, finely tuned models could be created.

For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model or seed is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.



Then, in the [Run Panel](#), by clicking the Optimize button, an evolutionary process starts in which all the subsequent models will be descendants of the seed you introduced. A note of caution though: if your seed has a very small fitness, you risk losing it early in the run as better models could be randomly created by GeneXproTools 4.0 and your seed would, most probably, not be selected for breeding new models. If your seed has zero fitness, though, you will receive a warning so that you could modify your seed until it becomes a viable seed capable of breeding new models.

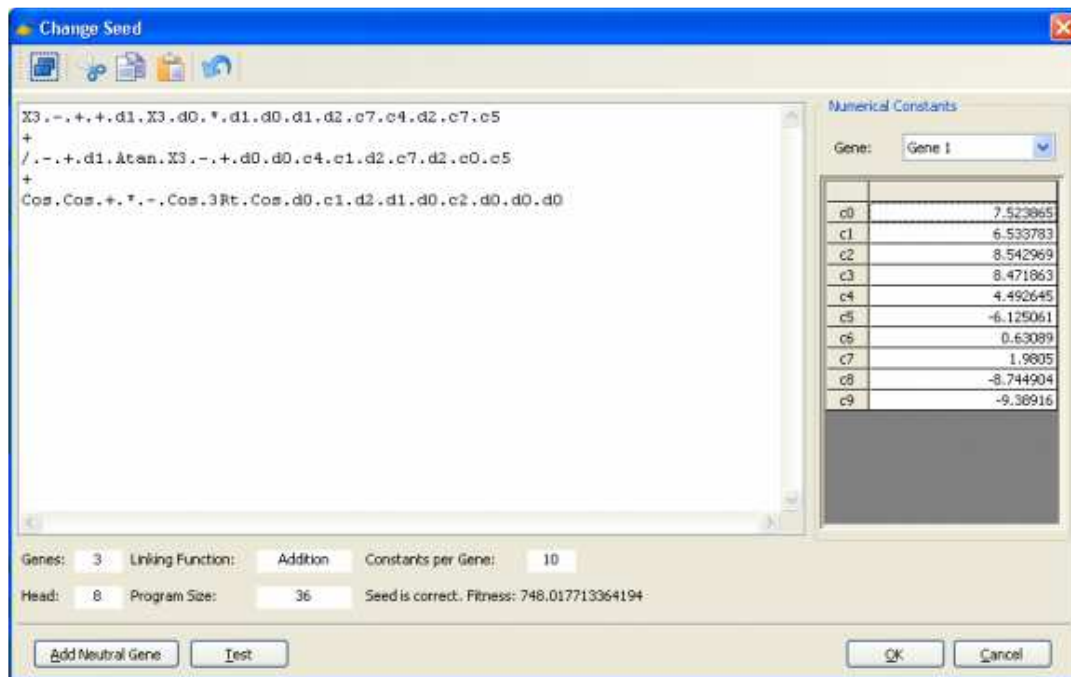
For models created in the GeneXproTools 4.0 environment, the seed is fed to the algorithm every time you click the Optimize or Simplify buttons during modeling, that is, the seed (in this case, the best model of the previous run) is introduced automatically by GeneXproTools.



Adding a Neutral Gene

The addition of a neutral gene to a seed might seem, at first sight, the wrong thing to do as most of the times we are interested in creating efficient and parsimonious models. But one should look at this as modeling in progress as, for really complex phenomena, it is not uncommon to approximate a complex function progressively.

Thus, being able of introducing extra terms in your seed is a powerful modeling tool and GeneXproTools 4.0 allows you to do that by selecting Add Neutral Gene in the Edit Menu or through the Change Seed window.



Here, by pressing the Add Neutral Gene button, you will see a neutral gene being added to your model. By doing this, you are giving the learning algorithm more room to play and, hopefully, a better, more complex program will evolve.

Neutral genes can also be introduced automatically by GeneXproTools 4.0 using the Complexity Increase Engine.

Complexity Increase Engine

GeneXproTools 4.0 also allows you to introduce neutral genes automatically by activating the Complexity Increase Engine in the Settings Panel -> General Settings Tab.

General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 200 [Reset]

Testing Samples: 246 [Reset]

Number of Chromosomes: 30

Head Size: 7

Number of Genes: 3

Linking Function: Addition

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 5

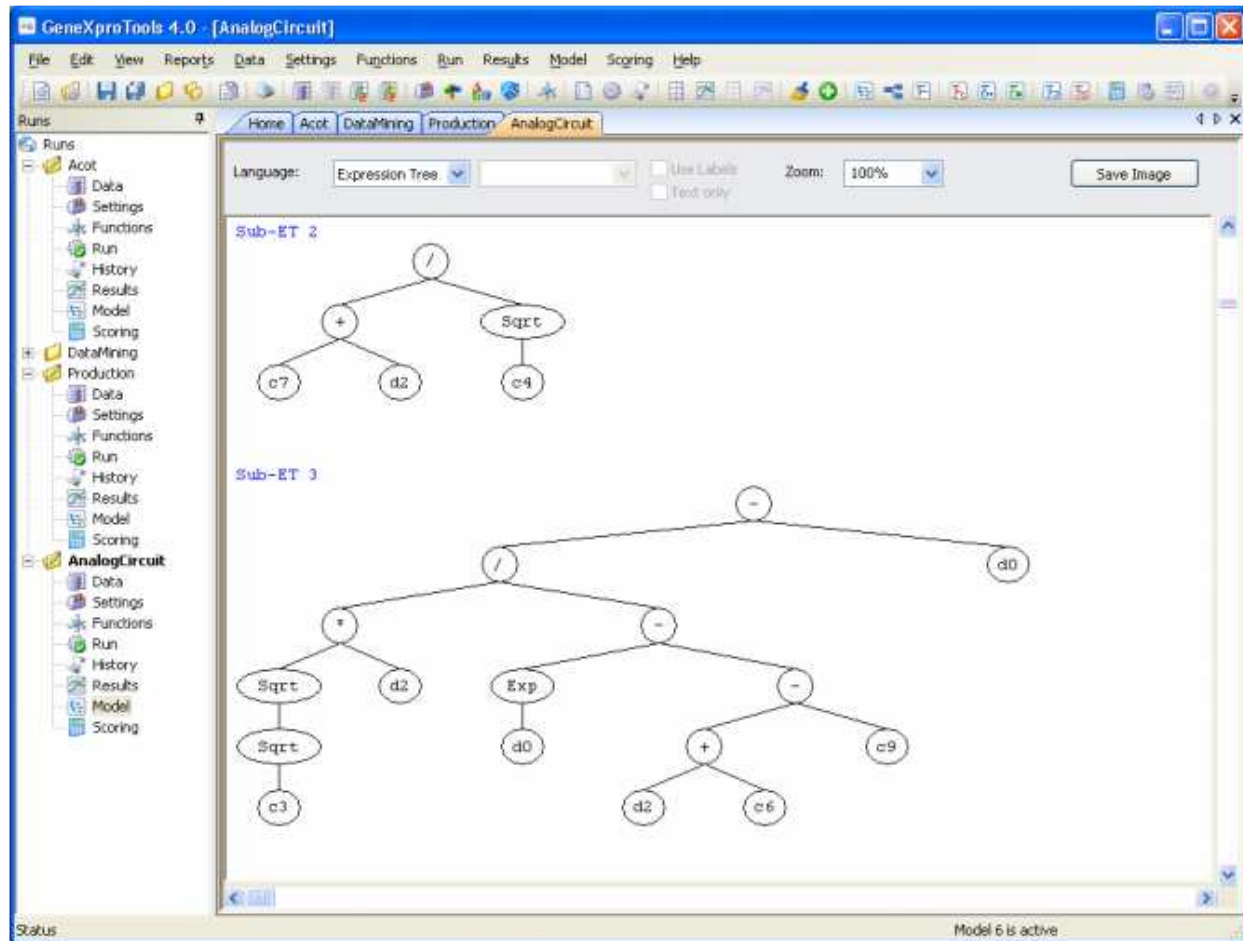
☒ Save Run History

Whenever you are using the Complexity Increase Engine of GeneXproTools 4.0, you must fill the Generations Without Change box to set the period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction or a neutral gene (an extra term) is automatically added to your model; the Number of Tries box corresponds to the number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models; in the Max. Complexity box you write the maximum number of terms (genes) you'll allow in your model and no other terms will be introduced beyond this threshold.

The Complexity Increase Engine of GeneXproTools 4.0 might be a very powerful modeling tool, but you must be careful not to create excessively complex models for, most of the times, a greater complexity does not necessarily imply a greater efficiency.

Visualizing Models as Parse Trees

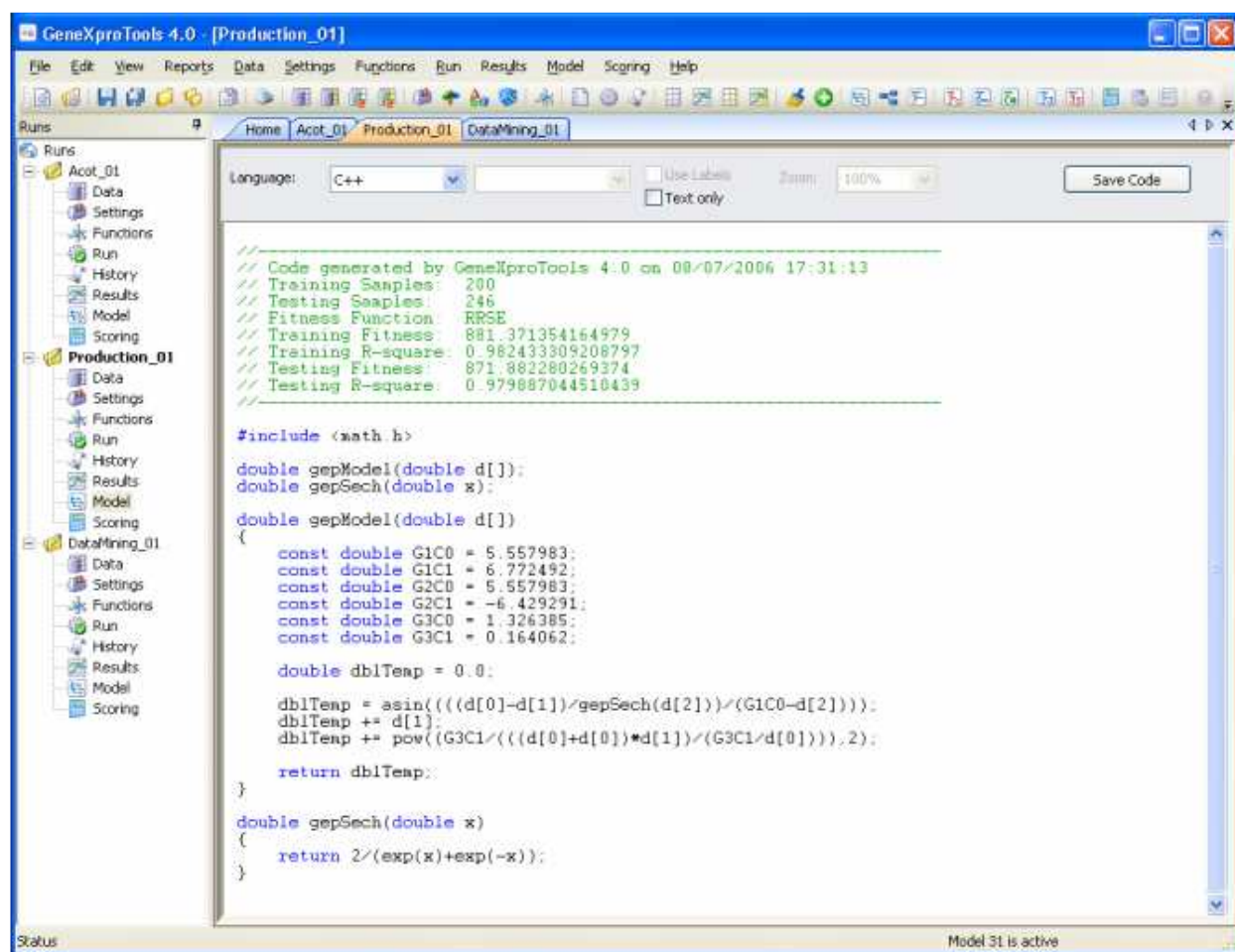
GeneXproTools 4.0 comes equipped with a parse tree generator that automatically converts your models into diagram representations or parse trees.



Indeed, all the models evolved by GeneXproTools in its native [Karva language](#) can be automatically parsed into visually appealing expression trees, allowing a quicker and more complete understanding of their mathematical intricacies.

Generating Code Automatically Using GeneXproTools Built-in Grammars

GeneXproTools 4.0 offers a total of 15 [built-in grammars](#) so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the most commonly used programming languages (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL). This code can then be used in other applications.



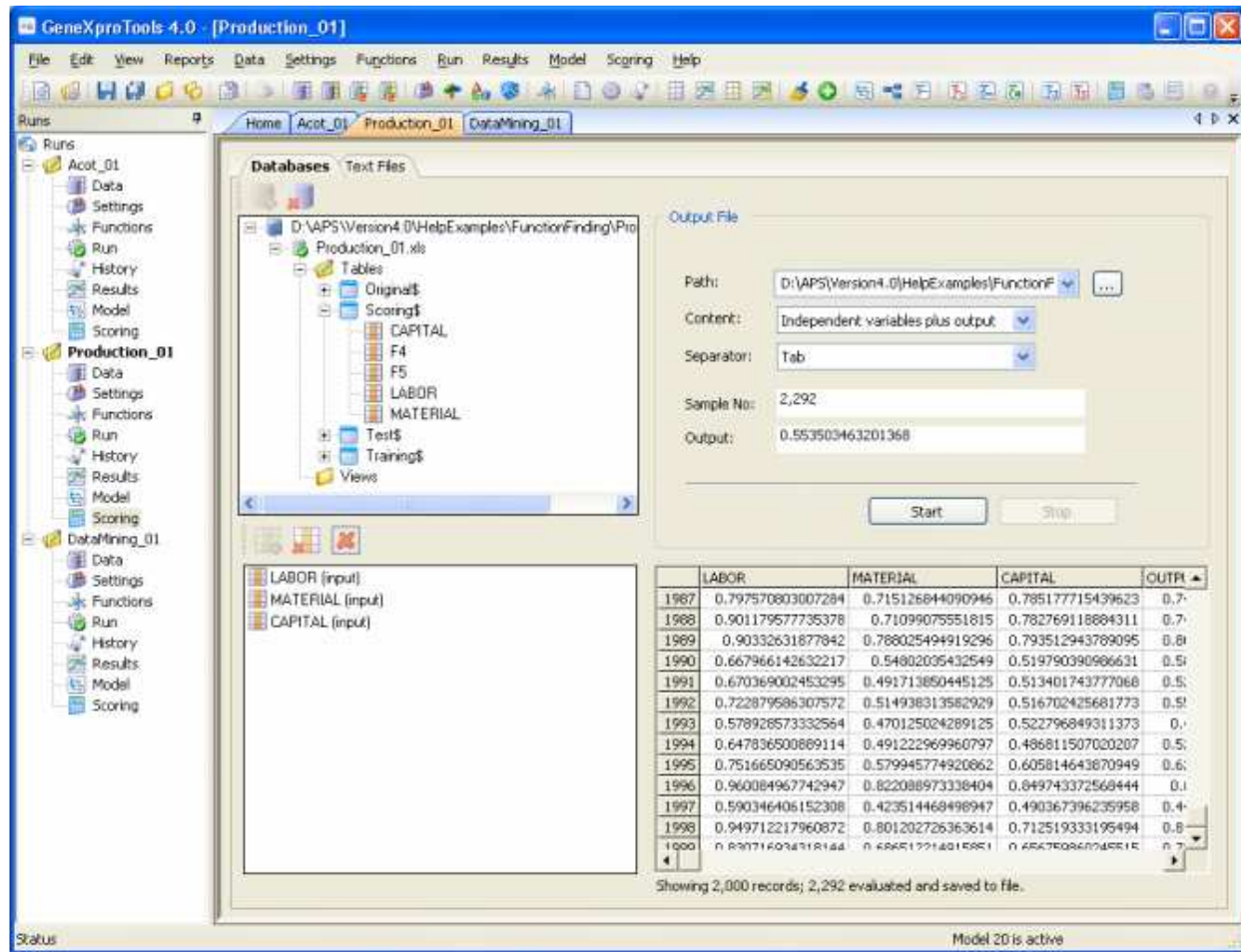
Generating Code Automatically Using User Defined Grammars

GeneXproTools 4.0 allows the design of [User Defined Grammars](#) so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the programming language of your choice if you happen to prefer a programming language not already covered by the 15 [built-in grammars](#) of GeneXproTools 4.0 (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL).

As an illustration, the C++ grammar of GeneXproTools 4.0 is shown [here](#). Other grammars may be easily created using this or other GeneXproTools built-in grammars as reference.

Scoring Engine

GeneXproTools 4.0 comes equipped with a scoring engine for scoring your databases/Excel or text files in the GeneXproTools environment. The scoring engine of GeneXproTools uses the JavaScript code of your models to score your data.



After its creation, a model is usually used multiple times to make predictions or extrapolations. The scoring engine of GeneXproTools 4.0 allows the immediate scoring of your databases as soon as a model is created, saving the scores to Excel (csv) or text files.



GeneXproTools Environment

Classification Tools

The successful creation of Classification models requires the use of very sharp modeling tools and the most important are the following:

- Data Screening Engine
- Data Visualization Tool
- Built-in Fitness Functions
- User Defined Fitness Functions
- Built-in Mathematical Functions
- Function Selection Tool
- Dynamic User Defined Functions
- Static User Defined Functions

- [Learning Algorithms](#)
- [Monitoring the Modeling Process](#)
- [Analyzing Intermediate Models](#)
- [Simplifying a Model](#)
- [Comparing Actual and Predicted Values](#)
- [Essential Statistics](#)
- [Modeling from Seed Models](#)
- [Adding a Neutral Gene](#)
- [Complexity Increase Engine](#)
- [Visualizing Models as Parse Trees](#)
- [Generating Code Automatically Using Built-in Grammars](#)
- [Generating Code Automatically Using User Defined Grammars](#)
- [Scoring Engine](#)

Data Screening Engine

GeneXproTools accepts two different data formats: the standard Samples x Variables format where samples are in rows and variables in columns, with the class occupying the rightmost position; and Gene Expression Matrixes commonly used in DNA microarray studies where samples are in columns and variables in rows, with the class occupying the topmost position. Note, however, that GeneXproTools uses the Samples x Variables format and therefore all formats are automatically converted and shown in this format.

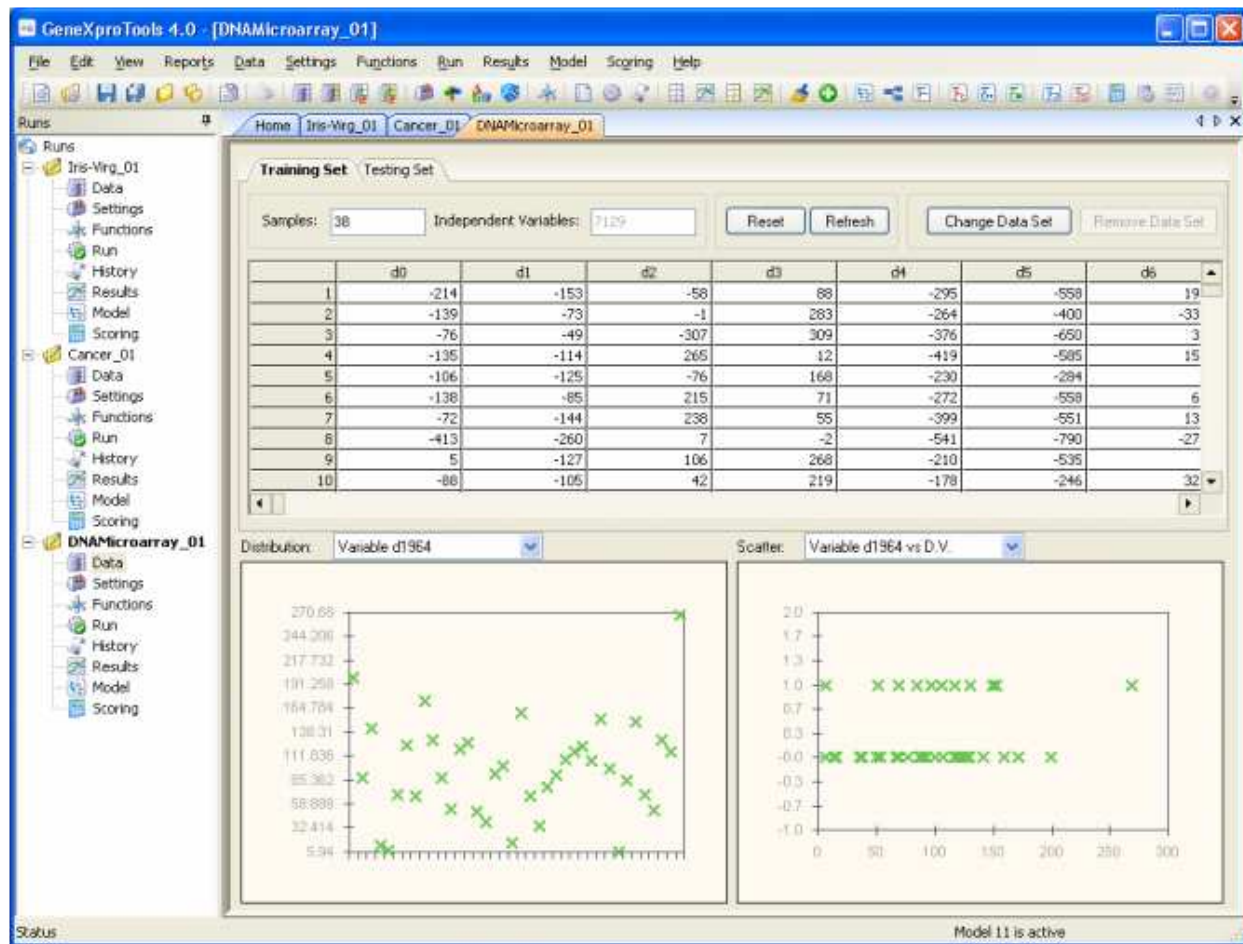
The data screening engine of GeneXproTools 4.0 checks the validity of all the data sets used both in the creation and testing of the model. The data screening engine is operative every time you load data sets either for training or testing.

Classification problems require data files with numerical inputs in which the dependent variable can take only two values: zero or one. The data screening engine of GeneXproTools makes sure that all the input files for classification respect this format. But it is common for the input files to have missing or nominal values which render them useless for modeling or testing. GeneXproTools 4.0 data screening engine can help you identify these faulty samples and correct them.

Data Visualization Tool

The data visualization tool of GeneXproTools 4.0 enables you to analyze the distribution of all the inputs and to spot simple relationships between the independent variables and the dependent variable.

The visualization of both the distribution and relationships between variables is a valuable modeling tool as it can help you detect outliers or simple relationships between your variables that can be used to evolve more efficient models.



Built-in Fitness Functions

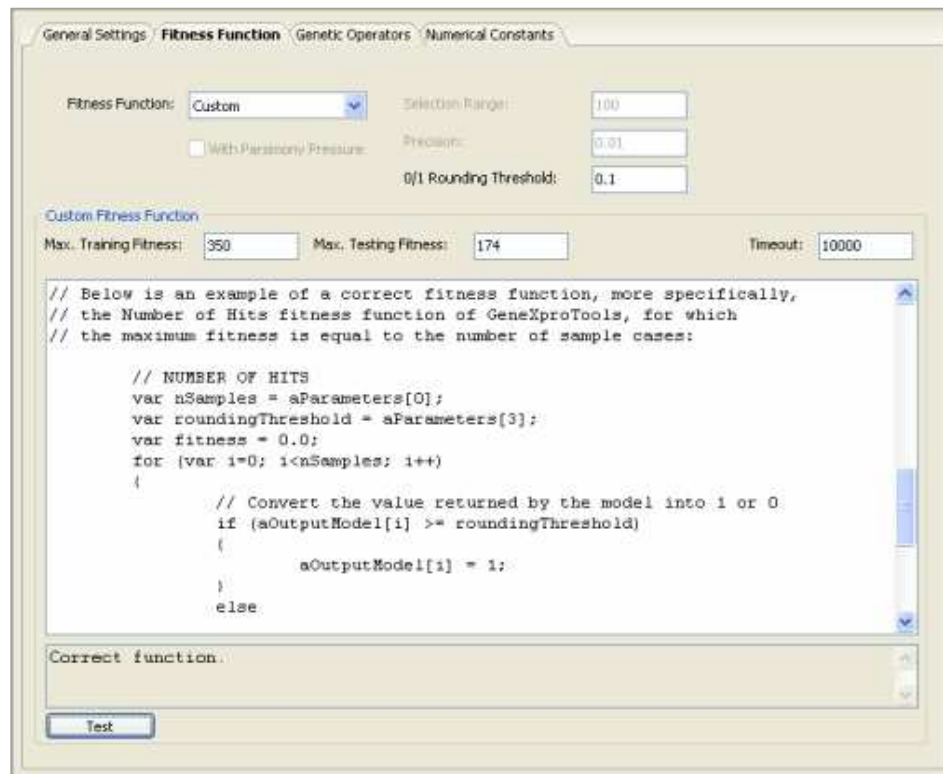
GeneXproTools 4.0 offers 30 [built-in fitness functions](#) based on well-known statistical functions.

Fitness functions are fundamental modeling tools for they determine the nature of the search landscape. And different fitness functions open and explore different paths in the solution space. The 30 built-in fitness functions of GeneXproTools 4.0 are named as follows: [Number of Hits](#), [Hits with Penalty](#), [Accuracy](#), [Squared Accuracy](#), [Sensitivity/Specificity](#), [PPV/NPV](#), [SSPN](#), [R-square](#), [Correlation Coefficient](#), [MSE](#), [RMSE](#), [MAE](#), [RSE](#), [RRSE](#), and [RAE](#). Furthermore, all these fitness functions have their own counterpart with [parsimony pressure](#).

User Defined Fitness Functions

Despite the wide set of GeneXproTools [built-in fitness functions](#), some users sometimes want to experiment with fitness functions of their own. GeneXproTools 4.0 gives the user the possibility of creating custom tailored fitness functions and evolve models with them.

The code for the [custom fitness function](#) must be in JavaScript and is written in the Custom Fitness Function window and can be tested before evolving a model with it.



Built-in Mathematical Functions

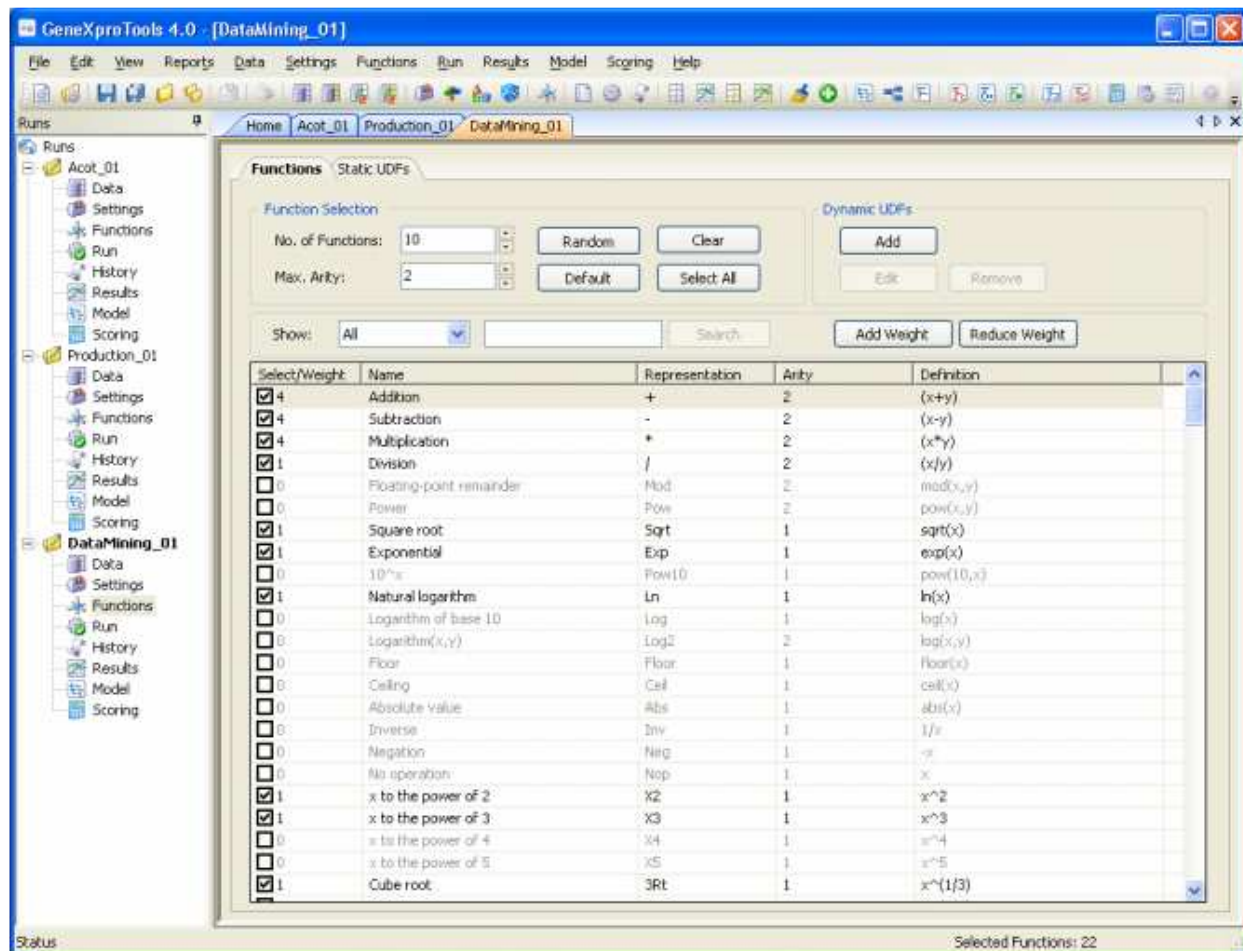
GeneXproTools 4.0 offers a total of 279 [built-in mathematical functions](#), including 186 different IF THEN ELSE rules. This wide set of functions allows the evolution of complex and rigorous models quickly built with the most appropriate mathematical functions or rules.

From the simple arithmetic operators to complex mathematical functions available in most programming languages to more complex mathematical functions commonly used by engineers and scientists, the modeling algorithms of GeneXproTools 4.0 enable you the integration of the most appropriate functions or rules in your models.

Function Selection Tool

For some problems, the simple arithmetic operators are more than enough to create extremely accurate and elegant models. But for other problems more complex functions are necessary to create good models and when inside knowledge of the problem at hand is not producing the expected results, it might be interesting to experiment with different combinations of functions. The Function Selection Tool of GeneXproTools 4.0 helps you to experiment with different function sets very quickly either by pressing the Random button in the Functions Panel or by selecting certain sub-sets of functions.

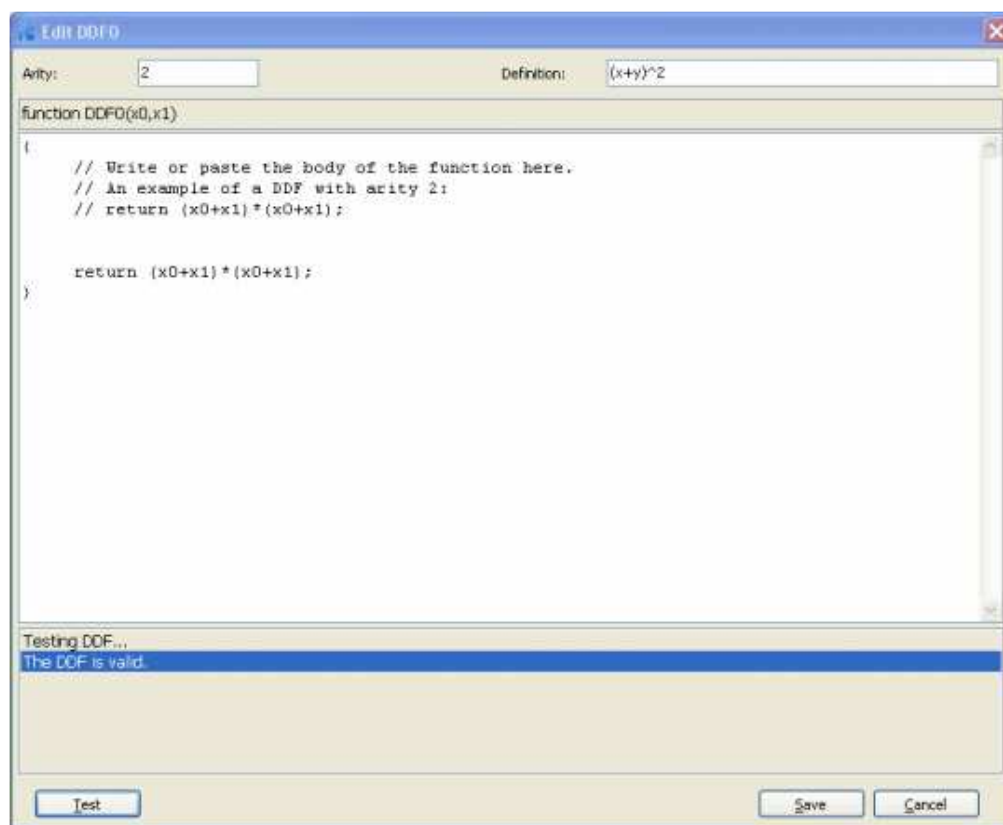
In addition, although you can always handcraft your function set so that it is well-balanced relatively to the number of independent variables in your data, in GeneXproTools 4.0 this is no longer essential as GeneXproTools balances automatically all function sets. This is particularly important for high-dimensional datasets, such as the huge DNA microarray data.



Dynamic User Defined Functions (DDFs)

Despite the wide set of GeneXproTools built-in mathematical functions, some users sometimes want to model with different functions or rules. GeneXproTools 4.0 gives the user the possibility of creating custom tailored functions and evolve models with them. A note of caution though: the use of DDFs slows considerably the evolutionary process and therefore should be used with moderation.

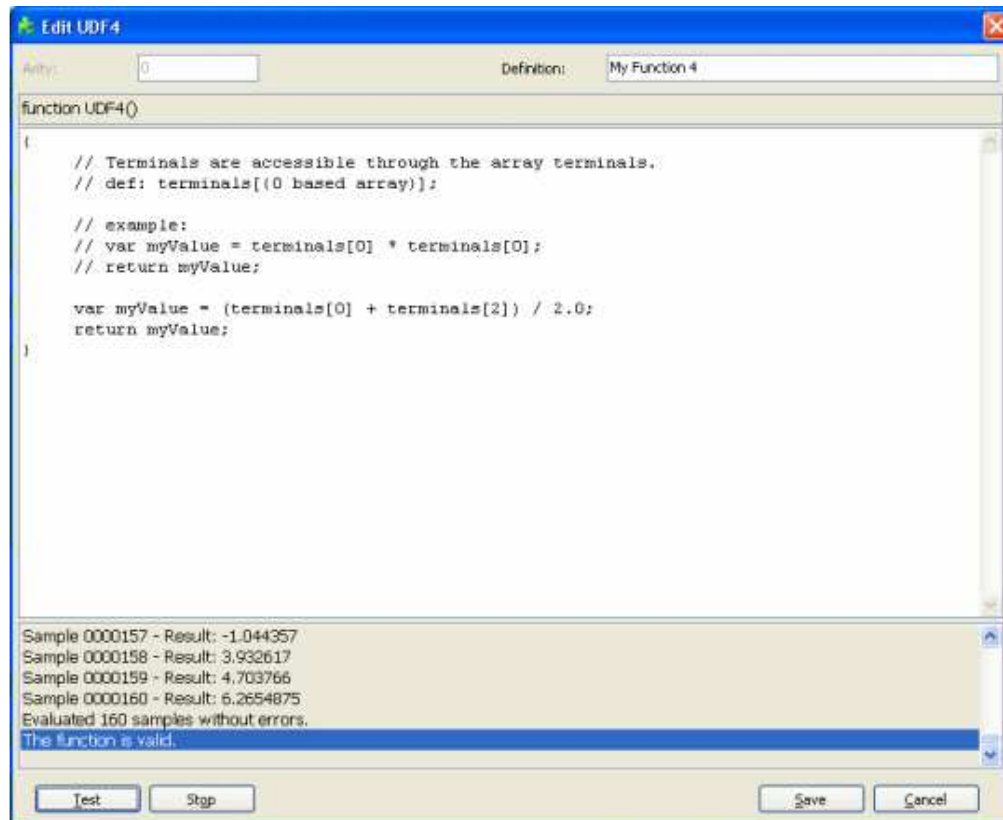
The code for the DDFs must be in JavaScript and is written in the Edit DDF window (in the Functions Panel, select the Functions Tab and then click Add in the Dynamic UDFs tool box).



Static User Defined Functions (UDFs)

Sometimes it is possible to spot relatively simple relationships between certain variables in your data, and GeneXproTools 4.0 allows you to use this information to build custom tailored functions (UDFs). These functions allow the discovery of more complex models composed of several simpler models.

The code for the UDFs must be in JavaScript and is written in the Edit UDF window (in the Functions Panel, select the Static UDFs Tab and then click Add in the Static UDFs tool box).



Learning Algorithms

GeneXproTools 4.0 is equipped with two different [learning algorithms](#) for evolving complex nonlinear classification models.

The first is the fastest and simplest of the two. The second algorithm is very similar to the first, with the difference that it gives the user the possibility of choosing the range and type of numerical constants that will be used during the learning process. And because numerical constants are crucial to a wide set of problems, the second algorithm is the default in GeneXproTools 4.0. Keep in mind, however, that for problems where numerical constants are not necessary it pays to use the first algorithm as the models are not only much simpler but also the evolutionary process is much quicker.

In order to evolve models using the second algorithm, select Numerical Constants in the Settings Panel and then check the box that activates this algorithm. You will notice that additional parameters become available, including a small set of genetic operators especially developed for handling numerical constants (if you are not familiar with these operators, please use the default values for they work very well in all cases). To use the first algorithm, just uncheck the Use Random Numerical Constants box.

Monitoring the Modeling Process

GeneXproTools 4.0 allows you to monitor the modeling process by showing the essential parameters of a run during the discovery process, including the fitness, [classification accuracy](#), and [confusion matrix](#) (true positives, true negatives, false positives, and false negatives) of the best-of-generation model.

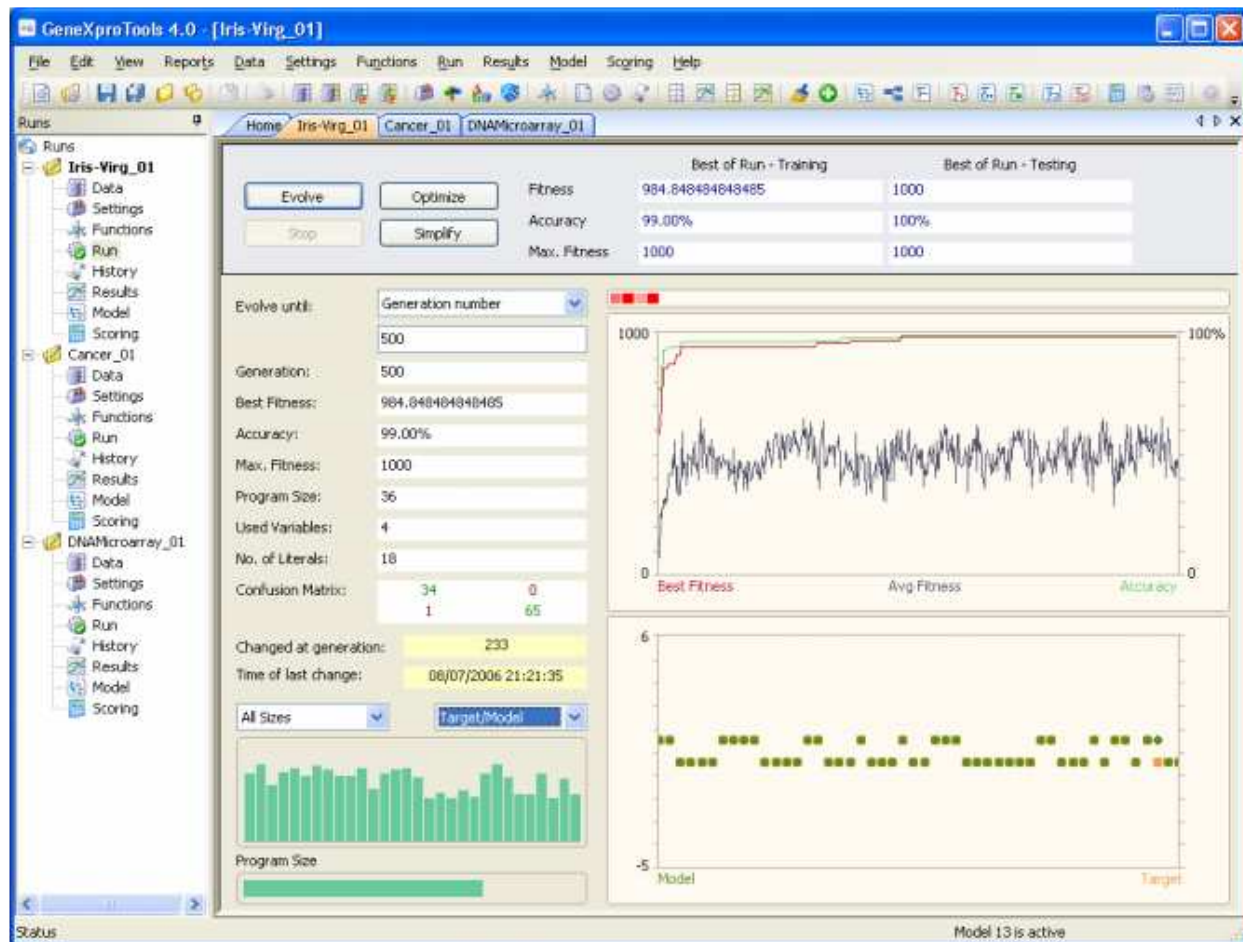
Furthermore, GeneXproTools allows you to visualize the modeling process by plotting the essential features of a run during the discovery process. In the [Run Panel](#), a total of eight different charts are shown:

1. The Variables Usage map.
This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square you can access the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic.
2. The Evolutionary Dynamics chart.
This chart shows the average fitness of the population plus the fitness and accuracy of the best-of-generation model.
3. The Average/Best Size chart.
This chart shows the average program size of the population versus the program size of the best-of-generation model. This chart is especially useful during simplification and can be activated any time during evolution by selecting Avg/Best Size in the rightmost combo box in the bottom.
4. The Target/Model chart.
This chart plots the first 50 data points of the training set and shows how well the evolving models are fitting the target. The Target/Model chart can be activated any time during evolution by selecting Target/Model in the rightmost combo box in the bottom.
5. The Sub-Program Sizes chart.
This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be activated any time during evolution by selecting Sub-Program Sizes in the leftmost combo box in the bottom.
6. The All Sizes chart.
This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be activated any time during evolution by selecting All Sizes in the leftmost combo box in the bottom.
7. The All Fitnesses chart.
This chart shows the fitnesses of all the models in the population. Not only after a run but also

during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be activated any time during evolution by selecting All Fitnesses in the leftmost combo box in the bottom.

8. The Program Size chart.

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model.



The evolutionary process can be stopped whenever you are satisfied with the results by pressing the Stop button or you can use one of the stop criteria of GeneXproTools for picking up your model exactly as you want it.

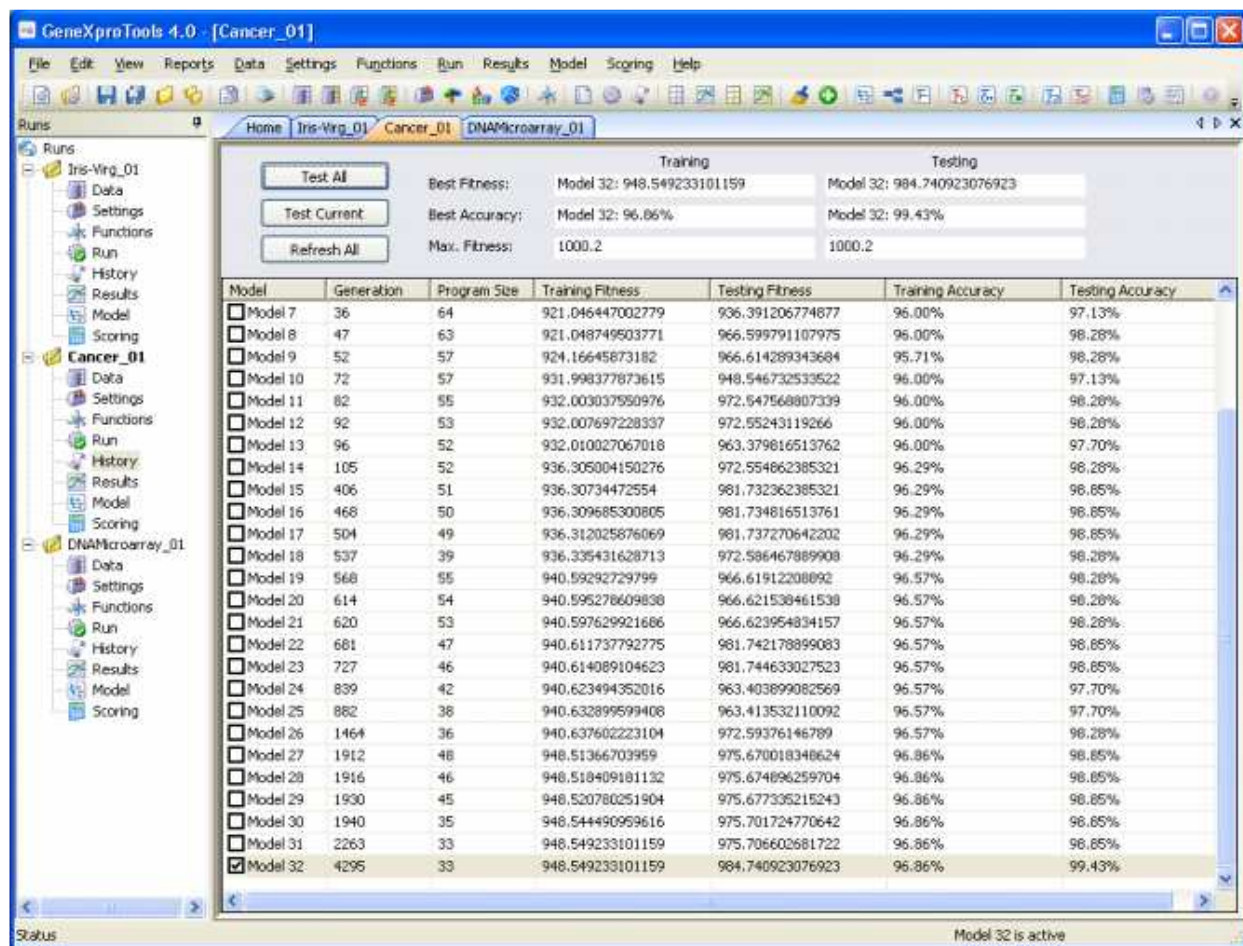
When the evolutionary process stops, the best-of-run model is ready either for analysis or scoring. And if you are still not happy with the results, you can still try to fine-tune the evolved model by pressing the Optimize button and repeat this process until you are completely satisfied with your model. In addition, you might also wish to try and simplify your model and for that you just have to press the Simplify button.

Analyzing Intermediate Models

GeneXproTools 4.0 gives you the possibility of gaining some insights into the modeling process by analyzing all the best-of-generation intermediate models.

All the best-of-generation models discovered during a run are saved and you can pick them up for analysis at the [History Panel](#) by checking the model you are interested in. Each of these intermediate models can then be analyzed exactly as you do for the best-of-run model, that is, you can check their

performance in the testing set, check their vital statistics, automatically generate code with them, visualize their parse trees, use them as seed to create better models from them and so forth.



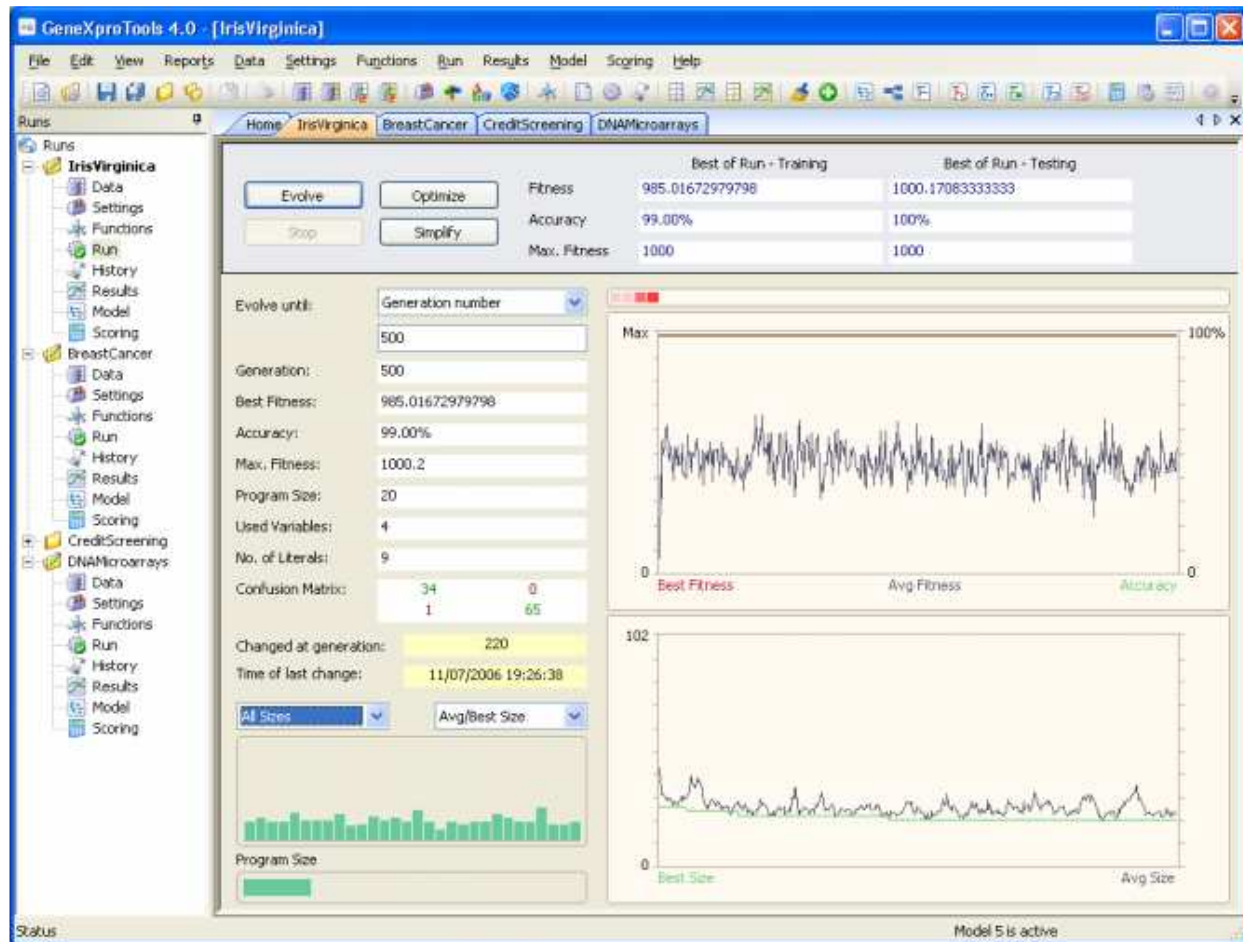
Simplifying a Model

GeneXproTools 4.0 allows you to simplify an existing model (which could have been either generated by GeneXproTools or by another modeling tool) either by pressing the Simplify button on the [Run Panel](#) or by choosing a fitness function with parsimony pressure and then press Optimize or Simplify on the Run Panel.

For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.

Then, in the [Run Panel](#), by clicking the Simplify button, an evolutionary process starts in which all the subsequent models will be descendants of the model you want to simplify. Keep in mind, however, that the simplification algorithm used by GeneXproTools is an evolutionary one and, despite parsimony pressure being applied, models continue to be selected preferentially by fitness and sometimes their complexity might even increase temporarily if the gain in fitness outweighs the loss in simplicity.

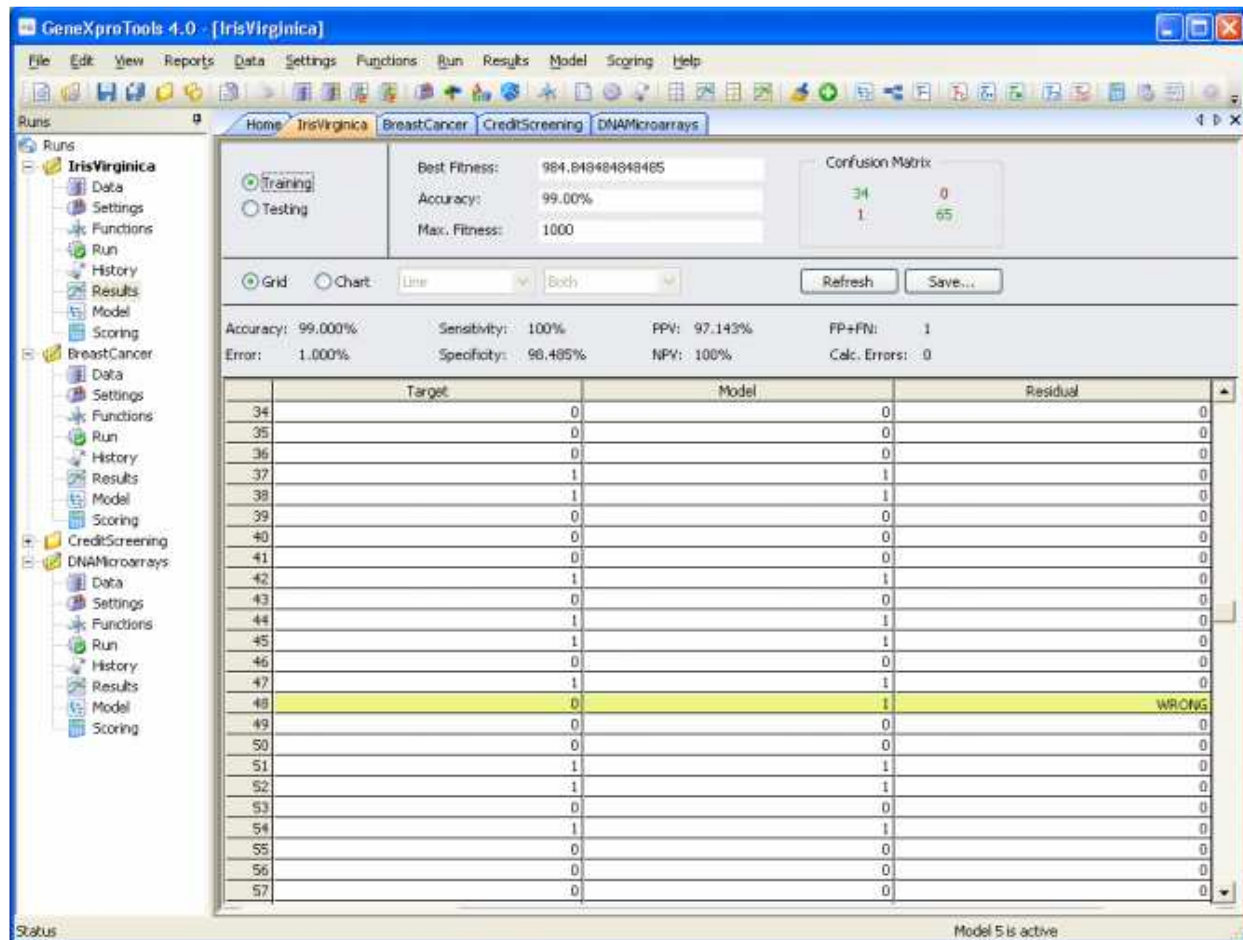
For models created in the GeneXproTools 4.0 environment, you just have to select the model you want to simplify (either the best-of-run or an intermediate model picked up in the [History Panel](#)) and then click the Simplify button and let the algorithm create better descendants not only in terms of fitness but also in terms of size.



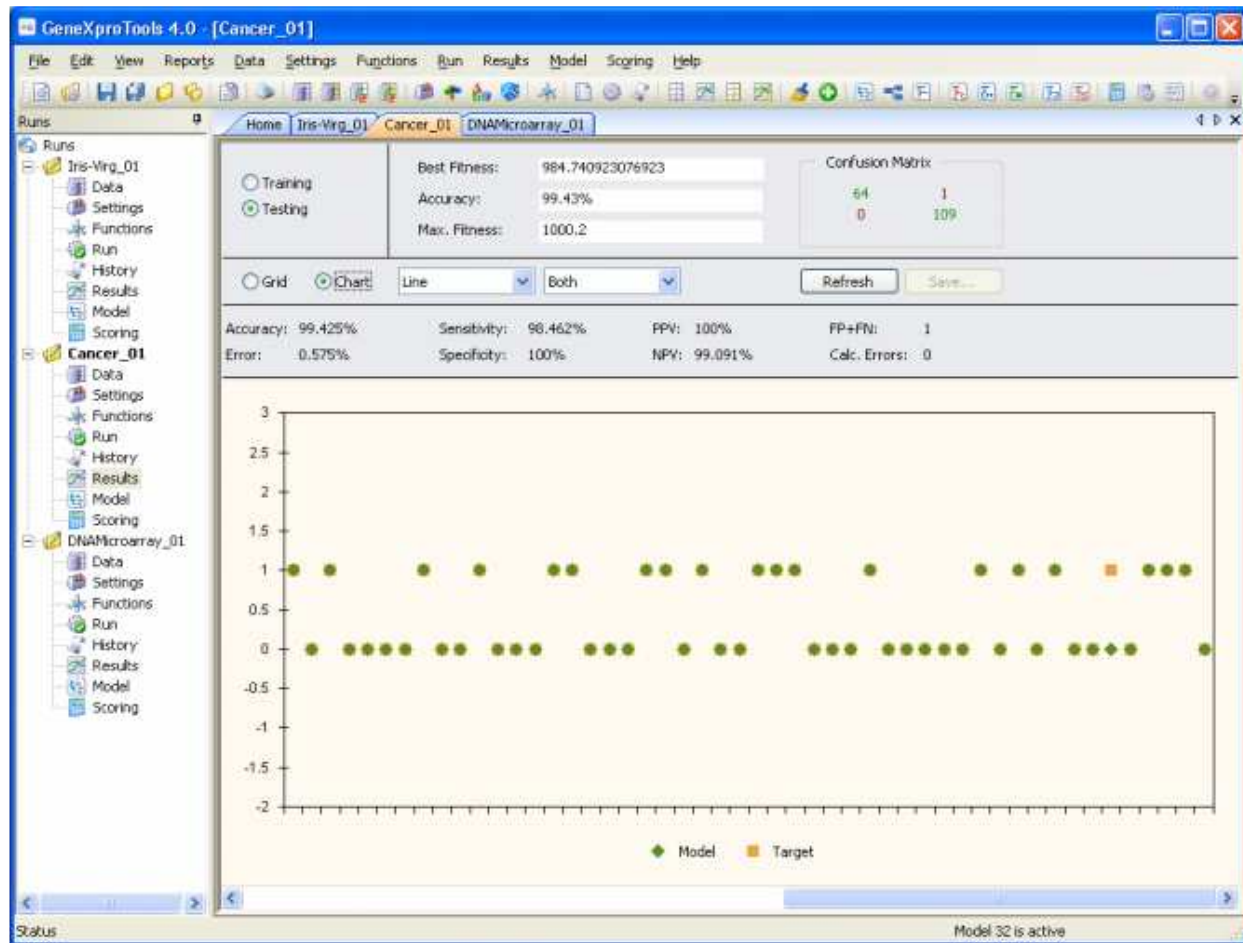
Comparing Actual and Predicted Values

GeneXproTools 4.0 offers two different ways of analyzing and comparing the output of your model with the actual or target values both for the training and testing sets.

In the first, the target or actual values are listed in a table side by side with the predicted or model values.



In the second, the target and predicted values are plotted in a Target/Model chart for easy visualization. By moving the scroll bar in the bottom, you can see the results for the entire dataset.

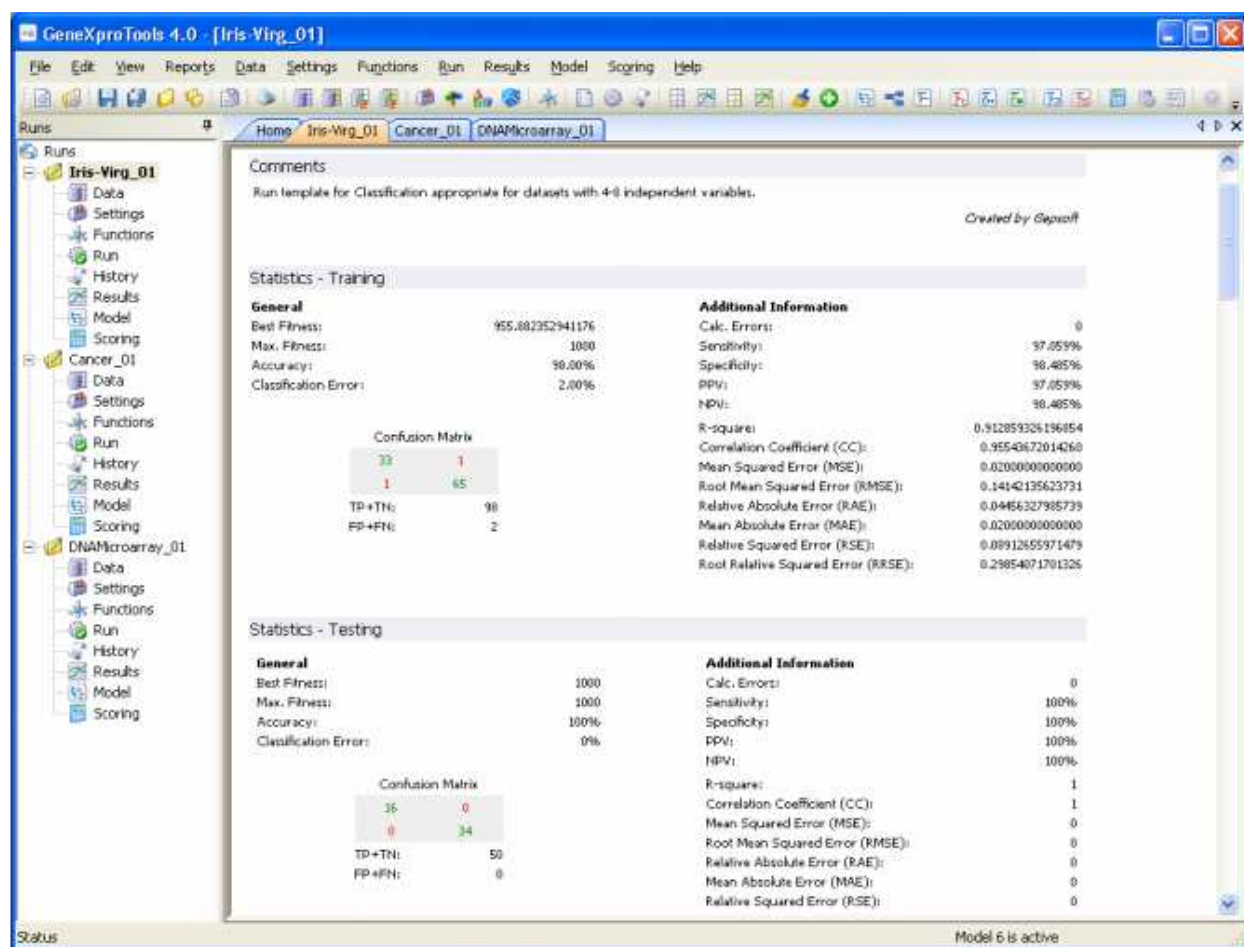


Essential Statistics

GeneXproTools 4.0 allows a quick and easy assessment of a wide set of [statistical functions](#). Some of them are immediately computed and shown every time you check the [Results Panel](#) ([classification error](#), [classification accuracy](#), [confusion matrix](#) (true positives, true negatives, false positives, and false negatives), [sensitivity](#), [specificity](#), [positive predictive value](#), and [negative predictive value](#)).



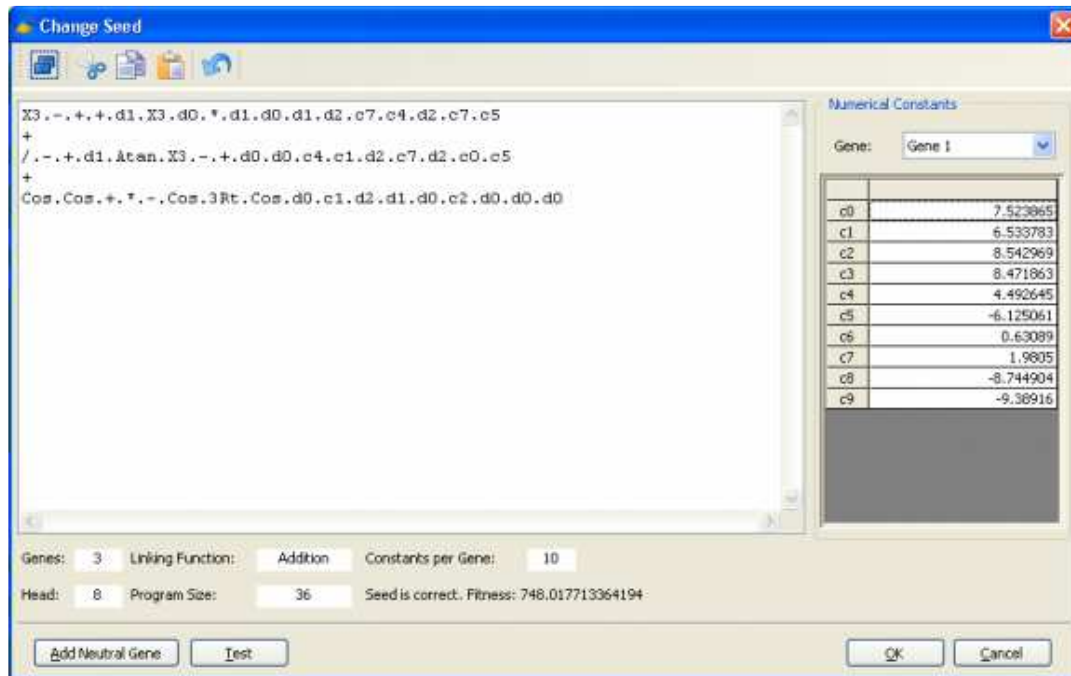
Others, though, are only shown in the [Report Panel](#) (mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, R-square, and correlation coefficient) after their evaluation in the [Results Panel](#).



Modeling from Seed

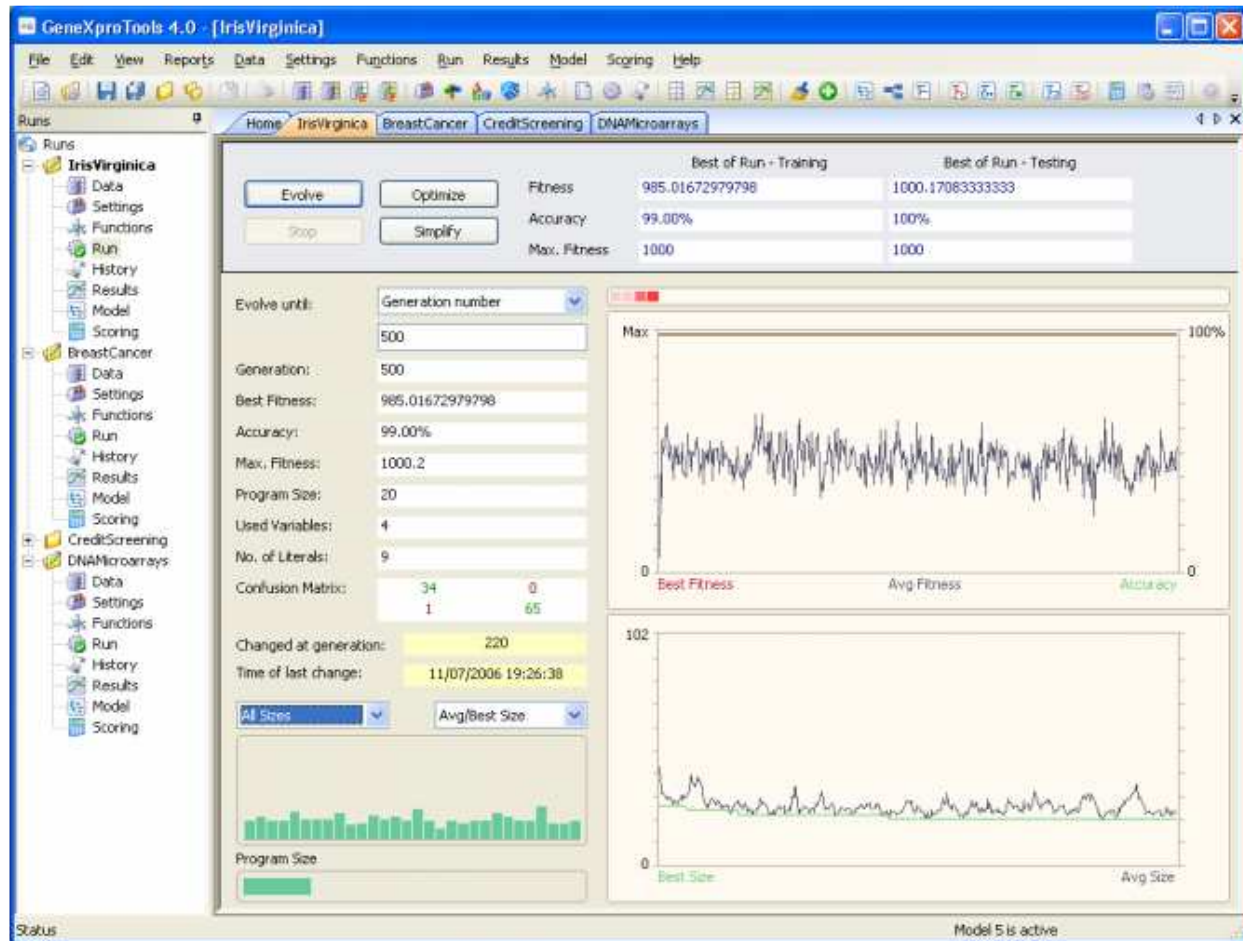
GeneXproTools 4.0 allows the use of an existing model (which could have been either generated by GeneXproTools or by another modeling tool) as the starting point of an evolutionary process so that more complex, finely tuned models could be created.

For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model or seed is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.



Then, in the [Run Panel](#), by clicking the Optimize button, an evolutionary process starts in which all the subsequent models will be descendants of the seed you introduced. A note of caution though: if your seed has a very small fitness, you risk losing it early in the run as better models could be randomly created by GeneXproTools and your seed would, most probably, not be selected for breeding new models. If your seed has zero fitness, though, you will receive a warning so that you could modify your seed until it becomes a viable seed capable of breeding new models.

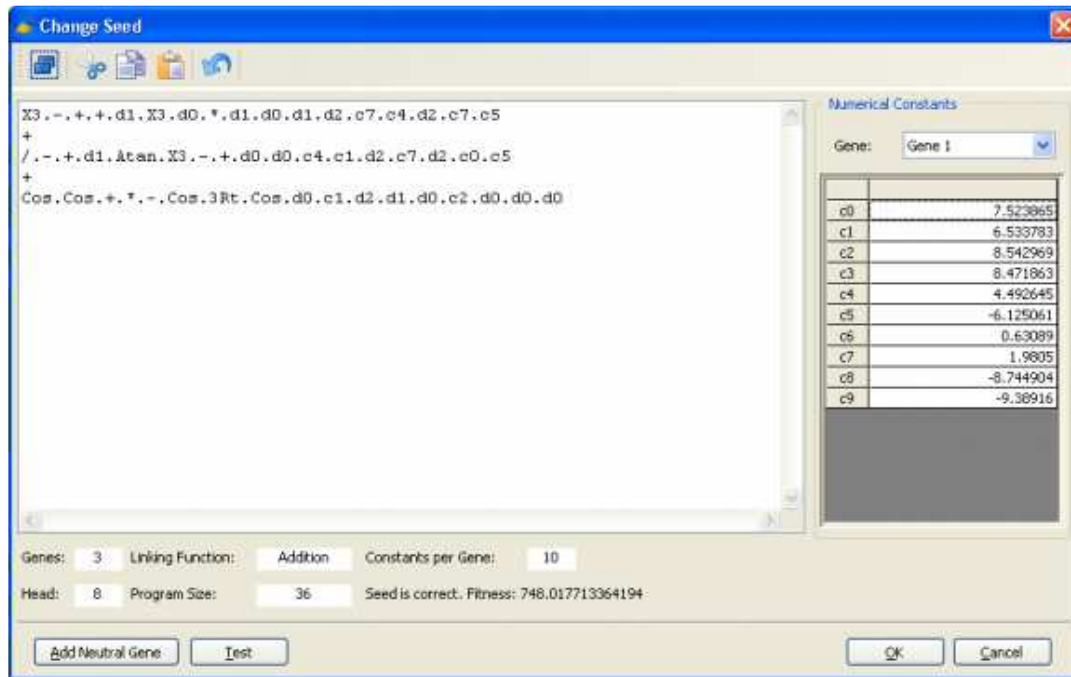
For models created in the GeneXproTools 4.0 environment, the seed is fed to the algorithm every time you click the Optimize or Simplify buttons during modeling, that is, the seed (in this case, the best model of the previous run) is introduced automatically by GeneXproTools.



Adding a Neutral Gene

The addition of a neutral gene to a seed might seem, at first sight, the wrong thing to do as most of the times we are interested in creating efficient and parsimonious models. But one should look at this as modeling in progress as, for really complex phenomena, it is not uncommon to approximate a complex function progressively.

Thus, being able of introducing extra terms in your seed is a powerful modeling tool and GeneXproTools 4.0 allows you to do that by selecting Add Neutral Gene in the Edit Menu or through the Change Seed window.



Here, by pressing the Add Neutral Gene button, you will see a neutral gene being added to your model. By doing this, you are giving the learning algorithm more room to play and, hopefully, a better and more complex program will evolve.

Neutral genes can also be introduced automatically by GeneXproTools 4.0 using the Complexity Increase Engine.

Complexity Increase Engine

GeneXproTools 4.0 also allows you to introduce neutral genes automatically by activating the Complexity Increase Engine in the Settings Panel -> General Settings Tab.

General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 200 [Reset]

Testing Samples: 246 [Reset]

Number of Chromosomes: 30

Head Size: 7

Number of Genes: 3

Linking Function: Addition

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 5

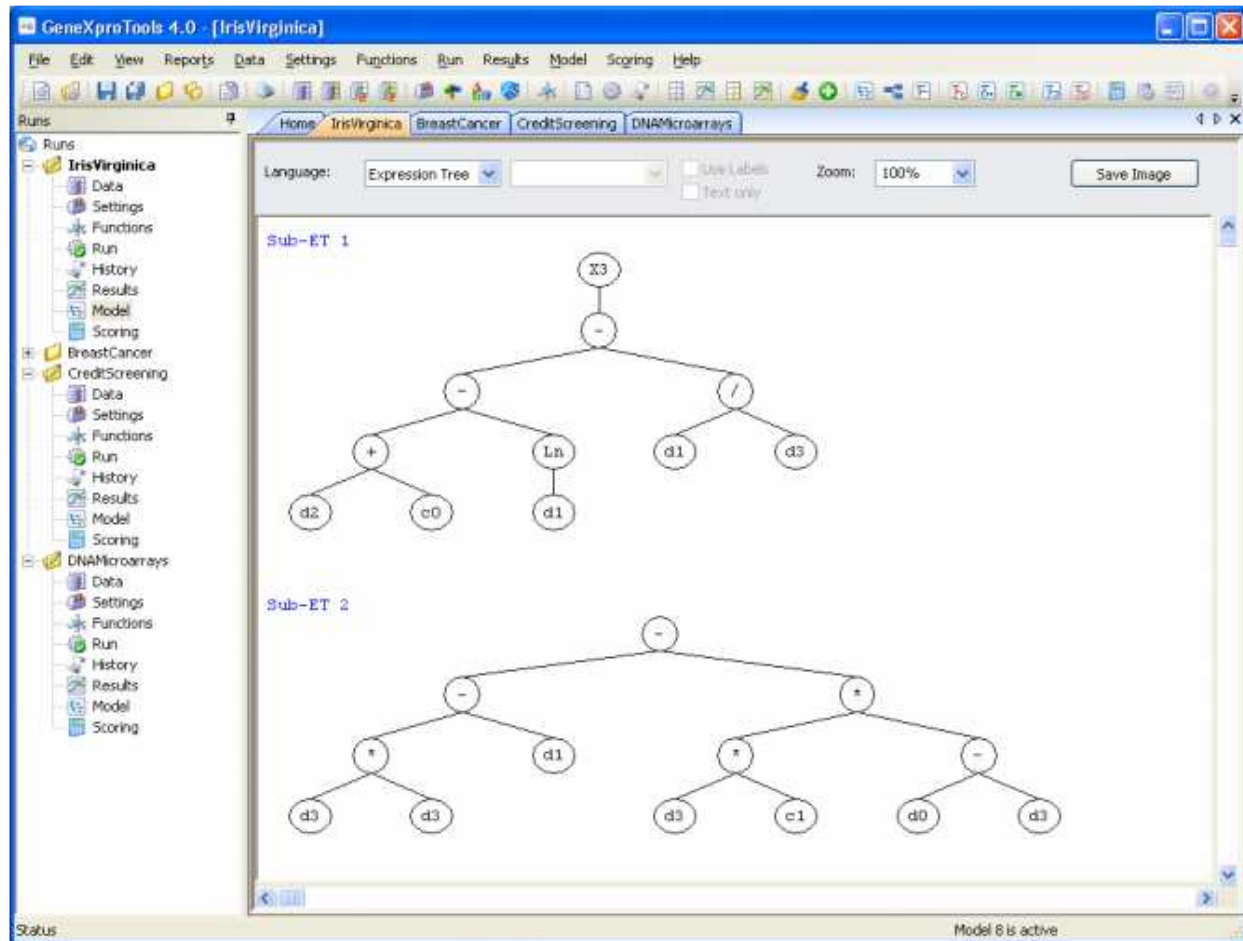
☒ Save Run History

Whenever you are using the Complexity Increase engine of GeneXproTools 4.0, you must fill the Generations Without Change box to set the period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction or a neutral gene (an extra term) is automatically added to your model; the Number of Tries box corresponds to the number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models; in the Max. Complexity box you write the maximum number of terms (genes) you'll allow in your model and no other terms will be introduced beyond this threshold.

The Complexity Increase Engine of GeneXproTools 4.0 might be a very powerful modeling tool, but you must be careful not to create excessively complex models for, most of the times, a greater complexity does not necessarily imply a greater efficiency.

Visualizing Models as Parse Trees

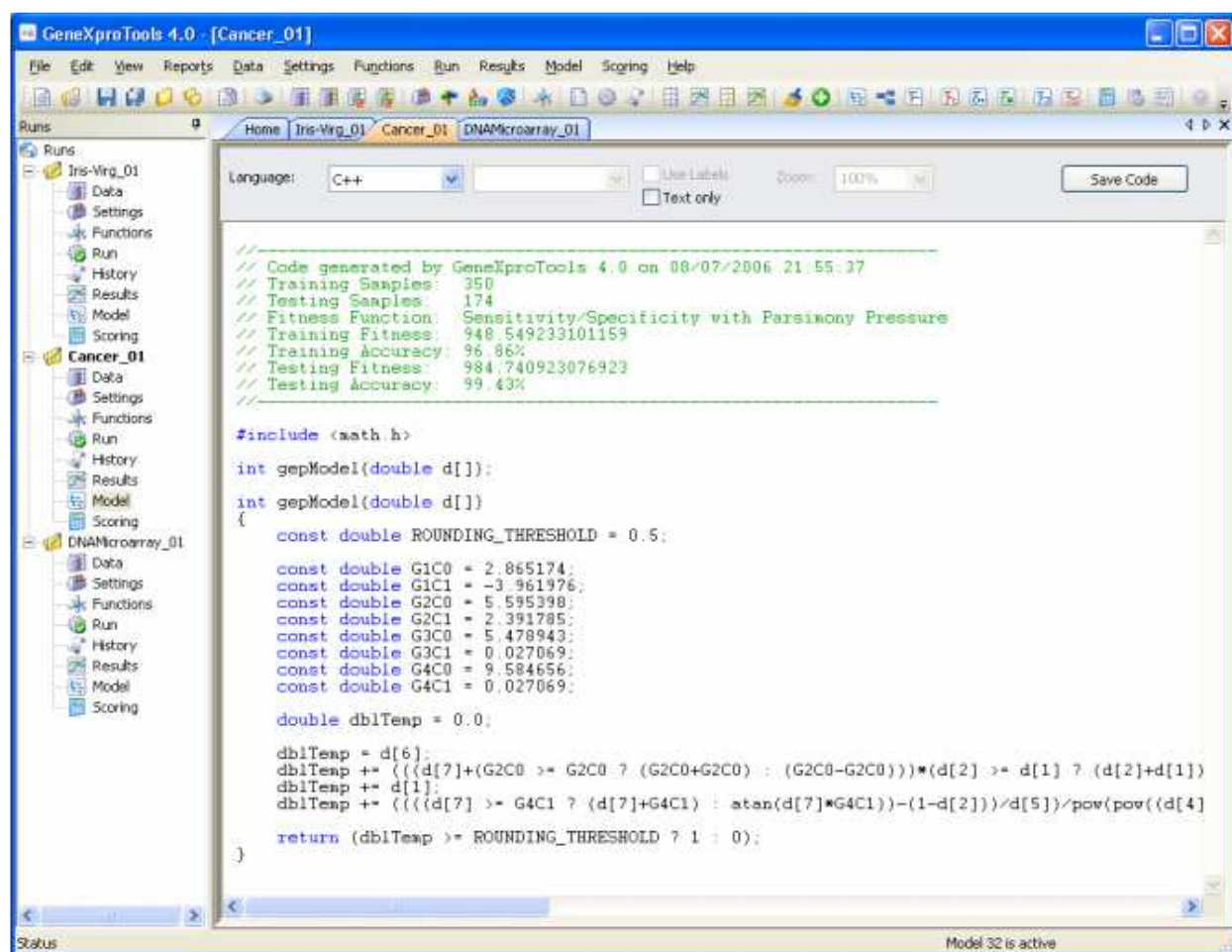
GeneXproTools 4.0 comes equipped with a parse tree generator that automatically converts your models into diagram representations or parse trees.



Indeed, all the models evolved by GeneXproTools in its native [Karva language](#) can be automatically parsed into visually appealing expression trees, allowing a quicker and more complete understanding of their mathematical intricacies.

Generating Code Automatically Using GeneXproTools Built-in Grammars

GeneXproTools 4.0 offers a total of 15 [built-in grammars](#) so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the most commonly used programming languages (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL). This code can then be used in other applications.



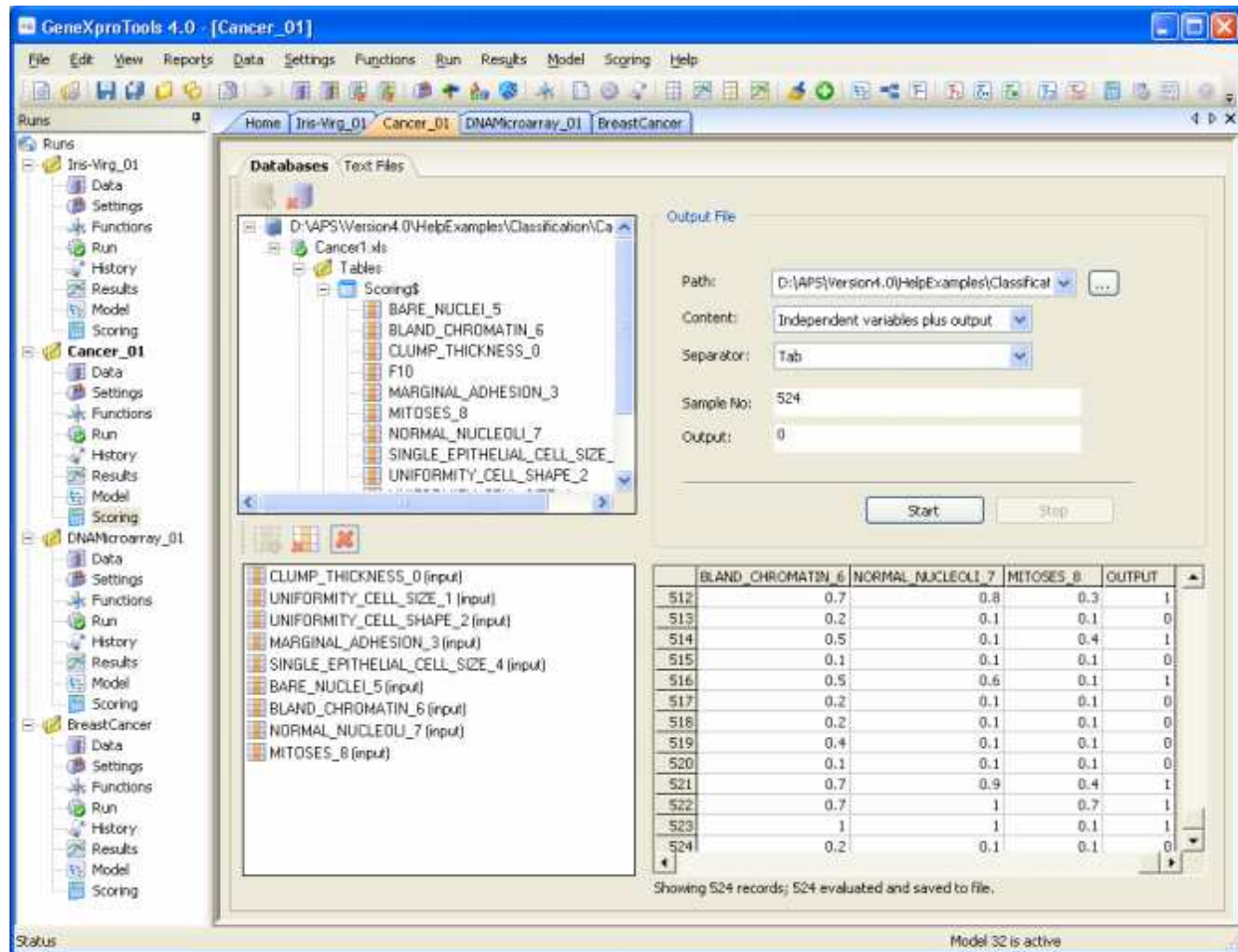
Generating Code Automatically Using User Defined Grammars

GeneXproTools 4.0 allows the design of [User Defined Grammars](#) so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the programming language of your choice if you happen to prefer a programming language not already covered by the 15 [built-in grammars](#) of GeneXproTools 4.0 (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL).

As an illustration, the C++ grammar of GeneXproTools 4.0 is shown [here](#). Other grammars may be easily created using this or other GeneXproTools built-in grammars as reference.

Scoring Engine

GeneXproTools 4.0 comes equipped with a scoring engine for scoring your databases/Excel or text files in the GeneXproTools environment. The scoring engine of GeneXproTools uses the JavaScript code of your models to score your data.



After its creation, a model is usually used multiple times to make predictions or extrapolations. The scoring engine of GeneXproTools 4.0 allows the immediate scoring of your databases as soon as a model is created, saving the scores to Excel (csv) or text files.



GeneXproTools Environment

Time Series Prediction Tools

The successful creation of Time Series Prediction models requires the use of very sharp modeling tools and the most important are the following:

- Time Series Transformation Engine
- Data Screening Engine
- Data Visualization Tool
- Built-in Fitness Functions
- User Defined Fitness Functions
- Built-in Mathematical Functions
- Function Selection Tool
- Dynamic User Defined Functions

- [Static User Defined Functions](#)
- [Learning Algorithms](#)
- [Monitoring the Modeling Process](#)
- [Analyzing Intermediate Models](#)
- [Simplifying a Model](#)
- [Comparing Actual and Predicted Values](#)
- [Essential Statistics](#)
- [Modeling from Seed Models](#)
- [Adding a Neutral Gene](#)
- [Complexity Increase Engine](#)
- [Visualizing Models as Parse Trees](#)
- [Generating Code Automatically Using Built-in Grammars](#)
- [Generating Code Automatically Using User Defined Grammars](#)
- [Making Predictions](#)

Time Series Transformation Engine

Before modeling, the data in a time series must be transformed so that it can be used to create prediction models.

GeneXproTools 4.0 comes with a time series transforming engine that transforms the time series according to the Embedding Dimension and Delay Time, but it also transforms the time series depending on whether you wish to make predictions about the future or test the predictive power of the evolved models on known past behavior.

GeneXproTools 4.0 Wizard - Run Category

New Run Wizard

This wizard will help you create a new run. You must enter a run name, the run category and the datasets. The training dataset is mandatory but the testing dataset is optional as it is only used to check the predictive accuracy of the evolved models.

Run Name:

Run Category:

Data Source Type:

Embedding Dimension:

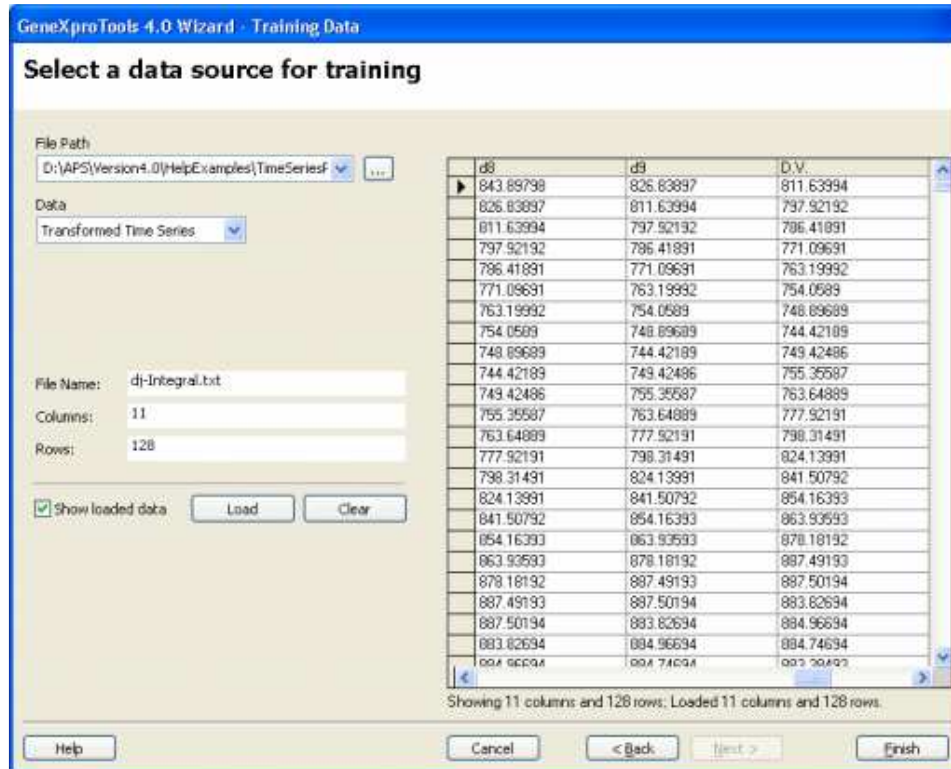
Delay Time:

Prediction Mode:

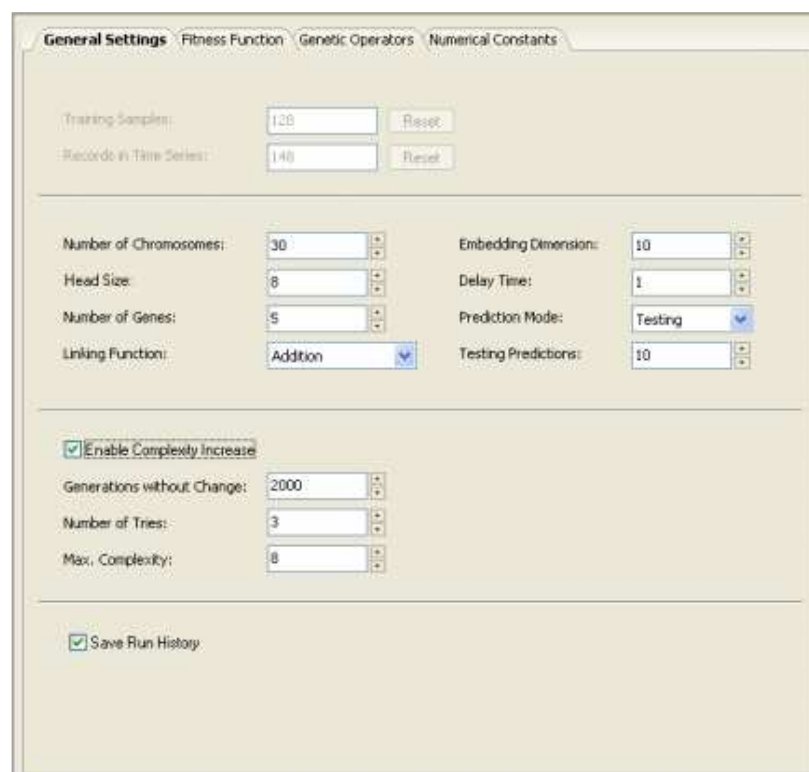
Testing Predictions:

Help Cancel < Back Next > Finish

You can then observe the transformation on the Training Data window before proceeding with the new run.



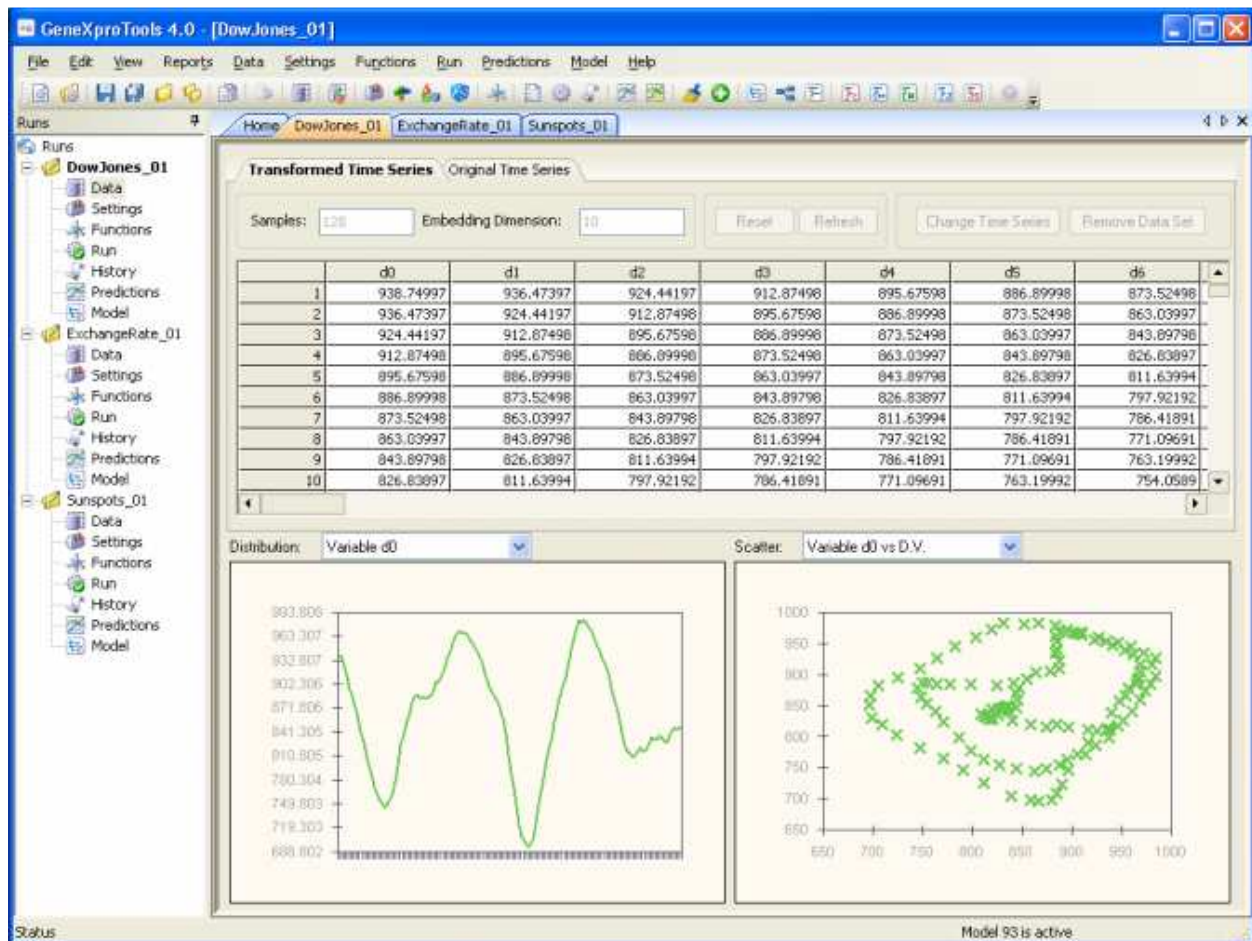
The time series transforming engine of GeneXproTools 4.0 is operational every time you create a new run or every time you change either the Embedding Dimension or the Delay Time, the Prediction Mode or the number of Testing Predictions during a run in the General Settings Tab.



In practice, the embedding dimension corresponds to the number of independent variables (terminals) after your time series has been transformed. The delay time τ determines how data are processed, that is, continuously if $\tau = 1$ or at τ intervals.

These two parameters, together with the size of the time series and the prediction mode, will determine the final number of training samples after the transformation of the time series.

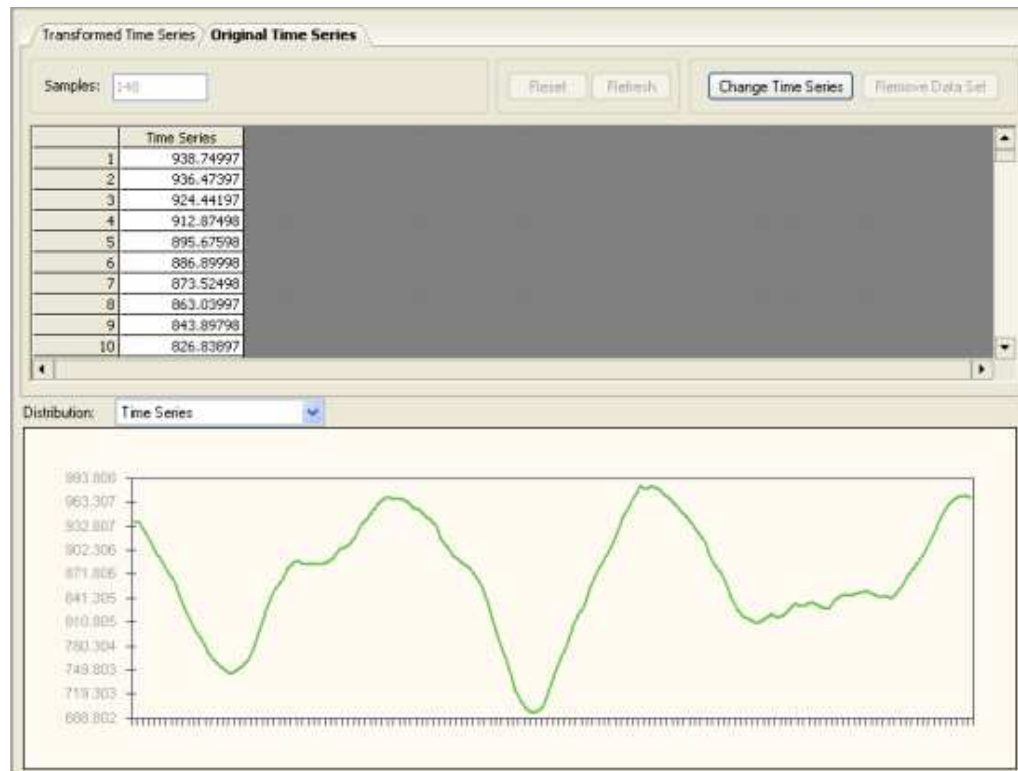
The data are then ready for evolving prediction models with them and can be visualized in the [Data Panel](#) before evolving a new model.



Data Screening Engine

Time series have a simple data format which consists of a column of time series observations.

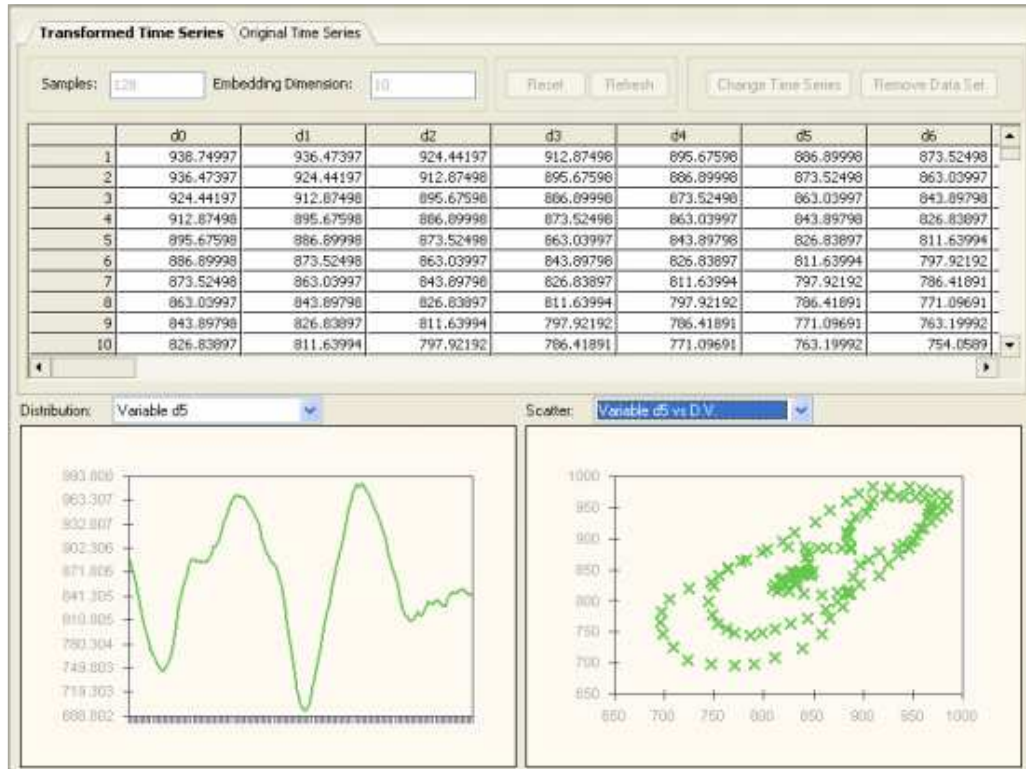
GeneXproTools 4.0 data screening engine makes sure that the right format is fed into the time series transforming engine of GeneXproTools so that the time series is transformed correctly either to create a model for making predictions or testing the predictive potential of the model against known behavior.



Data Visualization Tool

The data visualization tool of GeneXproTools 4.0 enables you to analyze the distribution of all the inputs.

The visualization of the distribution of all data points is a valuable modeling tool as it can help you detect outliers in your data that can then be removed so that better models are created. Typographical or measurement errors generally cause outliers that can be detected by analyzing the time series graphically.



Built-in Fitness Functions

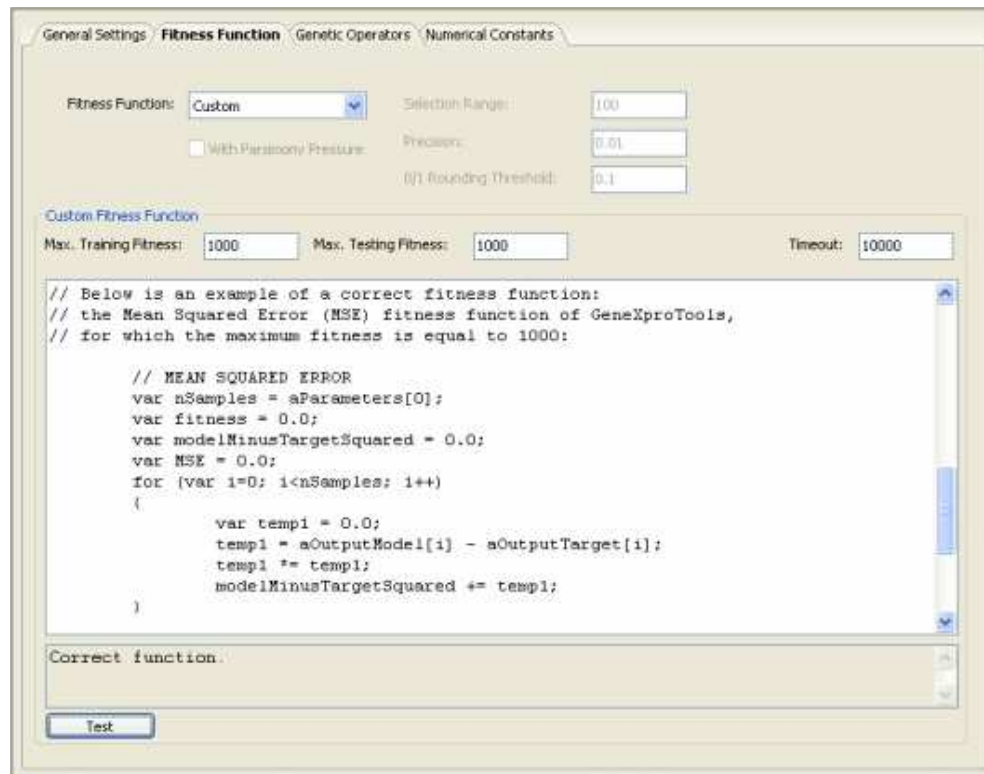
GeneXproTools 4.0 offers a total of 36 [built-in fitness functions](#) based on well-known statistical functions.

Fitness functions are fundamental modeling tools for they determine the nature of the search landscape. And different fitness functions open and explore different paths in the solution space. The fitness functions of GeneXproTools 4.0 are named as follows: [Absolute with SR](#), [Absolute/Hits](#), [MSE](#), [RMSE](#), [MAE](#), [RSE](#), [RRSE](#), [RAE](#), [Relative with SR](#), [Relative/Hits](#), [rMSE](#), [rRMSE](#), [rMAE](#), [rRSE](#), [rRRSE](#), [rRAE](#), [R-square](#), and [Correlation Coefficient](#). Furthermore, all these fitness functions have their own counterpart with [parsimony pressure](#).

User Defined Fitness Functions

Despite the wide set of GeneXproTools [built-in fitness functions](#), some users sometimes want to experiment with fitness functions of their own. GeneXproTools 4.0 gives the user the possibility of creating custom tailored fitness functions and evolve models with them.

The code for the custom fitness function must be in JavaScript and is written in the Custom Fitness Function window and can be tested before evolving a model with it.



Built-in Mathematical Functions

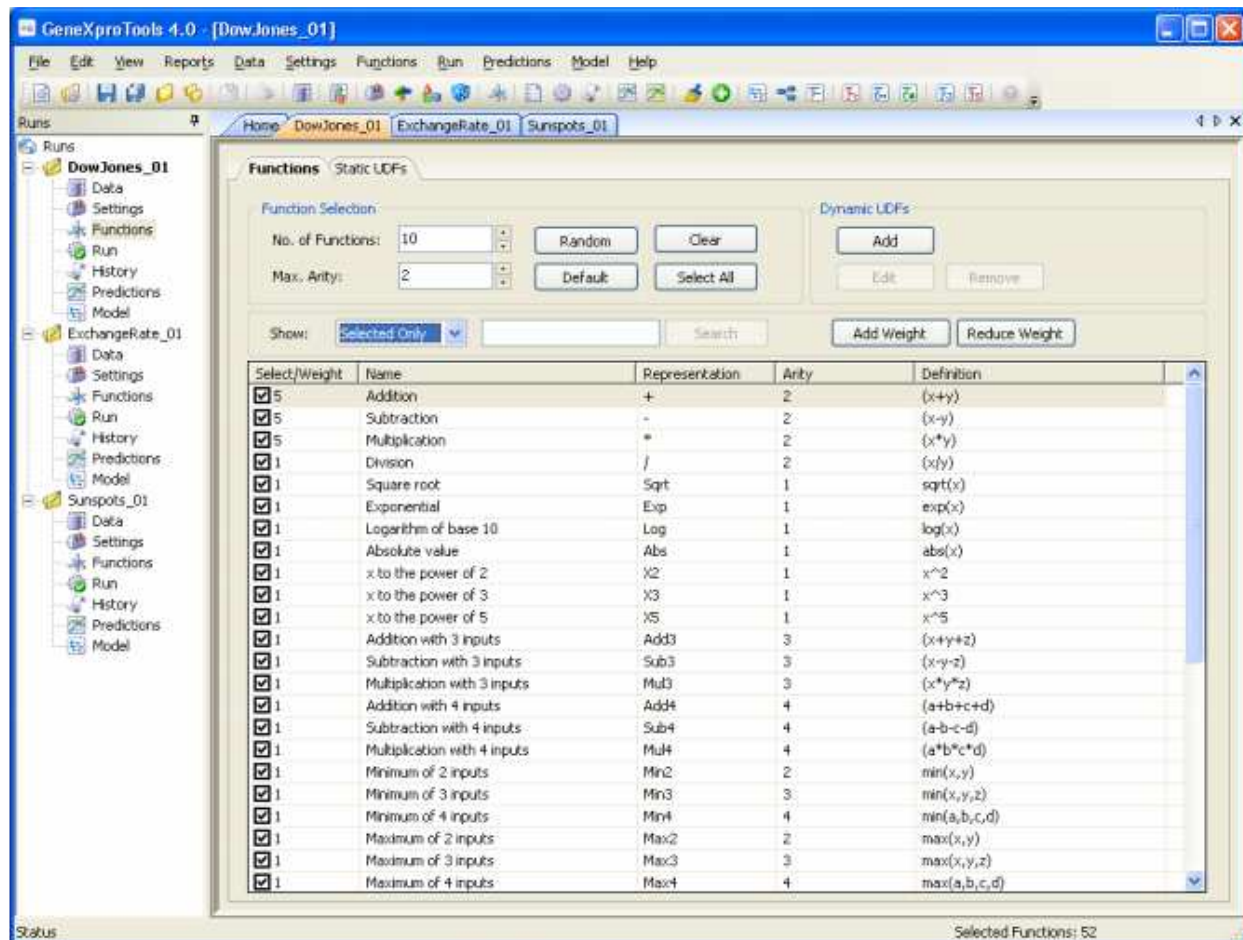
GeneXproTools 4.0 offers a total of 279 [built-in mathematical functions](#). This wide set of mathematical functions allows the evolution of complex and rigorous models quickly built with the most appropriate functions.

From the simple arithmetic operators to complex mathematical functions available in most programming languages to more complex mathematical functions commonly used by engineers and scientists, the modeling algorithms of GeneXproTools 4.0 enable you the integration of the most appropriate functions in your models.

Function Selection Tool

For some problems, the simple arithmetic operators are more than enough to create extremely accurate and elegant models. But for other problems more complex functions are necessary to create good models and when inside knowledge of the problem at hand is not producing the expected results, it might be interesting to experiment with different combinations of functions. The Function Selection Tool of GeneXproTools 4.0 helps you to experiment with different function sets very quickly either by pressing the Random button in the Functions Panel or by selecting certain sub-sets of functions.

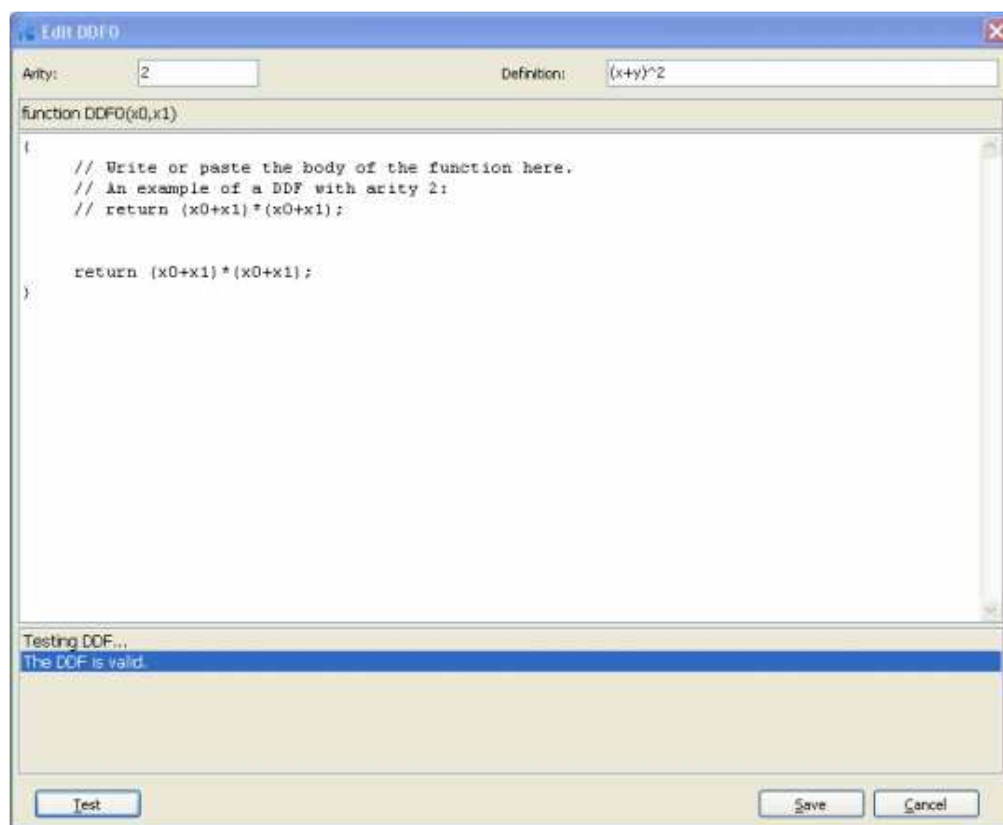
In addition, although you can always handcraft your function set so that it is well-balanced relatively to the embedding dimension in the problem at hand, in GeneXproTools 4.0 this is no longer essential as GeneXproTools balances automatically all function sets. This is particularly important for high-dimensional datasets.



Dynamic User Defined Functions (DDFs)

Despite the wide set of GeneXproTools [built-in mathematical functions](#), some users sometimes want to model with different ones. GeneXproTools 4.0 gives the user the possibility of creating custom tailored functions and evolve models with them. A note of warning though: the use of dynamic UDFs slows considerably the evolutionary process and therefore should be used with moderation.

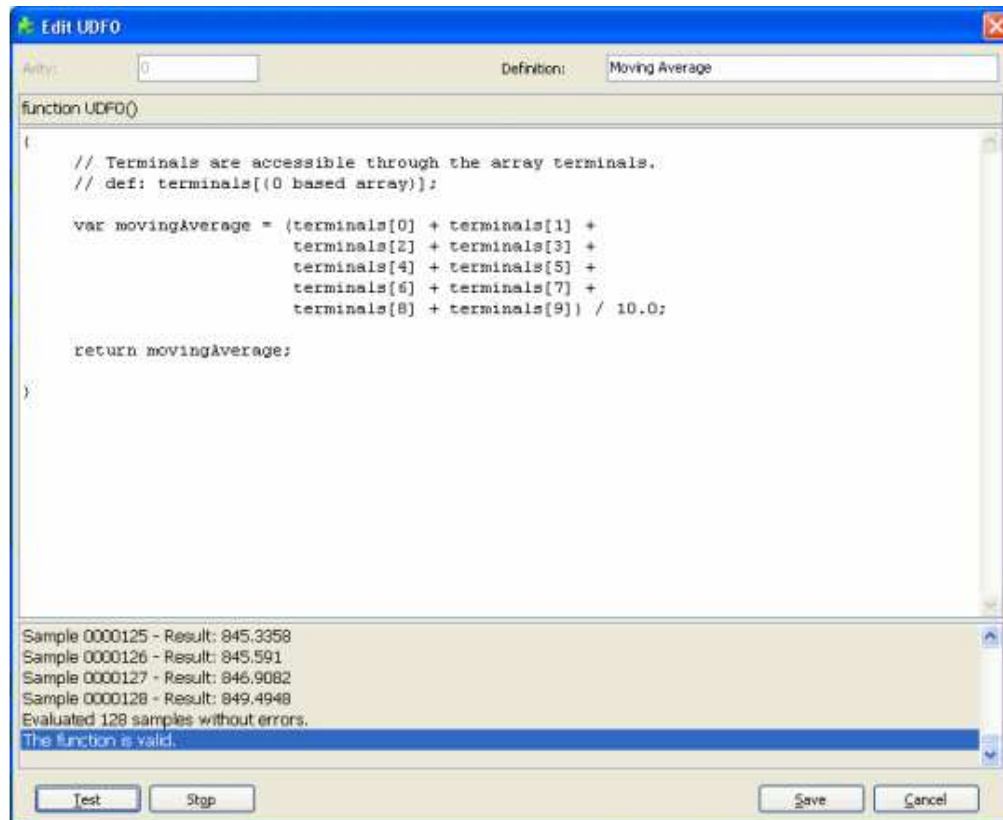
The code for the DDFs must be in JavaScript and is written in the Edit DDF window (in the Functions Panel, select the Functions Tab and then click Add in the Dynamic UDFs tool box).



Static User Defined Functions (UDFs)

Sometimes it is possible to spot relatively simple relationships between certain variables in your data, and GeneXproTools 4.0 allows you to use this information to build custom tailored functions (UDFs). These functions allow the discovery of more complex models composed of several simpler models.

The code for the UDFs must be in JavaScript and is written in the Edit UDF window (in the Functions Panel, select the Static UDFs Tab and then click Add in the Static UDFs tool box).



Learning Algorithms

GeneXproTools 4.0 is equipped with two different [learning algorithms](#) for evolving complex nonlinear prediction models.

The first is the fastest and simplest of the two. The second algorithm is very similar to the first, with the difference that it gives the user the possibility of choosing the range and type of numerical constants that will be used during the learning process. And because numerical constants are crucial to a wide set of problems, the second algorithm is the default in GeneXproTools 4.0. Keep in mind, however, that for problems where numerical constants are not necessary it pays to use the first algorithm as the models are not only much simpler but also the evolutionary process is much quicker.

In order to evolve models using the second algorithm, select Numerical Constants in the Settings Panel and then check the box that activates this algorithm. You will notice that additional parameters become available, including a small set of genetic operators especially developed for handling numerical constants (if you are not familiar with these operators, please use the default values for they work very well in all cases). To use the first algorithm, just uncheck the Use Random Numerical Constants box.

Monitoring the Modeling Process

GeneXproTools 4.0 allows you to monitor the modeling process by plotting the essential parameters of a run during the discovery process, including average fitness of the population and the fitness and R-square of the best-of-generation model.

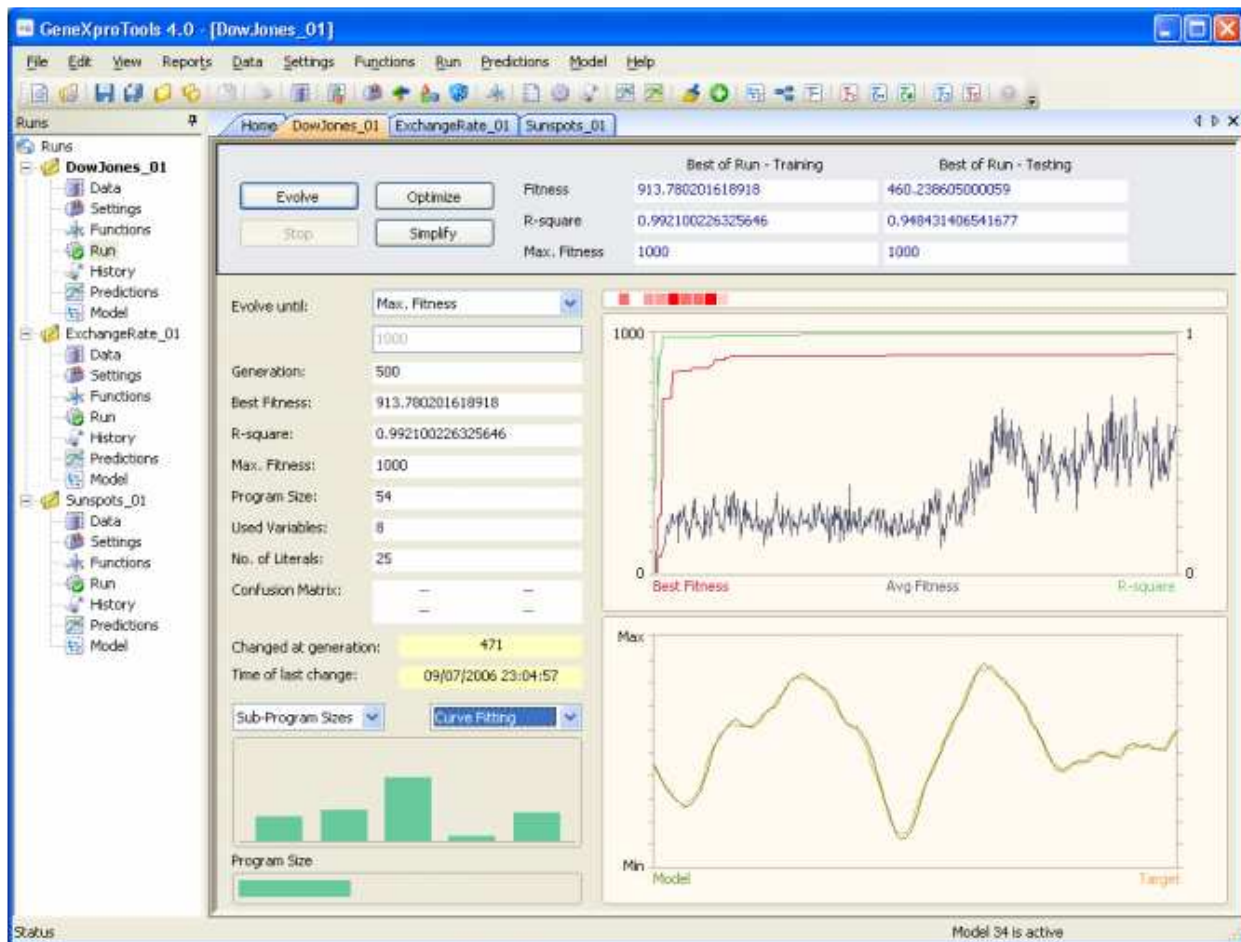
In addition, GeneXproTools allows you to visualize the modeling process by plotting the essential features of a run during the discovery process. In the [Run Panel](#), a total of eight different charts are shown:

1. The Variables Usage map.
This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square you can access the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic.
2. The Evolutionary Dynamics chart.
This chart shows the average fitness of the population plus the fitness and R-square of the best-of-generation model.
3. The Average/Best Size chart.
This chart shows the average program size of the population versus the program size of the best-of-generation model. This chart is especially useful during simplification and can be activated any time during evolution by selecting Avg/Best Size in the rightmost combo box in the bottom.
4. The Curve Fitting chart.
This chart plots the first 250 data points of the training set and shows how well the evolving models are fitting the target. The Curve Fitting chart can be activated any time during evolution by selecting Curve Fitting in the rightmost combo box in the bottom.
5. The Sub-Program Sizes chart.
This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be activated any time during evolution by selecting Sub-Program Sizes in the leftmost combo box in the bottom.
6. The All Sizes chart.
This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be activated any time during evolution by selecting All Sizes in the leftmost combo box in the bottom.
7. The All Fitnesses chart.
This chart shows the fitnesses of all the models in the population. Not only after a run but also

during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be activated any time during evolution by selecting All Fitnesses in the leftmost combo box in the bottom.

8. The Program Size chart.

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model.



The evolutionary process can be stopped whenever you are satisfied with the results by pressing the Stop button or you can use one of the stop criteria of GeneXproTools for picking up your model exactly as you want it.

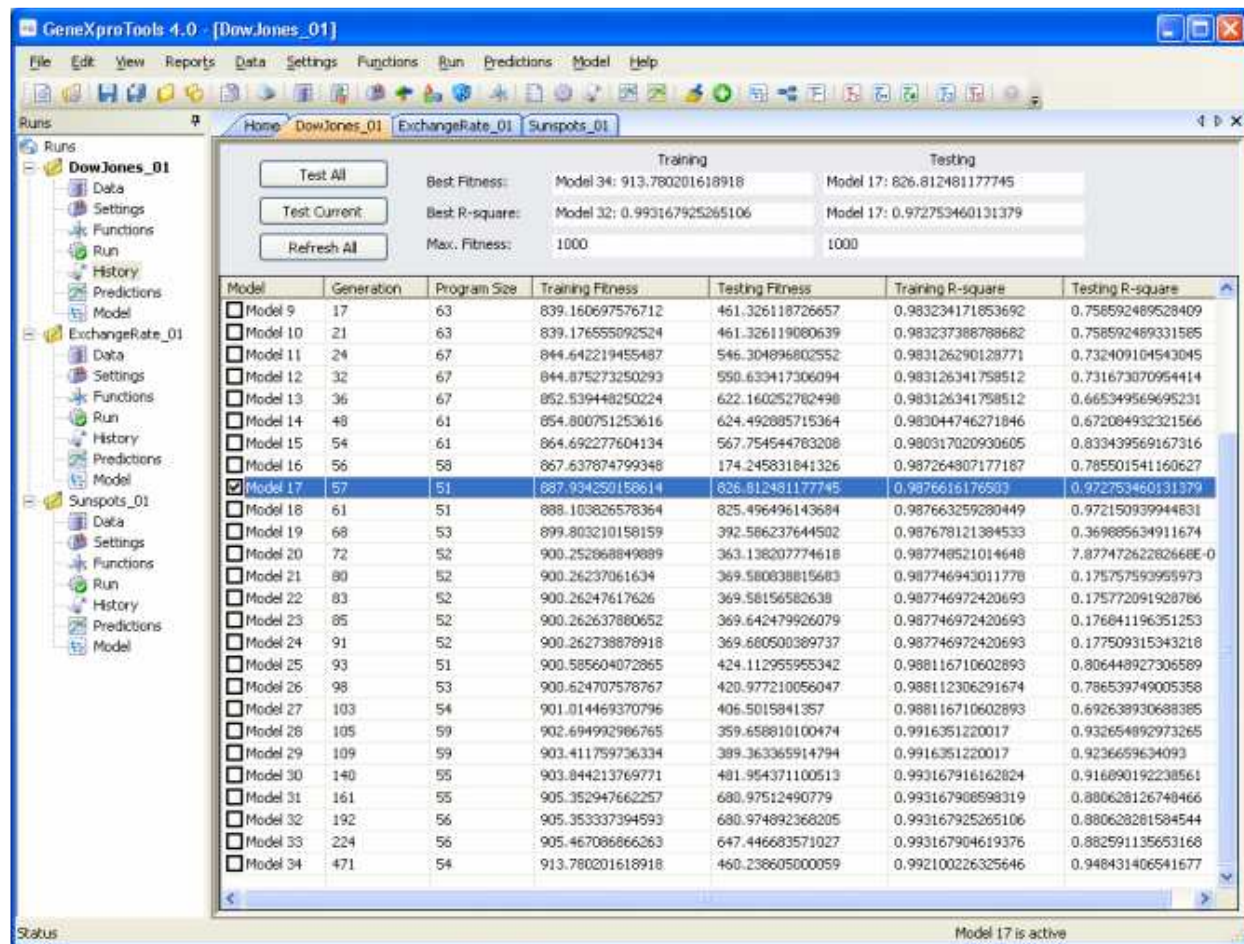
When the evolutionary process stops, the best-of-run model is ready either for analysis or for making predictions. And if you are still not happy with the results, you can still try to fine-tune the evolved model by pressing the Optimize button and repeat this process until you are completely satisfied with your model. In addition, you might also wish to try and simplify your model and for that you just have to press the Simplify button.

Analyzing Intermediate Models

GeneXproTools 4.0 gives you the possibility of gaining some insights into the modeling process by analyzing all the best-of-generation intermediate models.

All the best-of-generation models discovered during a run are saved and you can pick them up for analysis at the [History Panel](#) by checking the model you are interested in. Each of these intermediate models can then be analyzed exactly as you do for the best-of-run model, that is, you can either check

their performance in the testing samples if you are working on Testing Mode or use them to make predictions if you are in Prediction Mode, check their vital statistics, automatically generate code with them, visualize their parse trees, use them as seed to create better models from them and so forth.



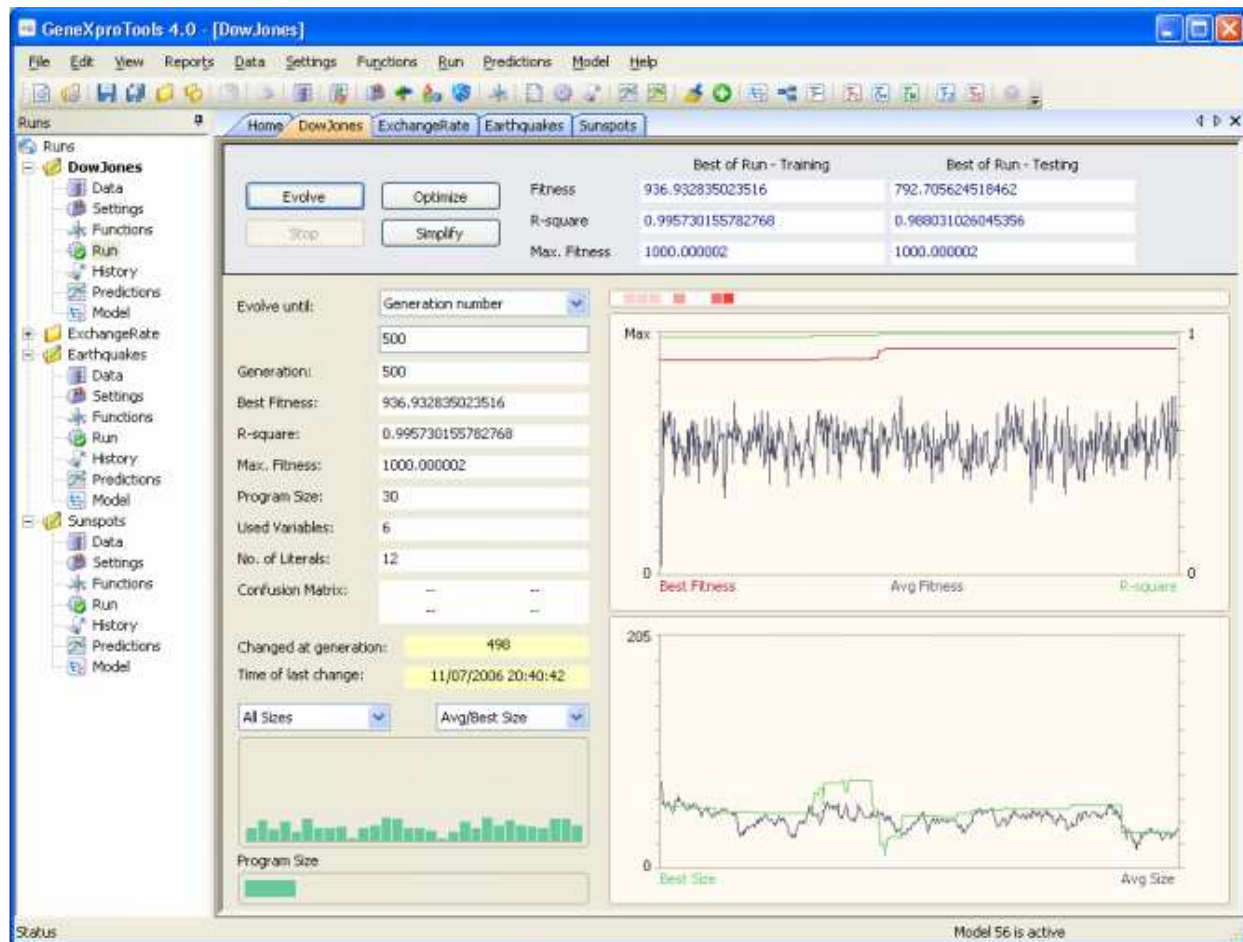
Simplifying a Model

GeneXproTools 4.0 allows you to simplify an existing model (which could have been either generated by GeneXproTools or by another modeling tool) either by pressing the Simplify button on the [Run Panel](#) or by choosing a fitness function with parsimony pressure and then press Optimize or Simplify on the Run Panel.

For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.

Then, in the [Run Panel](#), by clicking the Simplify button, an evolutionary process starts in which all the subsequent models will be descendants of the model you want to simplify. Keep in mind, however, that the simplification algorithm used by GeneXproTools is an evolutionary one and, despite parsimony pressure being applied, models continue to be selected preferentially by fitness and sometimes their complexity might even increase temporarily if the gain in fitness outweighs the loss in simplicity.

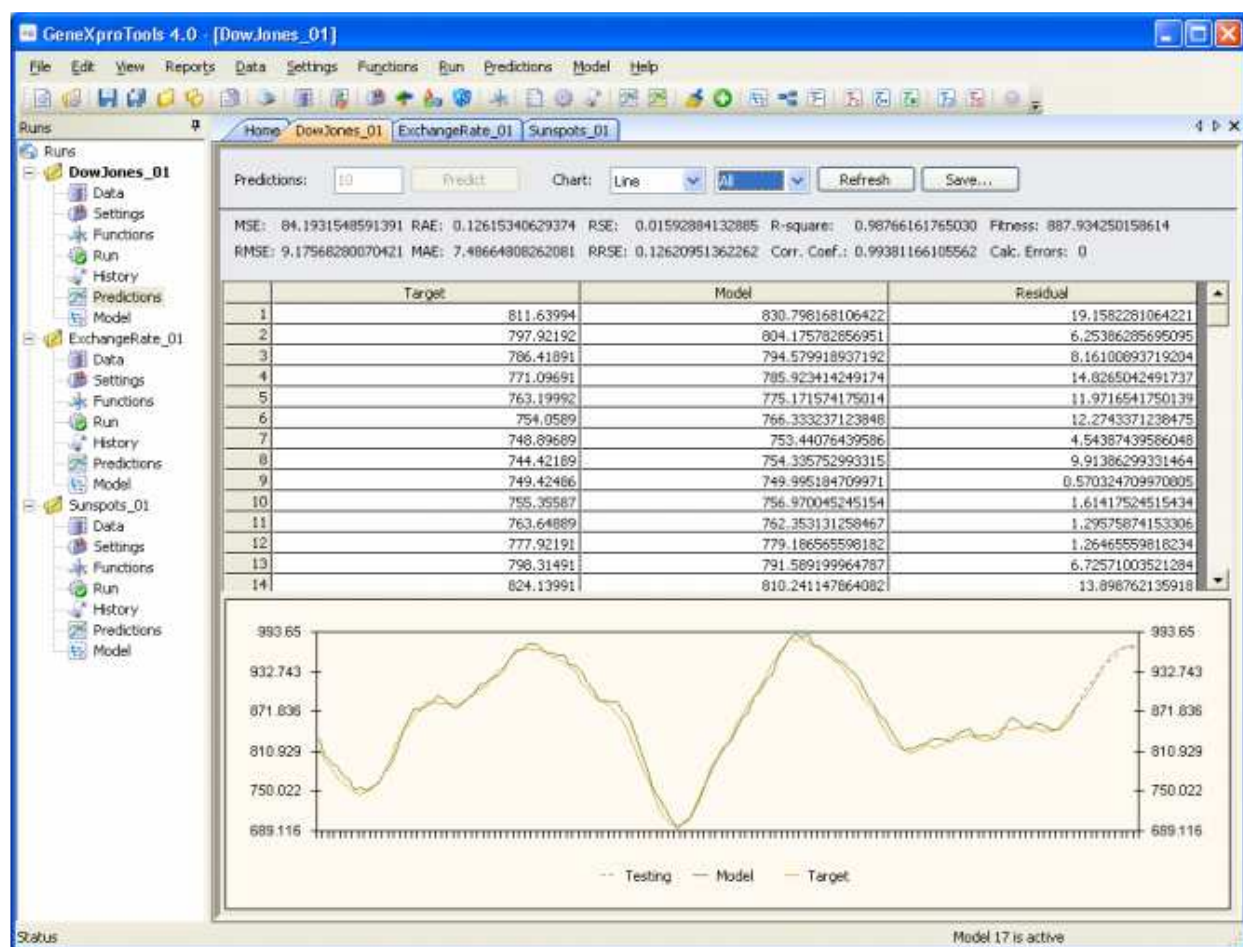
For models created in the GeneXproTools 4.0 environment, you just have to select the model you want to simplify (either the best-of-run or an intermediate model picked up in the [History Panel](#)) and then click the Simplify button and let the algorithm create better descendants not only in terms of fitness but also in terms of size.



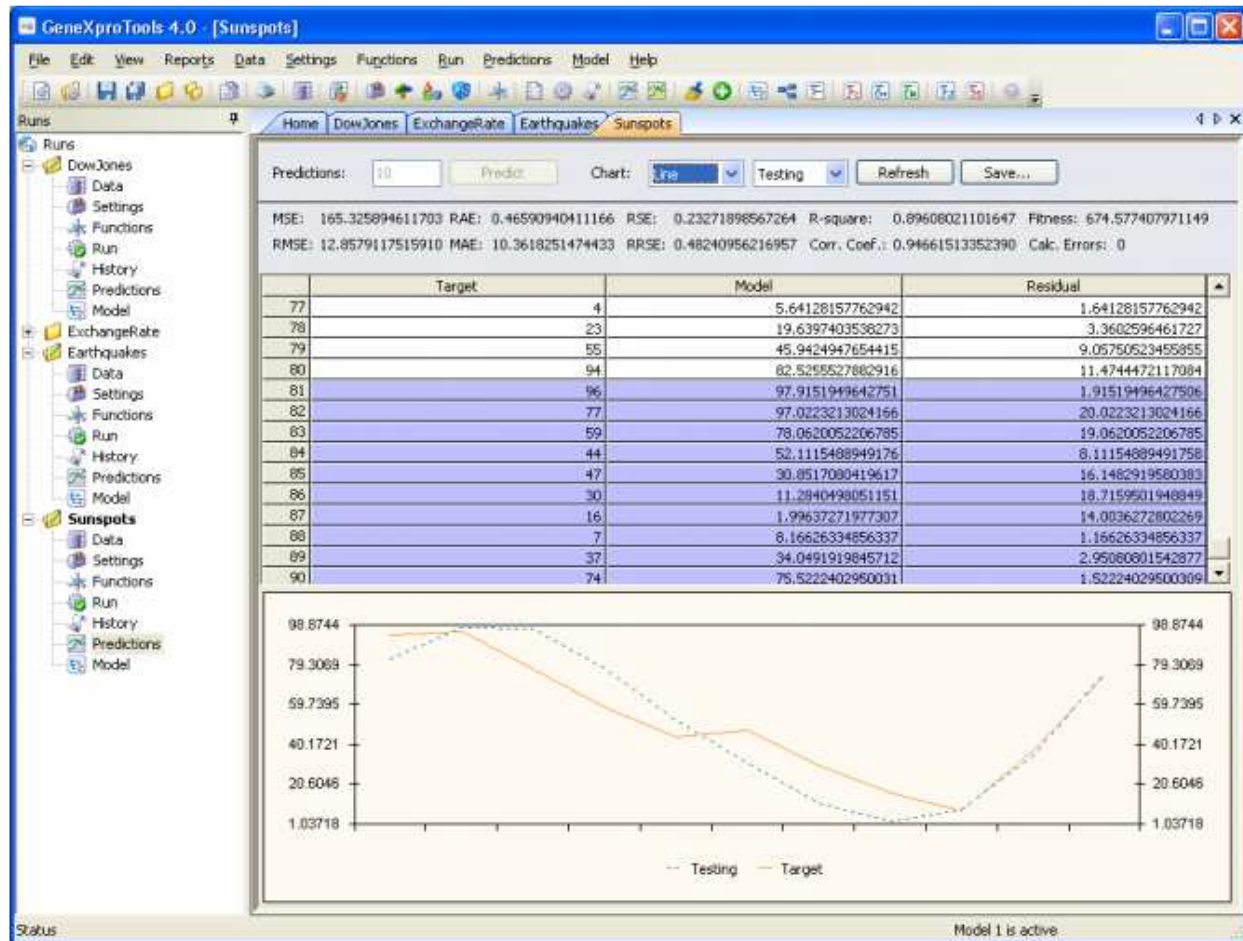
Comparing Actual and Predicted Values

GeneXproTools 4.0 offers two different ways of analyzing and comparing the output of your model with the actual or target values both for training and testing.

In the first, the target or actual values are listed in a table side by side with the predicted or model values. In the second, the target and predicted values are plotted in a Curve Fitting chart for easy visualization. And both table and chart are shown simultaneously in the [Predictions Panel](#). By moving the scroll bar in the right, you can see the results for all the data points.

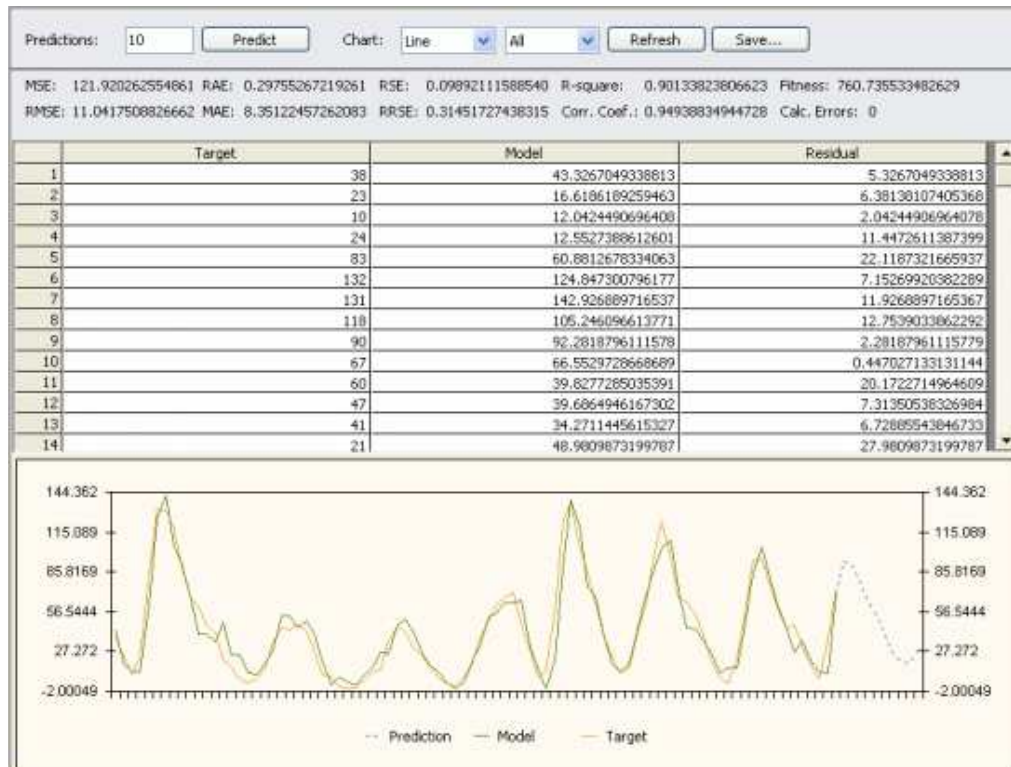


If you are testing the predictive power of your model using past known observations, the comparison of the predictions with the actual values are listed at the end of the table and highlighted in blue and are also plotted in the chart. In addition, these testing predictions can either be observed together with the training results or separately in a different Curve Fitting plot by selecting Testing in the Chart options.

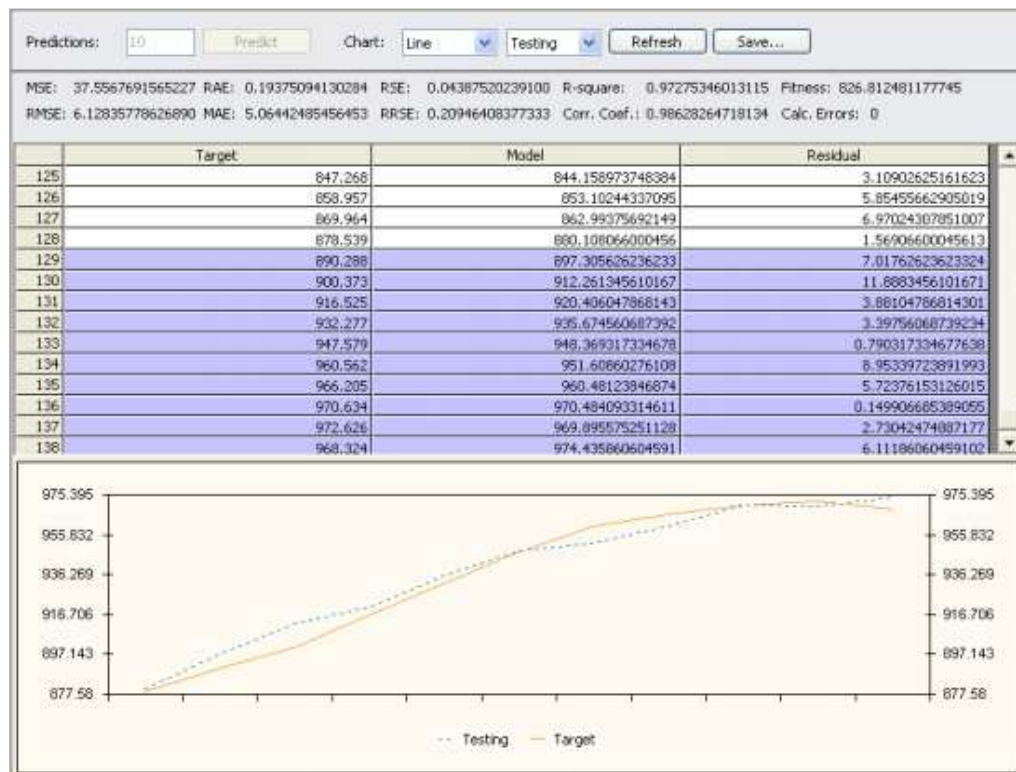


Essential Statistics

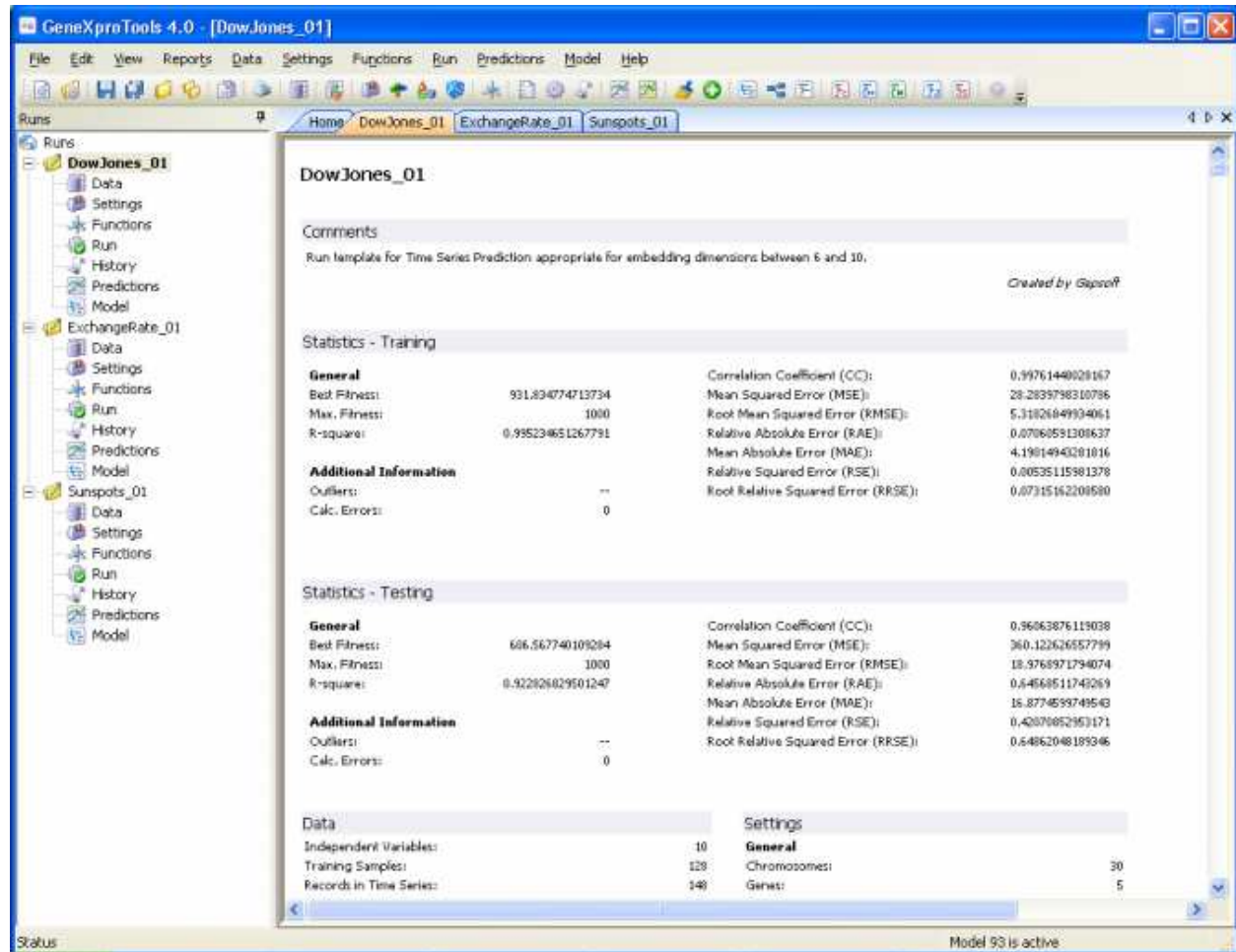
GeneXproTools 4.0 allows a quick and easy assessment of a wide set of [statistical functions](#). Most of them are immediately computed and shown every time you check the Predictions Panel (for instance, [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [R-square](#), and [correlation coefficient](#)).



If you are testing the predictive power of your model using past known observations, you can also check the same set of statistical indexes ([mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [R-square](#), and [correlation coefficient](#)) obtained for the recursive testing, by selecting Testing in the Chart options. You'll notice that the values of the statistical functions are updated, and they refer to the results obtained in the testing.



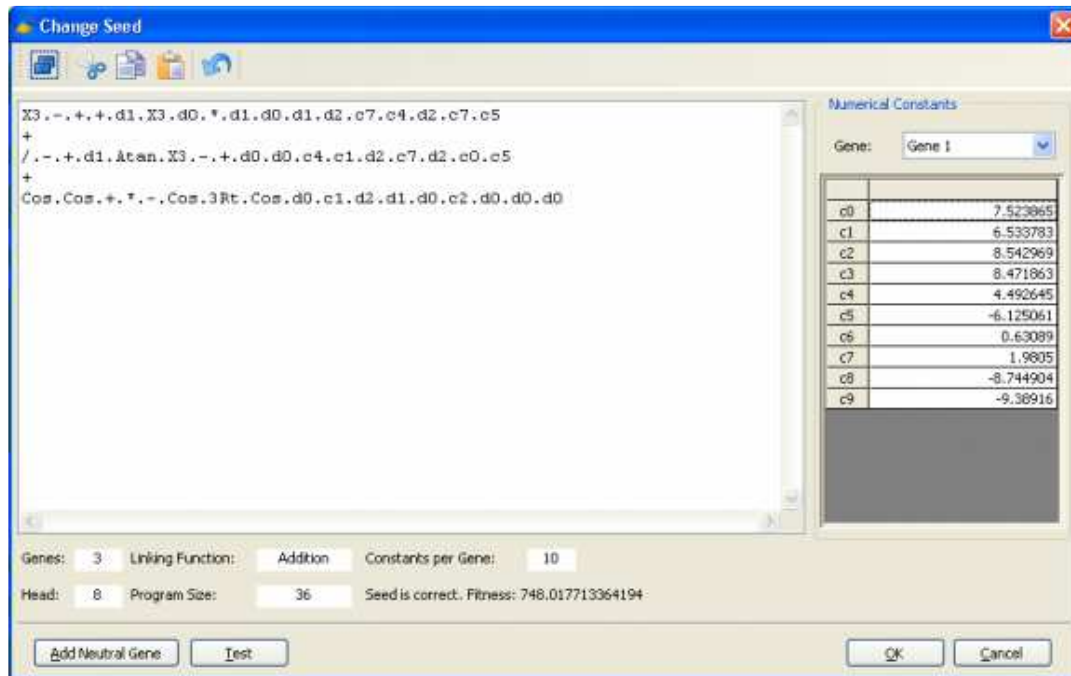
These and other statistical functions ([R-square](#), [correlation coefficient](#), and [fitness](#)) are also shown in the [Report Panel](#) so that they can be used for future reference, but you must check them first in the [Predictions Panel](#).



Modeling from Seed

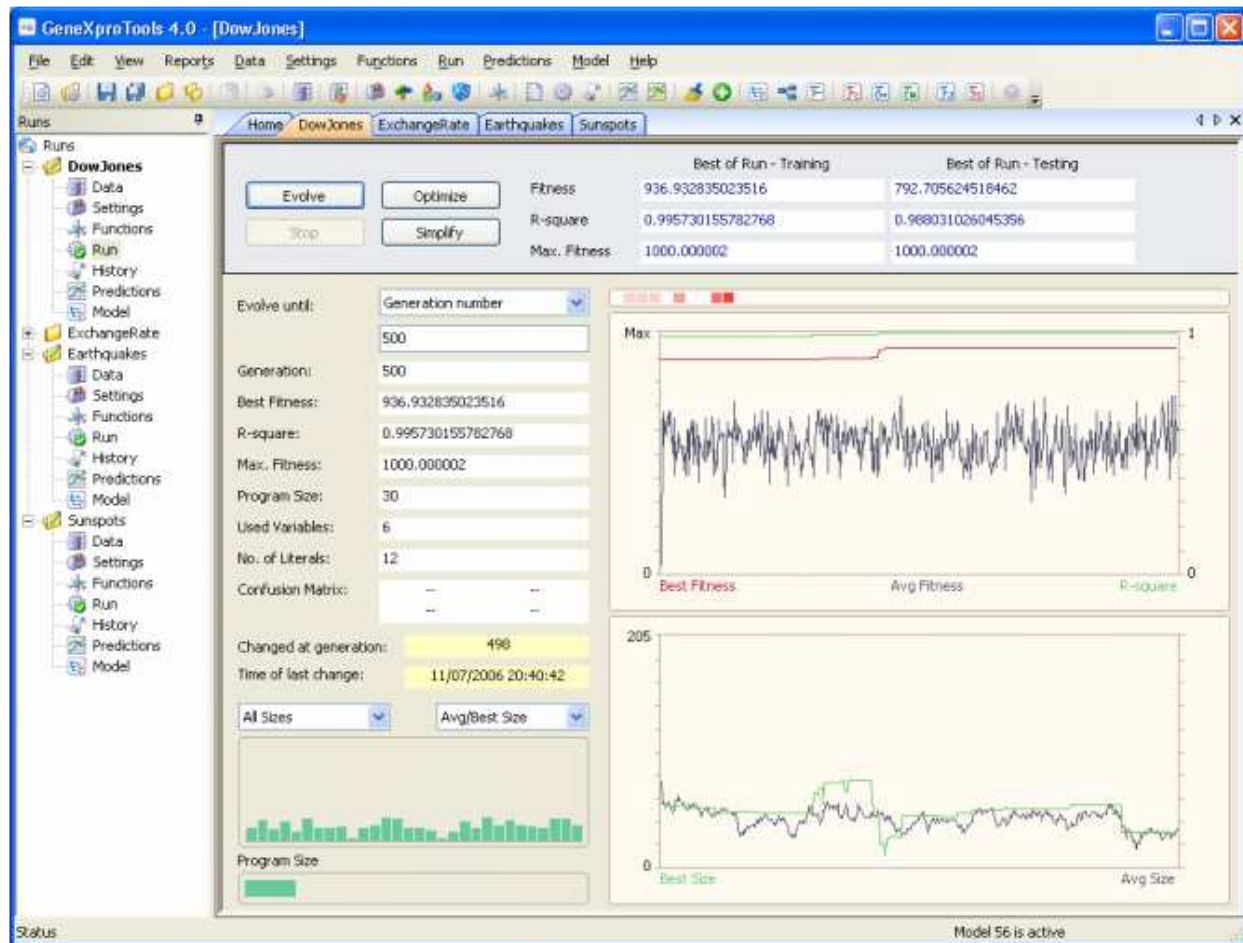
GeneXproTools 4.0 allows the use of an existing model (which could have been either generated by GeneXproTools or by another modeling tool) as the starting point of an evolutionary process so that more complex, finely tuned models could be created.

For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model or seed is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.



Then, in the [Run Panel](#), by clicking the Optimize button, an evolutionary process starts in which all the subsequent models will be descendants of the seed you introduced. A note of caution though: if your seed has a very small fitness, you risk losing it early in the run as better models could be randomly created by GeneXproTools and your seed, most probably, would not be selected for breeding new models. If your seed has zero fitness, though, you will receive a warning so that you could modify your seed until it becomes a viable seed capable of breeding new models.

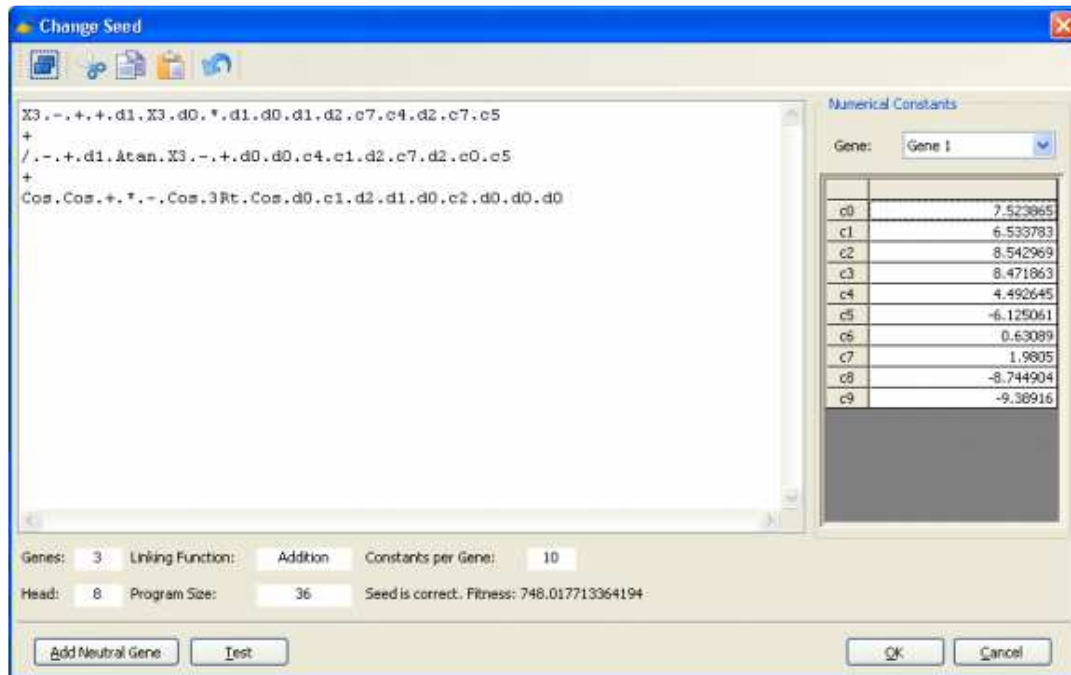
For models created in the GeneXproTools 4.0 environment, the seed is fed to the algorithm every time you click the Optimize or Simplify buttons during modeling, that is, the seed (in this case, the best model of the previous run) is introduced automatically by GeneXproTools.



Adding a Neutral Gene

The addition of a neutral gene to a seed might seem, at first sight, the wrong thing to do as most of the times we are interested in creating efficient and parsimonious models. But one should look at this as modeling in progress as, for really complex phenomena, it is not uncommon to approximate a complex function progressively.

Thus, being able of introducing extra terms in your seed is a powerful modeling tool and GeneXproTools 4.0 allows you to do that by selecting Add Neutral Gene in the Edit Menu or through the Change Seed window.



Here, by pressing the Add Neutral Gene button, you will see a neutral gene being added to your model. By doing this, you are giving the learning algorithm more room to play and, hopefully, a better, more complex program will evolve.

Neutral genes can also be introduced automatically by GeneXproTools 4.0 using the Complexity Increase Engine.

Complexity Increase Engine

GeneXproTools 4.0 also allows you to introduce neutral genes automatically by activating the Complexity Increase Engine in the Settings Panel -> General Settings Tab.

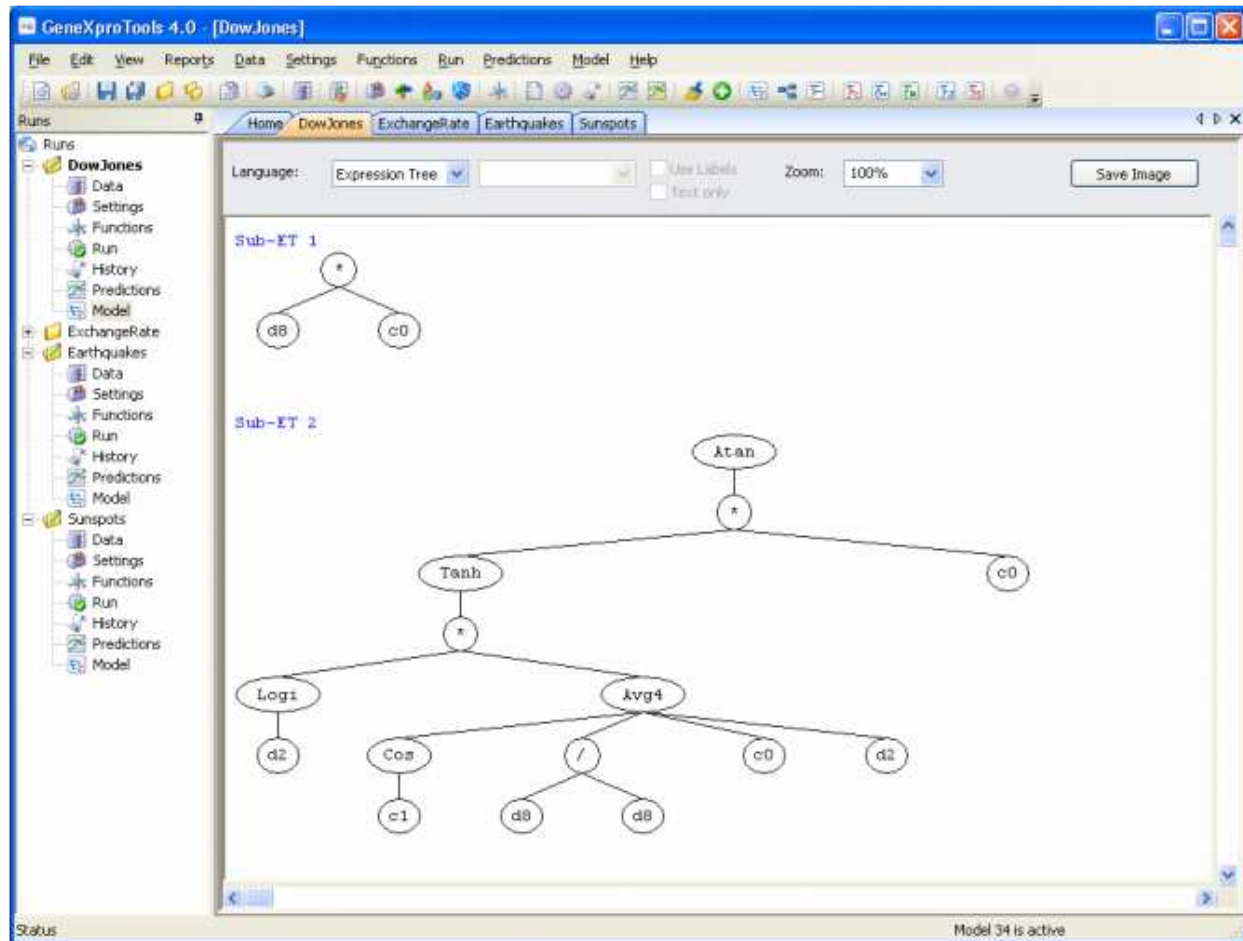
The screenshot shows the 'General Settings' tab of the GeneXproTools 4.0 interface. It features several input fields and checkboxes for configuring the evolutionary model. The fields include 'Training Samples' (128), 'Records in Time Series' (148), 'Number of Chromosomes' (30), 'Embedding Dimension' (10), 'Head Size' (8), 'Delay Time' (1), 'Number of Genes' (5), 'Linking Function' (Addition), 'Prediction Mode' (Testing), and 'Testing Predictions' (10). There are also checkboxes for 'Enable Complexity Increase' and 'Save Run History', both of which are checked. The 'Generations without Change' is set to 2000, 'Number of Tries' is 3, and 'Max. Complexity' is 8.

Whenever you are using the Complexity Increase Engine of GeneXproTools 4.0, you must fill the Generations Without Change box to set the period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction or a neutral gene (an extra term) is automatically added to your model; the Number of Tries box corresponds to the number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models; in the Max. Complexity box you write the maximum number of terms (genes) you'll allow in your model and no other terms will be introduced beyond this threshold.

The Complexity Increase Engine of GeneXproTools 4.0 might be a very powerful modeling tool, especially for time series prediction models where good models could take longer to come by, but you must be careful not to create excessively complex models as a greater complexity does not necessarily imply a greater predictive power.

Visualizing Models as Parse Trees

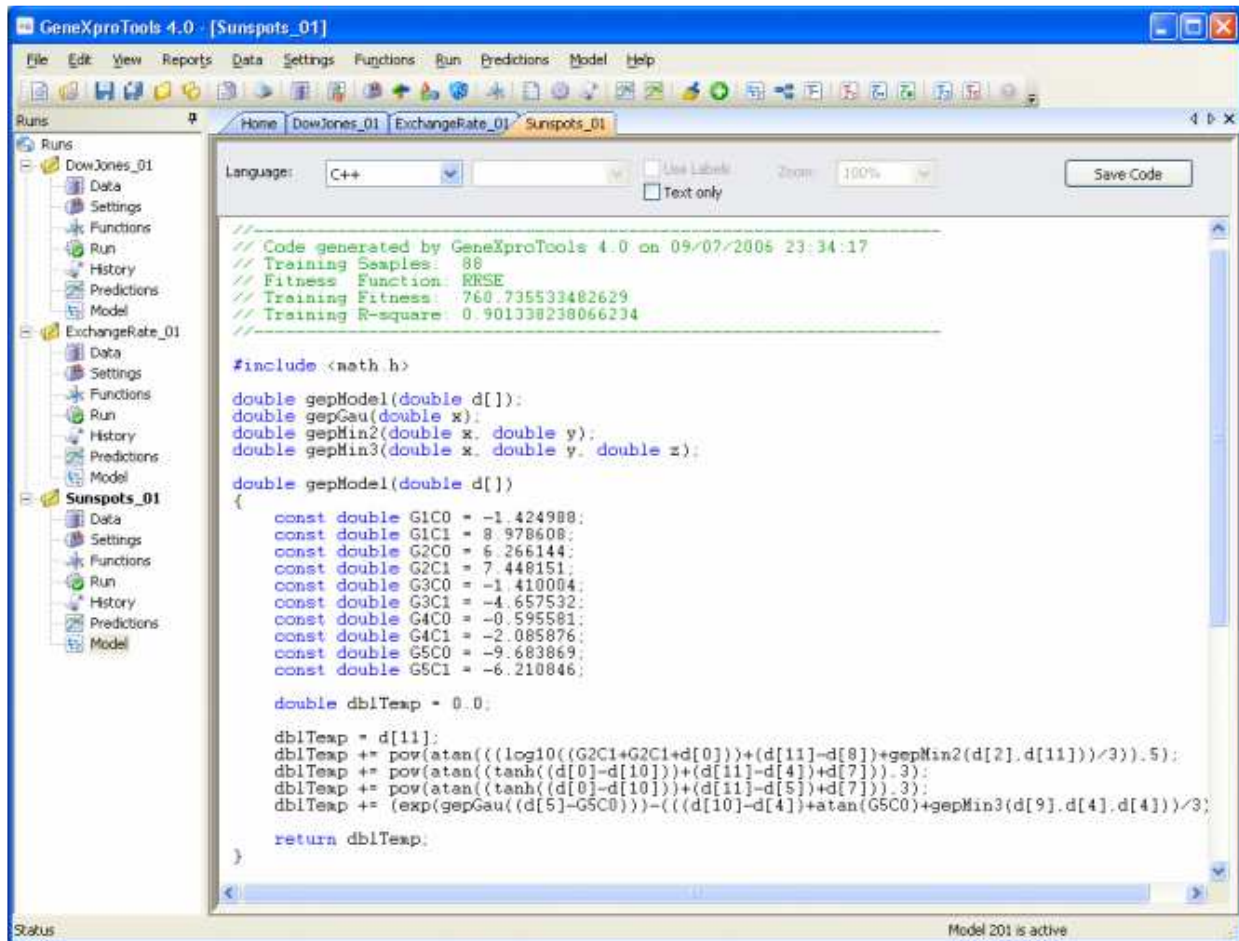
GeneXproTools 4.0 comes equipped with a parse tree generator that automatically converts your models into diagram representations or parse trees.



Indeed, the models evolved by GeneXproTools in its native [Karva language](#) can be automatically parsed into visually appealing expression trees, allowing a quicker and more complete understanding of their mathematical intricacies.

Generating Code Automatically Using GeneXproTools Built-in Grammars

GeneXproTools 4.0 offers a total of 15 [built-in grammars](#) so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the most commonly used programming languages (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL). This code can then be used in other applications.



Generating Code Automatically Using User Defined Grammars

GeneXproTools 4.0 allows the design of [User Defined Grammars](#) so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the programming language of your choice if you happen to prefer a programming language not already covered by the 15 [built-in grammars](#) of GeneXproTools 4.0 (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL).

As an illustration, the C++ grammar of GeneXproTools 4.0 is shown [here](#). Other grammars may be easily created using this or other GeneXproTools built-in grammars as reference.

Making Predictions

The algorithms of GeneXproTools 4.0 used for Time Series Prediction allow not only the evolution of models but also to use these models to make predictions.

GeneXproTools 4.0 allows you to make two kinds of predictions: one for testing past known events and another for making predictions about the future. In both cases, though, predictions are made recursively, by evaluating the forecast at $t + 1$, then using it to forecast $t + 2$, and so on.

You must choose either one of these methods while loading your time series data, as this imposes some constraints on the restructuring of the time series for training. Namely, n testing records (the n last ones) are saved for testing and sometimes a small number of records from the top must be deleted. However, you can also change these parameters later in the GeneXproTools modeling environment.

General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples:

Records in Time Series:

Number of Chromosomes:

Embedding Dimension:

Head Size:

Delay Time:

Number of Genes:

Prediction Mode:

Linking Function:

Testing Predictions:

☒ Enable Complexity Increase

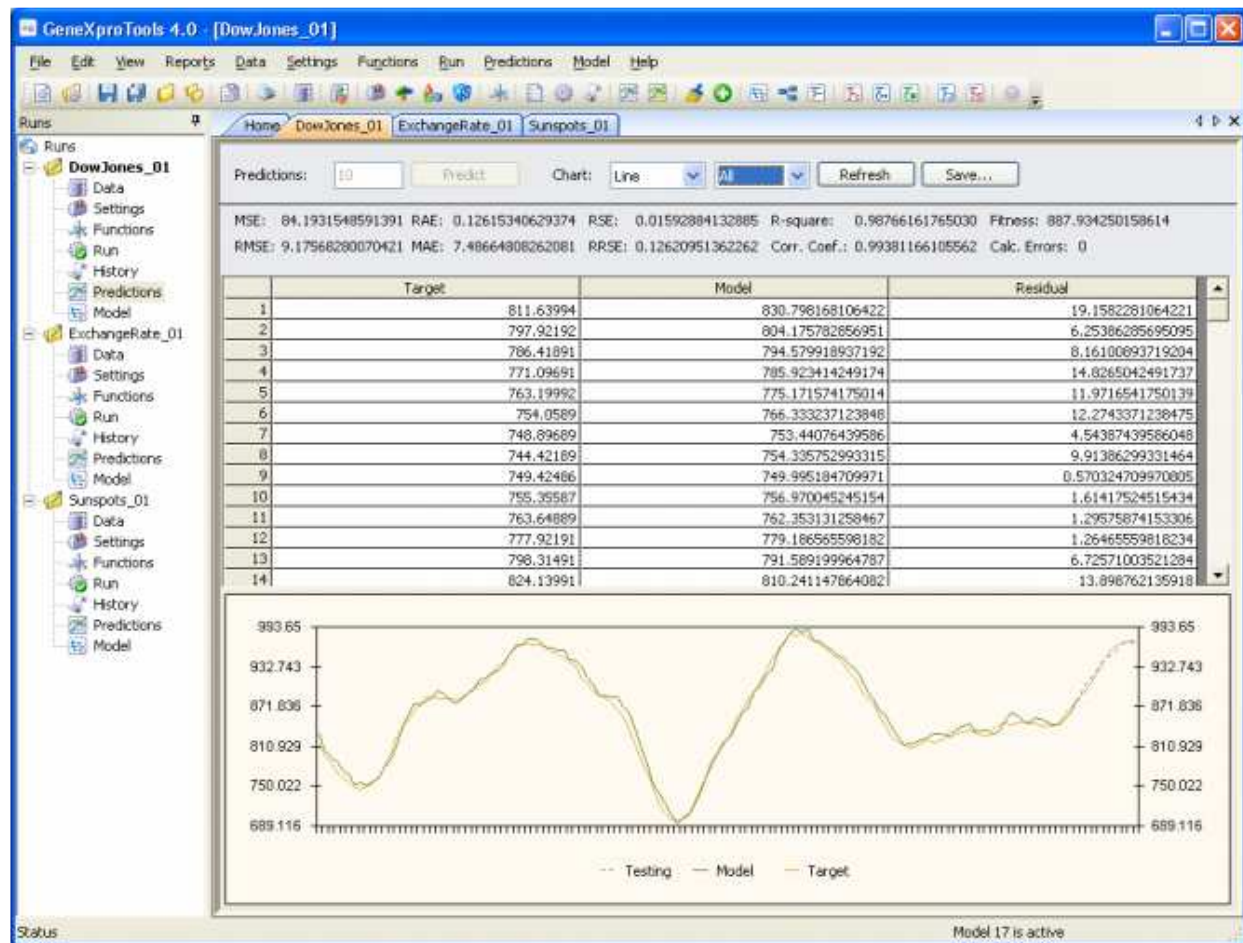
Generations without Change:

Number of Tries:

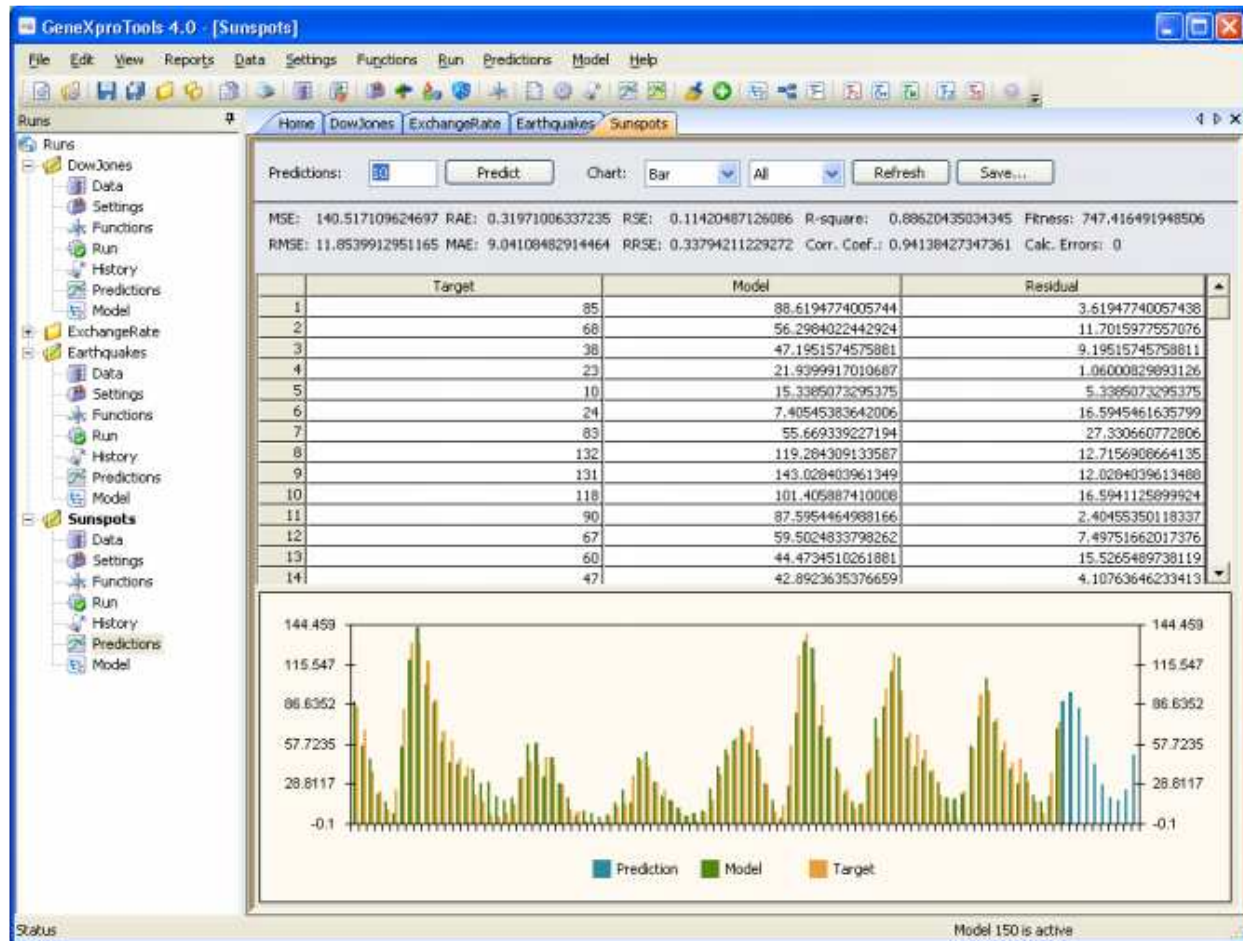
Max. Complexity:

☒ Save Run History

Thus, the first type – Testing Mode – can be used for research or pre-evaluation purposes, as it allows you to test the forecasting capabilities of your model on a set of test observations.



The second type – Prediction Mode – is used obviously to predict unknown behavior, and GeneXproTools 4.0 allows you to venture into the future as far as you see fit, by setting the number of predictions you want to make and then click the Predict button. Like for the Testing Mode, the predictions can be plotted together with the Target and the Model output on the training set or be shown separately in a different plot.



GeneXproTools Environment

Logic Synthesis Tools

Sum-of-products and product-of-sums expressions are alright as solutions to Boolean problems, but most of the times they are far from parsimonious solutions. On the other hand, limiting oneself to the use of just AND-OR-NOT might not be the right approach to finding good solutions to more complex problems. With GeneXproTools you can use an extremely wide spectrum of logical functions (a total of 258 **built-in functions** plus all kinds of **user defined functions**) to design your logic circuits. And, for instance, if you are just interested in NAND-only or NOR-only circuits, you might try and find a solution to your problem using just this particular function instead of applying De Morgan's theorem to your sum-of-products solutions to design your circuits, as most of the times a more parsimonious expression can be found. And if you are interested in less conventional circuits, say a MUX-only circuit or a AND-XOR-NOT circuit, GeneXproTools has no problems working with just these functions to build your circuits. Below are briefly introduced the most important tools you have at your disposal in GeneXproTools:

- [Data Screening Engine](#)

- [Data Visualization Tool](#)
- [Built-in Fitness Functions](#)
- [User Defined Fitness Functions](#)
- [Built-in Logical Functions](#)
- [Built-in Logical Systems](#)
- [Function Selection Tool](#)
- [Dynamic User Defined Functions](#)
- [Static User Defined Functions](#)
- [Learning Algorithms](#)
- [Monitoring the Modeling Process](#)
- [Analyzing Intermediate Models](#)
- [Simplifying a Model](#)
- [Comparing Actual and Predicted Values](#)
- [Essential Statistics](#)
- [Modeling from Seed Models](#)
- [Adding a Neutral Gene](#)
- [Complexity Increase Engine](#)
- [Visualizing Models as Parse Trees](#)
- [Generating Code Automatically Using Built-in Grammars](#)
- [Generating Code Automatically Using User Defined Grammars](#)
- [Scoring Engine](#)

Data Screening Engine

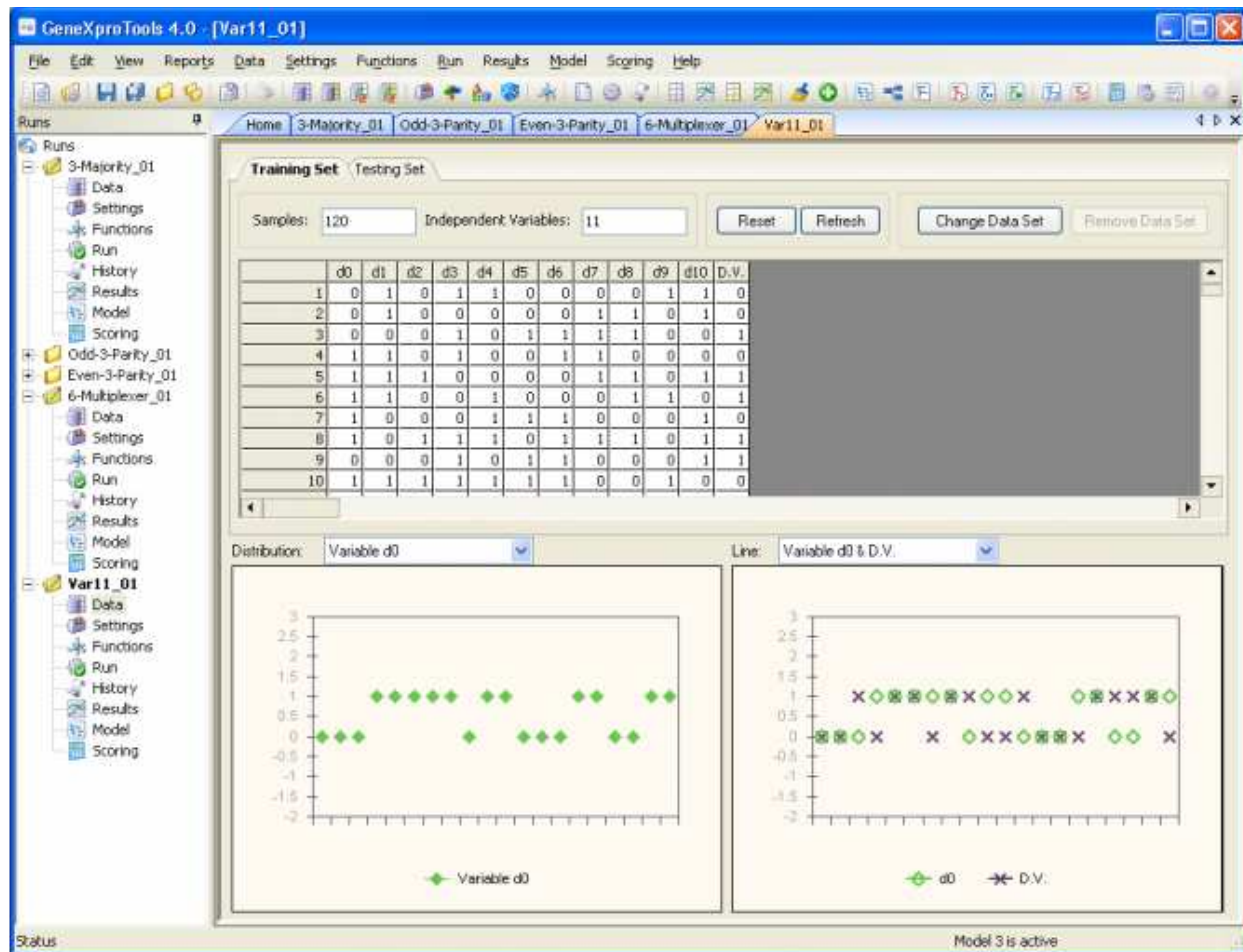
GeneXproTools accepts two different data formats: the standard Samples x Variables format where samples are in rows and variables in columns, with the function output occupying the rightmost position (this format corresponds obviously to the standard truth table used in logic); and Gene Expression Matrixes commonly used in DNA microarray studies where samples are in columns and variables in rows, with the function output occupying the topmost position. Note, however, that GeneXproTools uses the Samples x Variables format and therefore all formats are automatically converted and shown in this format.

The data screening engine of GeneXproTools 4.0 checks the validity of all the data sets used both in the creation and testing of the model. The data screening engine is operative every time you load data sets either for training or testing. Only the values of False and True (case insensitive) or "0" and "1" are valid as input. Note, however, that GeneXproTools uses 0's and 1's internally and, therefore, False and True are respectively converted into "0" and "1" and shown in this format. Missing or invalid values can be detected by the data screening engine so that you can identify these faulty samples and correct them.

Data Visualization Tool

The data visualization tool of GeneXproTools 4.0 enables you to analyze the distribution of all the inputs and to spot simple relationships between the independent variables and the dependent variable.

The visualization of both the distribution and relationships between variables is a valuable modeling tool as it can help you detect simple relationships between your variables that can be used to evolve more efficient models.



Built-in Fitness Functions

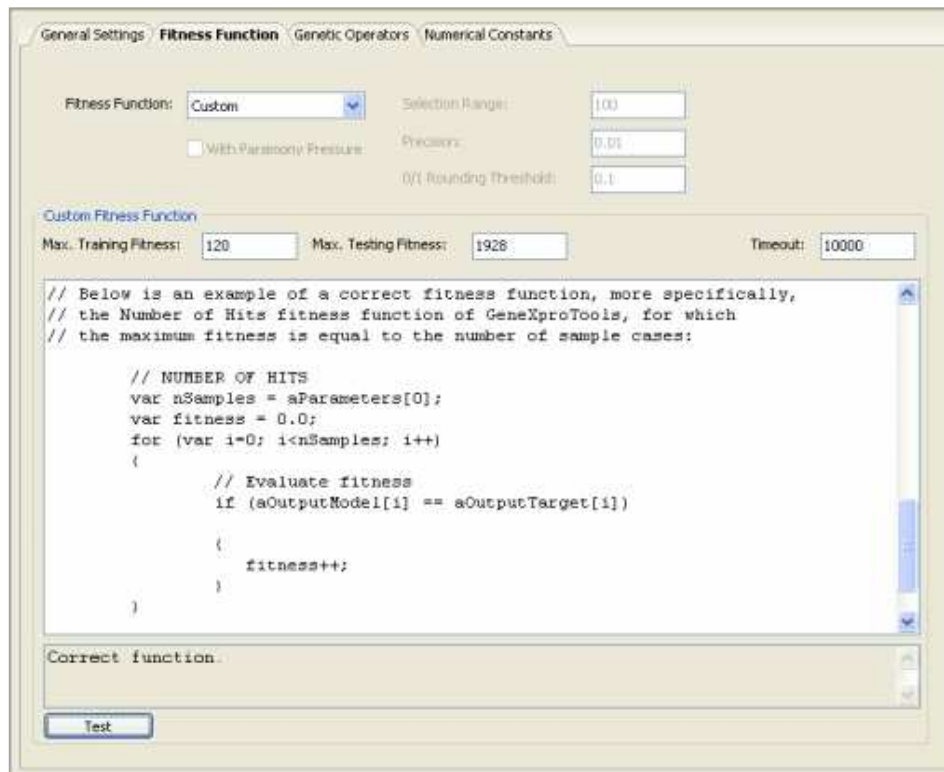
GeneXproTools 4.0 offers 30 [built-in fitness functions](#) based on well-known statistical functions.

Fitness functions are fundamental modeling tools for they determine the nature of the search landscape. And different fitness functions open and explore different paths in the solution space. The 30 built-in fitness functions of GeneXproTools 4.0 are named as follows: [Number of Hits](#), [Hits with Penalty](#), [Accuracy](#), [Squared Accuracy](#), [Sensitivity/Specificity](#), [PPV/NPV](#), [SSPN](#), [MSE](#), [RMSE](#), [MAE](#), [RSE](#), [RRSE](#), [RAE](#), [R-square](#), and [Correlation Coefficient](#). Furthermore, all these fitness functions have their own counterpart with [parsimony pressure](#).

User Defined Fitness Functions

Despite the wide set of GeneXproTools [built-in fitness functions](#), some users sometimes want to experiment with fitness functions of their own. GeneXproTools 4.0 gives the user the possibility of creating custom tailored fitness functions and evolve models with them.

The code for the custom fitness function must be in JavaScript and is written in the Custom Fitness Function window and can be tested before evolving a model with it.



Built-in Logical Functions

GeneXproTools 4.0 offers a total of 258 [built-in logical functions](#), including all the Boolean functions with one and two inputs, 92 functions with three inputs, and 146 functions with four inputs. This wide set of logical functions opens new doors to the creation of new logical circuits that might then be automatically redesigned using certain kinds of gates (All Gates, Not-And-Or Only, Nand Only, Nor Only, Mux System, Reed-Muller System).

Sum-of-products and product-of-sums expressions are alright as solutions to Boolean problems, but most of the times they are far from parsimonious solutions. On the other hand, limiting oneself to the use of just AND-OR-NOT might not be the right approach to finding good solutions to more complex problems or you might just be interested in a different kind of circuit. The broad palette of GeneXproTools functions offers you unlimited possibilities in terms of design. For instance, if you are just interested in NAND-only or NOR-only circuits, you might try and find a solution to your problem using just this particular function instead of applying De Morgan's theorem to your sum-of-products solutions to design your circuits, as most of the times a more parsimonious expression can be found if the circuit is built from scratch using these gates. And if you are interested in less conventional circuits, say a MUX-only circuit or a AND-XOR-NOT circuit, GeneXproTools has no problems working with just these functions to build your circuits.

Built-in Logical Systems

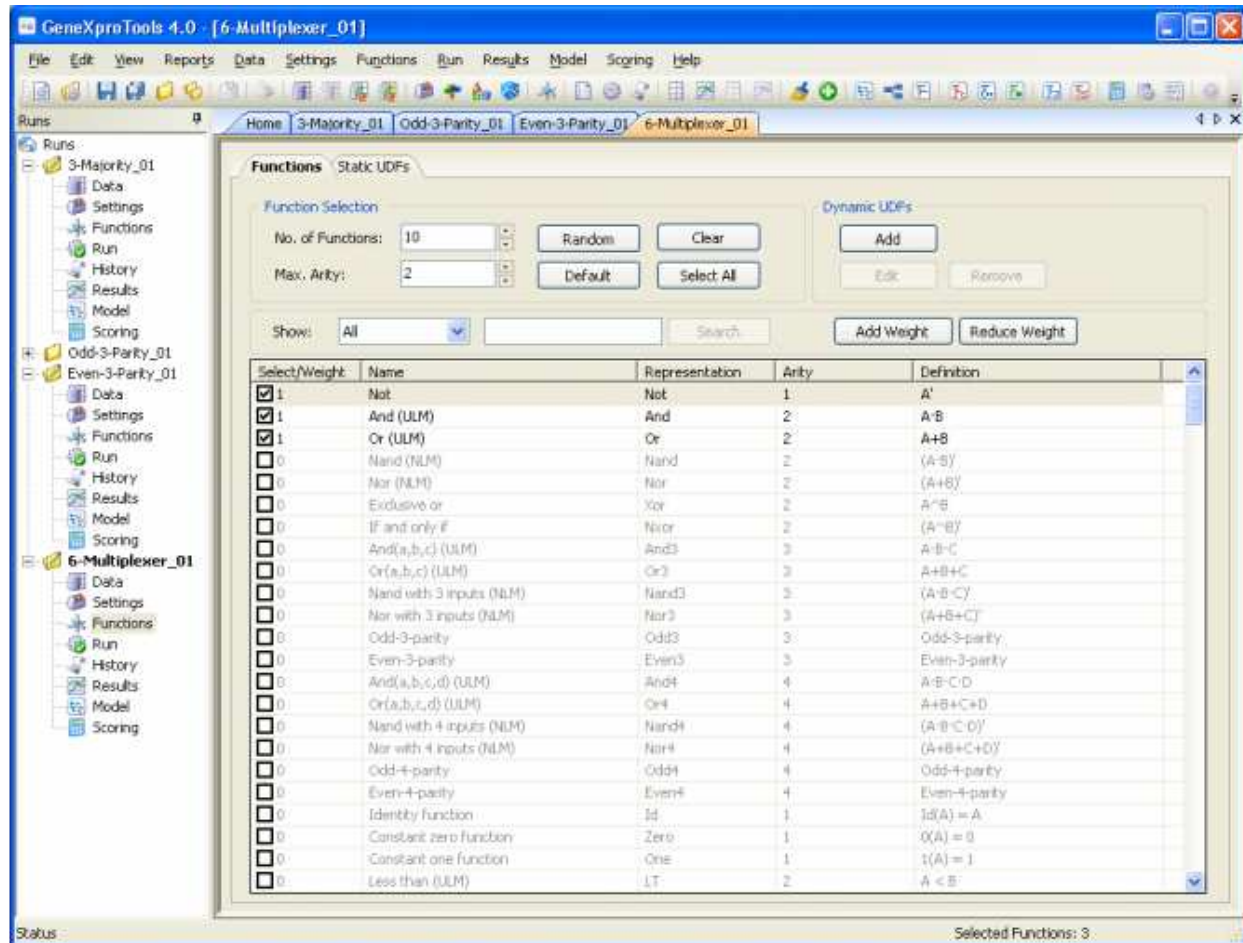
GeneXproTools 4.0 allows you to design logic circuits using a total of 258 [built-in logical functions](#), which are the logical functions at the disposal of GeneXproTools learning algorithms and also part of the basic Boolean grammar of all [built-in grammars](#) (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL). But it also allows you to redesign immediately the generated circuits using six different kinds of [logical systems](#) (All Gates, Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System).

Function Selection Tool

For some problems, the basic Boolean operators AND-OR-NOT are more than enough to create elegant and efficient solutions. But for other problems more complex functions are necessary to create good models and when inside knowledge of the problem at hand is not producing the expected results, it

might be interesting to experiment with different combinations of functions. The Function Selection Tool of GeneXproTools 4.0 helps you to experiment with different function sets very quickly either by pressing the Random button in the Functions Panel or by selecting certain sub-sets of functions.

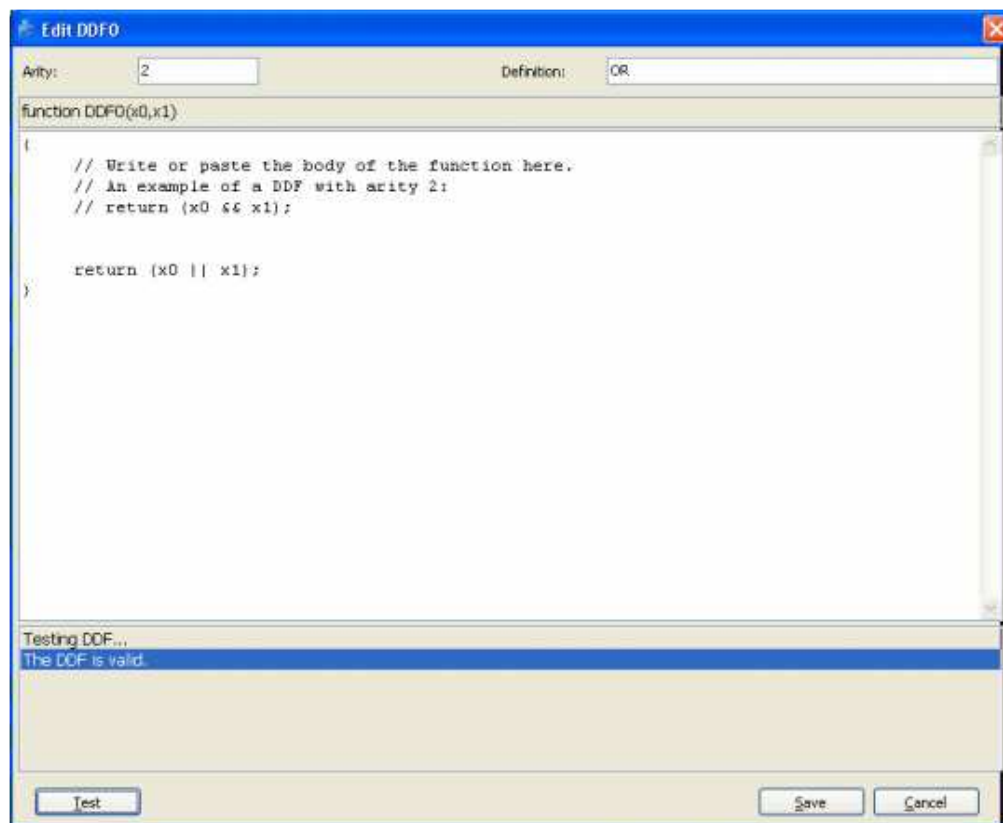
In addition, although you can always handcraft your function set so that it is well-balanced relatively to the number of independent variables in your data, in GeneXproTools 4.0 this is no longer essential as GeneXproTools balances automatically all function sets. This is particularly important for high-dimensional datasets, say more than 20 variables.



Dynamic User Defined Functions (DDFs)

Despite the wide set of GeneXproTools [built-in logical functions](#), some users sometimes want to model with different functions. GeneXproTools 4.0 gives the user the possibility of creating custom tailored functions and evolve models with them. So, if the 92 functions with three inputs plus the 146 functions with four inputs of GeneXproTools are not enough for you, you can implement all the remaining functions of three and four inputs as DDFs and work with them as building blocks. A note of caution though: the use of DDFs slows considerably the evolutionary process and therefore should be used with moderation.

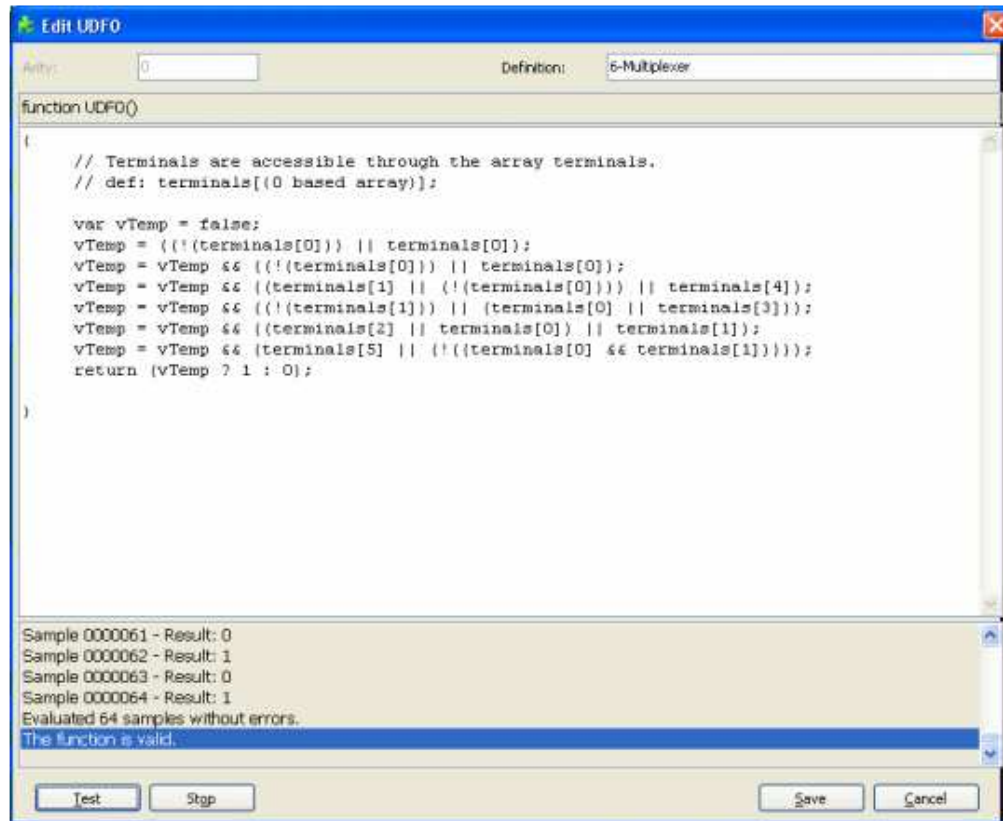
The code for the DDFs must be in JavaScript and is written in the Edit DDF window (in the Functions Panel, select the Functions Tab and then click Add in the Dynamic UDFs tool box).



Static User Defined Functions (UDFs)

Sometimes it is fundamental to use complex building blocks to build even more complex ones, and GeneXproTools 4.0 allows you to design your own basic building blocks (UDFs). These building blocks might represent any kind of logical expression (not just functions of 1-4 arguments) and are therefore ideal to design complex models composed of several simpler components.

The code for the UDFs must be in JavaScript and is written in the Edit UDF window (in the Functions Panel, select the Static UDFs Tab and then click Add in the Static UDFs tool box).



Learning Algorithms

GeneXproTools 4.0 is equipped with two different [learning algorithms](#) for evolving logic circuits.

The first is the fastest and simplest of the two. And because it is also the most efficient, this algorithm is the default in GeneXproTools 4.0. The second algorithm is very similar to the first, with the difference that it gives the user the possibility of sprinkling "0" and "1" to help in the design.

In order to evolve models using the second algorithm, select Numerical Constants in the Settings Panel and then check the box that activates this algorithm. You will notice that additional parameters become available, including a small set of genetic operators especially developed for handling numerical constants (if you are not familiar with these operators, please use the default values for they work very well in all cases). To use the first algorithm, just uncheck the Use Random Numerical Constants box.

Monitoring the Modeling Process

GeneXproTools 4.0 allows you to monitor the modeling process by showing the essential parameters of a run during the discovery process, including the [fitness](#), [classification accuracy](#), and [confusion matrix](#) (true positives, true negatives, false positives, and false negatives) of the best-of-generation model.

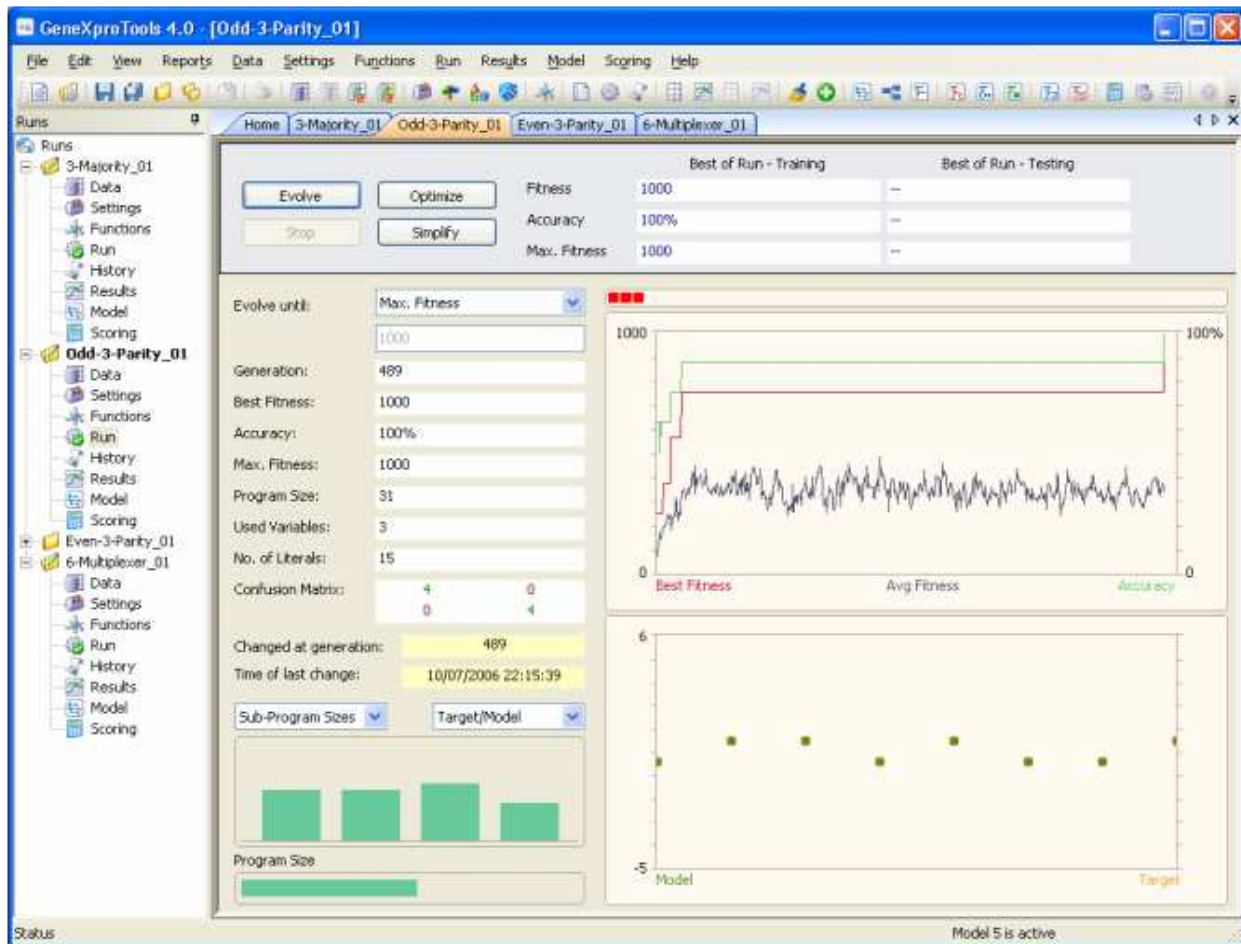
Furthermore, GeneXproTools allows you to visualize the modeling process by plotting the essential features of a run during the discovery process. In the [Run Panel](#), a total of eight different charts are shown:

1. The Variables Usage map.
This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square you can access the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic.
2. The Evolutionary Dynamics chart.
This chart shows the average fitness of the population plus the fitness and accuracy of the best-of-generation model.
3. The Average/Best Size chart.
This chart shows the average program size of the population versus the program size of the best-of-generation model. This chart is especially useful during simplification and can be activated any time during evolution by selecting Avg/Best Size in the rightmost combo box in the bottom.
4. The Target/Model chart.
This chart plots the first 50 data points of the training set and shows how well the evolving models are fitting the target. The Target/Model chart can be activated any time during evolution by selecting Target/Model in the rightmost combo box in the bottom.
5. The Sub-Program Sizes chart.
This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be activated any time during evolution by selecting Sub-Program Sizes in the leftmost combo box in the bottom.
6. The All Sizes chart.
This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be activated any time during evolution by selecting All Sizes in the leftmost combo box in the bottom.
7. The All Fitnesses chart.
This chart shows the fitnesses of all the models in the population. Not only after a run but also

during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be activated any time during evolution by selecting All Fitnesses in the leftmost combo box in the bottom.

8. The Program Size chart.

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model.



The evolutionary process can be stopped whenever you are satisfied with the results by pressing the Stop button or you can use one of the stop criteria of GeneXproTools for picking up your model exactly as you want it.

When the evolutionary process stops, the best-of-run model is ready either for analysis or scoring. And if you are still not happy with the results, you can still try to fine-tune the evolved model by pressing the Optimize button and repeat this process until you are completely satisfied with your model. In addition, you might also wish to try and simplify your model and for that you just have to press the Simplify button.

Analyzing Intermediate Models

GeneXproTools 4.0 gives you the possibility of gaining some insights into the modeling process by analyzing all the best-of-generation intermediate models.

All the best-of-generation models discovered during a run are saved and you can pick them up for analysis at the [History Panel](#) by checking the model you are interested in. Each of these intermediate models can then be analyzed exactly as you do for the best-of-run model, that is, check their vital

statistics, automatically generate code with them, visualize their parse trees, use them as seed to create more parsimonious models and so forth.

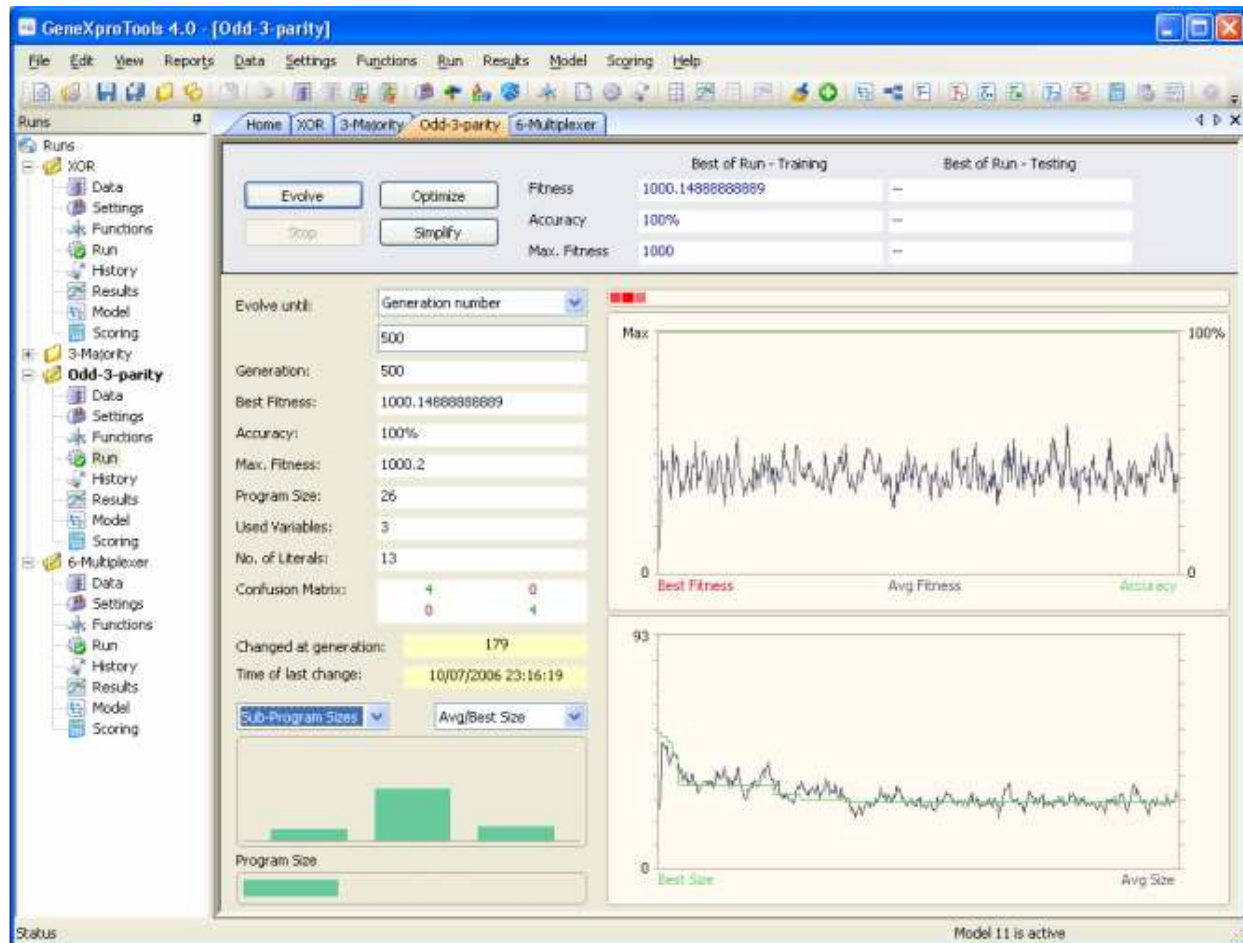
The screenshot displays the GeneXproTools 4.0 software interface. The title bar indicates the current run is '6-Multiplexer_01'. The menu bar includes File, Edit, View, Reports, Data, Settings, Functions, Run, Results, Model, Scoring, and Help. The toolbar contains various icons for file operations and analysis. The 'Runs' panel on the left shows a tree view of runs, with '6-Multiplexer_01' selected. The main window features buttons for 'Test All', 'Test Current', and 'Refresh All'. Below these buttons, a table lists the results of the simplification process for various models.

Model	Generation	Program Size	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
☐ Model 32	150	46	738.404690625	--	85.94%	--
☐ Model 33	165	45	738.40528125	--	85.94%	--
☐ Model 34	186	44	738.405871875	--	85.94%	--
☐ Model 35	220	40	765.7566875	--	87.50%	--
☐ Model 36	271	39	765.7573	--	87.50%	--
☐ Model 37	308	37	765.758525	--	87.50%	--
☐ Model 38	315	29	765.763425	--	87.50%	--
☐ Model 39	317	26	765.7652625	--	87.50%	--
☐ Model 40	577	36	820.45621875	--	90.63%	--
☐ Model 41	591	35	820.456875	--	90.63%	--
☐ Model 42	873	35	875.154	--	93.75%	--
☐ Model 43	974	34	875.1547	--	93.75%	--
☐ Model 44	1793	31	875.1568	--	93.75%	--
☐ Model 45	1975	29	875.1582	--	93.75%	--
☐ Model 46	2031	27	875.1596	--	93.75%	--
☐ Model 47	2094	25	875.161	--	93.75%	--
☐ Model 48	2971	35	937.665	--	96.88%	--
☐ Model 49	2977	48	968.910425	--	98.44%	--
☐ Model 50	3000	37	1000.1744	--	100%	--
☐ Model 51	3002	36	1000.1752	--	100%	--
☐ Model 52	3008	34	1000.1768	--	100%	--
☐ Model 53	3076	33	1000.1776	--	100%	--
☐ Model 54	3164	31	1000.1792	--	100%	--
☐ Model 55	3254	29	1000.1808	--	100%	--
☐ Model 56	3588	28	1000.1816	--	100%	--
☒ Model 57	4683	26	1000.1816	--	100%	--

The status bar at the bottom indicates 'Model 57 is active'.

Simplifying a Model

The whole business of circuit design revolves around finding the most parsimonious circuits and GeneXproTools deals with this task excellently. No matter in what form your circuits are (sum-of-products, product-of-sums, NAND-only, NOR-only, AND-XOR-NOT-only, MUX-only, and so on), you can feed them to GeneXproTools and let it have a go at simplifying them. The results of its efforts can be clearly observed in the Run Panel where both the size and number of literals are plotted during the simplification process.



GeneXproTools 4.0 allows you to simplify an existing model (which could have been either generated by GeneXproTools or by another modeling tool) either by pressing the Simplify button on the [Run Panel](#) or by choosing a fitness function with parsimony pressure and then press Optimize or Simplify on the Run Panel.

For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.

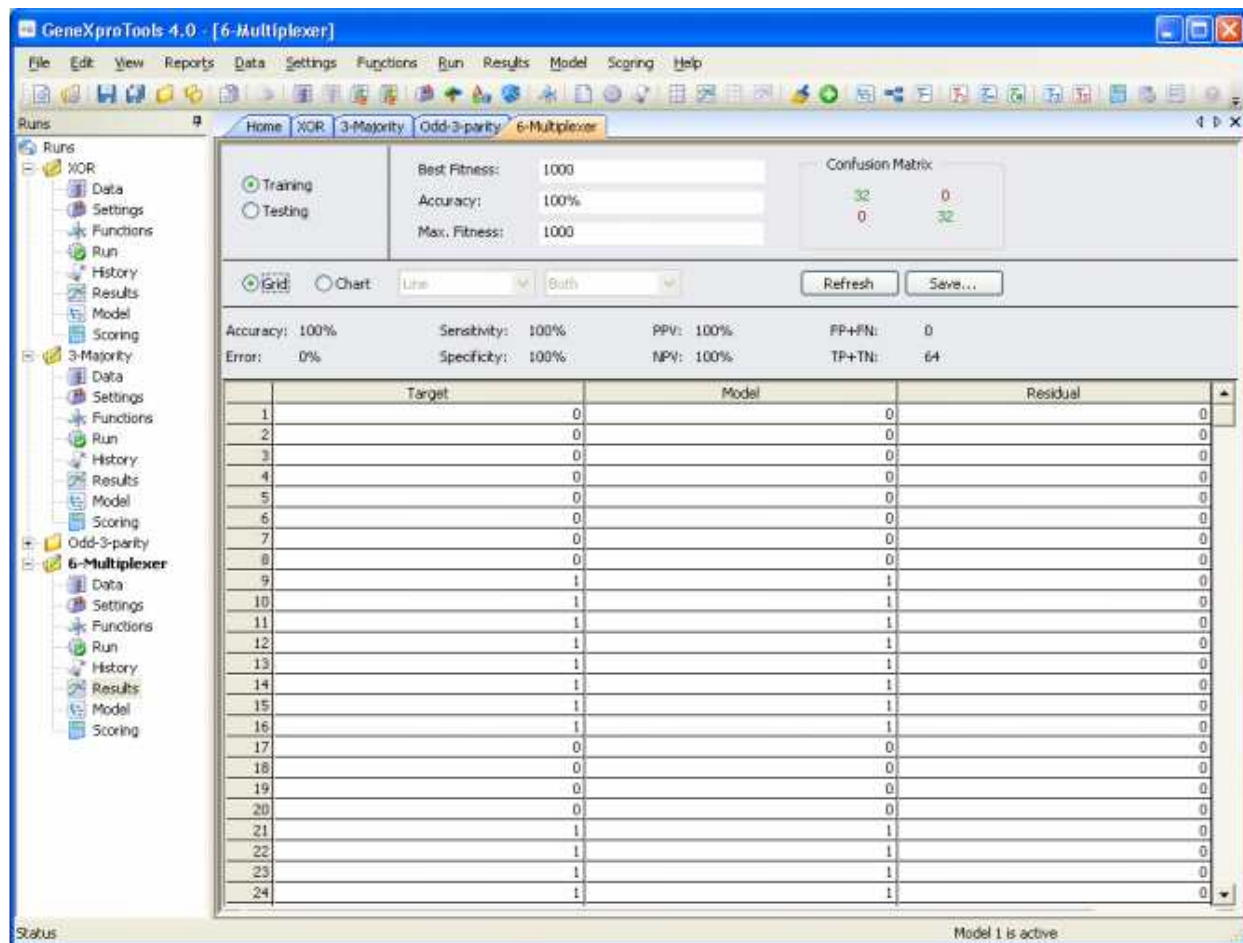
Then, in the [Run Panel](#), by clicking the Simplify button, an evolutionary process starts in which all the subsequent models will be descendants of the model you want to simplify. Keep in mind, however, that the simplification algorithm used by GeneXproTools is an evolutionary one and, despite parsimony pressure being applied, models continue to be selected preferentially by fitness and sometimes their complexity might even increase temporarily if the gain in fitness outweighs the loss in simplicity. But, obviously, if your model has already maximum fitness, it can only improve by becoming more and more parsimonious.

For models created in the GeneXproTools 4.0 environment, you just have to select the model you want to simplify (either the best-of-run or an intermediate model picked up in the [History Panel](#)) and then click the Simplify button and let the algorithm create better descendants not only in terms of fitness but also in terms of size.

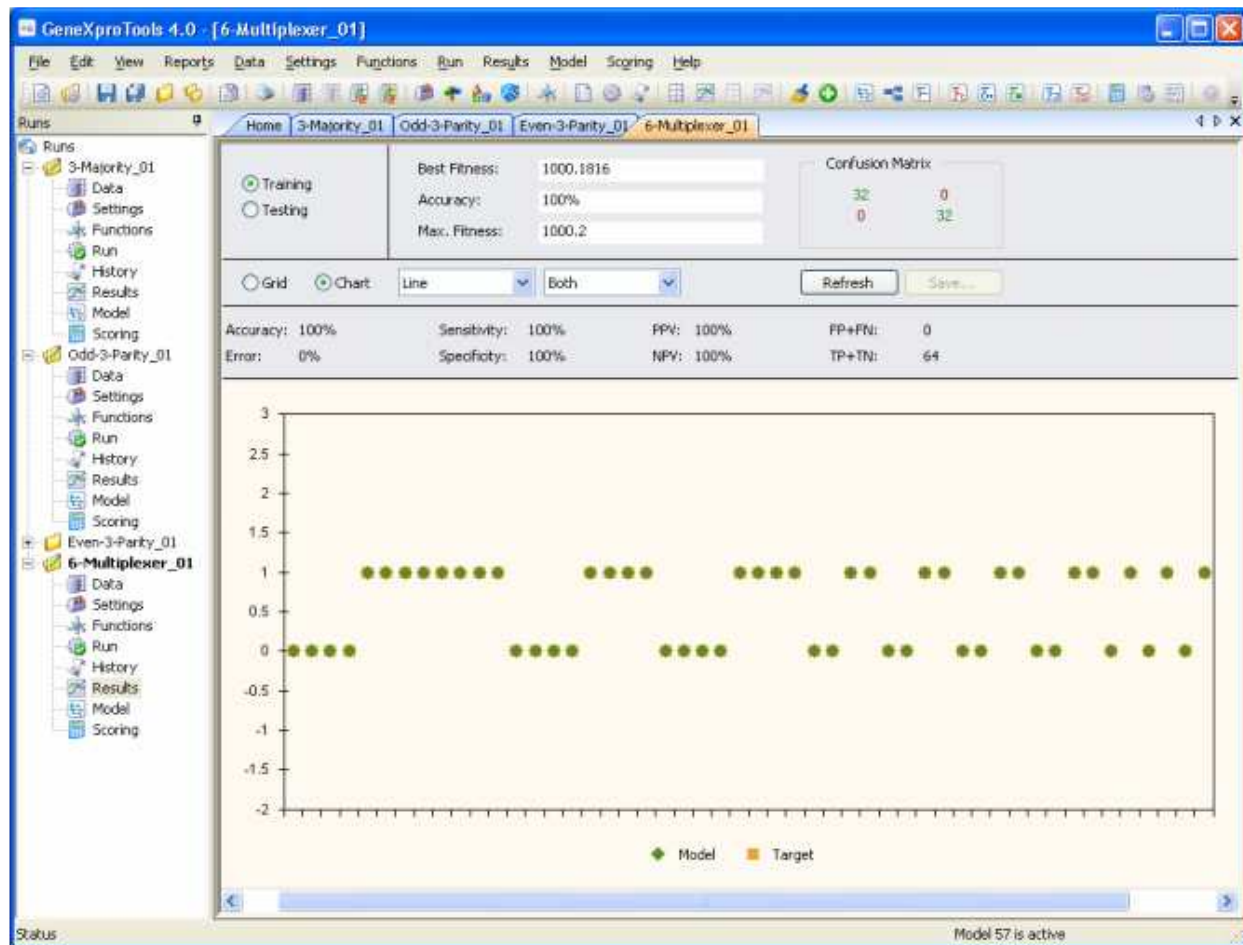
Comparing Actual and Predicted Values

GeneXproTools 4.0 offers two different ways of analyzing and comparing the output of your model with the actual or target values both for the training and testing sets.

In the first, the target or actual values are listed in a table side by side with the predicted or model values.



In the second, the target and predicted values are plotted in a Target/Model chart for easy visualization. By moving the scroll bar in the bottom, you can see the results for the entire dataset.

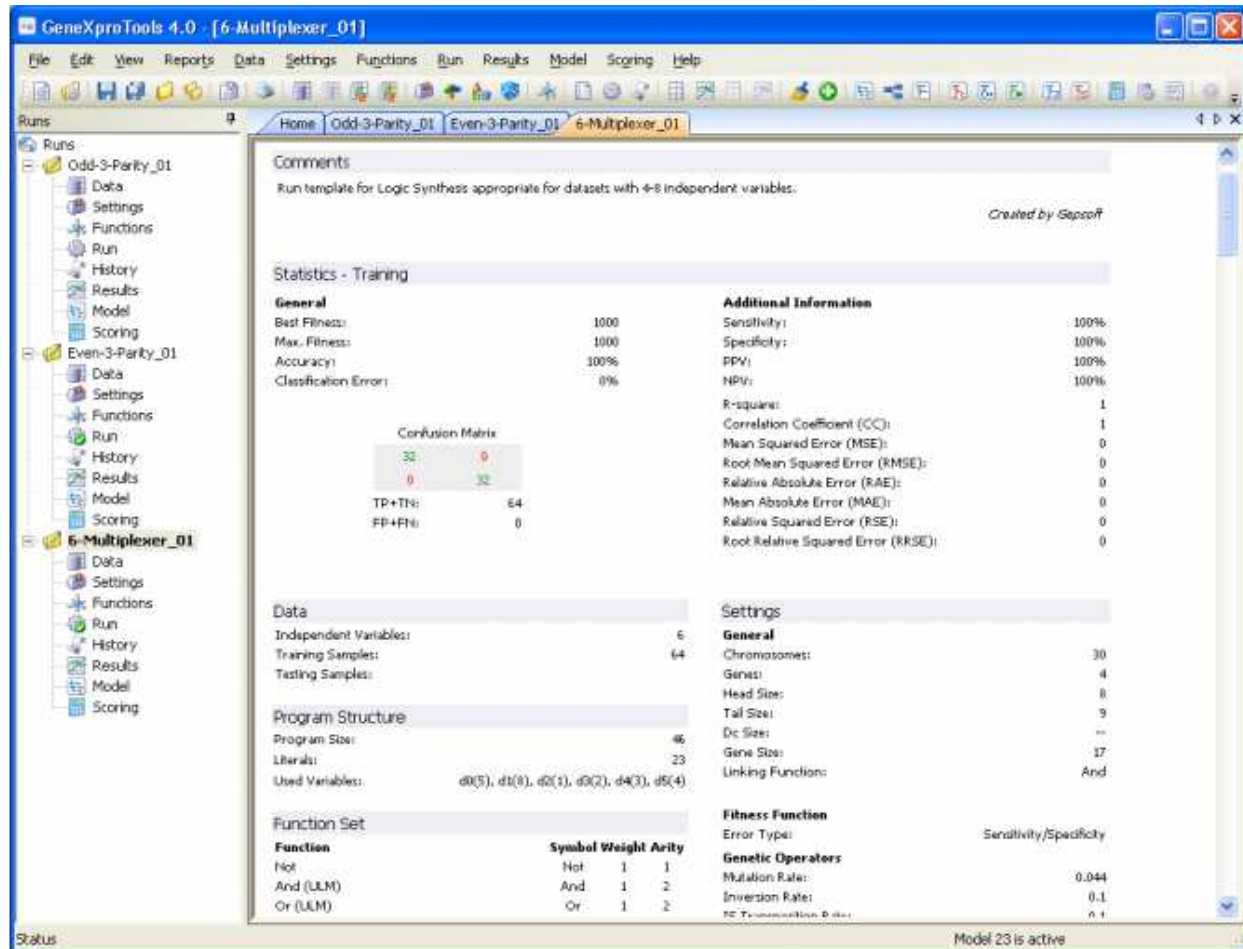


Essential Statistics

GeneXproTools 4.0 allows a quick and easy assessment of a wide set of [statistical functions](#). Some of them are immediately computed and shown every time you check the [Results Panel](#) ([classification error](#), [classification accuracy](#), [confusion matrix](#) (true positives, true negatives, false positives, and false negatives), [sensitivity](#), [specificity](#), [positive predictive value](#), and [negative predictive value](#)).



Others, though, are only shown in the [Report Panel](#) (mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, R-square, and correlation coefficient) after their evaluation in the [Results Panel](#).

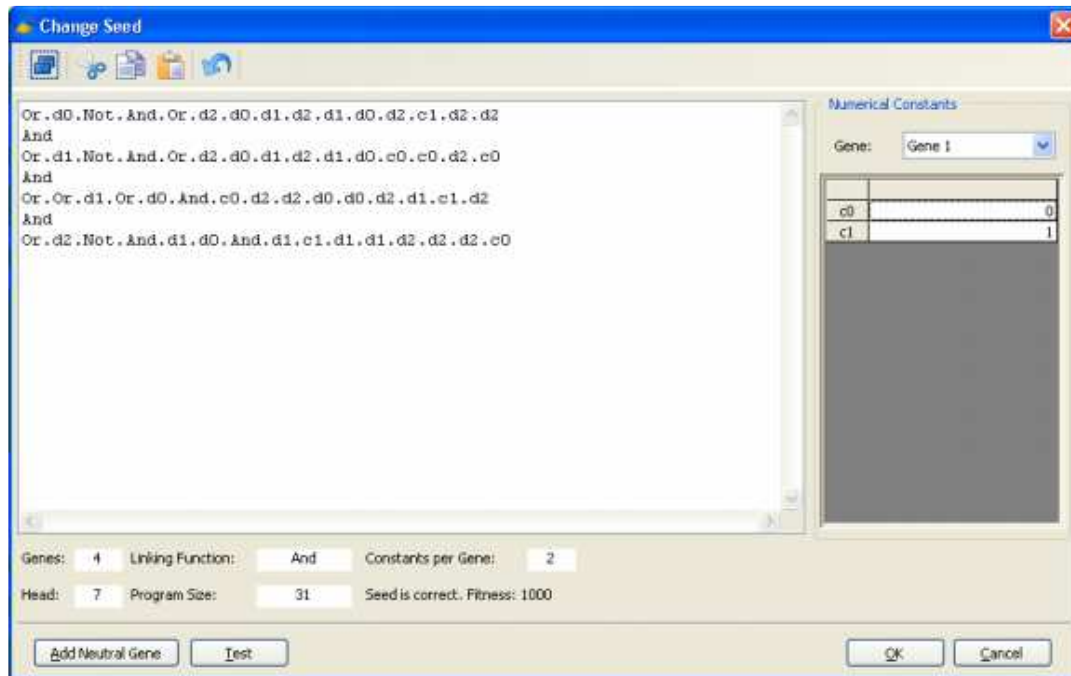


Modeling from Seed

The whole business of circuit design revolves around finding the most parsimonious circuits and GeneXproTools deals with this task excellently. No matter in what form your circuits are (sum-of-products, product-of-sums, NAND-only, NOR-only, AND-XOR-NOT-only, MUX-only, and so on), you can use them as seed and let GeneXproTools have a go at simplifying them. The results of its efforts can be clearly observed in the Run Panel where both the size and number of literals are plotted during the simplification process.

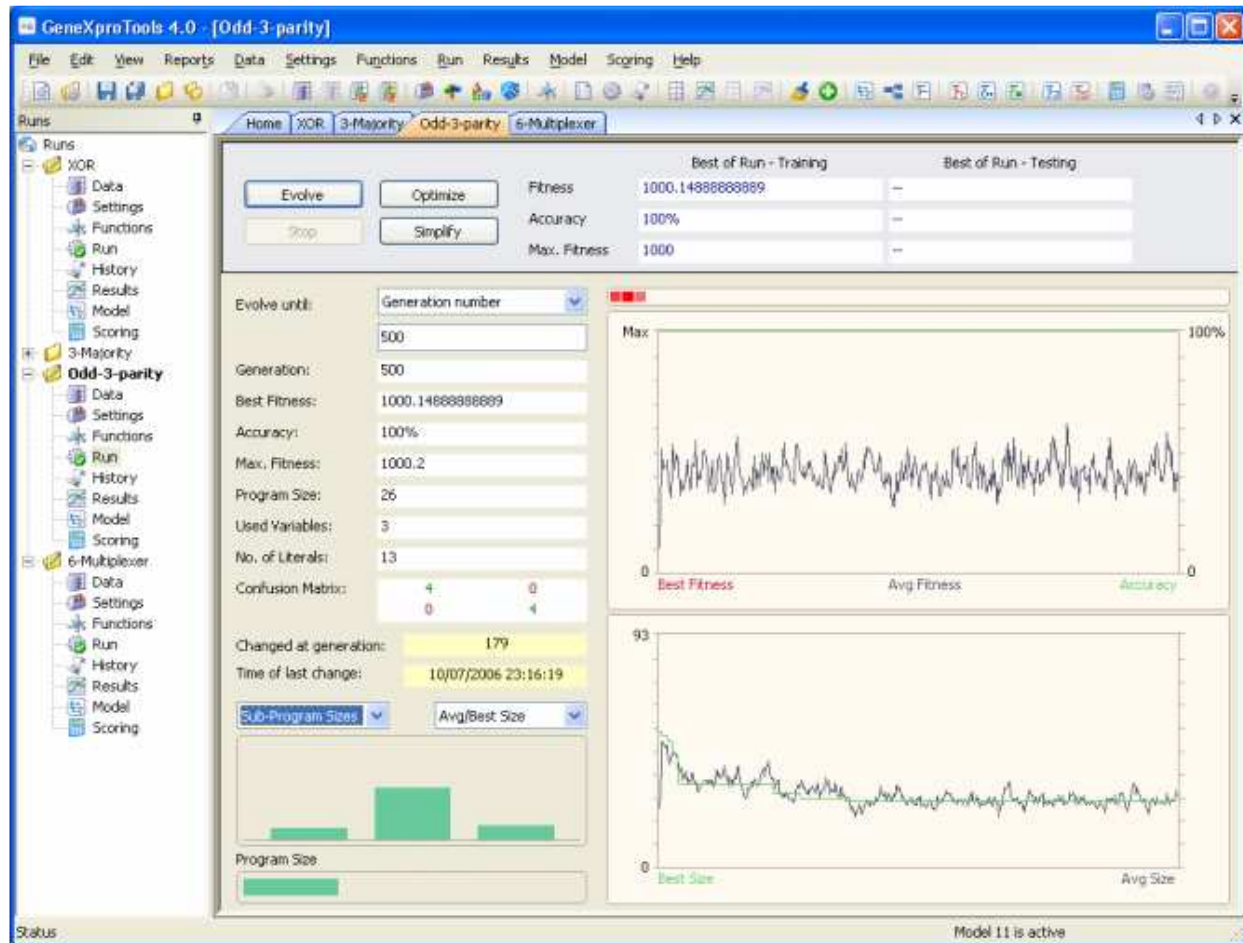
GeneXproTools 4.0 allows the use of an existing model (which could have been either generated by GeneXproTools or by another modeling tool) as the starting point of an evolutionary process so that more rigorous or more parsimonious models could be created.

For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model or seed is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.



Then, in the [Run Panel](#), by clicking the Optimize button, an evolutionary process starts in which all the subsequent models will be descendants of the seed you introduced. A note of caution though: if your seed has a very small fitness, you risk losing it early in the run as better models could be randomly created by GeneXproTools and your seed would, most probably, not be selected for breeding new models. If your seed has zero fitness, though, you will receive a warning so that you could modify your seed until it becomes a viable seed capable of breeding new models.

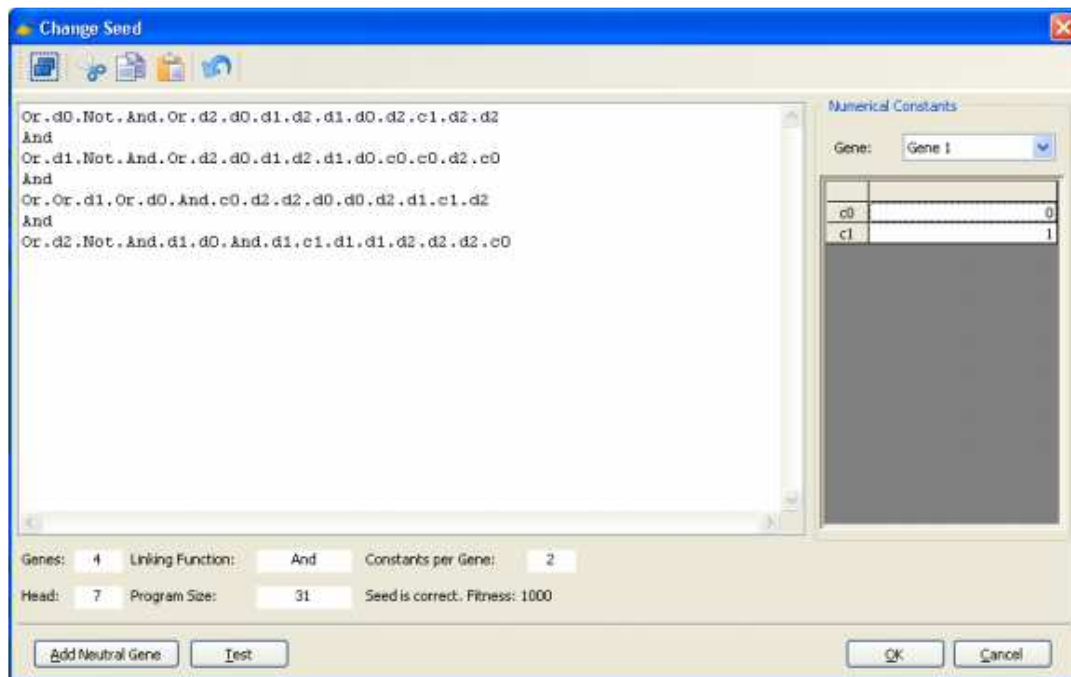
For models created in the GeneXproTools 4.0 environment, the seed is fed to the algorithm every time you click the Optimize or Simplify buttons during modeling, that is, the seed (in this case, the best model of the previous run) is introduced automatically by GeneXproTools.



Adding a Neutral Gene

The addition of a neutral gene to a seed might seem, at first sight, the wrong thing to do as most of the times we are interested in creating parsimonious models. But one should look at this as modeling in progress as, for really complex circuits, it is not uncommon to design them by progressively incorporating simpler elements.

Thus, being able of introducing extra terms in your seed is a powerful modeling tool and GeneXproTools 4.0 allows you to do that by selecting Add Neutral Gene in the Edit Menu or through the Change Seed window. For instance, by choosing OR as linking function, you could use a different gene for each minterm of your circuit, and then try and evolve a more parsimonious solution from here.



By pressing the Add Neutral Gene button in the Change Seed window, you will see a neutral gene being added to your model. By doing this, you are giving the learning algorithm more room to play and, hopefully, a better and more complex program will evolve.

Neutral genes can also be introduced automatically by GeneXproTools 4.0 using the Complexity Increase Engine.

Complexity Increase Engine

GeneXproTools 4.0 also allows you to introduce neutral genes automatically by activating the Complexity Increase Engine in the Settings Panel -> General Settings Tab.

General Settings Fitness Function Genetic Operators Numerical Constants

Training Samples: 120

Testing Samples: 2048

Number of Chromosomes: 30

Head Size: 10

Number of Genes: 7

Linking Function: And

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 7

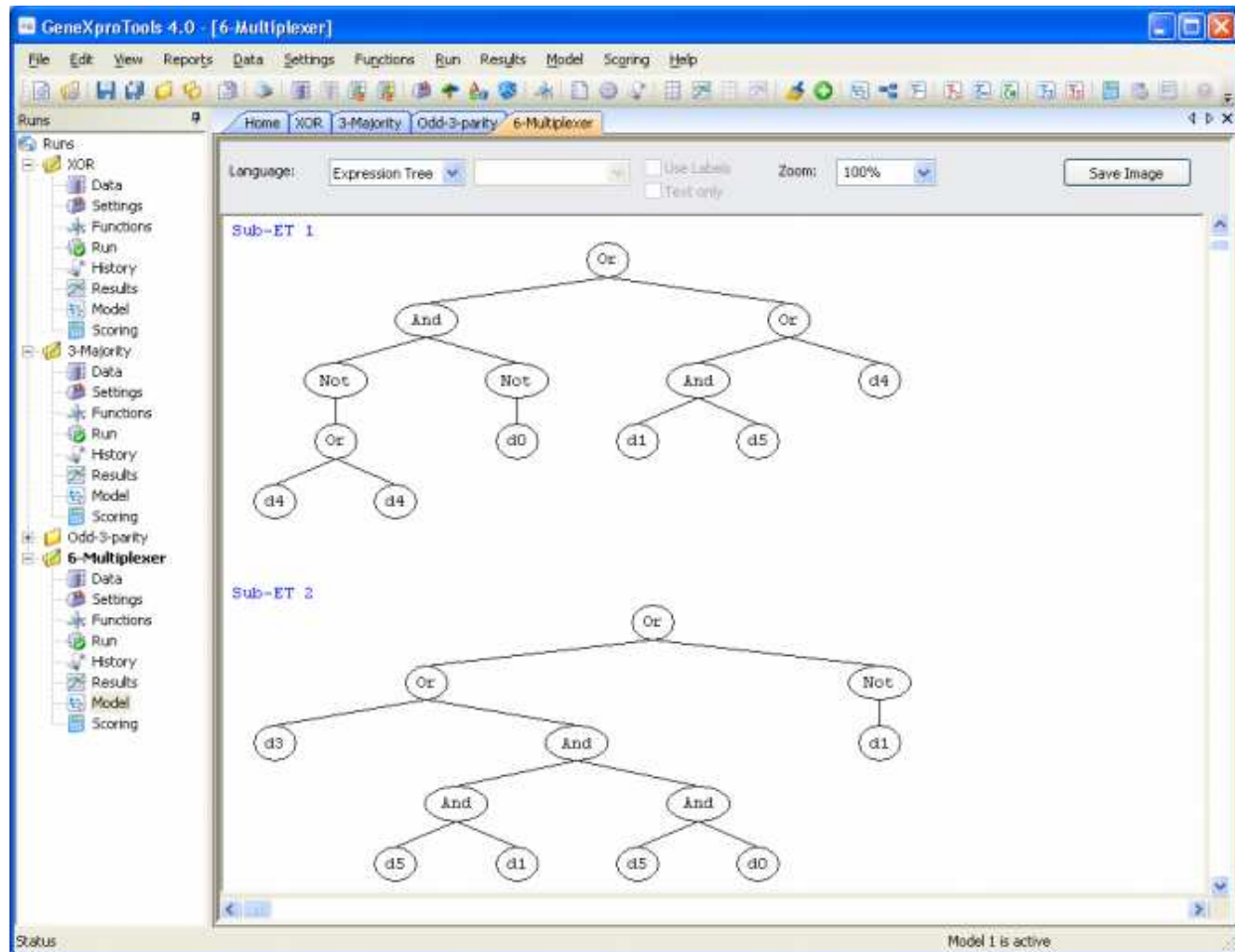
☒ Save Run History

Whenever you are using the Complexity Increase engine of GeneXproTools 4.0, you must fill the Generations Without Change box to set the period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction or a neutral gene (an extra term) is automatically added to your model; the Number of Tries box corresponds to the number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models; in the Max. Complexity box you write the maximum number of terms (genes) you'll allow in your model and no other terms will be introduced beyond this threshold.

The Complexity Increase Engine of GeneXproTools 4.0 might be a very powerful modeling tool, but you must be careful not to create excessively complex models as a greater complexity does not necessarily imply a greater efficiency.

Visualizing Models as Parse Trees

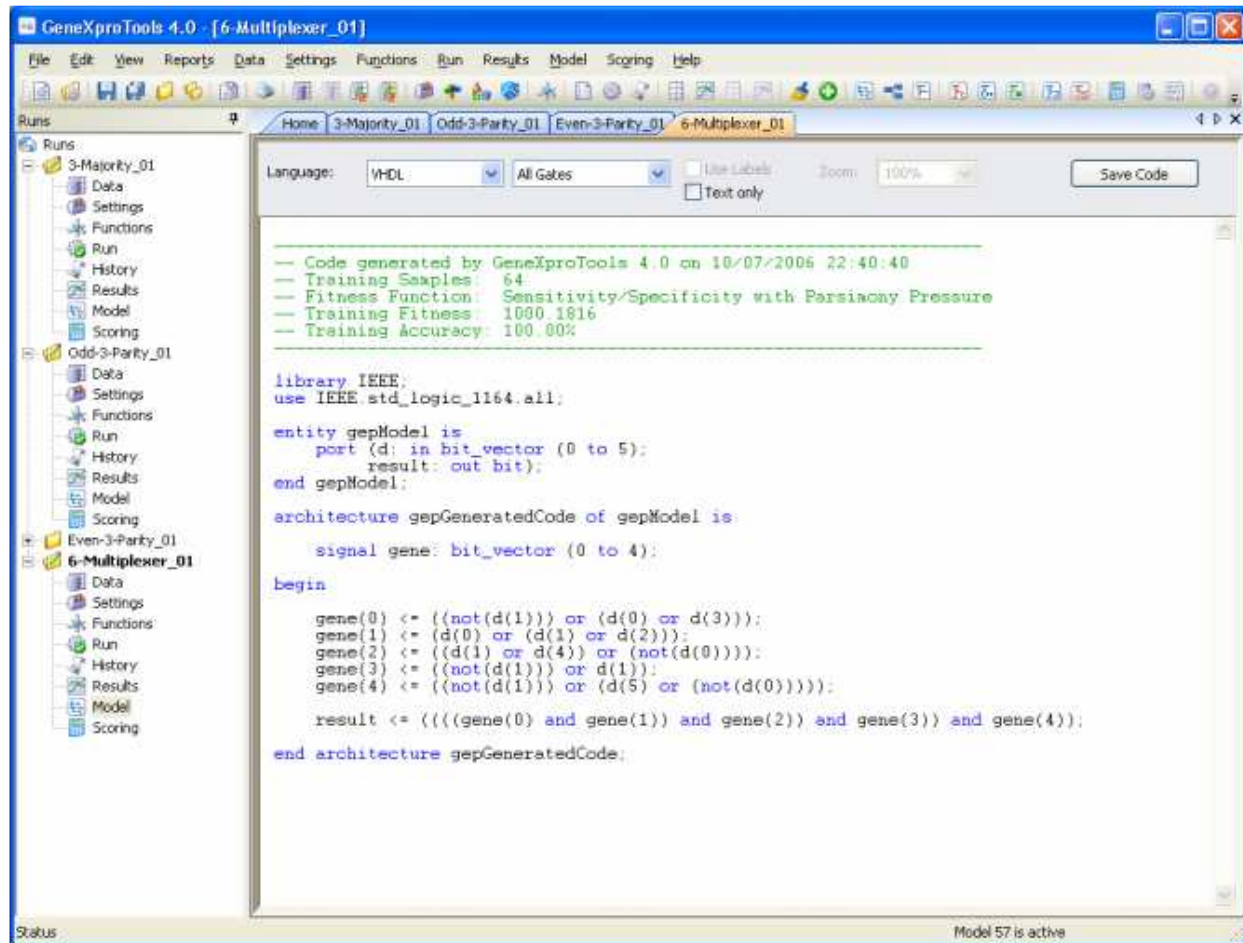
GeneXproTools 4.0 comes equipped with a parse tree generator that automatically converts your models into diagram representations or parse trees.



Indeed, all the models evolved by GeneXproTools in its native [Karva language](#) can be automatically parsed into visually appealing expression trees, allowing a quicker and more complete understanding of their structural intricacies.

Generating Code Automatically Using GeneXproTools Built-in Grammars

GeneXproTools 4.0 offers a total of 16 [built-in grammars](#) so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the most commonly used programming languages (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL). Furthermore, for all languages, the generated code can be automatically transcribed to six different [logical systems](#) (All Gates, Not-And-Or Only, Nand Only, Nor Only, Mux System, Reed-Muller System) using six different Boolean grammars. This code can then be used in other applications.



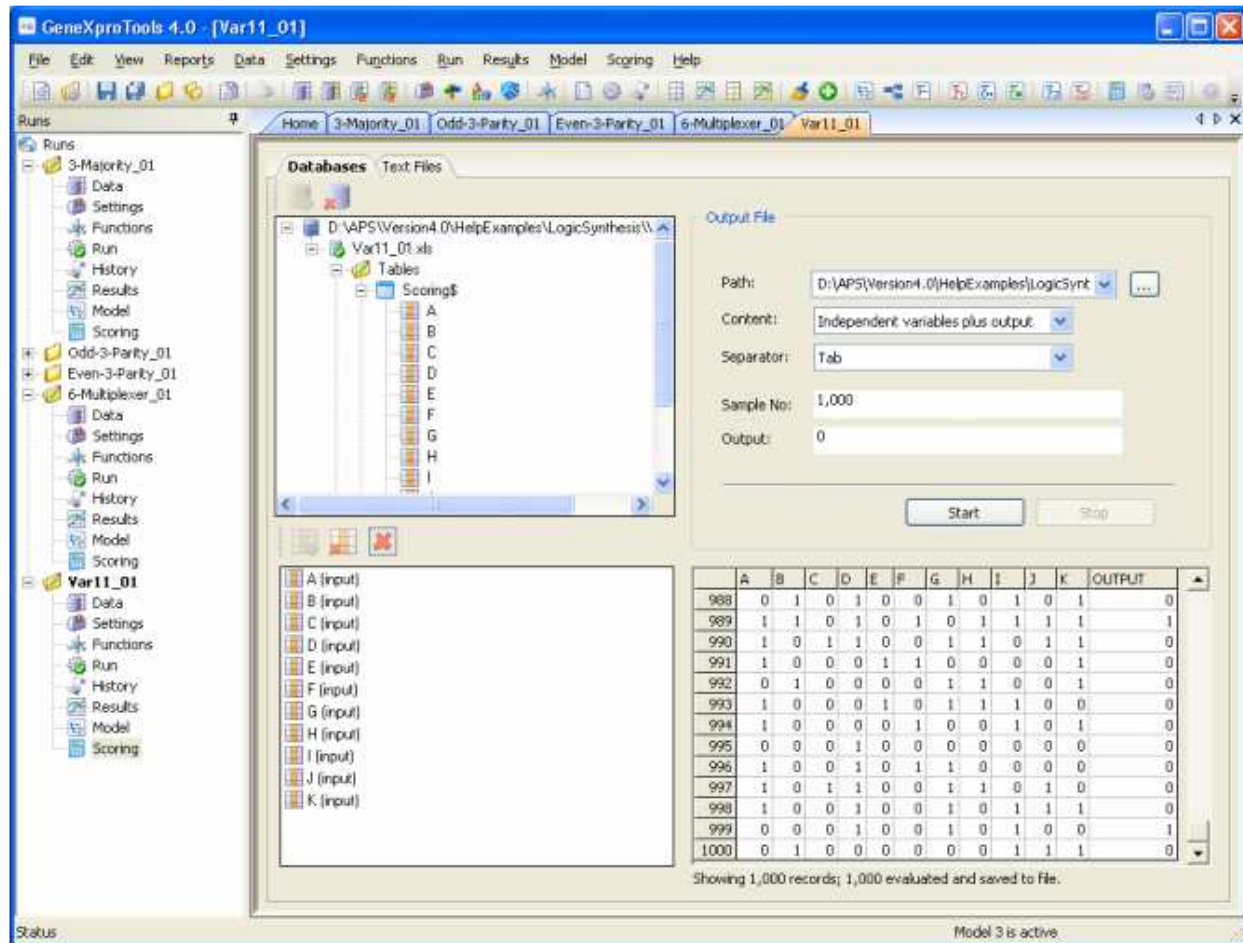
Generating Code Automatically Using User Defined Grammars

GeneXproTools 4.0 allows the design of [User Defined Grammars](#) so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the programming language of your choice if you happen to prefer a programming language not already covered by the 16 [built-in grammars](#) of GeneXproTools 4.0 (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL).

As an illustration, the All Gates VHDL grammar of GeneXproTools 4.0 is shown [here](#). Other grammars may be easily created using this or other GeneXproTools built-in grammars as reference.

Scoring Engine

GeneXproTools 4.0 comes equipped with a scoring engine for scoring your databases/Excel or text files in the GeneXproTools environment. The scoring engine of GeneXproTools uses the All Gates JavaScript code of your models to score your data.



Scoring logic circuits is not current practice, but you can make use of the scoring engine of GeneXproTools 4.0 just to check your code or for the fun of it. As usual, the scores are saved to Excel (csv) or text files.



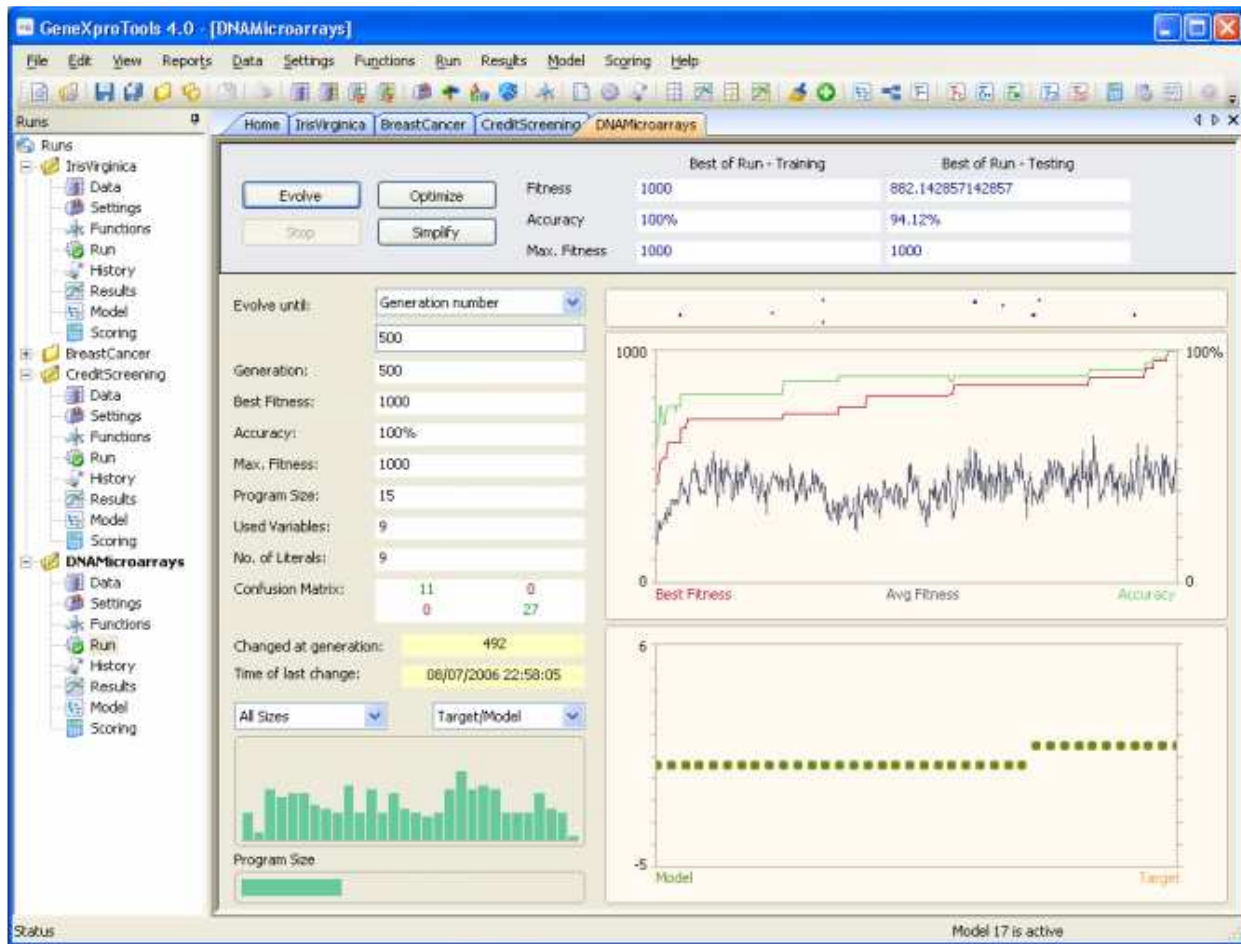
GeneXproTools Environment

GeneXproTools Panels

The modeling environment of GeneXproTools 4.0 is structured into the following basic panels:

- [Report](#)
- [Data](#)
- [Settings](#)
- [Functions](#)
- [Run](#)
- [History](#)
- [Results](#)
- [Predictions](#)
- [Model](#)

- Scoring

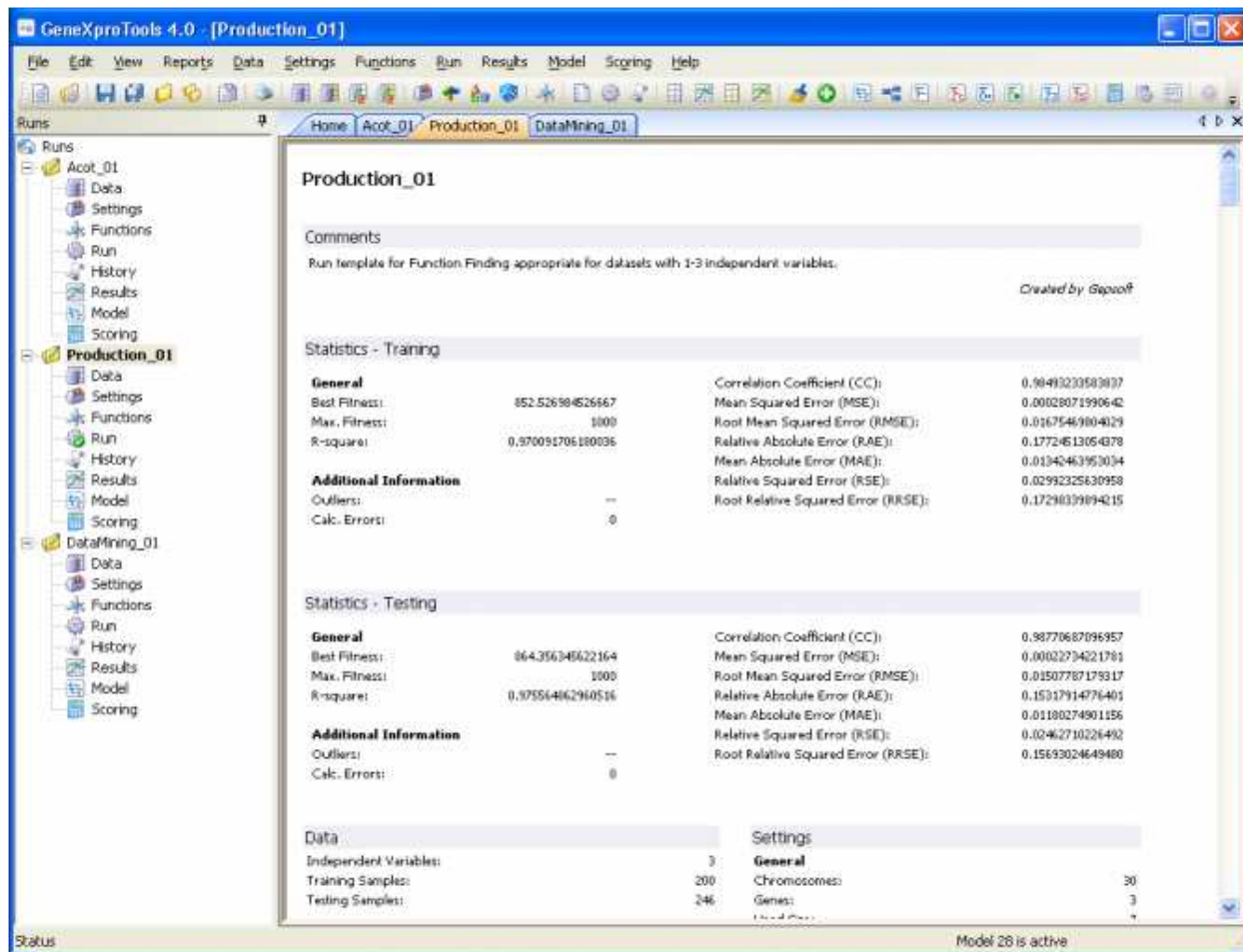


◀ | ▶

GeneXproTools Environment

Report

The **Report Panel** of GeneXproTools 4.0 keeps a summary of all the relevant information of a run, organized into the following headings:



Comments

This heading is usually used to give a small description of the problem, the objectives, the kind of data used, their source, etc. Both the comments and the author's name are introduced by choosing Properties in the File Menu.

Statistics – Training

Under this heading are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the training set. Depending on the problem category, these statistics include: [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), [R-square](#), [classification error](#), [classification accuracy](#), [confusion matrix](#) (true positives, true negatives, false positives, and false negatives), [sensitivity](#), [specificity](#), [positive predictive value](#), and [negative predictive value](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, number of calculation errors (they should always be zero, unless the seed was modified or the training set was changed after the model had been created), and number of outliers (when the fitness function is one of the following: [Relative with SR](#), [Relative/Hits](#), [Absolute with SR](#), and [Absolute/Hits](#)).

A note of warning, though: All the parameters listed here refer to the last evaluated model in the [Results Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	24	9	di(5), di(3), di(1)	847.282339126748	859.152192900566	0.960006472679246	0.973587454532183
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Sqrt} + \text{Atan} X3 \cdot d0 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d0 \cdot d2 \cdot d0} \\ & + \\ & \sqrt{X2} / \sqrt{\text{Sqrt} + \text{Cos} d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d0 \cdot d0} \\ & + \\ & X3 \cdot \text{Atan} d0 \cdot d0 \cdot \text{Cos} Ln \cdot \text{Sqrt} d0 \cdot d2 \cdot d2 \cdot d0 \cdot d1 \cdot d1 \cdot d0 \cdot d1 \end{aligned}$							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
12	25	9	di(4), di(4), di(1)	852.285418790887	864.046489090801	0.970678596225273	0.976046816260914
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Sqrt} + \text{Atan} X3 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0} \\ & + \\ & \sqrt{X2} / \sqrt{\text{Sqrt} + \text{Cos} d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d2 \cdot d0} \\ & + \\ & \text{Atan} X3 \cdot \text{Atan} d0 \cdot d0 \cdot \text{Cos} \text{Exp} d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d1 \cdot d2 \cdot d1 \end{aligned}$							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
23	25	9	di(3), di(4), di(2)	854.701706874743	864.588575207565	0.97254807136764	0.976472602618102
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Sqrt} + \text{Atan} X3 \cdot d1 \cdot d2 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d2 \cdot d0} \\ & + \\ & \sqrt{X2} / \sqrt{\text{Sqrt} + \text{Cos} d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d0 \cdot d0 \cdot d0} \\ & + \\ & \text{Atan} X3 \cdot \text{Atan} d0 \cdot d0 \cdot \text{Cos} \text{Exp} d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d2 \cdot d0 \cdot d0 \end{aligned}$							

Statistics – Testing

Under this heading are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the testing set, that is, the efficiency of your model at predicting unknown behavior. Depending on the problem category, these statistics include: [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), [R-square](#), [classification error](#), [classification accuracy](#), [confusion matrix](#) (true positives, true negatives, false positives, and false negatives), [sensitivity](#), [specificity](#), [positive predictive value](#), and [negative predictive value](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, number of calculation errors, and number of outliers (when the fitness function is one of the following: [Relative with SR](#), [Relative/Hits](#), [Absolute with SR](#), and [Absolute/Hits](#)).

Note again that all the parameters listed here refer to the last evaluated model in the [Results Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	11	4	d0(1), d1(2), d2(1)	858.429078313836	870.616091249945	0.976590014988823	0.982091633615242
<pre> d1.Sqrt Exp.-.Atan.X3.d2.c1.c1.c1.d0.d2.d0.d0.d0.c1 + Atan.X3/.Atan.Exp.3Rt.d1*.d2.d0.d2.d2.c1.d0.d0.c1.c0 </pre> Numerical Constants: Gene 1 c0 = 7.749878 c1 = 8.115112 Gene 2 c0 = -9.974549 c1 = -6.347351							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
2	10	3	d1(2), d2(1)	859.910797156278	872.16215754529	0.975751560710861	0.980563940942955
<pre> d1.Sqrt Exp.-.d2.d2.X3.d2.c1.c1.c1.d0.d2.d0.d0.d0.c1 + Atan.X3/.Atan.Exp.3Rt.d1.X2.d2.d0.c1.d2.c1.d0.d0.c0.c0 </pre> Numerical Constants: Gene 1 c0 = 7.749878 c1 = 8.115112 Gene 2 c0 = -9.974549 c1 = -6.347351							

Data

Under this heading are listed the number of samples in the training set actually used during training (remember that GeneXproTools 4.0 allows you to choose a smaller sub-set of the training set to use during training), the number of testing samples used to test the generalizing capabilities of your model (again, not all the samples on the testing set were necessarily used to test your model), and the number of terminals or independent variables in your data.

Program Structure

The program size, the number of literals, and a detailed list of all used variables and respective weights are shown here. This information refers to the active model highlighted in yellow in the History heading.

Program Structure:	
Program Size:	11
Literals:	4
Used Variables:	d0(1), d1(2), d2(1)

Function Set

Under this heading are listed all the functions used to create your model, together with the symbols used to represent them in [Karva notation](#), their weights and arities.

Function Set		
Function	Symbol	Weight Arity
Addition	+	1 2
Subtraction	-	1 2
Multiplication	*	1 2
Division	/	1 2
Square root	Sqrt	1 1
Exponential	Exp	1 1
Natural logarithm	Ln	1 1
x to the power of 2	X2	1 1
x to the power of 3	X3	1 1
Sine	Sin	1 1
Cosine	Cos	1 1
Cube root	3Rt	1 1
Arctangent	Atan	1 1

Note that the highest value of arity in your function set determines the structure of each gene, making the length of the **tail** t grow according to the equation:

$$t = h (n-1) + 1$$

where h is the length of the **head** and n is maximum arity.

Settings

Under this heading there are five entries: General Settings, Complexity Increase, Fitness Function, Genetic Operators, and Numerical Constants. Under General Settings are listed the number of chromosomes, number of genes, head size, tail size, gene size, and linking function. Under Complexity Increase are listed the number of Generations Without Change, the Number of Tries, and Max. Complexity. The name of the fitness function and all the relevant parameters for its design are given below the entry Fitness Function. Under Genetic Operators are listed the rates of all the genetic operators used to create genetic variation. And, finally, all the parameters relevant to the manipulation of random numerical constants are listed under Numerical Constants.

Settings	
General	
Chromosomes:	30
Genes:	3
Head Size:	8
Tail Size:	9
Dc Size:	9
Gene Size:	26
Linking Function:	Addition
Complexity Increase	
Generations without Change:	2000
Number of Tries:	3
Max. Complexity:	5
Fitness Function	
Error Type:	RRSE
Precision:	--
Selection Range:	--
Genetic Operators	
Mutation Rate:	0.044
Inversion Rate:	0.1
IS Transposition Rate:	0.1
RIS Transposition Rate:	0.1
One-Point Recombination Rate:	0.3
Two-Point Recombination Rate:	0.3
Gene Recombination Rate:	0.1
Gene Transposition Rate:	0.1
Numerical Constants	
Constants per Gene:	2
Data Type:	Floating-Point
Lower Bound:	-10
Upper Bound:	10
RNC Mutation:	0.01
Dc Mutation:	0.044
Dc Inversion:	0.1
Dc IS Transposition:	0.1

Data Sources

Under this heading the paths to the input files (training and testing files) used to create your model are given so that their origin could be easily identified.

File Path

Under this heading the path to the gep file that was used to save your model is given so that it can be easily traced for future reference.

History

Under this heading are listed all the best-of-generation models, including obviously the best-of-run model, discovered during the evolutionary process. Note, however, that the notation "best-of-run" and "best-of-generation" refers to the performance of these models on the training set, which could be slightly different on the testing set. This is why GeneXproTools 4.0 allows you to analyze the

History Panel.

Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	24	9	d0(5), d1(3), d2(1)	047.202339126740	059.152152900566	0.960006472679246	0.973507484632103
$\sqrt{X3} \cdot \sqrt{X1} + \text{Atan } X3 \cdot d0 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d0 \cdot d2 \cdot d0$ $+ \sqrt{X2} \cdot \sqrt{X1} + \text{Cos } d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d0 \cdot d0$ $+ X3 \cdot \text{Atan } d0 \cdot d0 \cdot \text{Cos } \ln \sqrt{X1} \cdot d0 \cdot d2 \cdot d2 \cdot d0 \cdot d1 \cdot d1 \cdot d0 \cdot d1$							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
12	25	9	d0(4), d1(4), d2(1)	052.205419790807	064.046469090001	0.970678956225273	0.976046016260914
$\sqrt{X3} \cdot \sqrt{X1} + \text{Atan } X3 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0$ $+ \sqrt{X2} \cdot \sqrt{X1} + \text{Cos } d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d2 \cdot d0$ $+ \text{Atan } X3 \cdot \text{Atan } d0 \cdot d0 \cdot \text{Cos } \exp d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d1 \cdot d2 \cdot d1$							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
23	25	9	d0(3), d1(4), d2(2)	054.701706874743	064.588675207565	0.97254807136764	0.976472602518102
$\sqrt{X3} \cdot \sqrt{X1} + \text{Atan } X3 \cdot d1 \cdot d2 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d2 \cdot d0$ $+ \sqrt{X2} \cdot \sqrt{X1} + \text{Cos } d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d0 \cdot d0 \cdot d0$ $+ \text{Atan } X3 \cdot \text{Atan } d0 \cdot d0 \cdot \text{Cos } \exp d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d2 \cdot d0 \cdot d0$							

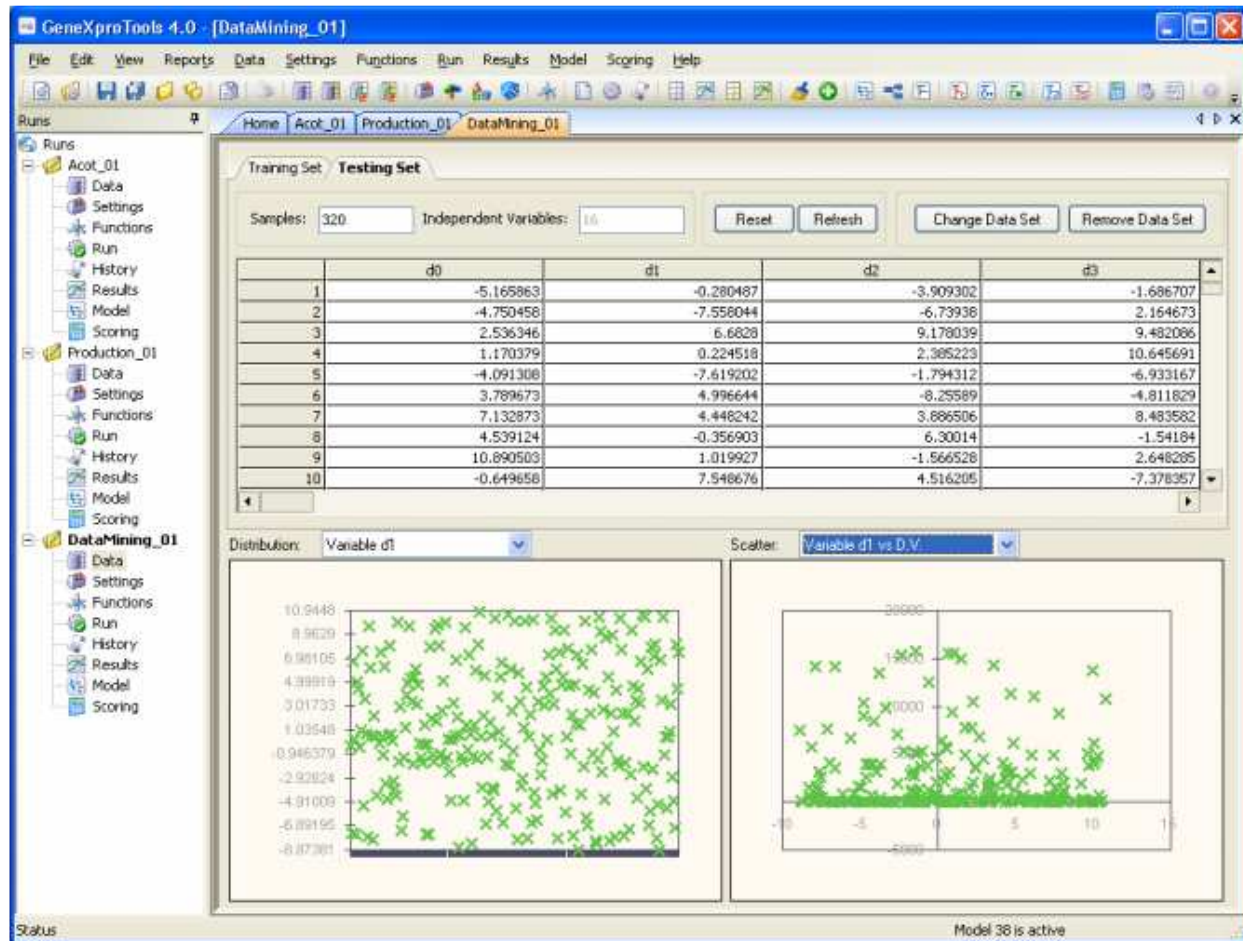
models in the [History Panel](#) and then translate it into the language of your choice in the [Model Panel](#).

GeneXproTools Environment

Data

In the **Data Panel** of GeneXproTools 4.0 you can access, visualize and change the:

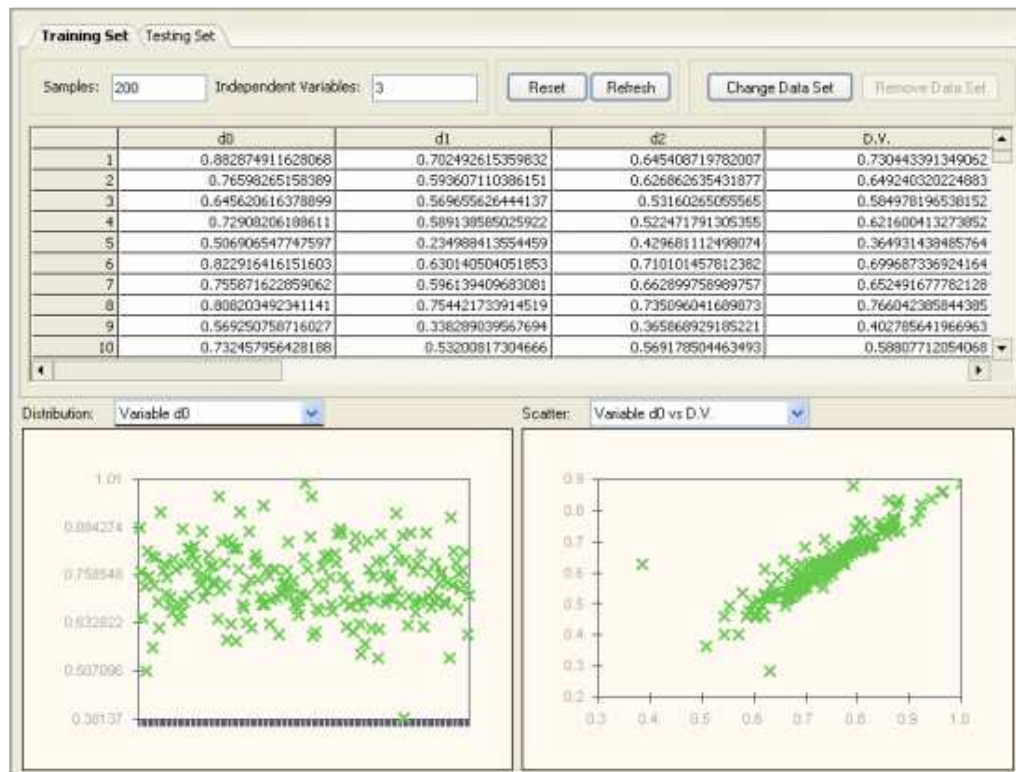
- Training Set
- Testing Set



GeneXproTools Environment

Training Data

By selecting the Training Set Tab in the Data Panel, you have full access to the training samples. Here you can analyze, visualize, choose a smaller sub-set for training, or change the data set altogether.



The data samples are shown on a spreadsheet for easy analysis and you can also use the plots at the bottom to analyze the distribution of all the inputs and to spot simple relationships between the independent variables and the dependent variable. By moving the scroll bar on the right you can easily visualize the entire dataset.

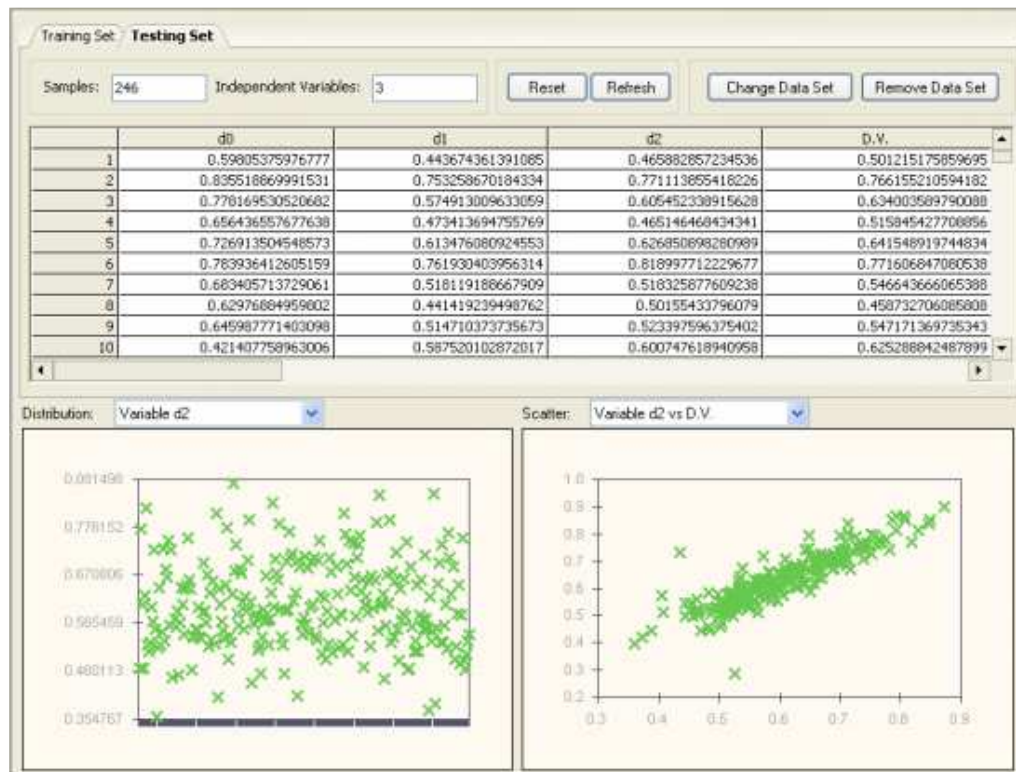
The visualization of both the distribution and relationships between variables is a valuable modeling tool as it can help you detect outliers or simple relationships between variables that can be used to evolve more efficient models.



GeneXproTools Environment

Testing Data

By selecting the Testing Set Tab in the Data Panel, you have full access to the testing samples. Here you can analyze, visualize, choose a smaller sub-set for testing, change or even remove the data set.



The data samples are shown on a spreadsheet for easy analysis and you can also use the plots at the bottom to analyze the distribution of all the inputs and to spot simple relationships between the independent variables and the dependent variable. By moving the scroll bar on the right you can easily visualize the entire dataset.

The visualization of both the distribution and relationships between variables is a valuable modeling tool as it can help you detect outliers or simple relationships between variables that can be used to evolve more efficient models.

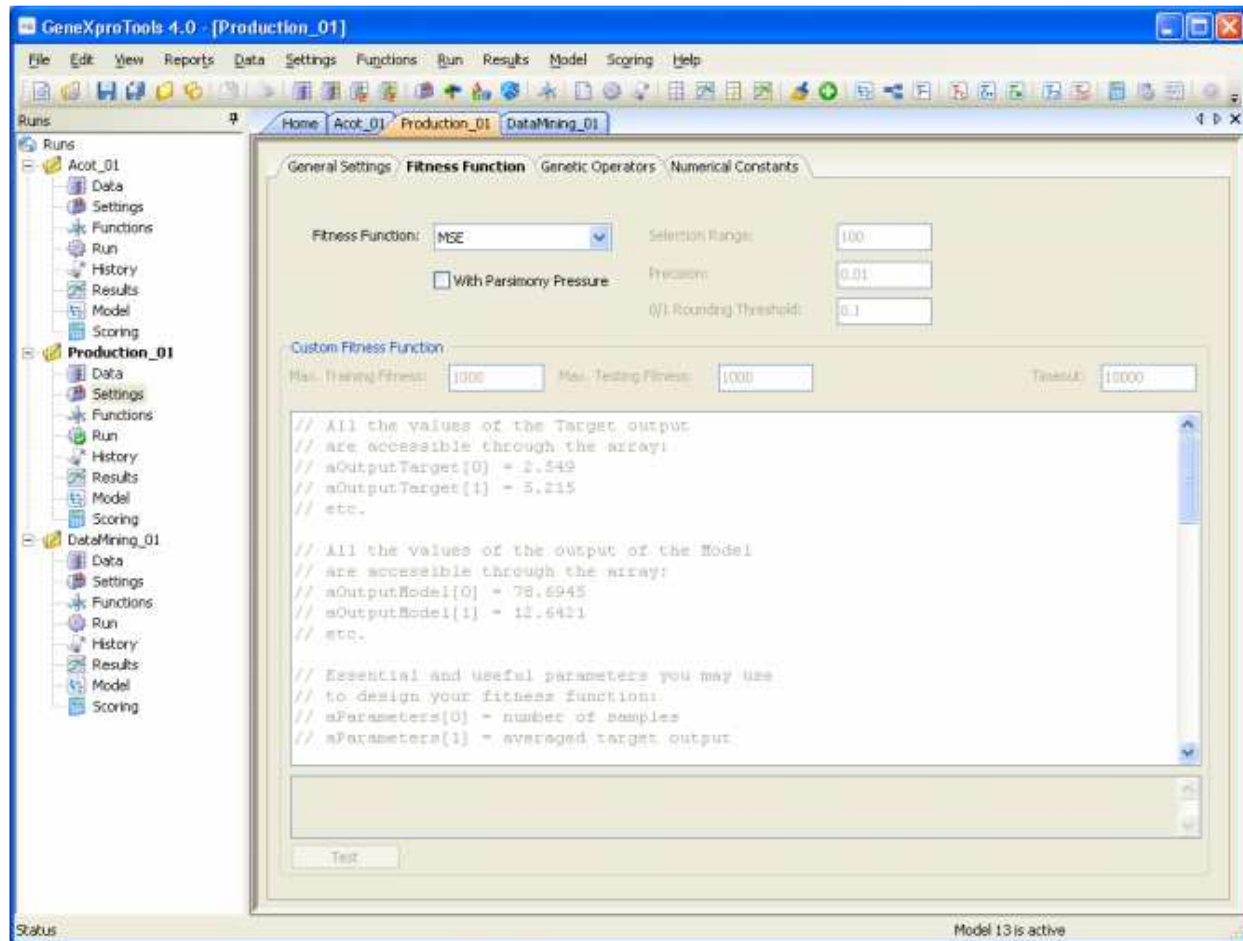


GeneXproTools Environment

Settings

The **Settings Panel** of GeneXproTools 4.0 is organized into the following sections:

- [General Settings](#)
- [Fitness Function](#)
- [Genetic Operators](#)
- [Numerical Constants](#)



GeneXproTools Environment

General Settings

Except for the different types of linking functions, the General Settings Tabs used both for Function Finding, Classification, and Logic Synthesis are exactly the same and look like this:

The screenshot shows the 'General Settings' tab of the GeneXproTools interface. It contains several configuration options:

- Training Samples:** 200 (with a 'Reset' button)
- Testing Samples:** 246 (with a 'Reset' button)
- Number of Chromosomes:** 30
- Head Size:** 7
- Number of Genes:** 3
- Linking Function:** Addition (dropdown menu)
- ☒ **Enable Complexity Increase**
 - Generations without Change:** 2000
 - Number of Tries:** 3
 - Max. Complexity:** 5
- ☒ **Save Run History**

Here you can change the number of samples used for training and the number of samples used for testing the generalizing capabilities of the evolved models.

Here is also the place where you choose the size of the population of evolving models (Number of Chromosomes) and where you design the [structural organization](#) of the chromosomes by choosing the Head Size, the Number of Genes, and the Linking Function. For mathematical problems, you can choose from four different linking functions (addition, subtraction, multiplication, and division); and for Boolean problems, you can choose from 10 different linking functions (And, Or, Nand, Nor, Xor, Nxor, Less Than, Greater Than, Less Or Equal, Greater Or Equal).

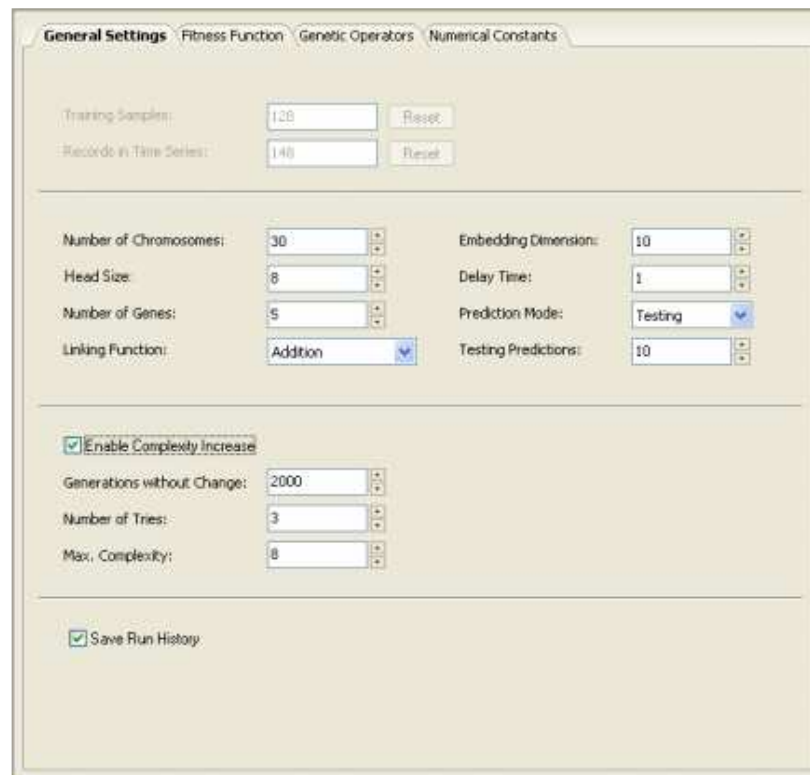
The Head Size and the Number of Genes are constrained by the maximum chromosome size allowed in GeneXproTools, which is 2049. And the chromosome size depends not only on the Number of Genes and Head Size but also on maximum arity and the [learning algorithm](#) (with or without random numerical constants).

And, finally, in the General Settings Tab you can also enter the parameters for the **Complexity Increase Engine**. In the Generations Without Change box you set the period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction or a neutral gene (an extra term) is automatically added to your model; in the Number of Tries box you enter the number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models; in the Max. Complexity box you write the maximum number of terms (genes) you'll allow in your model and no other terms will be introduced beyond this threshold.

The Complexity Increase Engine of GeneXproTools 4.0 might be a very powerful modeling tool, especially for Time Series Prediction where good models could take longer to come by, but you must be careful not to create excessively complex models for, most of the times, a greater complexity does not necessarily imply a greater efficiency.

The General Settings Tab used for Time Series Prediction gives access to all the parameters listed above, but has also additional parameters necessary for dealing with time series data (the Embedding

Dimension, the Delay Time, the Prediction Mode and the number of Testing Predictions).



General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 128 [Reset]

Records in Time Series: 148 [Reset]

Number of Chromosomes: 30 | Embedding Dimension: 10

Head Size: 8 | Delay Time: 1

Number of Genes: 5 | Prediction Mode: Testing

Linking Function: Addition | Testing Predictions: 10

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 8

☒ Save Run History

In practice, the embedding dimension corresponds to the number of independent variables (terminals) after your time series has been transformed. The delay time τ determines how data are processed, that is, continuously if $\tau = 1$ or at τ intervals.

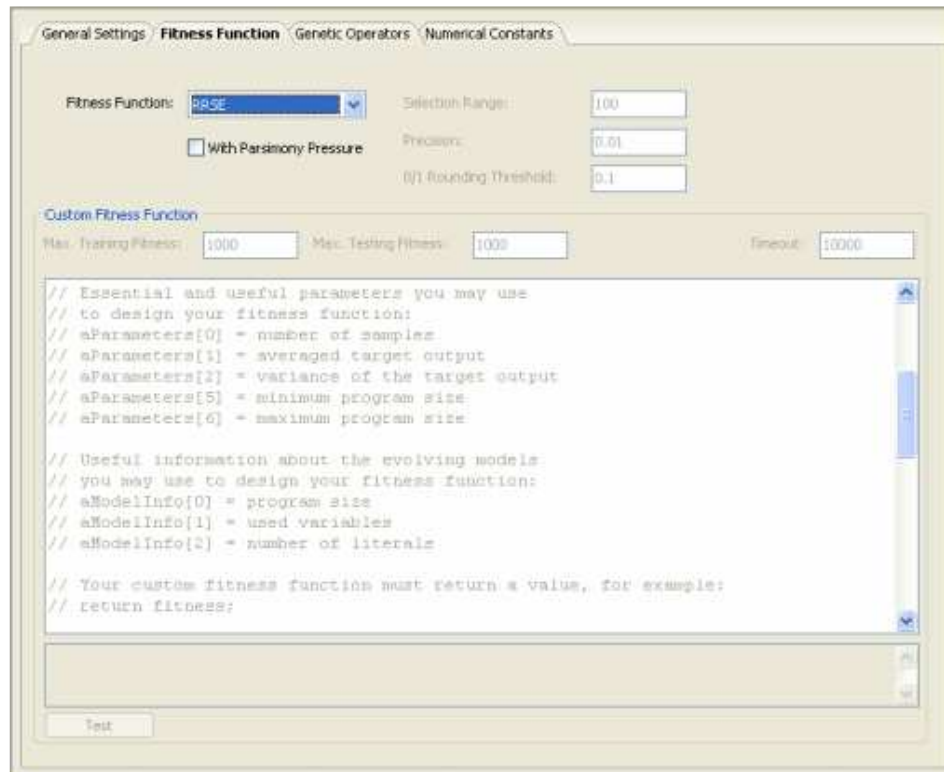
These two parameters, together with the size of the time series and the prediction mode, will determine the final number of training samples after the transformation of the time series.



GeneXproTools Environment

Fitness Function

The Fitness Function Tabs used both for Function Finding and Time Series Prediction are exactly the same and look like this:



Here you can select one of the 36 [built-in fitness functions](#) of GeneXproTools 4.0 ([Absolute with SR](#), [Absolute/Hits](#), [MSE](#), [RMSE](#), [MAE](#), [RSE](#), [RRSE](#), [RAE](#), [Relative with SR](#), [Relative/Hits](#), [rMSE](#), [rRMSE](#), [rMAE](#), [rRSE](#), [rRRSE](#), [rRAE](#), [R-square](#), and [Correlation Coefficient](#), together their respective counterparts with [parsimony pressure](#)) and, depending on the fitness function, adjust a few parameters, such as the Selection Range and Precision for [Relative with SR](#) and [Absolute with SR](#) or the Precision for [Relative/Hits](#) and [Absolute/Hits](#).

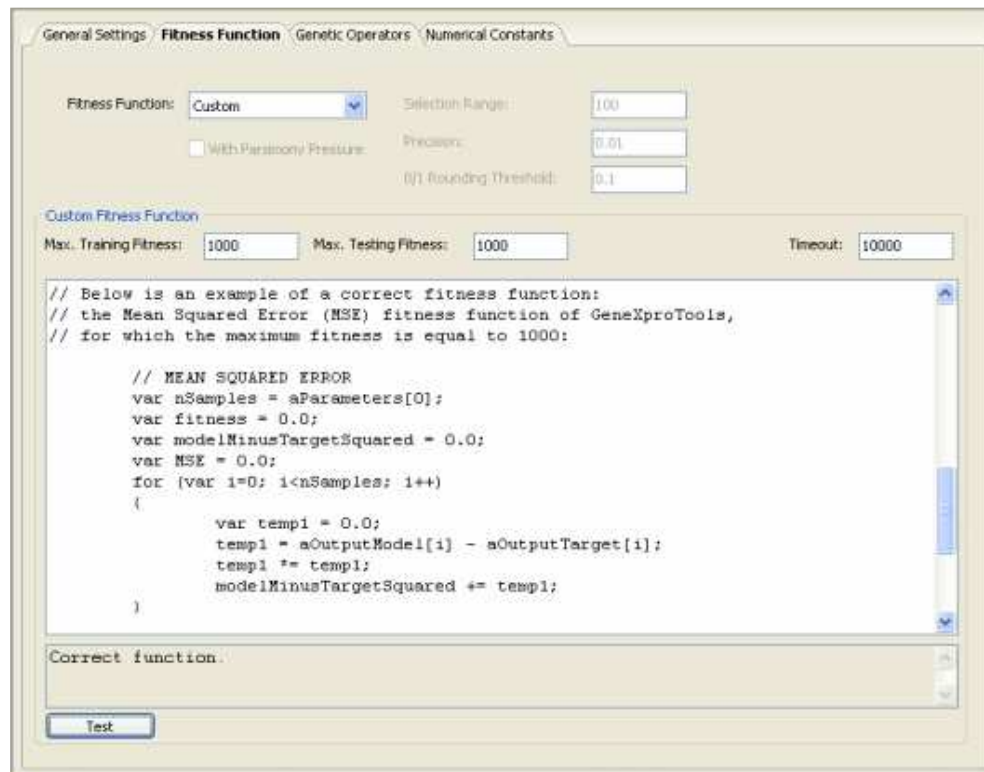
In the Fitness Function Tab used for Classification and Logic Synthesis the 30 [built-in fitness functions](#) of GeneXproTools 4.0 are listed ([Number of Hits](#), [Hits with Penalty](#), [Accuracy](#), [Squared Accuracy](#), [Sensitivity/Specificity](#), [PPV/NPV](#), [SSPN](#), [R-square](#), [Correlation Coefficient](#), [MSE](#), [RMSE](#), [MAE](#), [RSE](#), [RRSE](#), and [RAE](#)), together with the adjustable 0/1 Rounding Threshold used in Classification problems and fundamental for all fitness functions, including user defined fitness functions.

The screenshot shows the 'Fitness Function' tab in the GeneXproTools interface. It features a dropdown menu for 'Fitness Function' set to 'Sensitivity/Specificity', a 'With Parsimony Pressure' checkbox, and input fields for 'Selection Range' (100), 'Precision' (0.01), and '0/1 Rounding Threshold' (0.5). Below these is a 'Custom Fitness Function' section with input fields for 'Max. Training Fitness' (100), 'Max. Testing Fitness' (50), and 'Timeout' (10000). A large text area contains a JavaScript code template for a custom fitness function, including comments about parameters like 'aParameters[0]' (number of samples) and 'aModelInfo[0]' (program size). A 'Test' button is located at the bottom left of the window.

```
// Essential and useful parameters you may use
// to design your fitness functions:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[3] = 0/1 rounding threshold
// aParameters[4] = number of samples in the predominant class
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models:
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals
```

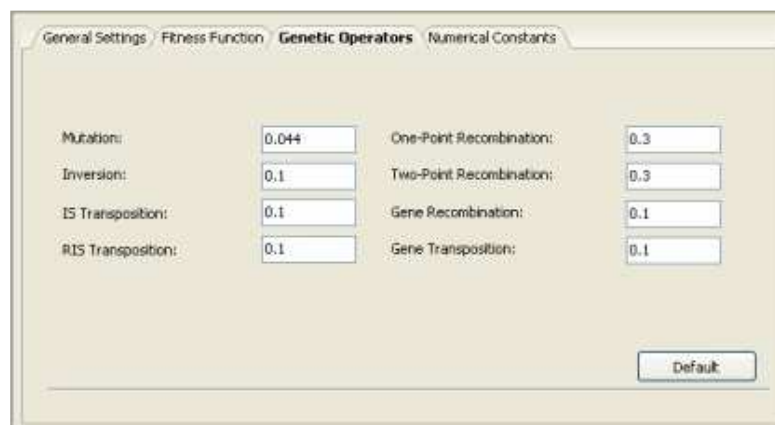
In addition to the wide variety of GeneXproTools built-in fitness functions, you can also design other fitness functions by selecting Custom in the Fitness Function list box. Indeed, for all problem categories, the Fitness Function Tab is also the place where you can design your own fitness function using the Custom Fitness Function window to write the code of your fitness function. The code for the [custom fitness function](#) must be in JavaScript and can be tested before evolving a model with it.



GeneXproTools Environment

Genetic Operators

By selecting the Genetic Operators Tab, you have full access to the genetic operators used to create genetic diversity in the population of evolving models.



All the learning algorithms of GeneXproTools 4.0 use the basic genetic operators listed in the Genetic

Operators Tab ([mutation](#), [inversion](#), [IS transposition](#), [RIS transposition](#), [one-point recombination](#), [two-point recombination](#), [gene recombination](#), and [gene transposition](#)). These operators allow you to introduce the ideal degree of [genetic modification](#) so that an efficient evolution could take place.

If you are overwhelmed by all these operators, please use the default values as they work very well for all problems. If you are curious, though, about the workings of evolution and the importance of each operator, you can see what happens by selecting the ones you are interested in and choosing different rates and then observe the evolutionary process in the Run Panel.



GeneXproTools Environment

Numerical Constants

By selecting the Numerical Constants Tab, you are now able to activate the [GEP-RNC learning algorithm](#) of GeneXproTools 4.0 by checking the Use Random Numerical Constants box.

General Settings Fitness Function Genetic Operators **Numerical Constants**

☒ Use Random Numerical Constants

Constants per Gene: 2 Lower Bound: -10

Data Type: Floating-Point Upper Bound: 10

RNC Mutation: 0.01

Dc Mutation: 0.044

Dc Inversion: 0.1

Dc IS Transposition: 0.1

Default

The [GEP-RNC algorithm](#) is very similar to the simplest [GEP](#) of GeneXproTools 4.0, with the difference that it gives the user the possibility of choosing the range and type of numerical constants that will be used during the learning process.

Here you have also access to a small set of genetic operators ([RNC mutation](#), [Dc mutation](#), [Dc inversion](#), and [Dc IS transposition](#)) especially developed for handling random numerical constants (if you are not familiar with these operators, please use the default values for they work very well in all cases).

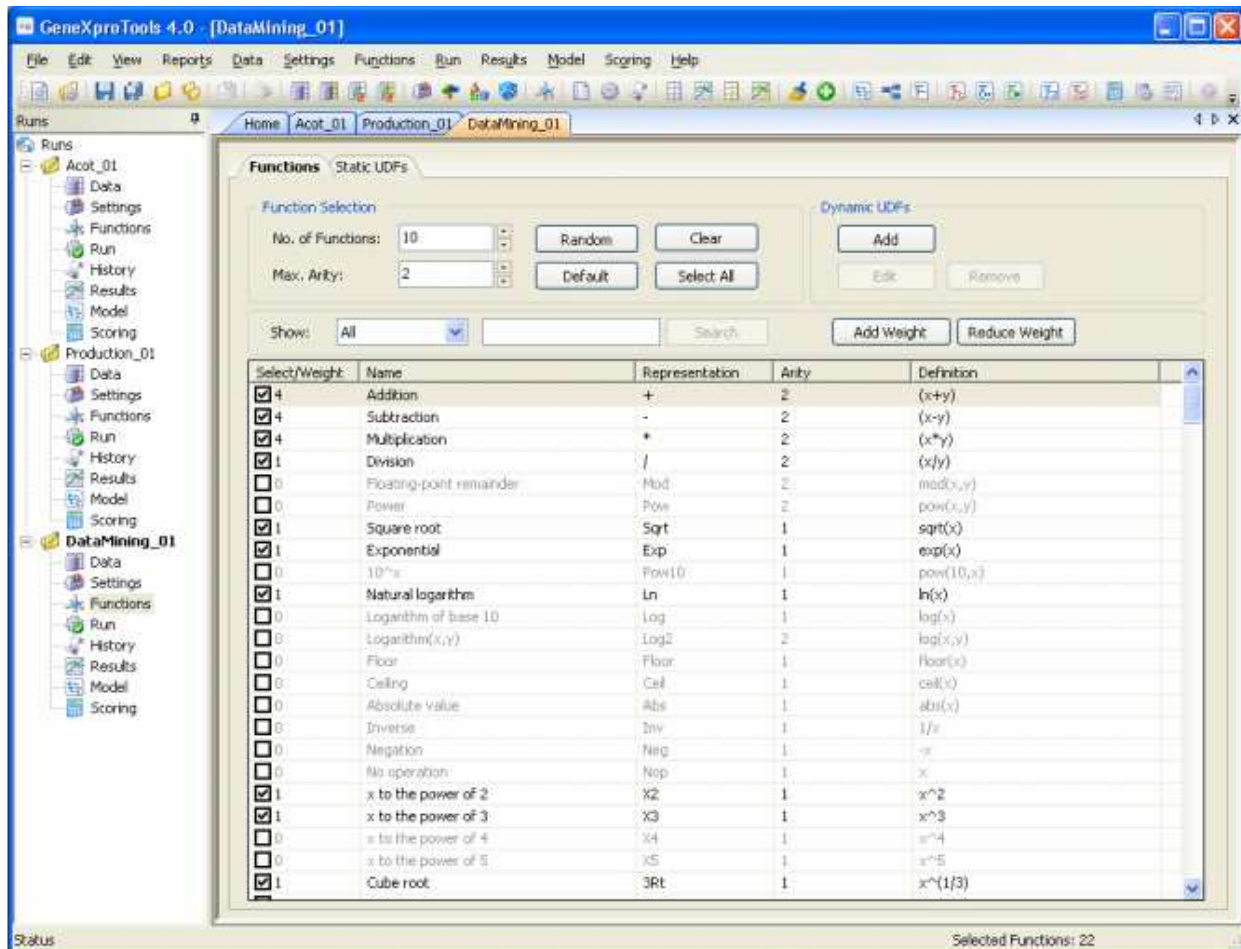


GeneXproTools Environment

Functions

The **Functions Panel** of GeneXproTools 4.0 gives access the following functions:

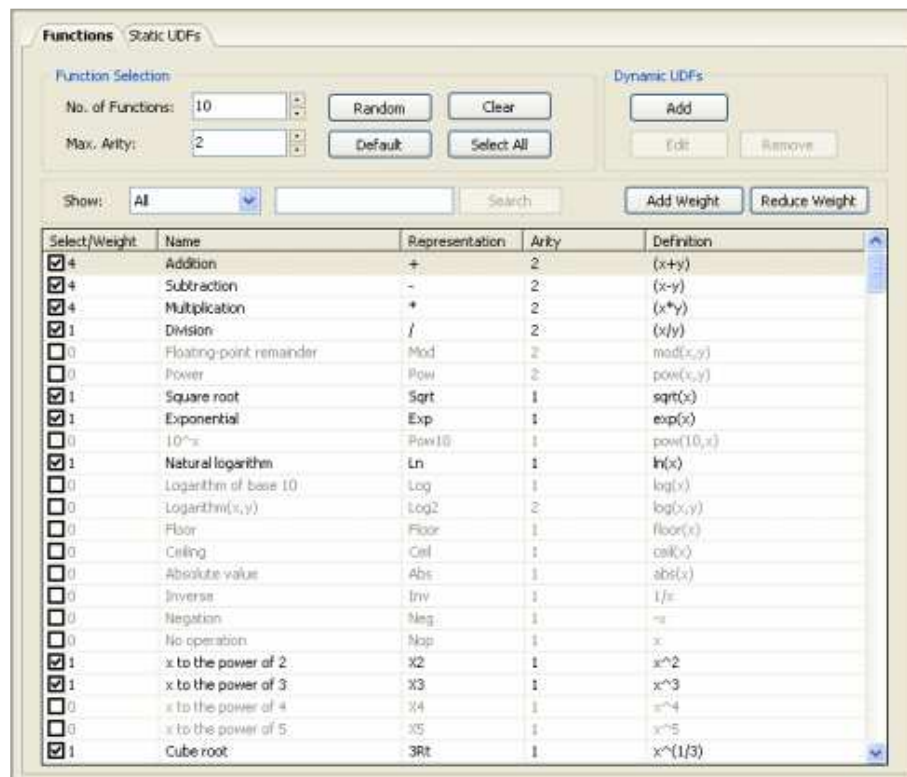
- [Mathematical Functions](#) (including Dynamic UDFs or DDFs).
- [Logical Functions](#) (including Dynamic UDFs or DDFs).
- [Static UDFs](#).



GeneXproTools Environment

Mathematical Functions

For mathematical problems (Function Finding, Classification, and Time Series Prediction), by selecting the Functions Tab in the Functions Panel, you have full access to the 279 [built-in mathematical functions](#) of GeneXproTools 4.0, including 186 IF THEN ELSE rules. Here is also the place where you can add [Dynamic UDFs](#) (DDFs) to your modeling kit.



To select a function, just check the box on the left. By default, the weight of each function is 1, but you can increase the probability of a function being included in your models by increasing its weight in the Select/Weight column.

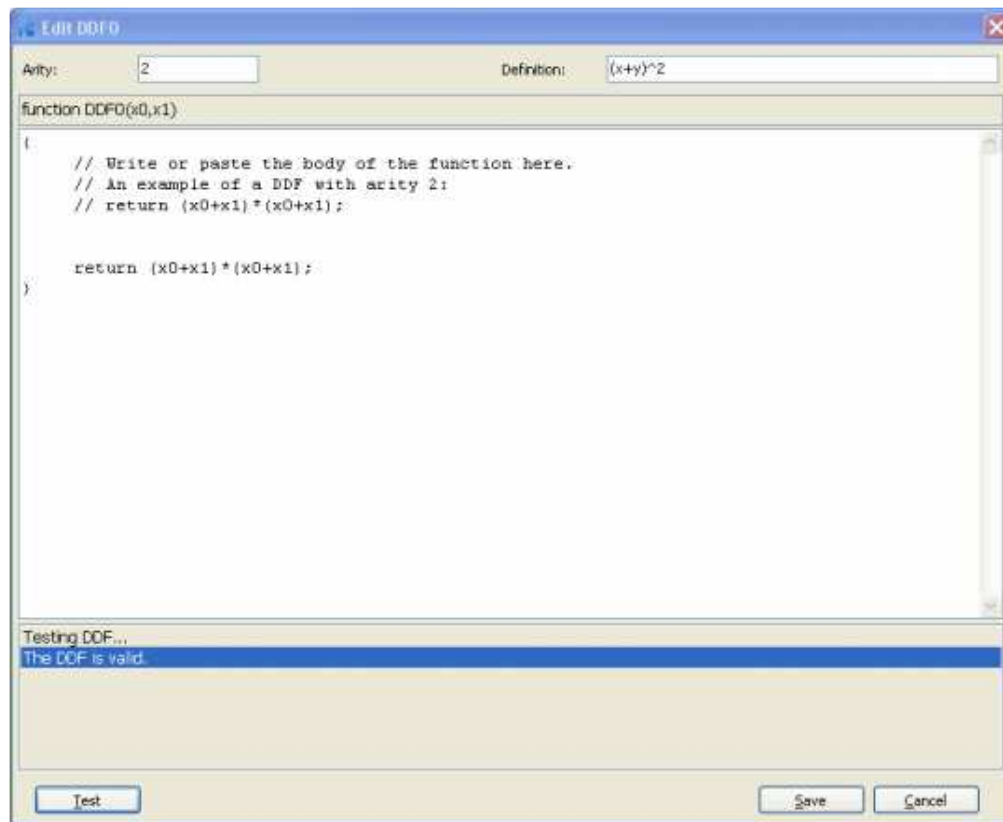
For some problems, the simple arithmetic operators are more than enough to create extremely accurate and elegant models. But for other problems more complex functions are necessary to create good models and when inside knowledge of the problem at hand is not producing the expected results, it might be interesting to experiment with different combinations of functions. The Function Selection facility of GeneXproTools 4.0 helps you to experiment with different function sets very quickly either by pressing the Random button or by selecting certain sub-sets of functions.

In addition, although you can always handcraft your function set so that it is well-balanced relatively to the number of independent variables in the problem at hand, in GeneXproTools 4.0 this is no longer essential as GeneXproTools balances automatically all function sets. This is particularly important for high-dimensional datasets, say above 100-1000 variables, where now you just have to choose the functions you want to use and their relative proportions as GeneXproTools will automatically balance them with the number of variables in your data.

Given the great quantity of functions, the Show box allows you to choose small sub-sets and then select just the ones you want or even select the entire sub-set by pressing the Select All button.

As you can see, before each function, it is given its common name, how it is represented in GeneXproTools 4.0, its arity (the number of arguments it takes), and its mathematical definition.

To add a **Dynamic UDF** (DDF) to your modeling kit, just click the Add button on the Dynamic UDFs frame and the DDF editor appears.



By choosing the arity (minimum is 1 and maximum is 4) in the Arity box, the function header appears below. Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be rapidly tested for compiling errors.

In the Definition box, you can write a brief description of the function for your future reference. The text you write here will appear in the Definition column.

Dynamic UDFs are extremely powerful and interesting tools as they are treated exactly like the built-in functions of GeneXproTools and therefore can be used to model all kinds of relationships between variables or complex expressions. For instance, you can design a DDF so that it will model the sum of four expressions, that is, $DDF = (\text{expression 1}) + (\text{expression 2}) + (\text{expression 3}) + (\text{expression 4})$, where the value of each expression will depend on the context of the DDF in the expression tree. A note of caution, though, although extremely interesting, DDFs decrease considerably the speed of the algorithm and therefore we advise you to choose your functions from the wide set of GeneXproTools 4.0 [built-in functions](#).



GeneXproTools Environment

Logical Functions

In Logic Synthesis mode, by selecting the Functions Tab in the Functions Panel, you have full access to the 258 [built-in logical functions](#) of GeneXproTools 4.0, including all the Boolean functions with one and two inputs, 92 functions with three inputs, and 146 functions with four inputs. Here is also the place where you can add your own functions ([Dynamic UDFs](#) or DDFs) to your modeling kit.

Select/Weight	Name	Representation	Arity	Definition
☒ 1	Not	Not	1	A'
☒ 1	And (ULM)	And	2	$A \cdot B$
☒ 1	Or (ULM)	Or	2	$A + B$
☐ 0	Nand (ULM)	Nand	2	$(A \cdot B)'$
☐ 0	Nor (ULM)	Nor	2	$(A + B)'$
☐ 0	Exclusive or	Xor	2	$A \oplus B$
☐ 0	If and only if	Nxor	2	$(A \oplus B)'$
☐ 0	And(a,b,c) (ULM)	And3	3	$A \cdot B \cdot C$
☐ 0	Or(a,b,c) (ULM)	Or3	3	$A + B + C$
☐ 0	Nand with 3 inputs (ULM)	Nand3	3	$(A \cdot B \cdot C)'$
☐ 0	Nor with 3 inputs (ULM)	Nor3	3	$(A + B + C)'$
☐ 0	Odd-3-parity	Odd3	3	Odd-3-parity
☐ 0	Even-3-parity	Even3	3	Even-3-parity
☐ 0	And(a,b,c,d) (ULM)	And4	4	$A \cdot B \cdot C \cdot D$
☐ 0	Or(a,b,c,d) (ULM)	Or4	4	$A + B + C + D$
☐ 0	Nand with 4 inputs (ULM)	Nand4	4	$(A \cdot B \cdot C \cdot D)'$
☐ 0	Nor with 4 inputs (ULM)	Nor4	4	$(A + B + C + D)'$
☐ 0	Odd-4-parity	Odd4	4	Odd-4-parity
☐ 0	Even-4-parity	Even4	4	Even-4-parity
☐ 0	Identity function	Id	1	$Id(A) = A$
☐ 0	Constant zero function	Zero	1	$O(A) = 0$
☐ 0	Constant one function	One	1	$I(A) = 1$
☐ 0	Less than (ULM)	LT	2	$A < B$

To select a function, just check the box on the left. By default, the weight of each function is 1, but you can increase the probability of a function being included in your models by increasing its weight in the Select/Weight column.

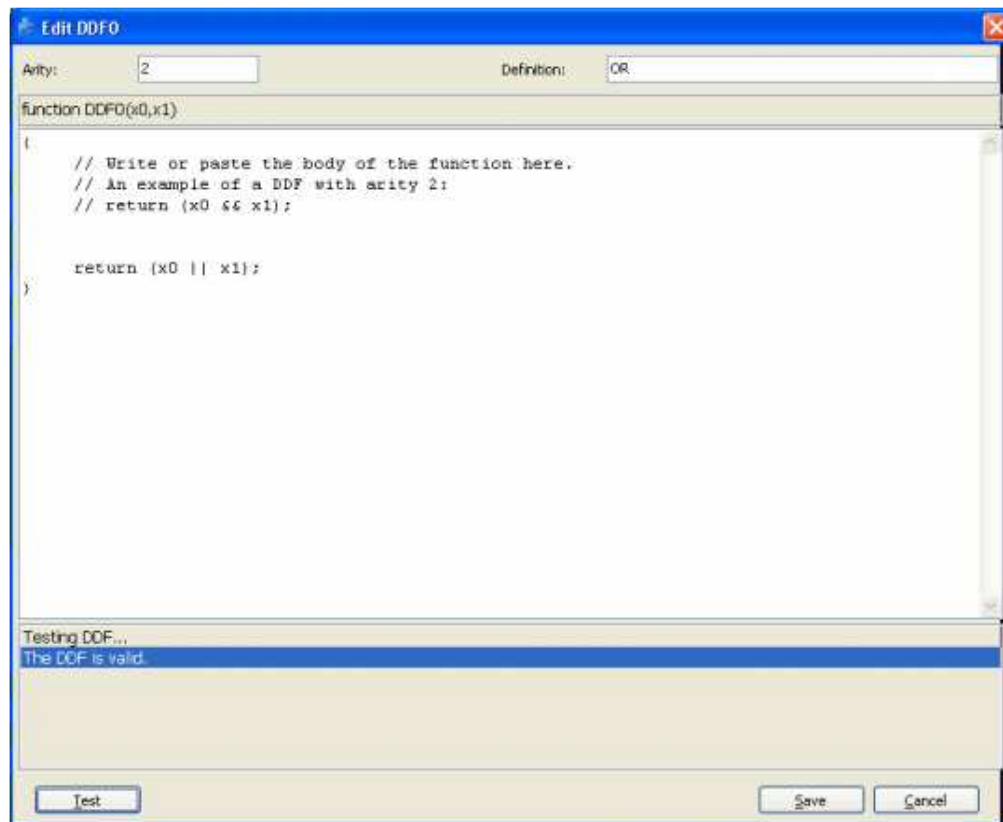
For some problems, the basic Boolean operators AND-OR-NOT are more than enough to create elegant and efficient solutions. But for other problems more complex functions are necessary to create good models and when inside knowledge of the problem at hand is not producing the expected results, it might be interesting to experiment with different combinations of functions. The Function Selection facility of GeneXproTools 4.0 helps you to experiment with different function sets very quickly either by pressing the Random button or by selecting certain sub-sets of functions.

In addition, although you can always handcraft your function set so that it is well-balanced relatively to the number of independent variables in your data, in GeneXproTools 4.0 this is no longer essential as GeneXproTools balances automatically all function sets. This is particularly important for high-dimensional datasets, say above 20 variables, where now you just have to choose the functions you want to use and their relative proportions as GeneXproTools will automatically balance them with the number of variables in your data.

Given the great quantity of functions, the Show box allows you to choose small sub-sets and then select just the ones you want or even select the entire sub-set by pressing the Select All button.

As you can see, before each function, it is given its common name, how it is represented in GeneXproTools 4.0, its arity (the number of arguments it takes), and its definition.

To add a **Dynamic UDF** (DDF) to your modeling kit, just click the Add button on the Dynamic UDFs frame and the DDF editor appears.



By choosing the arity (minimum is 1 and maximum is 4) in the Arity box, the function header appears below. Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be rapidly tested for compiling errors.

In the Definition box, you can write a brief description of the function for your future reference. The text you write here will appear in the Definition column.

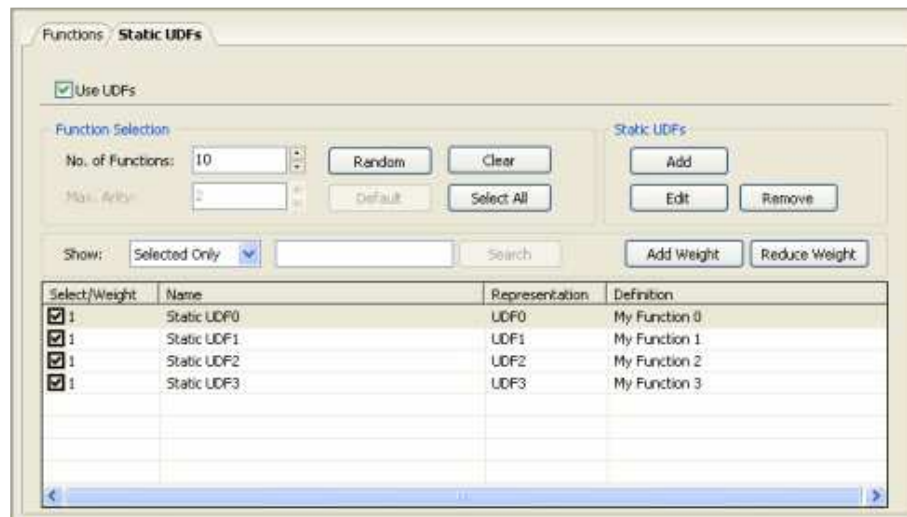
Dynamic UDFs are extremely powerful and interesting tools as they are treated exactly like the built-in functions of GeneXproTools and therefore can be used to model all kinds of relationships between variables or complex expressions. For instance, you can design a DDF so that it will express an OR of four arguments, that is, $DDF = (\text{expression } 1) \text{ OR } (\text{expression } 2) \text{ OR } (\text{expression } 3) \text{ OR } (\text{expression } 4)$, where the value of each expression will depend on the context of the DDF in the expression tree. A note of caution, though, although extremely interesting, DDFs decrease considerably the speed of the algorithm and therefore we advise you to choose your functions from the wide set of GeneXproTools 4.0 [built-in logical functions](#).



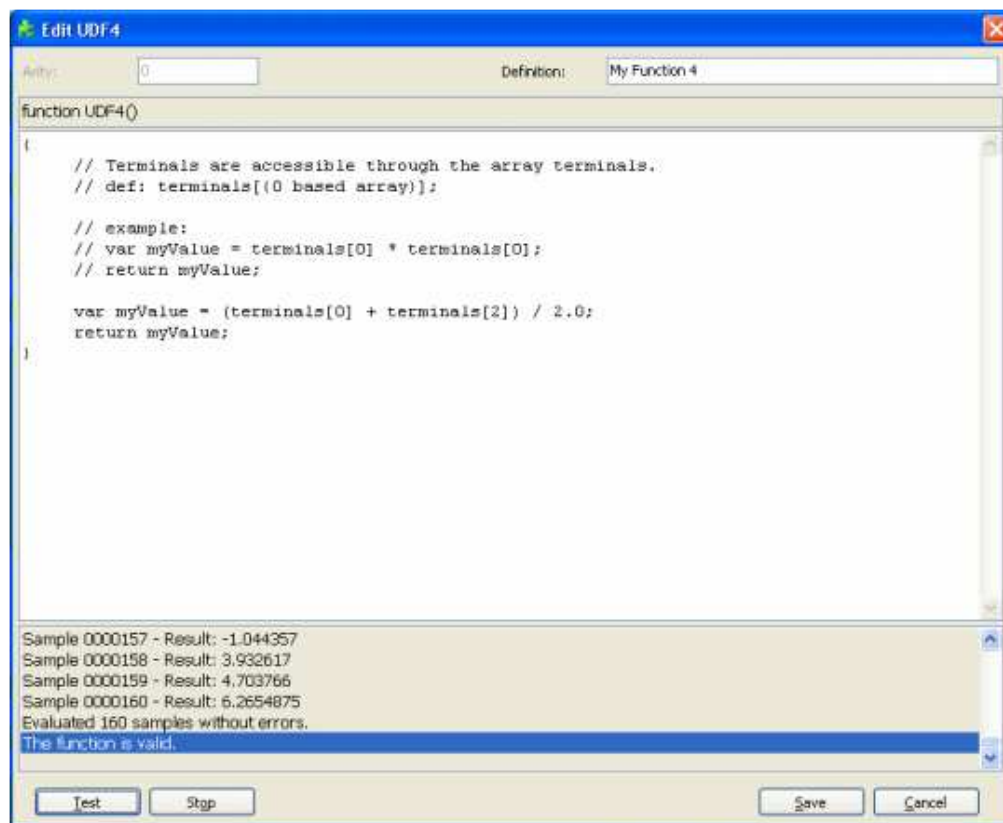
GeneXproTools Environment

Static UDFs

By selecting the Static UDFs Tab in the Functions Panel and checking the Use UDFs box, the Static UDFs framework lights up and you can now add your own user defined functions.



To add a **Static UDF** (UDF) to your modeling kit, just click the Add button on the Static UDFs frame and the UDF editor appears with the function header already present.



Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be rapidly tested for compiling errors.

You will notice that the Arity box is disabled and set to zero as all the UDFs are treated exactly as terminals.

In the Definition box, you can write a brief description of the function for your future reference. The text you write here will appear in the Definition column.

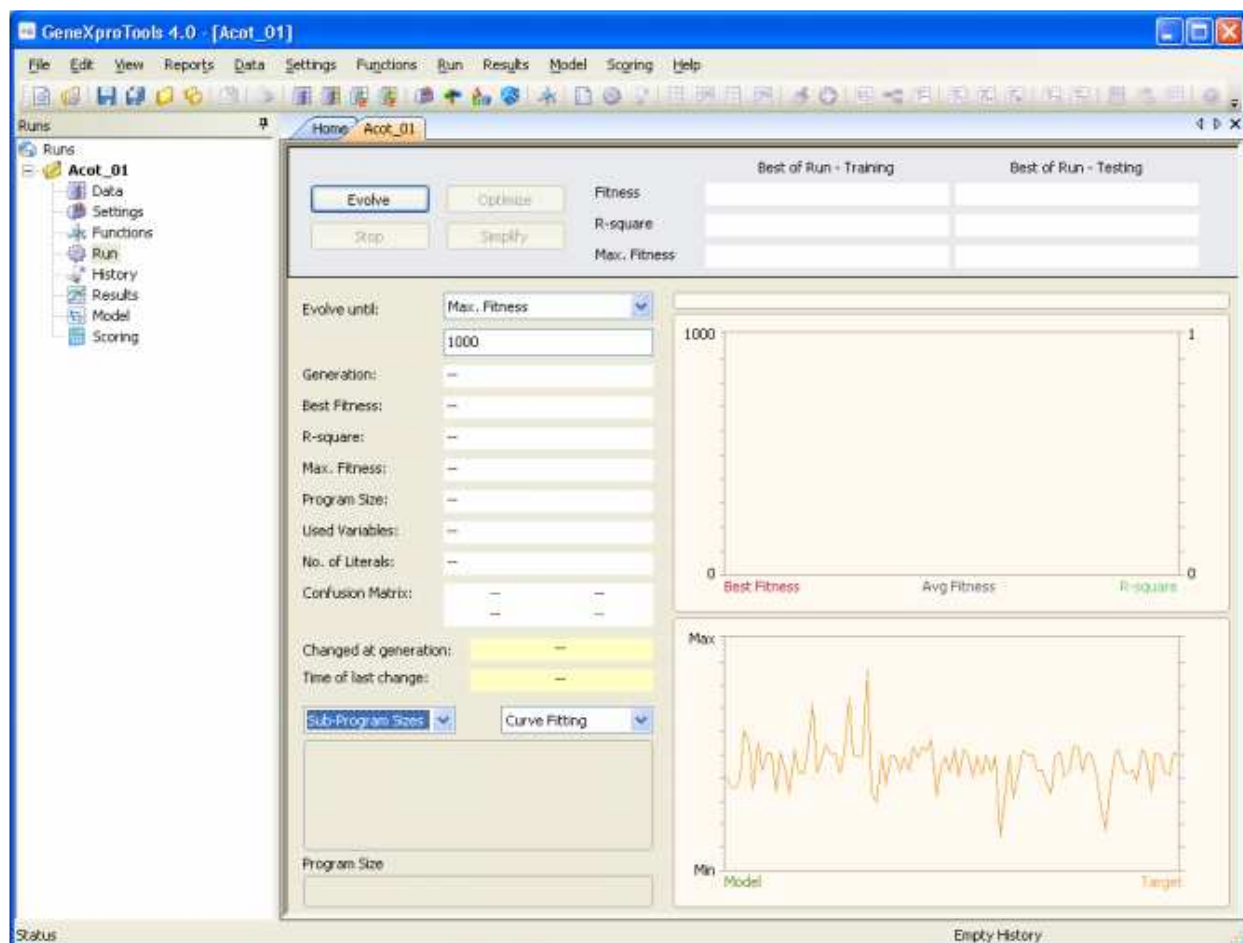
The use of UDFs is a very powerful tool as they allow you to explore the knowledge about relatively simple relationships that exist between certain variables in your data. For instance, suppose you wanted to explore the fact that the sum of six of your variables is relevant, say, $d_0 + d_1 + d_2 + d_3 + d_4 + d_5$. So you design a UDF to express this relationship, for instance, $UDF1 = d_0 + d_1 + d_2 + d_3 + d_4 + d_5$, and use this new UDF as a building block in your modeling kit. The advantage of well designed UDFs is that they make the discovery of more complex models composed of several simpler models much easier.



GeneXproTools Environment

Run

In the **Run Panel** is where modeling takes place and is where you see what is going on. To create a model, just click the Evolve button.

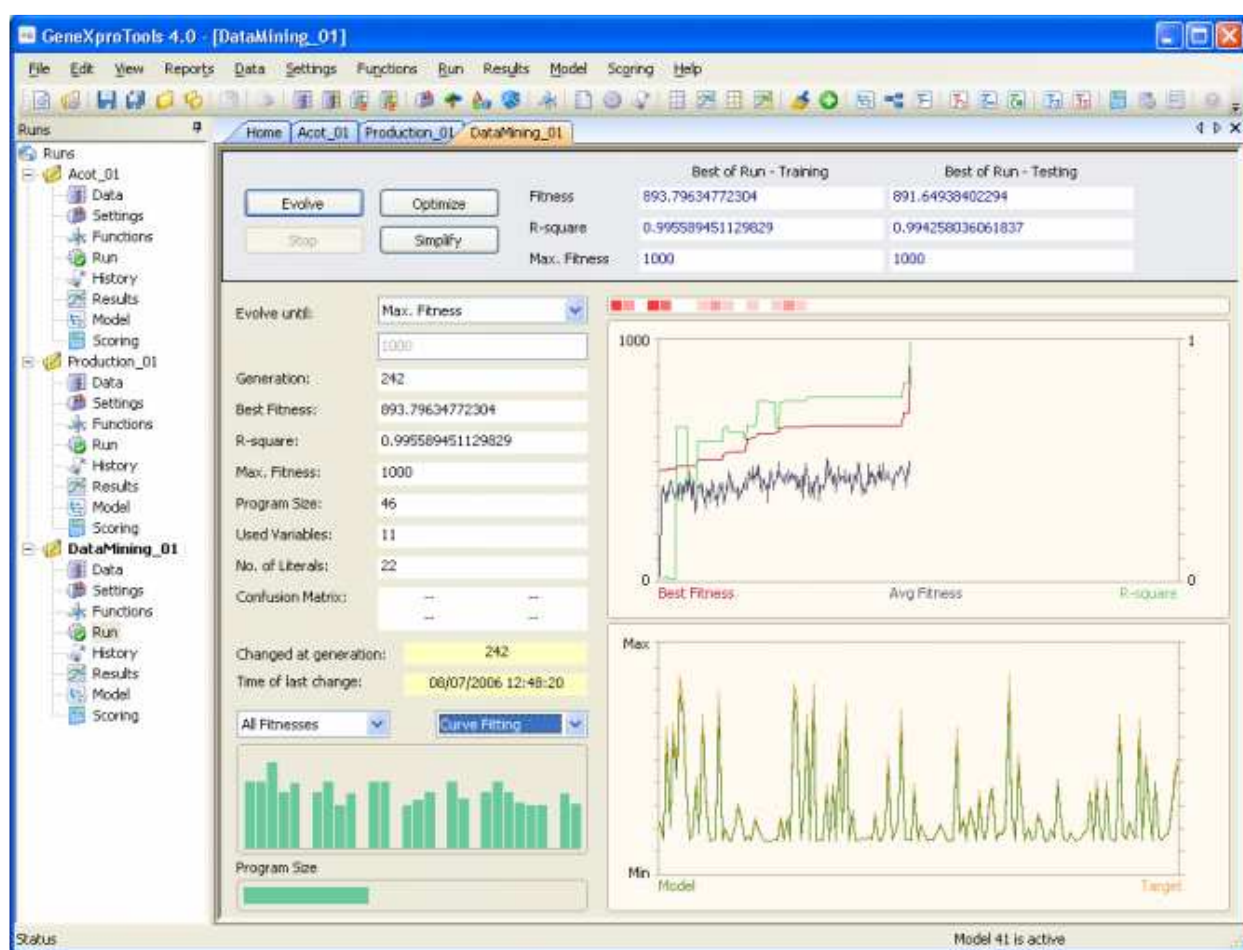


You will notice that while GeneXproTools 4.0 is modeling your data, the Stop button is switched on, enabling you to halt the learning process anytime you see fit.

Anytime you click the Stop button, a model has been selected among the thousands of models created and evaluated during a run. This is the best-of-run model and now you can:

1. Take a closer look at this model using the wide range of tools of GeneXproTools 4.0, such as looking at its structure (size, shape, and elements), translate it into the language of your preference, check its statistics and how it generalizes in the Results Panel, twinkle with its structure in the Change Seed window, evaluate its performance in a different testing set, and so on.
2. Use the model to [make predictions](#).
3. Try to evolve an even better model starting from exactly this point in the solution space ([evolve from seed](#)).
4. Try to simplify it.

The last options are accomplished here in the Run Panel by clicking the Optimize or Simplify buttons, which are activated after each run.



GeneXproTools 4.0 allows you to monitor the modeling process by plotting the essential parameters of a run during the discovery process, including average fitness of the population and the fitness and R-square/Accuracy of the best-of-generation model.

In addition, GeneXproTools allows you to visualize the modeling process by plotting the essential features of a run during the discovery process. In the Run Panel, a total of nine different charts are shown:

1. The Variables Usage map.

This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square you can access the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic.

2. The Evolutionary Dynamics chart.

This chart shows the average fitness of the population plus the fitness and R-square/Accuracy of the best-of-generation model. For a good adaptation, the plot for average fitness should never come near the plot for best fitness, otherwise the system is losing genetic diversity and becoming too uniform for an efficient evolution. Whenever this is happening in your system, you should obviously increase the [genetic diversity](#) of the evolving populations by choosing the right set of genetic operators and adjusting their rates.

3. The Average/Best Size chart.

This chart plots together the average program size of the population and the program size of the best-of-generation model. This chart is especially useful during simplification and can be activated any time during evolution by selecting Avg/Best Size in the rightmost combo box in the bottom.

4. The Curve Fitting chart (for Function Finding and Time Series Prediction).

This chart plots the first 250 data points of the training set and shows how well the evolving models are fitting the target. The Curve Fitting chart can be activated any time during evolution by selecting Curve Fitting in the rightmost combo box in the bottom.

5. The Target/Model chart (for Classification and Logic Synthesis).

This chart plots the first 50 data points of the training set and shows how well the evolving models are fitting the target. The Target/Model chart can be activated any time during evolution by selecting Target/Model in the rightmost combo box in the bottom.

6. The Sub-Program Sizes chart.

This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be activated any time during evolution by selecting Sub-Program Sizes in the leftmost combo box in the bottom.

7. The All Sizes chart.

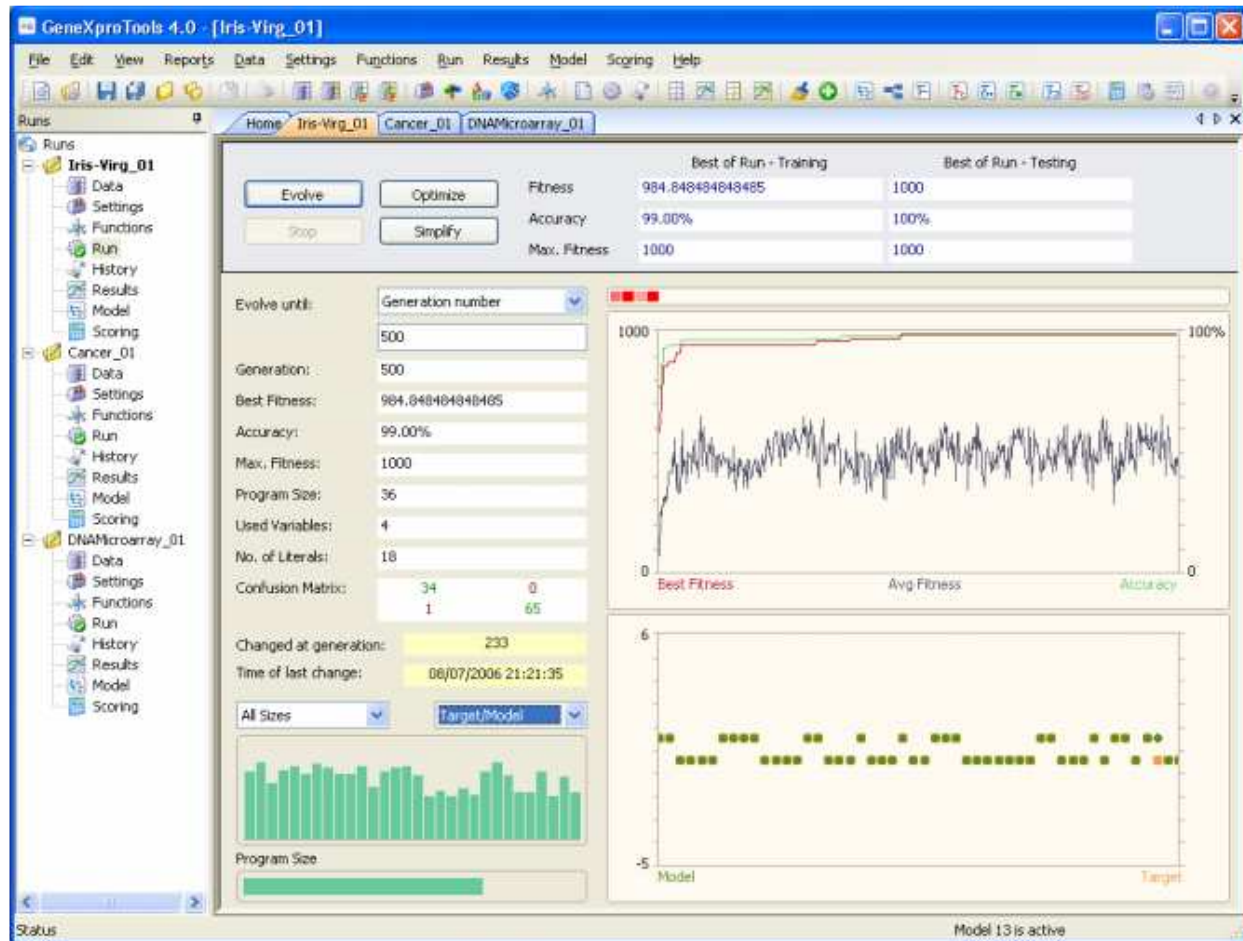
This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be activated any time during evolution by selecting All Sizes in the leftmost combo box in the bottom.

8. The All Fitnesses chart.

This chart shows the fitnesses of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be activated any time during evolution by selecting All Fitnesses in the leftmost combo box in the bottom.

9. The Program Size chart.

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model.



The evolutionary process can be stopped whenever you are satisfied with the results by pressing the Stop button or you can use one of the stop criteria of GeneXproTools and have GeneXproTools stop automatically after a certain condition had been met.

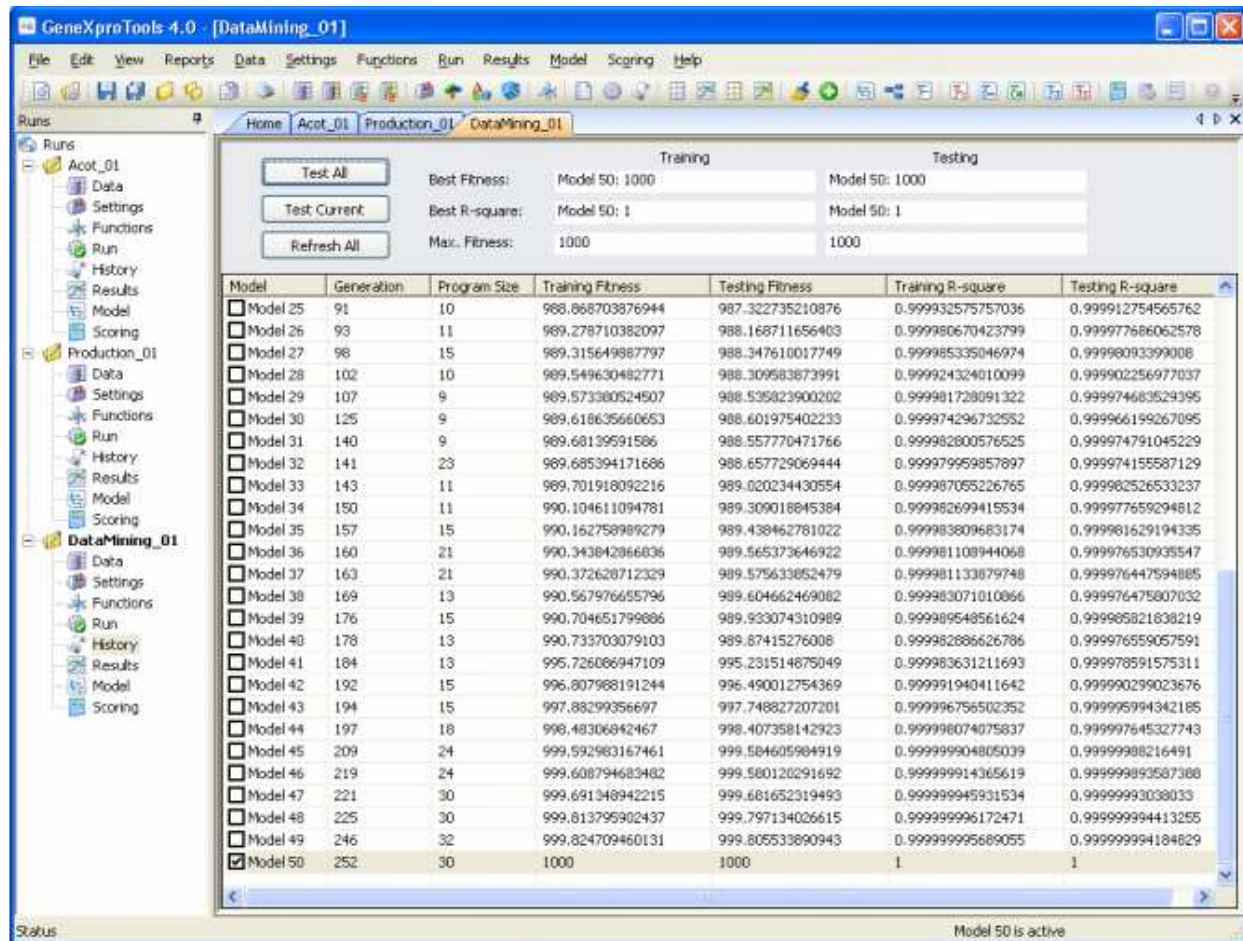
When the evolutionary process stops, the best-of-run model is ready either for analysis or scoring. And if you are still not happy with the results, you can still try to fine-tune the evolved model by pressing the Optimize button and repeat this process until you are completely satisfied with your model. In addition, you might also wish to try and simplify your model and for that you just have to press the Simplify button.



GeneXproTools Environment

History

In the **History Panel** you can dig in the evolutionary past of your model by analyzing all the best-of-generation solutions that were created during a particular run.



After a run, the **fitness** and **R-square/Accuracy** for the best-of-run model are shown both for the training and testing sets. For the remaining models only the fitness and R-square/Accuracy obtained during training are shown, but you can easily access the fitness and R-square/Accuracy for the testing set by clicking the Test All or Test Current buttons.

Each one of these intermediate models can be activated by clicking on it. And once a model is active, it can be used to make exactly the same kind of analysis and transformations one does with the best-of-run model, that is, look at its structure (size, shape, and elements), translate it into different programming languages, score a database, explore a different evolutionary path by using it as seed, check its statistics and how it generalizes in the Results Panel, twinkle with its structure in the Change Seed window, evaluate its performance in a different testing set, and so forth.

For your reference, in the History of the **Report Panel**, the active model is highlighted in yellow.

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	24	9	di(5), di(3), di(1)	847.282339126748	859.152192900566	0.960006472679246	0.973587464532183
$\begin{aligned} & \sqrt{X3} \cdot \sqrt{\text{Sqrt} + \text{Atan} \cdot X3 \cdot d0 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d0 \cdot d2 \cdot d0} \\ & + \\ & \sqrt{X2} \cdot \sqrt{\text{Sqrt} + \text{Cos} \cdot d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d0 \cdot d0} \\ & + \\ & X3 \cdot \text{Atan} \cdot d0 \cdot d0 \cdot \text{Cos} \cdot \text{Ln} \cdot \text{Sqrt} \cdot d0 \cdot d2 \cdot d2 \cdot d0 \cdot d1 \cdot d1 \cdot d0 \cdot d1 \end{aligned}$							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
12	25	9	di(4), di(4), di(1)	852.285419790807	864.046489990801	0.970678596225273	0.976046816260914
$\begin{aligned} & \sqrt{X3} \cdot \sqrt{\text{Sqrt} + \text{Atan} \cdot X3 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0} \\ & + \\ & \sqrt{X2} \cdot \sqrt{\text{Sqrt} + \text{Cos} \cdot d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d2 \cdot d0} \\ & + \\ & \text{Atan} \cdot X3 \cdot \text{Atan} \cdot d0 \cdot d0 \cdot \text{Cos} \cdot \text{Exp} \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d1 \cdot d2 \cdot d1 \end{aligned}$							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
23	25	9	di(3), di(4), di(2)	854.701706874743	864.588575207565	0.97254807136764	0.976472602618102
$\begin{aligned} & \sqrt{X3} \cdot \sqrt{\text{Sqrt} + \text{Atan} \cdot X3 \cdot d1 \cdot d2 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d2 \cdot d0} \\ & + \\ & \sqrt{X2} \cdot \sqrt{\text{Sqrt} + \text{Cos} \cdot d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d0 \cdot d0 \cdot d0} \\ & + \\ & \text{Atan} \cdot X3 \cdot \text{Atan} \cdot d0 \cdot d0 \cdot \text{Cos} \cdot \text{Exp} \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d2 \cdot d0 \cdot d0 \end{aligned}$							

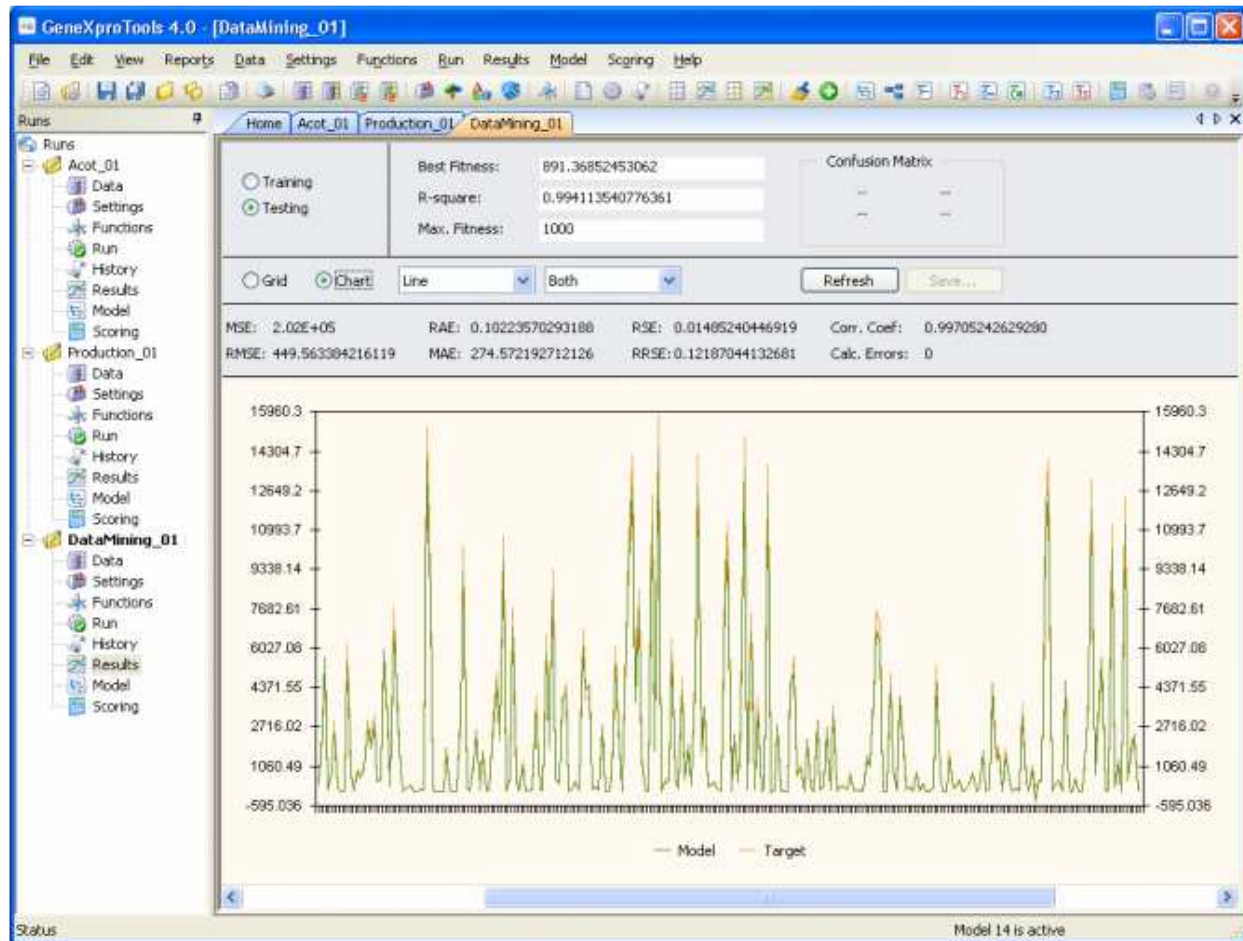


GeneXproTools Environment

Results

In the **Results Panel** of GeneXproTools 4.0 you can analyze and evaluate the performance of your model on the:

- [Training Set.](#)
- [Testing Set.](#)



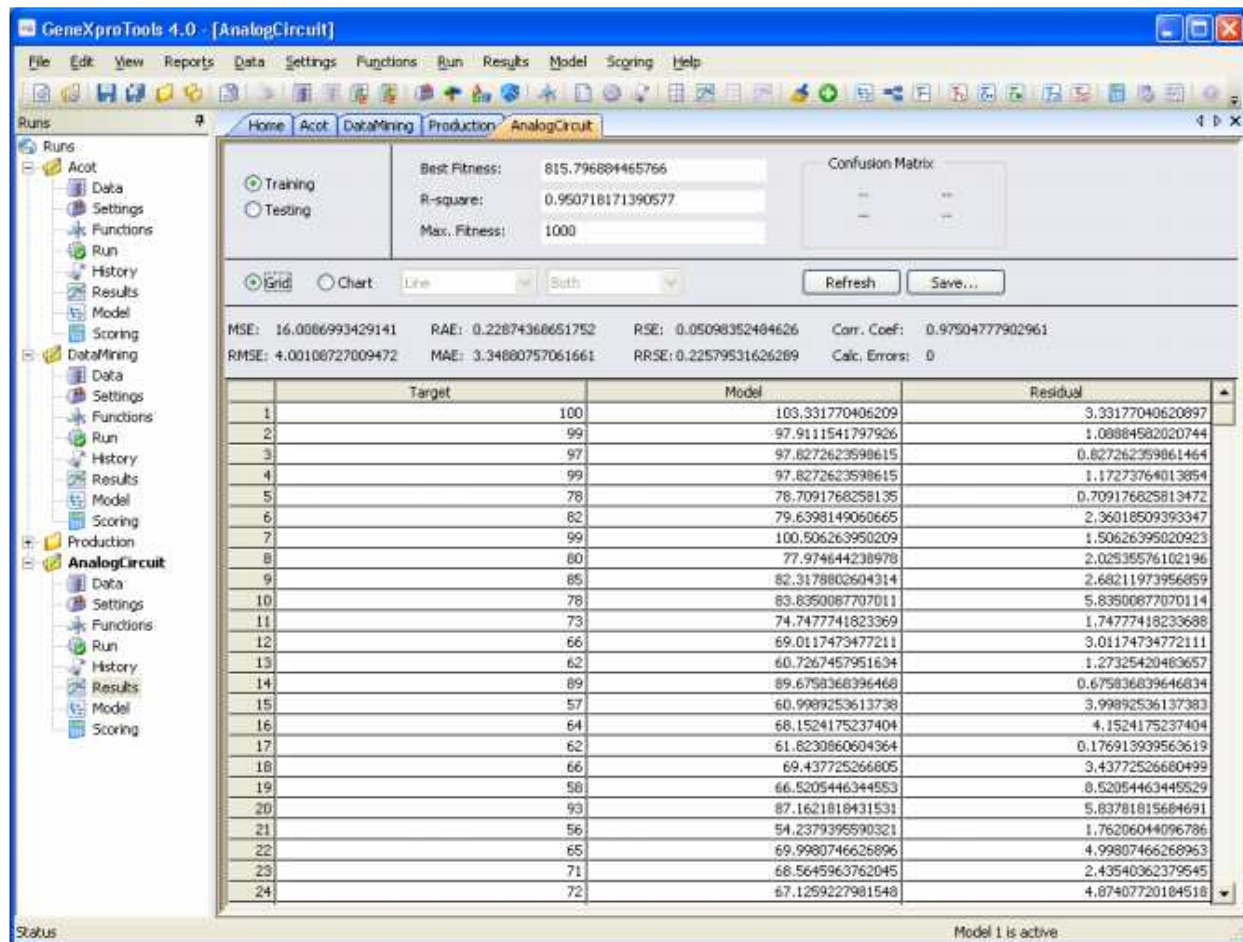
GeneXproTools Environment

Training Results

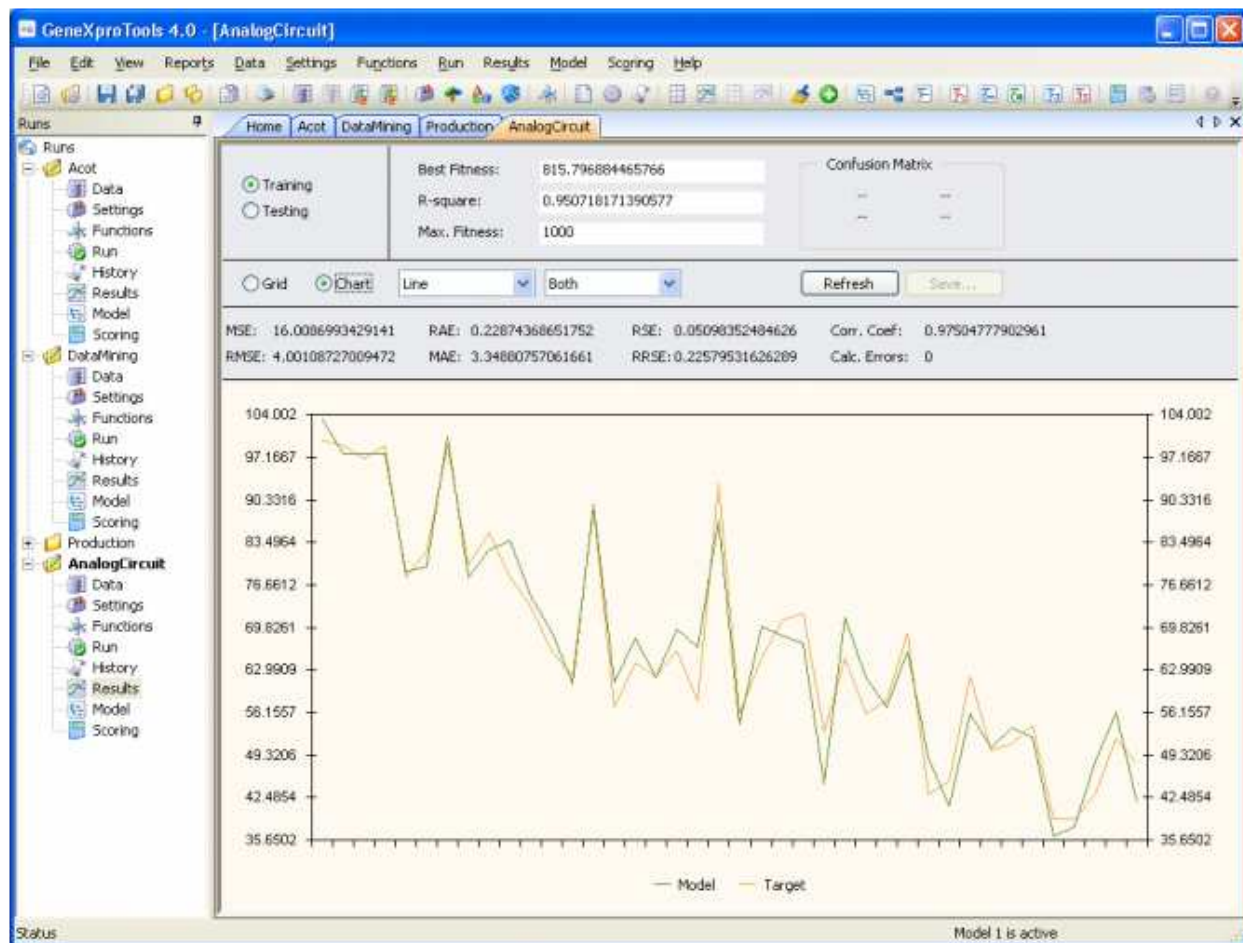
By selecting the Training option on the Results Panel, you can analyze the performance of your model in the training set.

GeneXproTools 4.0 offers two different ways of analyzing and comparing the output of your model with the actual or target values on the training set.

In the first, the target or actual values are listed in a spreadsheet side by side with the predicted or model values.



In the second, the target and predicted values are plotted in a Curve Fitting or Target/Model chart for easy visualization. By moving the scroll bar in the bottom, you can see the results for the entire dataset.

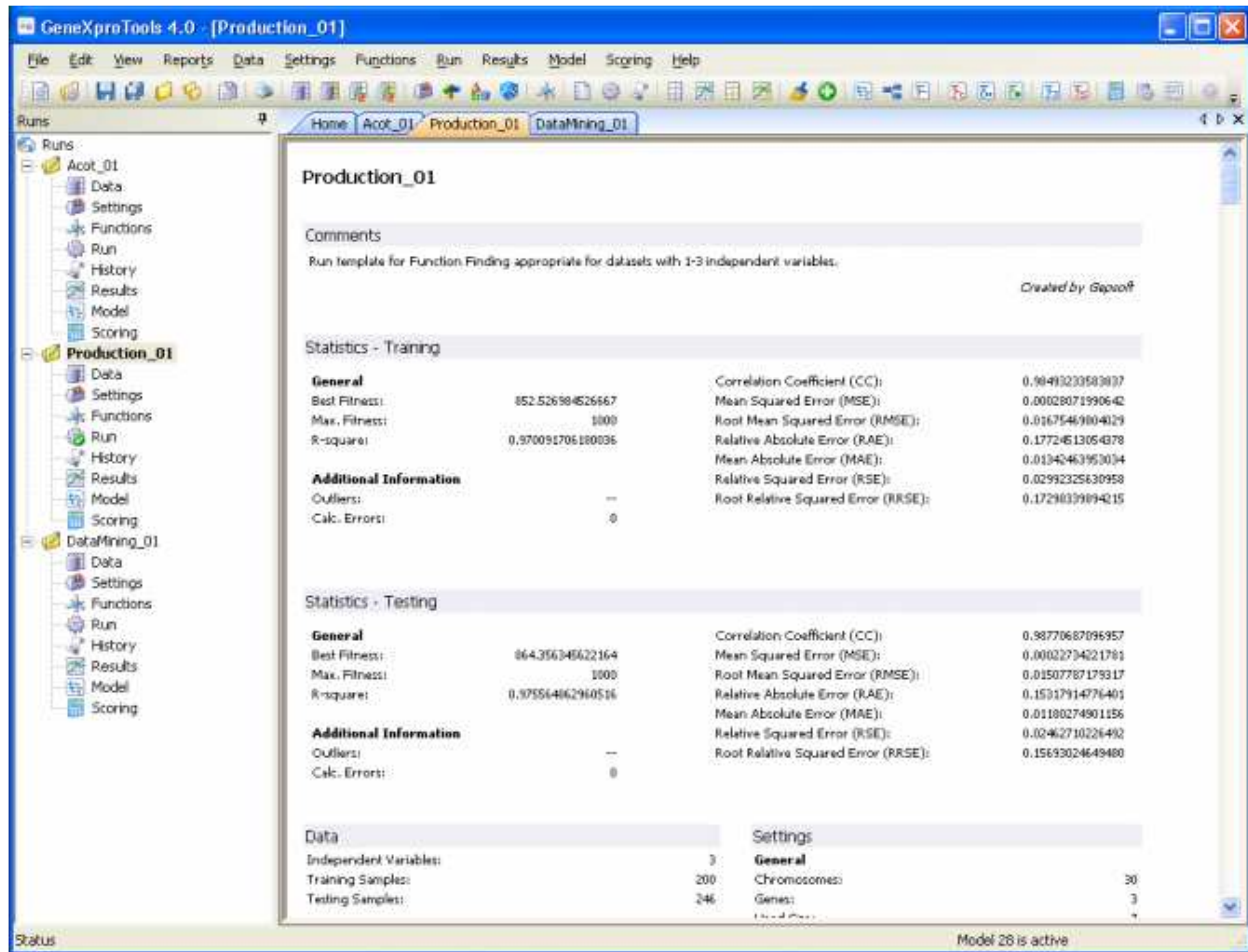


The charts of GeneXproTools 4.0 come in various flavors, and you can choose Line, Area, 3D Line, and 3D Area for analyzing your model. You can also choose to plot only the output of your model or the target values.

The 3D charts can be moved around for a different perspective by pressing Ctrl and dragging it with the mouse.

Additionally, GeneXproTools 4.0 performs on the fly an extremely complete [statistical analysis](#) of your model for the training set, including mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, correlation coefficient, and R-square for Function Finding and Time Series Prediction; and classification error, classification accuracy, confusion matrix (true positives, true negatives, false positives, and false negatives), sensitivity, specificity, positive predictive value, negative predictive value, correlation coefficient, and R-square for Classification and Logic Synthesis.

The output of all these statistical functions is conveniently shown in the Results Panel for a quick evaluation during the modeling process and are also shown, together with other indicators, in the [Report Panel](#) for future reference.



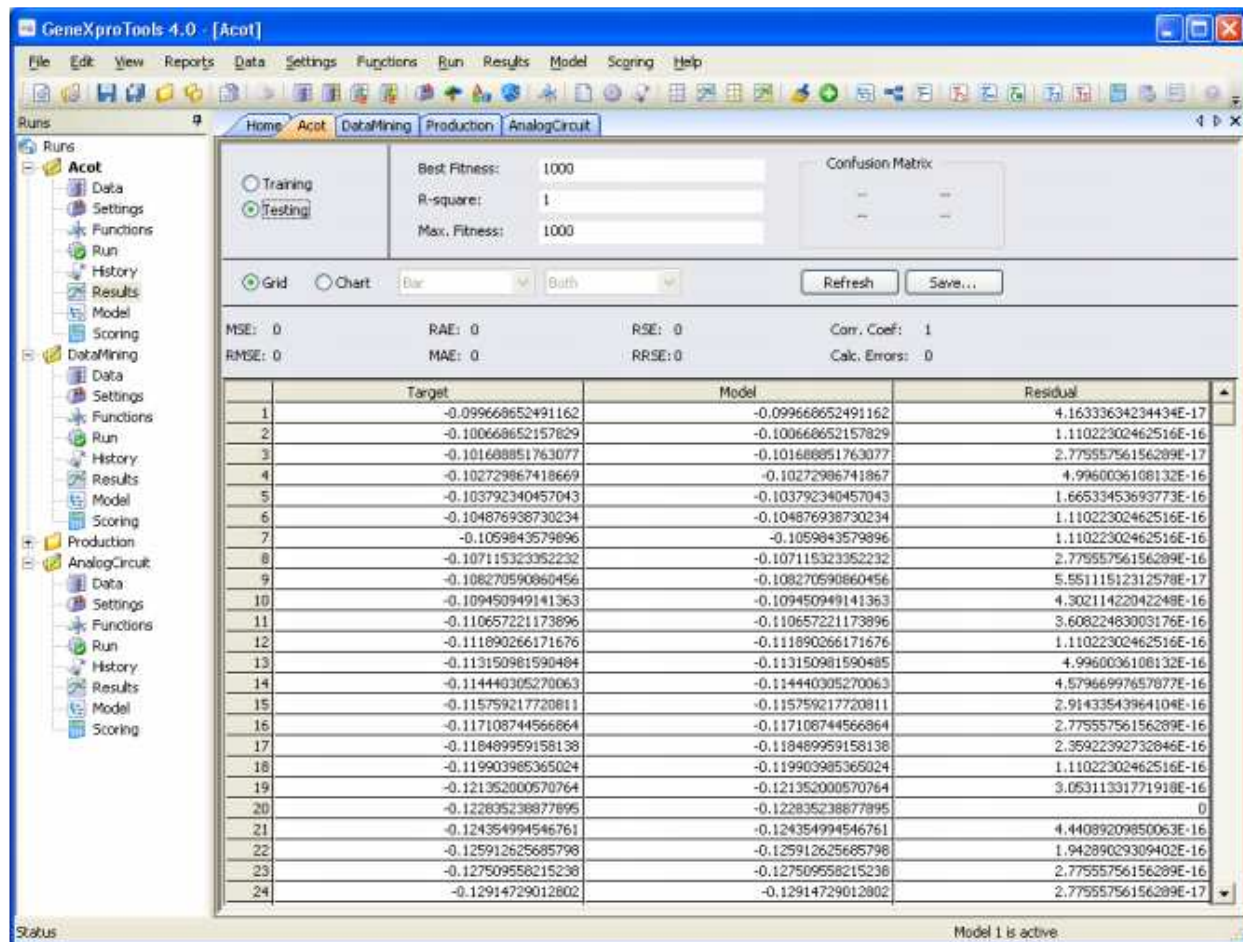
GeneXproTools Environment

Testing Results

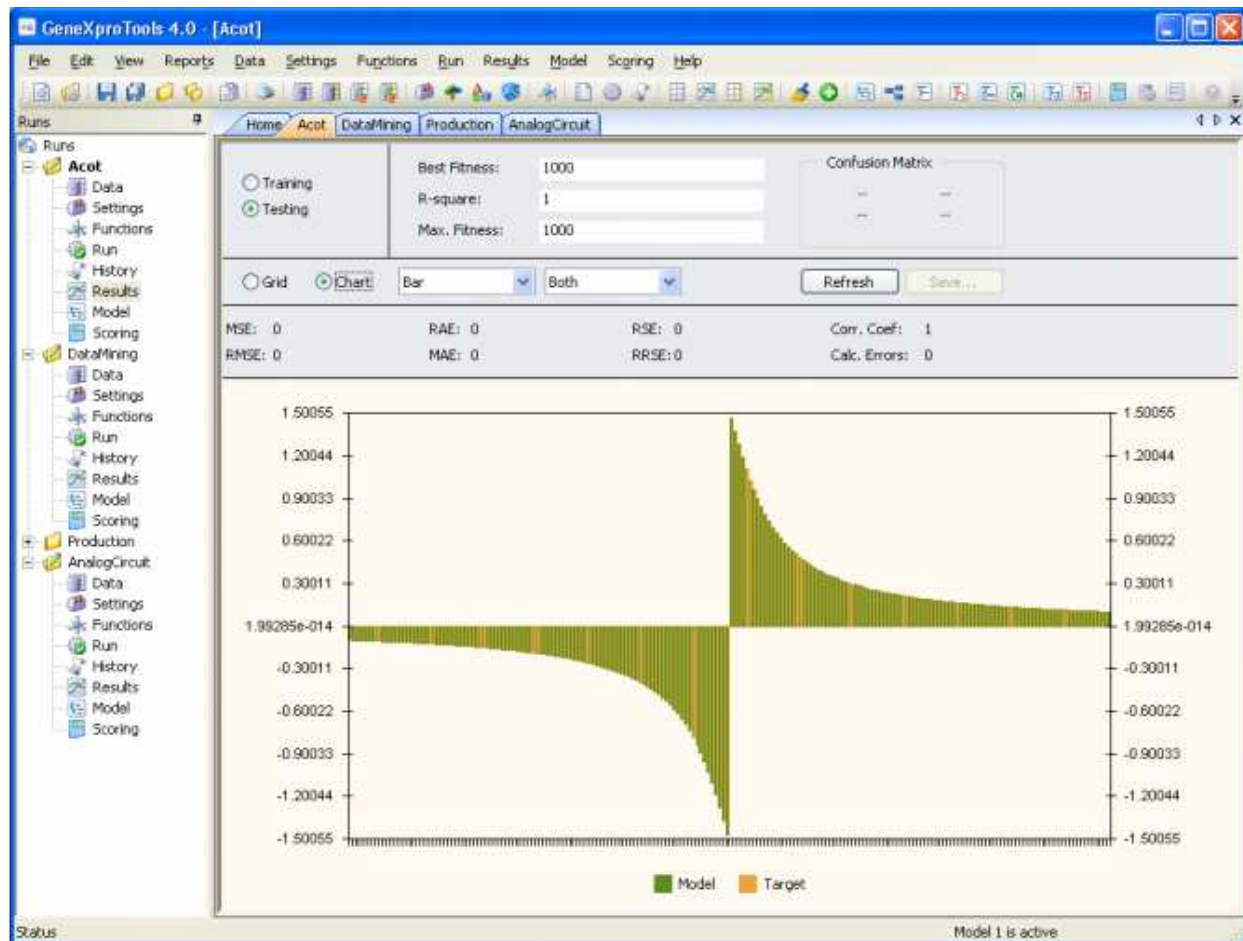
By selecting the Testing option on the Results Panel, you can analyze the predictive accuracy of your model using the testing set.

GeneXproTools 4.0 offers two different ways of analyzing and comparing the output of your model with the actual or target values on testing set.

In the first, the target or actual values are listed in a spreadsheet side by side with the predicted or model values.



In the second, the target and predicted values are plotted in a Curve Fitting or Target/Model chart for easy visualization. By moving the scroll bar in the bottom, you can see the results for the entire dataset.

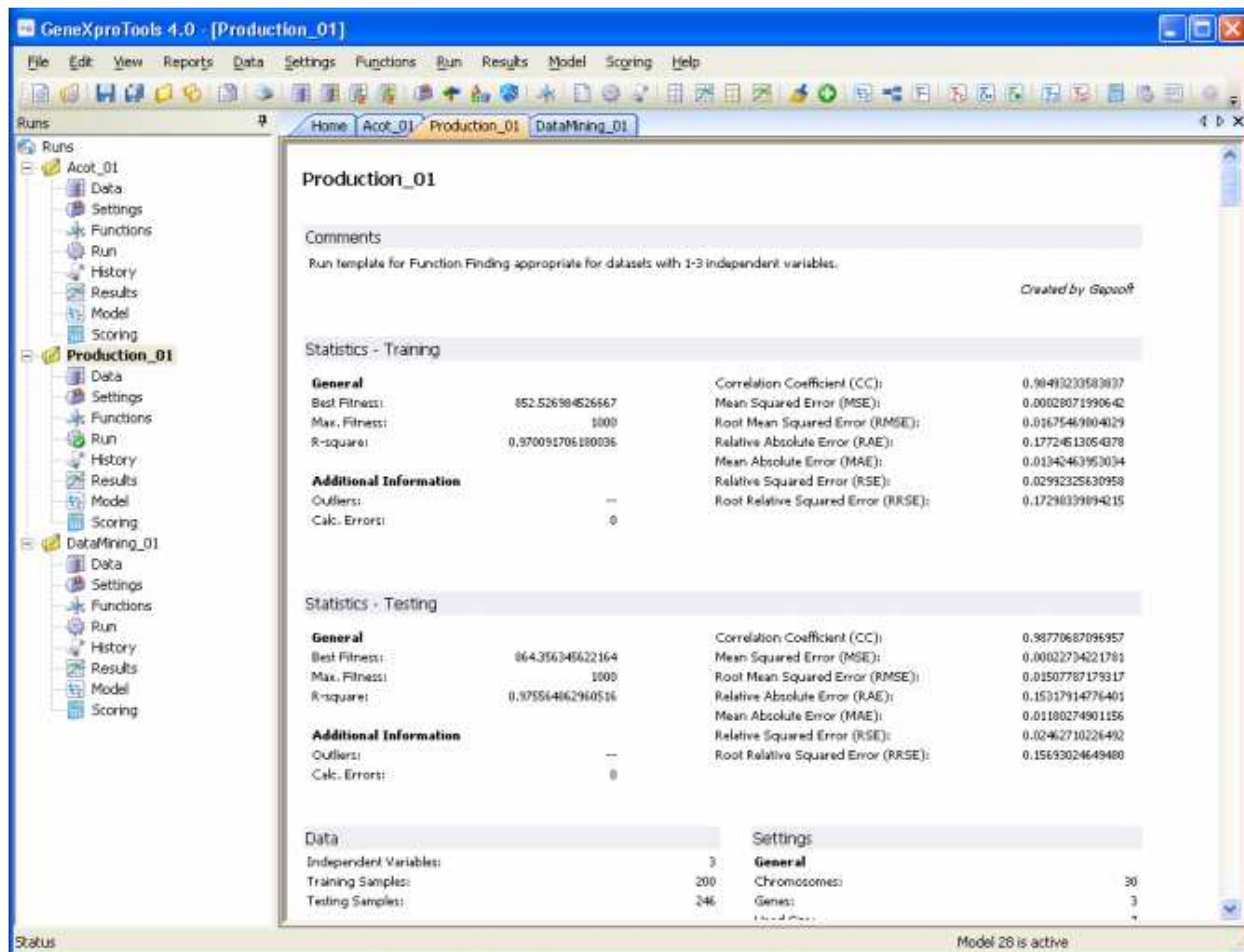


The charts of GeneXproTools 4.0 come in various flavors, and you can choose Line, Area, 3D Line, and 3D Area for analyzing your model. You can also choose to plot only the output of your model or the target values.

The 3D charts can be moved around for a different perspective by pressing Ctrl and dragging it with the mouse.

Additionally, GeneXproTools 4.0 performs an extremely complete [statistical analysis](#) of your model for the testing set, including mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, correlation coefficient, and R-square for Function Finding and Time Series Prediction; and classification error, classification accuracy, confusion matrix (true positives, true negatives, false positives, and false negatives), sensitivity, specificity, positive predictive value, negative predictive value, correlation coefficient, and R-square for Classification and Logic Synthesis.

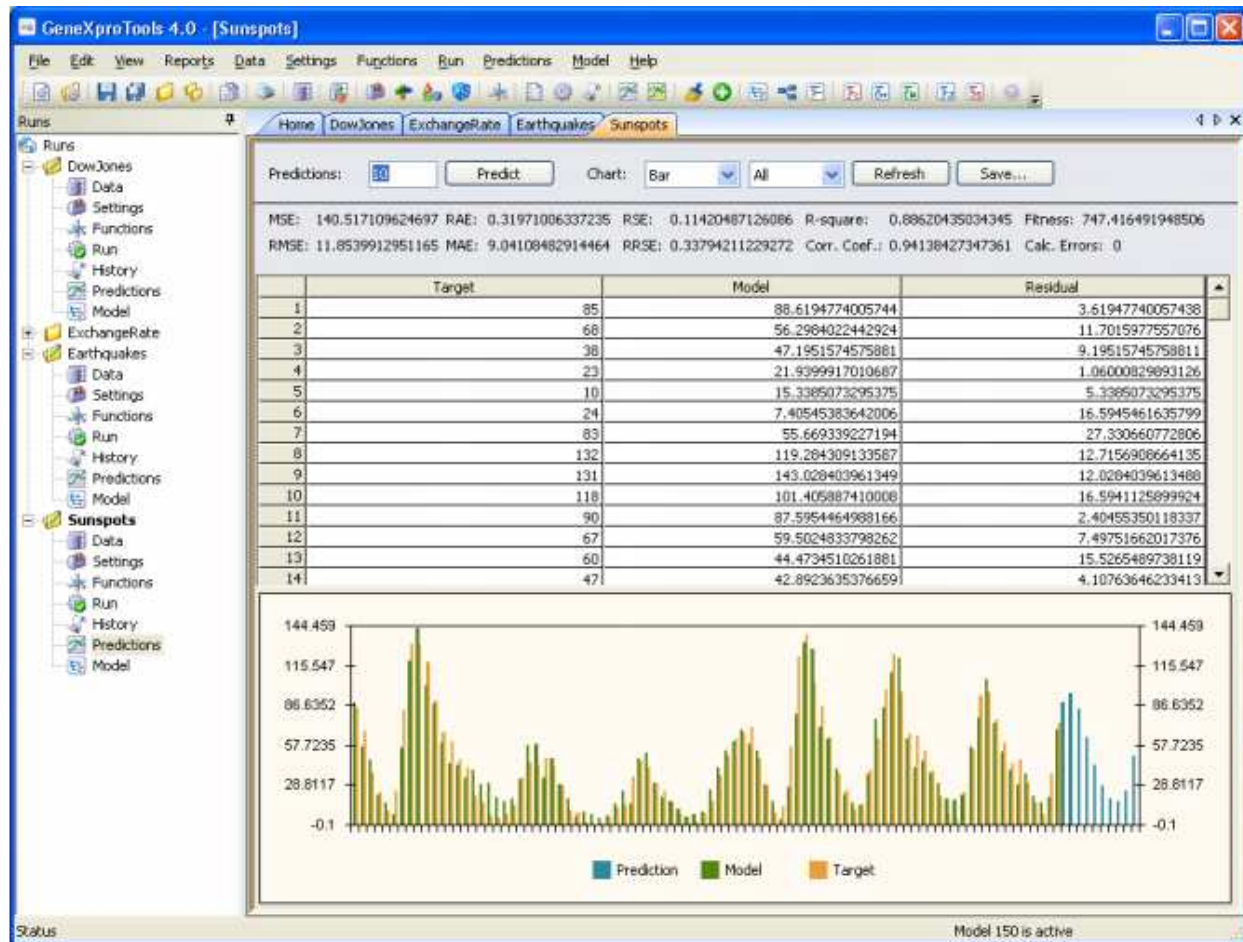
The output of all these statistical functions is conveniently shown in the Results Panel for a quick evaluation during the modeling process and are also shown, together with other indicators, in the [Report Panel](#) for future reference.



GeneXproTools Environment

Predictions

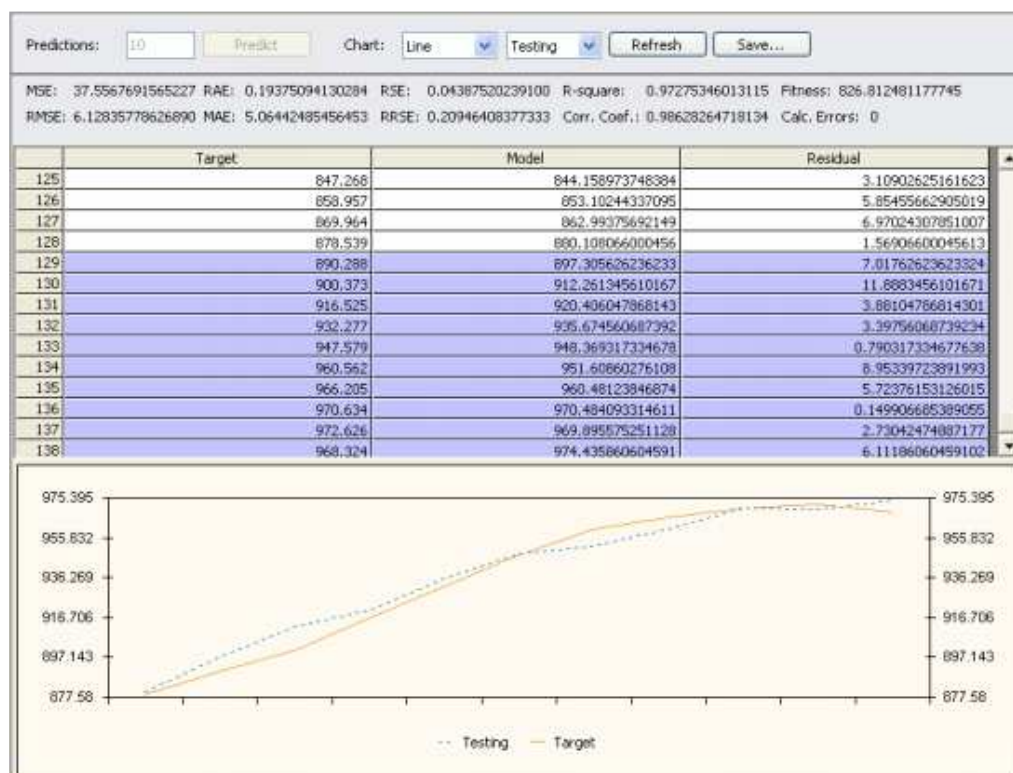
The algorithms of GeneXproTools 4.0 used for Time Series Prediction allow not only the evolution of models but also the use of these models to [make predictions](#). And you make predictions in the **Predictions Panel**.



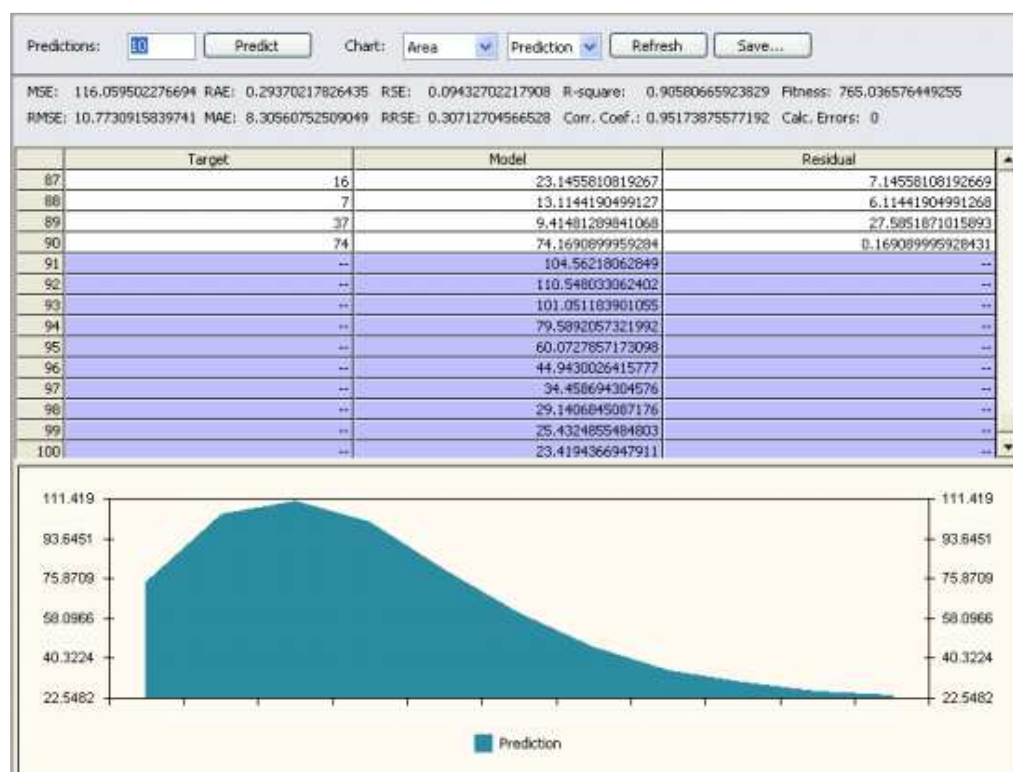
GeneXproTools 4.0 allows you to make two kinds of predictions: one for testing past known events and another for making predictions about the future. In both cases, though, predictions are made recursively, by evaluating the forecast at $t+1$, then using it to forecast $t+2$, and so on.

You must choose either one of these methods while loading your time series data, as this imposes some constraints on the restructuring of the time series for training. Namely, n testing records (the n last ones) are saved for testing and sometimes a small number of records from the top must be deleted. However, you can also change these parameters later in the [Settings Panel](#).

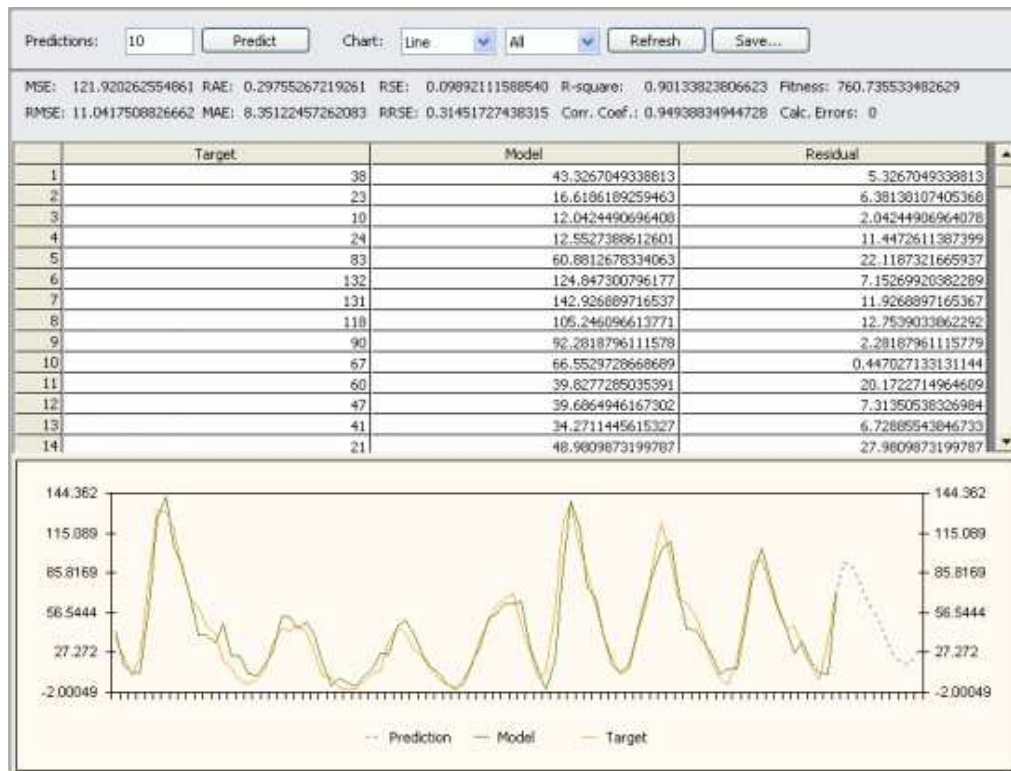
Thus, the first type of prediction can be used for research or pre-evaluation purposes, as it allows you to test the forecasting capabilities of your model on a set of test observations.



The second type is used obviously to predict unknown behavior, and GeneXproTools 4.0 allows you to venture into the future as far as you see fit, by setting the number of predictions you want to make and then click the Predict button.



Furthermore, on the Predictions Panel, you can analyze the performance of your model in the training set and compare its output with the actual values using both a spreadsheet and a Curve Fitting chart. By moving the scroll bar on the right you can visualize the entire data on the chart.

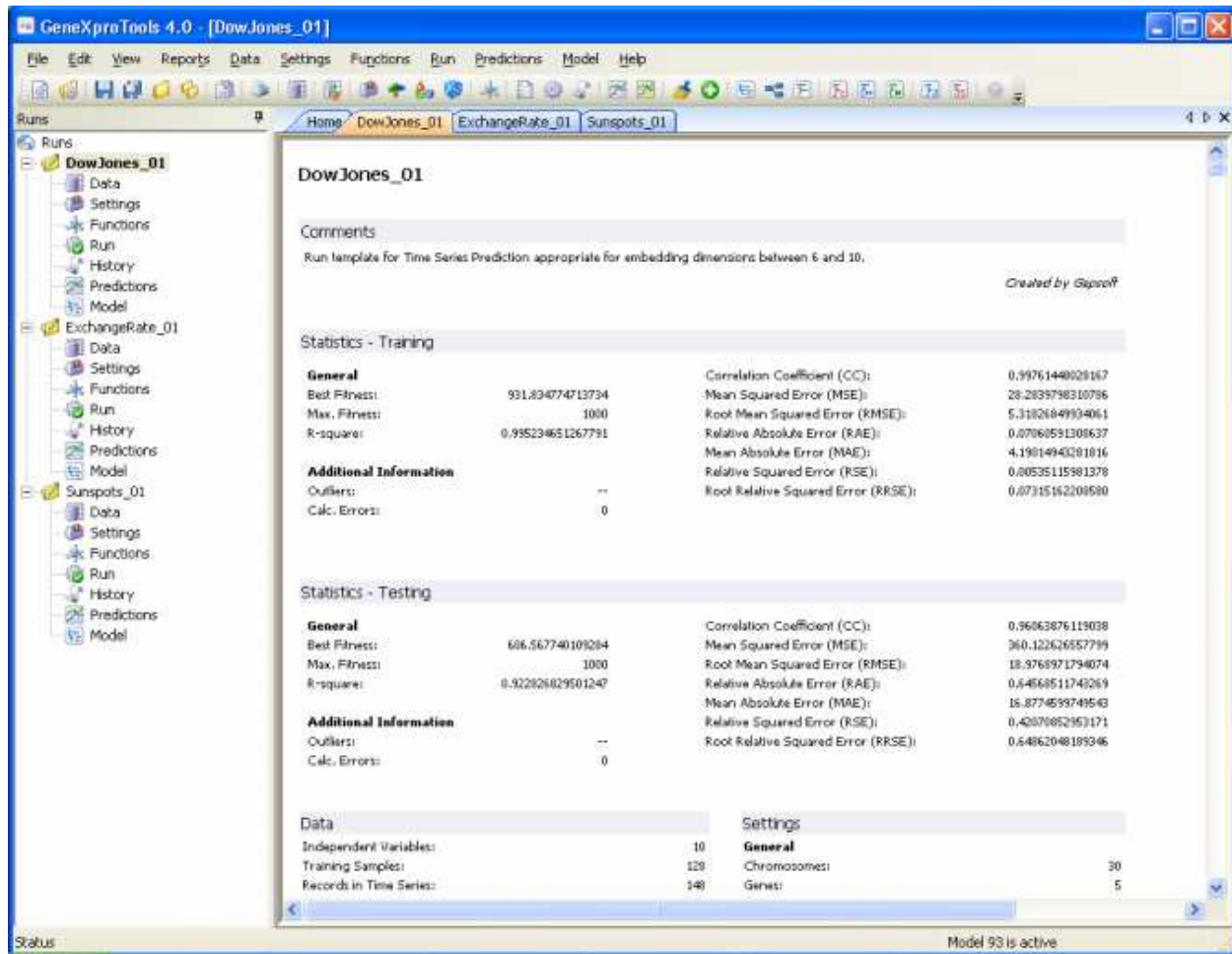


The charts of GeneXproTools 4.0 come in various flavors, and you can choose Line, Area, 3D Line, and 3D Area for analyzing your model. You can also choose to plot only the output of your model, or the target values, or the predictions (either testing or real ones, depending on the option previously chosen).

The 3D charts can be moved around for a different perspective by pressing Ctrl and dragging it with the mouse.

Additionally, GeneXproTools 4.0 performs an extremely complete [statistical analysis](#) of your model both for the training set and the testing predictions, including mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, correlation coefficient, and R-square.

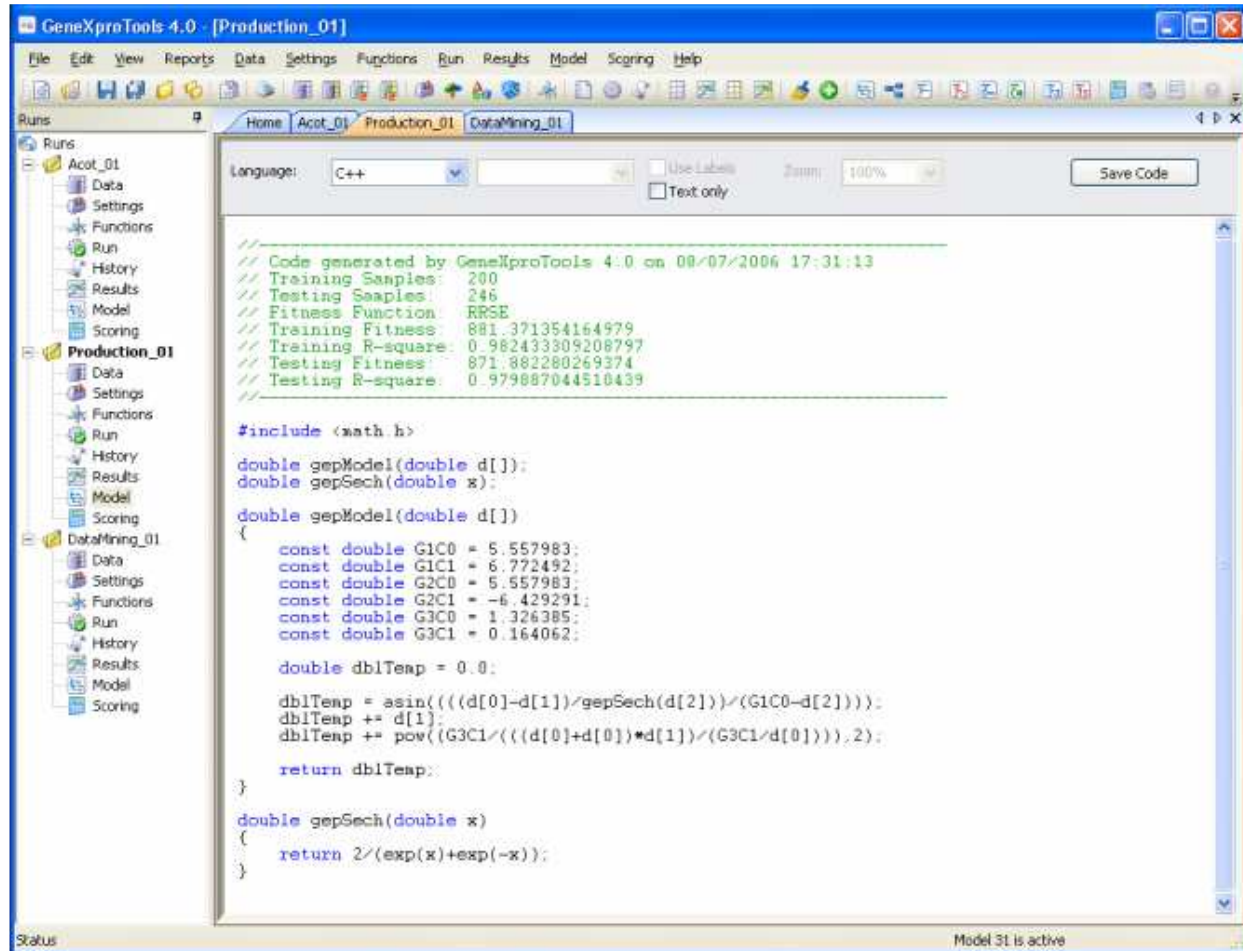
The output of all these statistical functions is conveniently shown in the Predictions Panel for a quick evaluation during the modeling process and are also shown in the [Report Panel](#) for future reference.



GeneXproTools Environment

Model

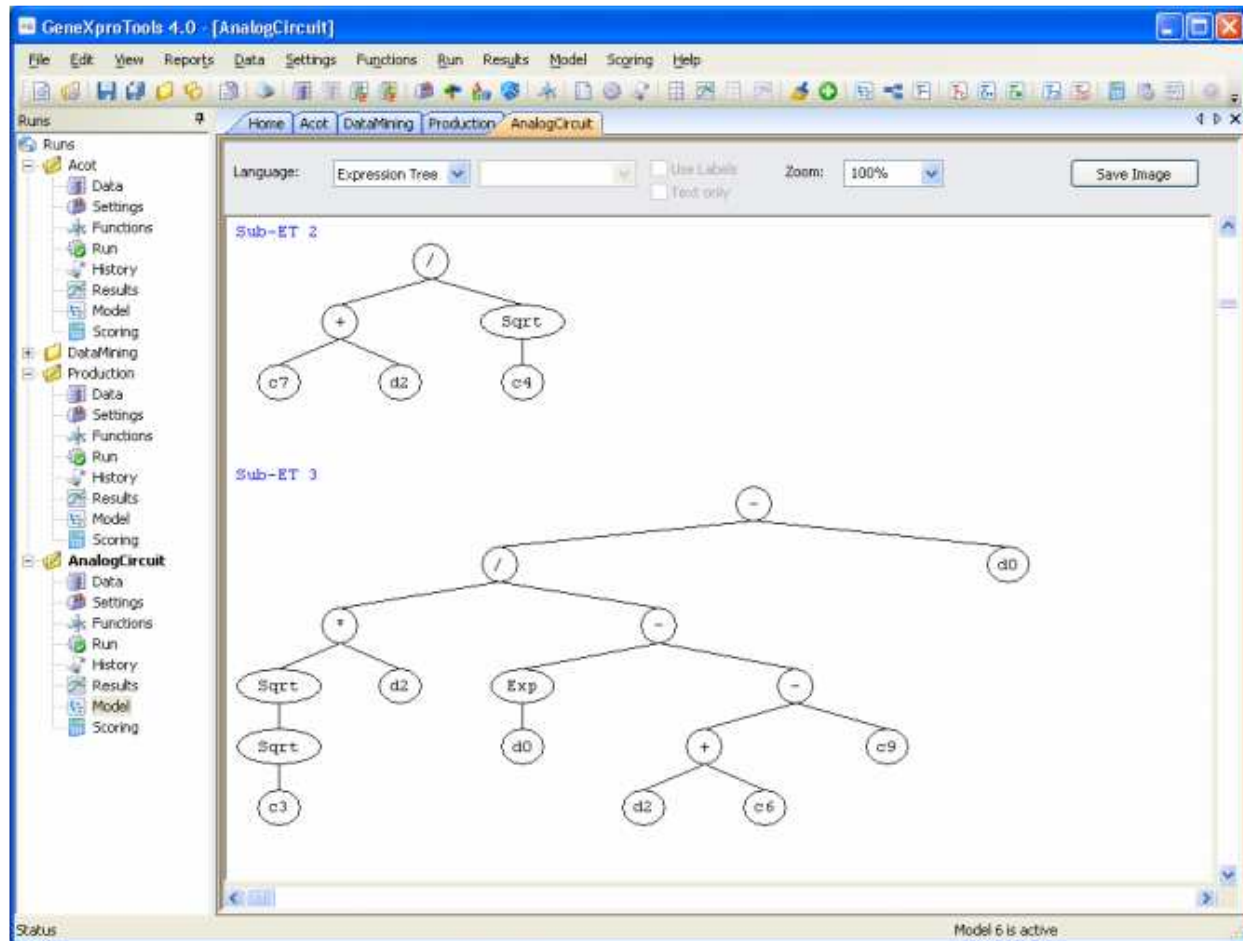
In the **Model Panel** you can visualize the code of your model by choosing the language or representation into which you want it translated.



GeneXproTools 4.0 comes equipped with 16 [built-in grammars](#) (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL) which allow the automatic translation of your models into the language of your choice. The code can then be saved and used in other applications.

If you happened to prefer another programming language into which to translate your models, GeneXproTools 4.0 allows you to do that through the use of [User Defined Grammars](#). Then, using your grammar, GeneXproTools 4.0 automatically translates your models into that particular language.

Extremely handy and popular is the diagram or [tree representation](#) of GeneXproTools models as it allows the quick and complete understanding of their mathematical/logical intricacies. These expression trees can be copied and pasted into other applications or saved as bit map files.

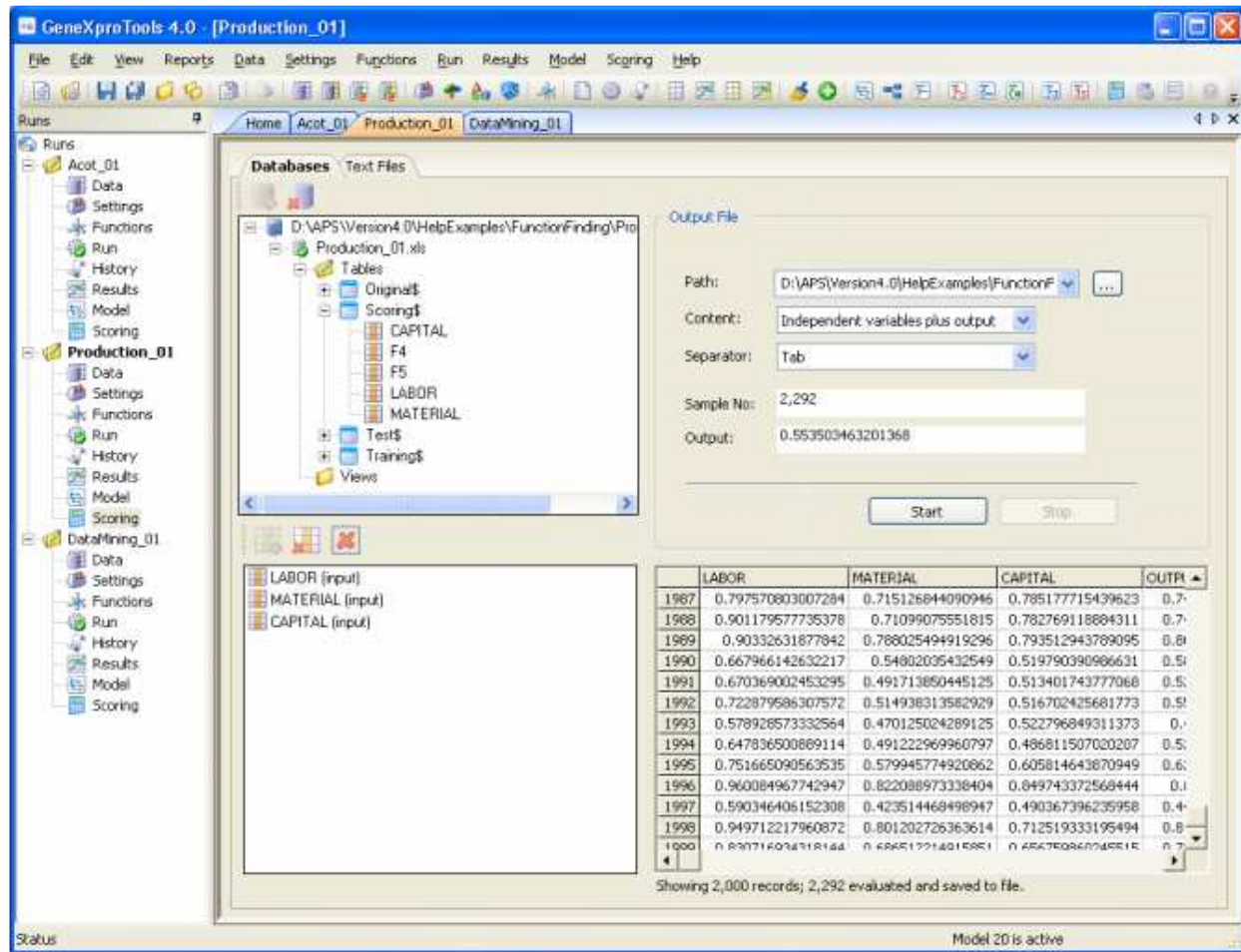


GeneXproTools Environment

Scoring

In the **Scoring Panel** of GeneXproTools 4.0 you can immediately apply the evolved models to score:

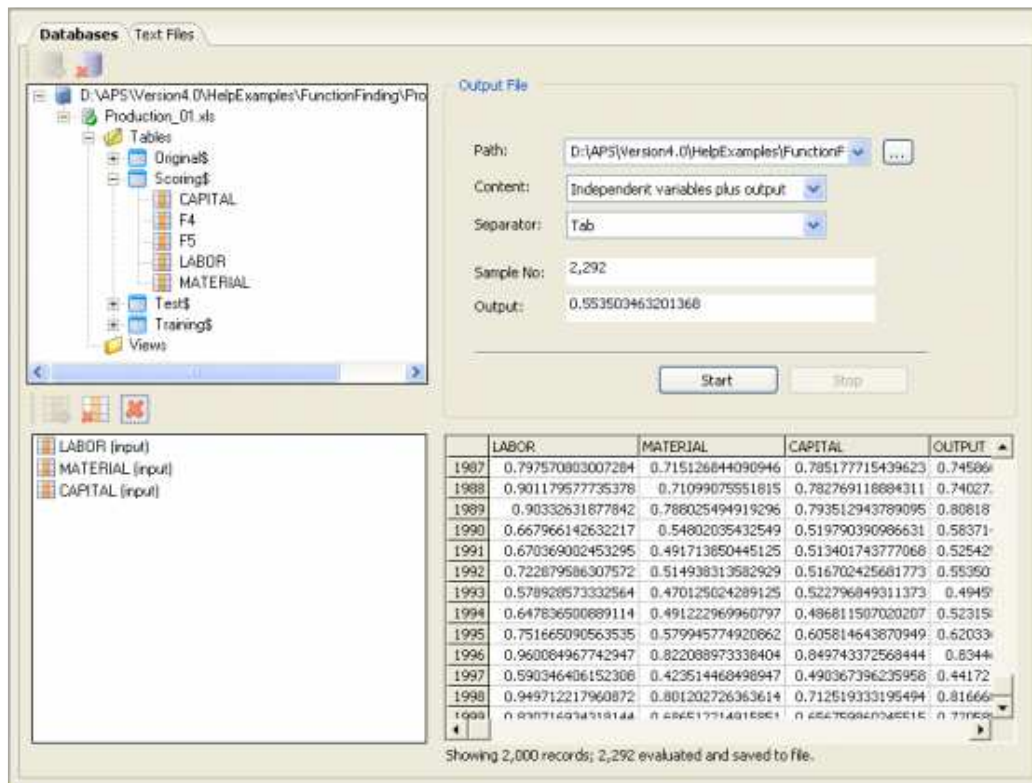
- [Databases or Excel Files.](#)
- [Text Files.](#)



GeneXproTools Environment

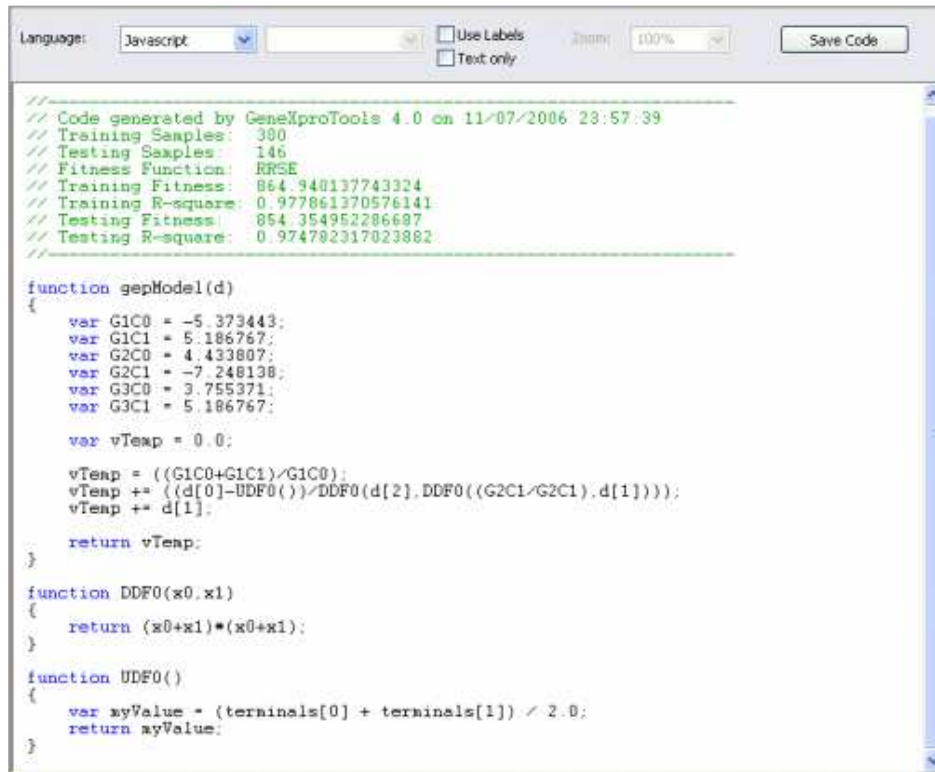
Databases

To score a database with your model, in the Scoring Panel select the Databases Tab.



Once a model has been created it can be used hundreds or thousands of times to predict unknown behavior. The scoring engine of GeneXproTools allows you to do that without leaving the GeneXproTools modeling environment where models can be not only created but also refreshed and kept up to date.

The scoring engine of GeneXproTools 4.0 uses the [JavaScript](#) code generated for your model to perform the computations since it already contains the code for the [UDFs](#) and [DDFs](#).



The screenshot shows the Scoring Panel of GeneXproTools. At the top, there is a 'Language' dropdown set to 'Javascript', a 'Zoom' slider at 100%, and checkboxes for 'Use Labels' and 'Text only'. A 'Save Code' button is on the right. The main area displays green text with model statistics and JavaScript code. The statistics include Training Samples (300), Testing Samples (146), Fitness Function (RRSE), Training Fitness (864.940137743324), Training R-square (0.977861370576141), Testing Fitness (864.354952286687), and Testing R-square (0.974782317023882). The code defines a 'gepModel(d)' function with several variables (G1C0, G1C1, G2C0, G2C1, G3C0, G3C1) and a loop to calculate 'vTemp'. It also defines 'DDF0(x0,x1)' and 'UDF0()' functions.

```
// Code generated by GeneXproTools 4.0 on 11/07/2006 23:57:39
// Training Samples: 300
// Testing Samples: 146
// Fitness Function: RRSE
// Training Fitness: 864.940137743324
// Training R-square: 0.977861370576141
// Testing Fitness: 864.354952286687
// Testing R-square: 0.974782317023882
//

function gepModel(d)
{
    var G1C0 = -5.373443;
    var G1C1 = 5.186767;
    var G2C0 = 4.433807;
    var G2C1 = -7.248138;
    var G3C0 = 3.755371;
    var G3C1 = 5.186767;

    var vTemp = 0.0;

    vTemp = ((G1C0+G1C1)/G1C0);
    vTemp += ((d[0]-UDF0())/DDF0(d[2],DDF0((G2C1/G2C1),d[1])));
    vTemp += d[1];

    return vTemp;
}

function DDF0(x0,x1)
{
    return (x0+x1)*(x0+x1);
}

function UDF0()
{
    var myValue = (terminals[0] + terminals[1]) / 2.0;
    return myValue;
}
```

The output is saved to a file which contains the predictions of your model for all the scores in the database. For small datasets (up to 16 variables and 2000 samples) GeneXproTools also shows the scoring results in the Scoring Panel.



GeneXproTools Environment

Text Files

To score a database with your model, in the Scoring Panel select the Text Files Tab.

Databases **Test Files**

Source File

File Path: ...

Data Matrix Format:

Separator: ☒ First row contains labels

Output File

Path: ...

Content:

Separator:

Sample No:

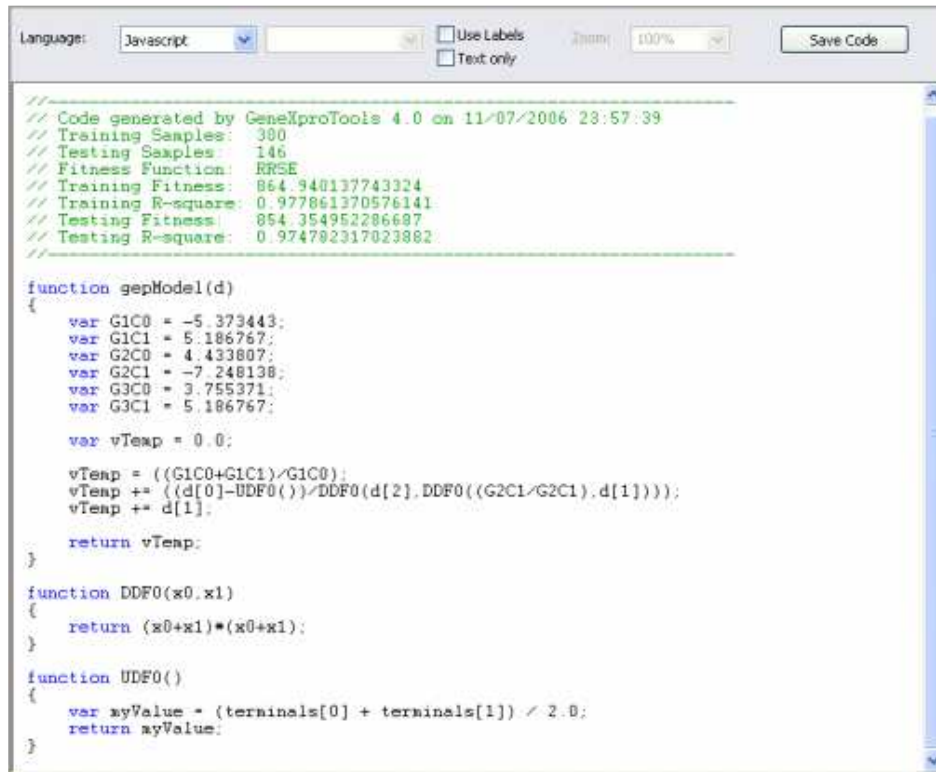
Output:

	LABOR	MATERIAL	CAPITAL	OUTPUT
1987	0.797570003007284	0.715126844090946	0.785177715439623	0.745860124335517
1988	0.901179577735378	0.71099075551815	0.782769118884311	0.740272671233412
1989	0.90332631877842	0.788025494919296	0.793512943789095	0.808187245837281
1990	0.667966142632217	0.54802035432549	0.519790390986631	0.583714519916069
1991	0.670369002453295	0.491713850445125	0.513401743777068	0.525425513025935
1992	0.722879586307572	0.514938313582929	0.516702425681773	0.553503463201368
1993	0.578928573332564	0.470125024289125	0.522796849311373	0.49455977578346
1994	0.647836500889114	0.491222969960797	0.486811507020207	0.523158403098033
1995	0.751665090563535	0.579945774920862	0.605814643870949	0.620336343118778
1996	0.960084967742947	0.822088973338404	0.849743372568444	0.83446603477471
1997	0.590346406152308	0.423514468498947	0.490367396235958	0.441721196745279
1998	0.949712217960872	0.801202726363614	0.712519333195494	0.816660616029998
1999	0.830716934318144	0.686512214915851	0.656759860245515	0.720585240805507
2000	0.85486473570545	0.674110700807944	0.678908982754438	0.708989324531767

Showing 2,000 records; 2,292 evaluated and saved to file.

Once a model has been created it can be used hundreds or thousands of times to predict unknown behavior. The scoring engine of GeneXproTools allows you to do that without leaving the GeneXproTools modeling environment where models can be not only created but also refreshed and kept up to date.

The scoring engine of GeneXproTools 4.0 uses the [JavaScript](#) code of your model to perform the computations since it already contains the code for the [UDFs](#) and [DDFs](#).



```

Language: Javascript
☐ Use Labels
☐ Text only
Zoom: 100%
Save Code

// Code generated by GeneXproTools 4.0 on 11/07/2006 23:57:39
// Training Samples: 300
// Testing Samples: 146
// Fitness Function: RRSE
// Training Fitness: 864.940137743324
// Training R-square: 0.977861370576141
// Testing Fitness: 864.354952286687
// Testing R-square: 0.974782317023882

function gepModel(d)
{
    var G1C0 = -5.373443;
    var G1C1 = 5.186767;
    var G2C0 = 4.433807;
    var G2C1 = -7.248138;
    var G3C0 = 3.755371;
    var G3C1 = 5.186767;

    var vTemp = 0.0;

    vTemp = ((G1C0+G1C1)/G1C0);
    vTemp += ((d[0]-UDF0())/DDF0(d[2],DDF0((G2C1/G2C1),d[1])));
    vTemp += d[1];

    return vTemp;
}

function DDF0(x0,x1)
{
    return (x0+x1)*(x0+x1);
}

function UDF0()
{
    var myValue = (terminals[0] + terminals[1]) / 2.0;
    return myValue;
}

```

The output is saved to a file which contains the predictions of your model for all the scores in the text file. For small datasets (up to 16 variables and 2000 samples) GeneXproTools also shows the scoring results in the Scoring Panel.

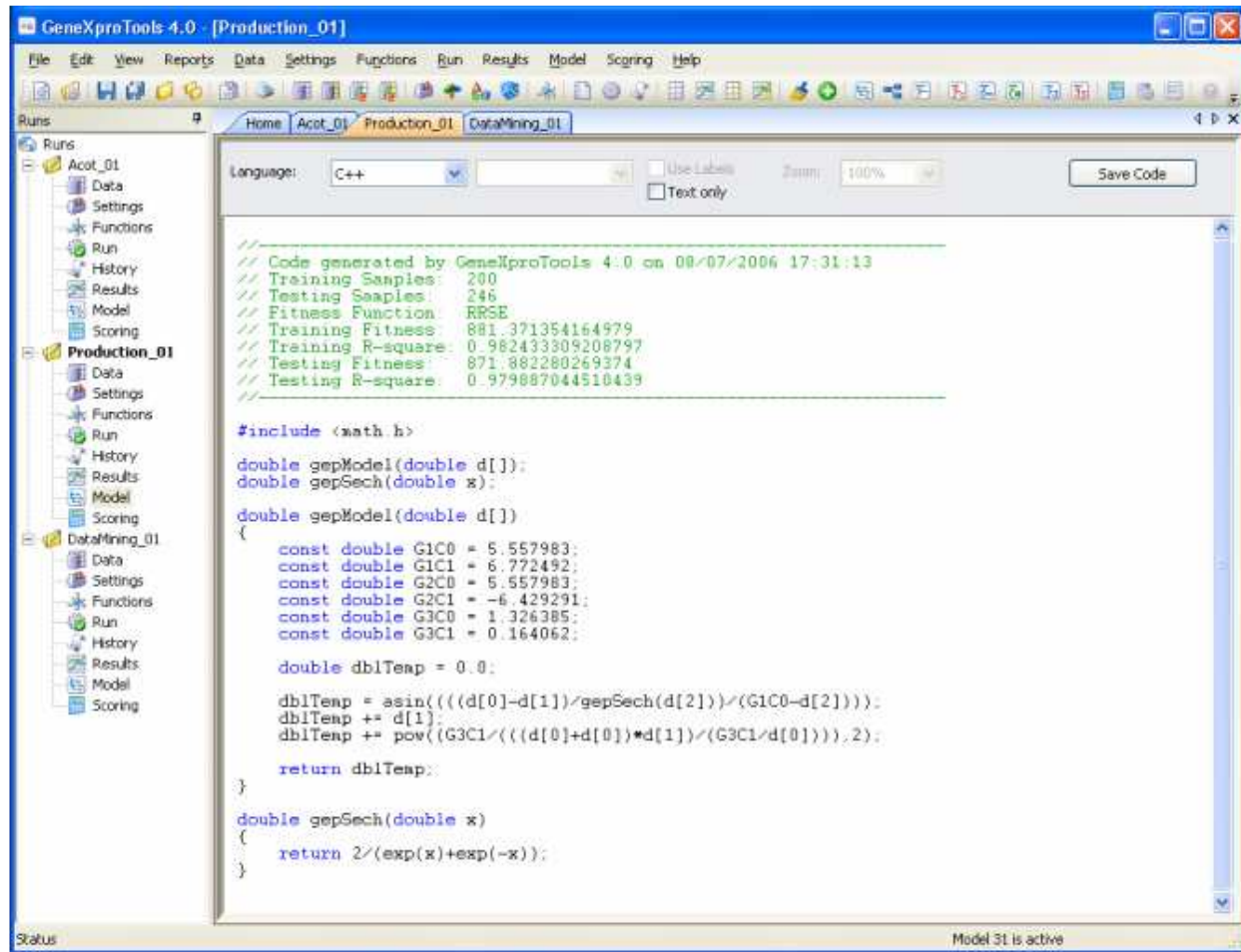


GeneXproTools Environment

Creating Models with GeneXproTools

The creation of good models involves the following steps:

- Preparing and analyzing your data carefully.
- Choosing the function set.
- Choosing the right chromosome architecture.
- Choosing the ideal population size.
- Choosing the right degree of genetic modification.
- Choosing a good fitness function.
- Creating your model.
- Evaluating your model.
- Creating better models from seed models.
- Simplifying your model.



GeneXproTools Environment

Preparing and Analyzing Data

In data mining, be it performed by learning algorithms or conventional statistical methods, it really pays to take a good look at your data before embarking on a complex, usually time consuming modeling process. It's true that evolutionary algorithms are particularly well equipped to deal with noisy data, but the better the data you feed them the better the models they produce.

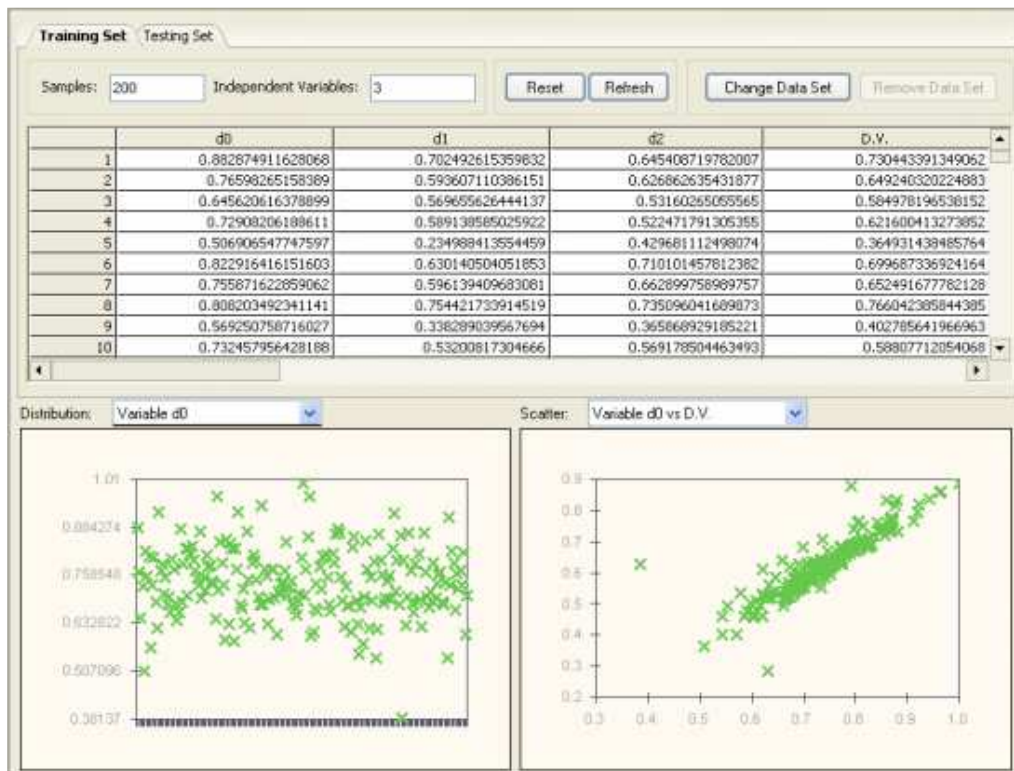
GeneXproTools helps you find missing and invalid (usually nominal) values in your data sets and prompts you to fix them before they are used for modeling. But the preparation of a well balanced data set should be done before loading the data into GeneXproTools, and we recommend you to particularly take care of the following:

1. Avoid using duplicated samples for they can bias the modeling process considerably.
 2. Choose a well balanced data set.
 3. Choose a reasonable number of samples for training.
- An excessively large data set will slow the modeling process unnecessarily. If you have access to

huge data bases it's good practice to use the surplus samples for testing instead. A good rule of thumb consists of using about 8-10 samples for each independent variable in your training data.

Check your data sets carefully for inaccurate values. Typographical or measurement errors generally cause outliers that can be detected by graphing one variable at a time. And you can do that easily in the GeneXproTools environment.

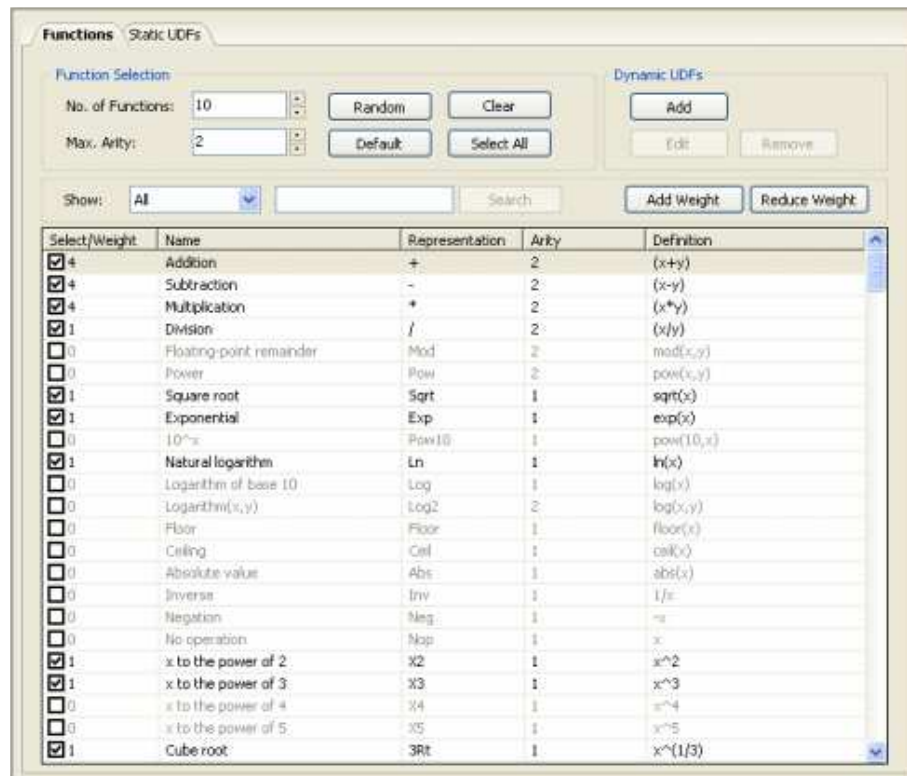
Data cleaning is a time-consuming and labor-intensive procedure, but one that is absolutely necessary for successful modeling. The [graphical visualization tools](#) of GeneXproTools 4.0 make it easy to identify outliers, which may well represent errors in the data files. After loading your data into GeneXproTools, in the [Data Panel](#) you can visualize the distribution of values for each variable and also plot each independent variable against the dependent variable, or each attribute against the class value for classification problems.



GeneXproTools Environment

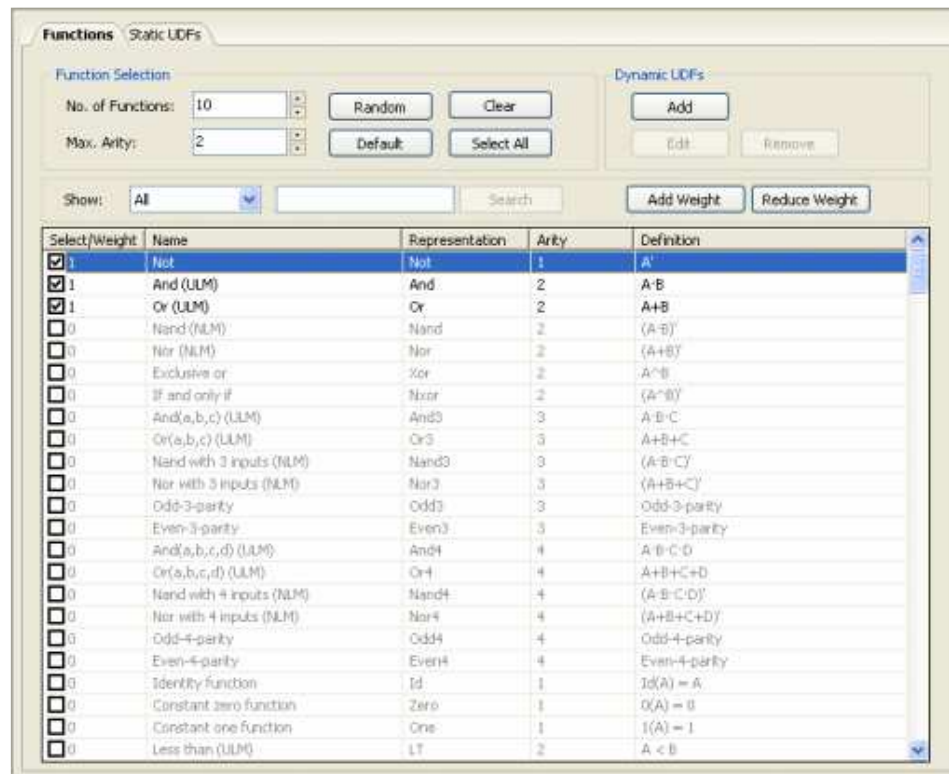
Choosing the Function Set

In the [Functions Panel](#) you choose the mathematical/logical and comparison functions you think appropriate to model your data. Here you also choose the probability (or the weight) each function has of being integrated in the chromosomes of the evolving models.



The functions you choose will consist of the function set. And besides the function set, the learning algorithms of GeneXproTools also use a terminal set. But unlike the functions in your function set which can be weighted as many times as you see fit, the terminal set is created automatically by GeneXproTools and consists of the independent variables in your data and, therefore, you cannot weight each terminal differently. But, during adaptation, the chromosomes of the evolving models are created and modified using the elements of both the function and terminal set, reflecting their relative representation. Although you can always handcraft your function set so that it is well-balanced relatively to the number of independent variables in the problem at hand, in GeneXproTools 4.0 this is no longer essential as GeneXproTools balances automatically all function sets. This is particularly important for high-dimensional datasets, say above 100-1000 variables.

GeneXproTools 4.0 offers a total of 279 [built-in mathematical functions](#) to choose from, including the mathematical, trigonometric and comparison functions most commonly used by sophisticated programming and modeling software. And for Logic Synthesis there are 258 [built-in logical functions](#) available, including all the Boolean functions with one and two inputs, 92 functions with three inputs, and 146 functions with four inputs.



For some problems, a few basic functions operators are more than enough to create extremely accurate and elegant models. But for other problems, more complex functions are necessary to create good models and when inside knowledge of the problem at hand is not producing the expected results, it might be interesting to experiment with different combinations of functions. The Function Selection Tool of GeneXproTools 4.0 helps you to experiment with different function sets very quickly either by pressing the Random button in the Functions Panel or by selecting certain sub-sets of functions.

The functions you select in the [Functions Panel](#) are the functions the learning algorithm will explore in order to create a good model for your data. The [learning algorithms](#) of GeneXproTools cope very well with virtually any combination of mathematical or logical functions. But sometimes for the sake of simplicity and intelligibility of your models it pays to be frugal and we recommend you give a try to one of the simplest function sets, the one composed of only the basic arithmetic operators. But, of course, the modeling of complex realities sometimes requires more complex functions and GeneXproTools 4.0 provides a wide range of such functions. Furthermore, you can also design [user defined functions](#) and add them to the function kit of GeneXproTools and explore new solution spaces with them.



GeneXproTools Environment

Choosing the Chromosome Architecture

The [chromosome architecture](#) of your models include the **head size**, the **number of genes** and the **linking function**. You choose these parameters in the Settings Panel -> General Settings Tab.

The Head Size determines the complexity of each term in your model. In the [heads of genes](#), the GeneXproTools learning algorithm tries out different arrangements of functions and terminals (variables and constants) in order to model your data. The plasticity of this architecture allows the discovery of a virtually infinite number of models of different sizes and shapes which are afterwards tested and selected during the learning process. The heads of genes are shown in blue in the compact, linear representation of your models in the Model Panel.

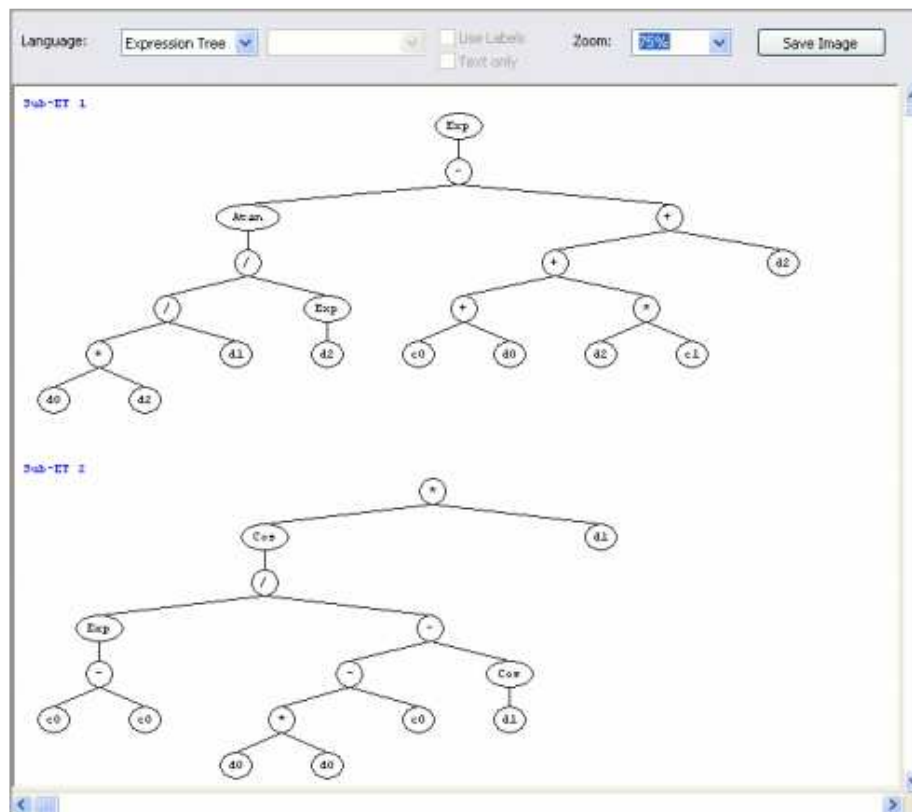
More specifically, the head size h of each gene determines the maximum width w and maximum depth d of the sub-expression trees encoded in the gene, which are given by the formulas:

$$w = (n - 1) * h + 1$$

$$d = ((h + 1) / m) * ((m + 1) / 2)$$

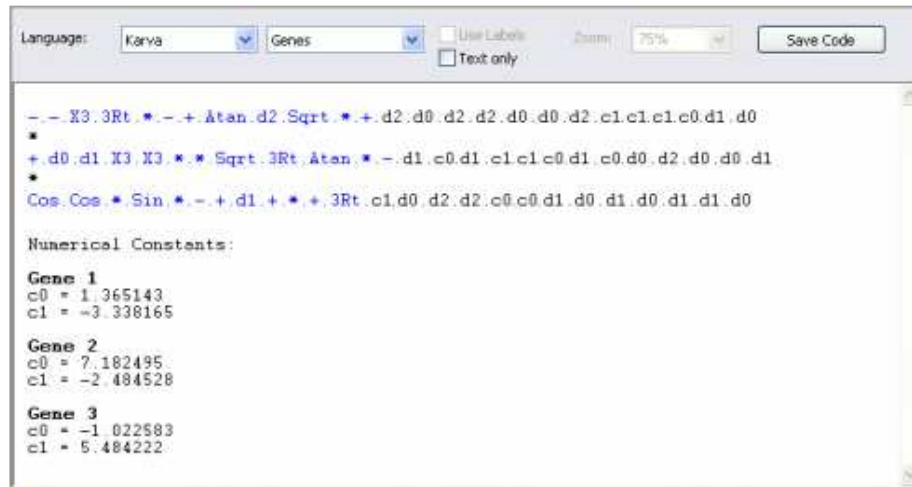
where m is minimum arity and n is maximum arity.

Thus, the GeneXproTools [learning algorithm](#) selects its models between these extreme cases, fine-tuning the ideal size and shape during the evolutionary process without human intervention.



The [number of genes](#) per chromosome is also an important parameter. It will determine the number of (complex) terms in your model as each gene codes for a different parse tree (sub-expression tree or sub-ET). Theoretically, one could just use a huge single gene in order to evolve very complex models. But the partition of the chromosome into simpler, more manageable units gives an edge to the learning process and more efficient and elegant models can be discovered using [multigenic chromosomes](#).

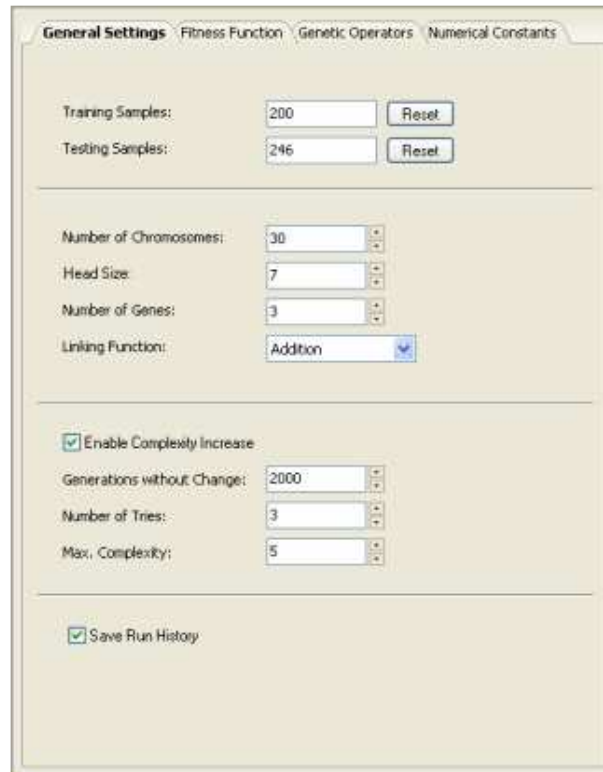
Whenever the number of genes is greater than one, you must also choose a suitable [linking function](#) to link the mathematical/logical terms encoded in each gene. GeneXproTools 4.0 allows you to choose **addition, subtraction, multiplication, or division** to link mathematical sub-ETs, and **And, Or, Nand, Nor, Xor, Nxor, Less Than, Greater Than, Less Or Equal, or Greater Or Equal** to link Boolean sub-ETs. As expected, addition/Or (and obviously subtraction and And) works very well for virtually all problems but sometimes one of the other linkers could be useful for searching different solution spaces and finding a very good, albeit unexpected model.



GeneXproTools Environment

Choosing the Population Size

To choose the population size, in the Settings Panel -> General Settings enter the Number of Chromosomes. The default in GeneXproTools 4.0 is 30, that is, a very small population, as it produces very good results for virtually all problems, even for such high-dimensional problems as DNA microarrays with more than 25,000 variables. But, obviously, if you'd rather work with larger populations, the limit is 4096 chromosomes in GeneXproTools 4.0.



General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 200 [Reset]

Testing Samples: 246 [Reset]

Number of Chromosomes: 30

Head Size: 7

Number of Genes: 3

Linking Function: Addition

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 5

☒ Save Run History

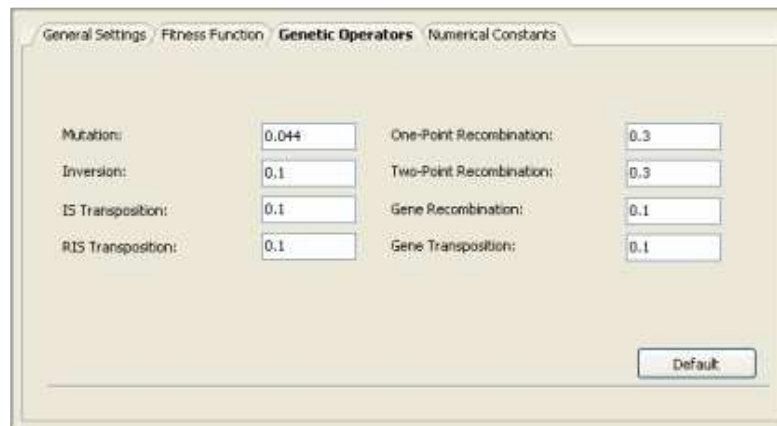
Learning and adaptation occur in evolutionary algorithms because [populations of individuals](#) (models) adapt with time. Obviously, the larger the population the faster (in terms of generations) adaptation. But there's a caveat here: in a computer, the higher the number of models the longer it takes to process them. So one must choose a population size that enables an efficient adaptation in record time. In [Gene Expression Programming](#), the learning algorithm used by GeneXproTools, this can be accomplished well enough with populations of 30-200 individuals provided that the right degree of [genetic modification](#) is being introduced in the evolving populations. How this is accomplished is explained in the next section.

◀ | ▶

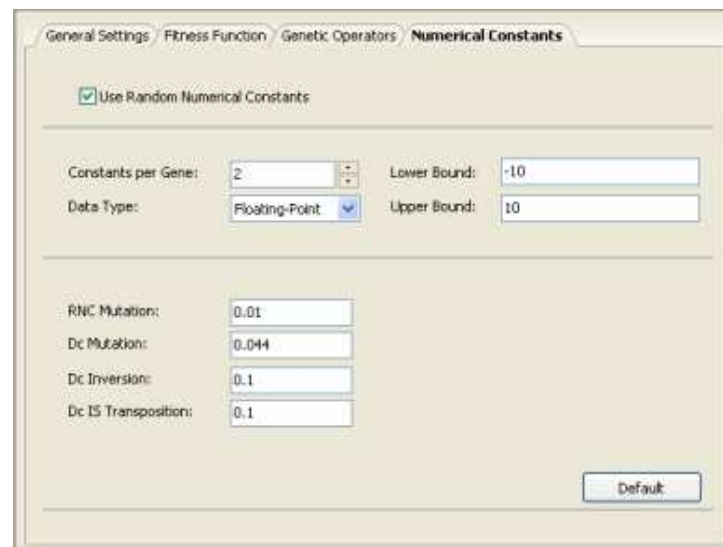
GeneXproTools Environment

Choosing the Degree of Genetic Modification

In the Settings Panel -> Genetic Operators you enter the rates for the set of [genetic operators](#) most commonly used in GeneXproTools 4.0.



In case you are using [random numerical constants](#) you have also access to [additional genetic operators](#) especially designed to allow the efficient adaptation of the numerical constants used during the evolution of your models.



If you are not familiar with the role and power of all these operators and if you are not sure which rates to use, please use the default values for they work very well for all cases.

If you want to learn about their role and power, you can choose different operators and different rates and then analyze their behavior in the plots shown in the [Run Panel](#).

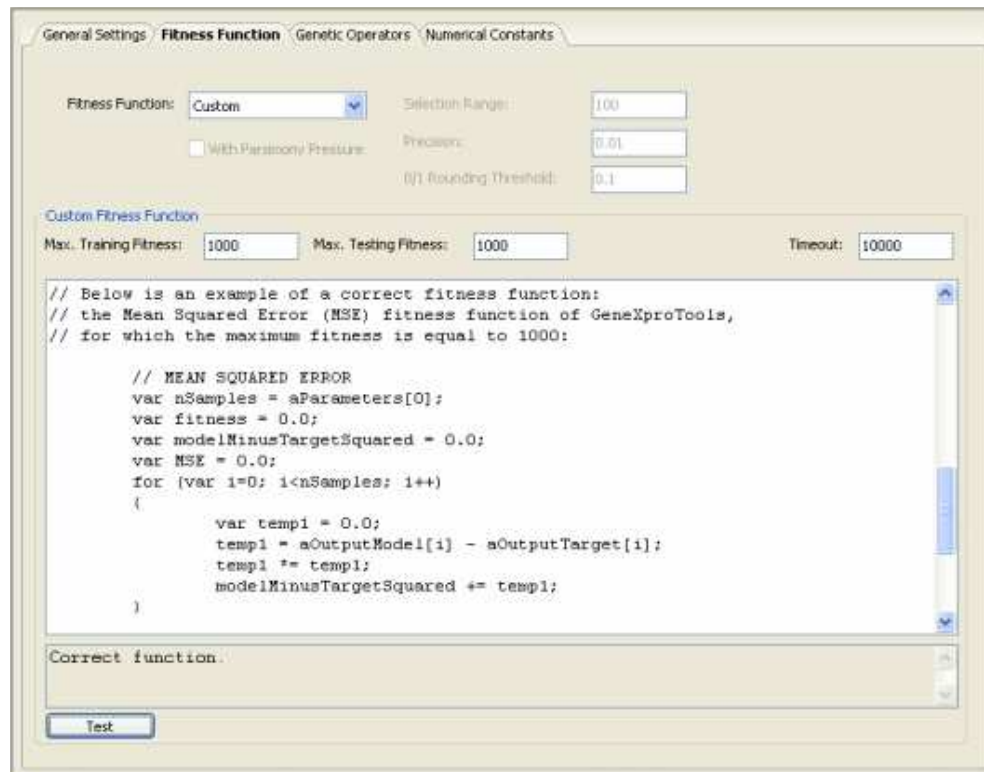
And if you want to know the mechanisms of all the genetic operators of GeneXproTools 4.0, you can find all about it in the chapter [Genetic Operators](#).



GeneXproTools Environment

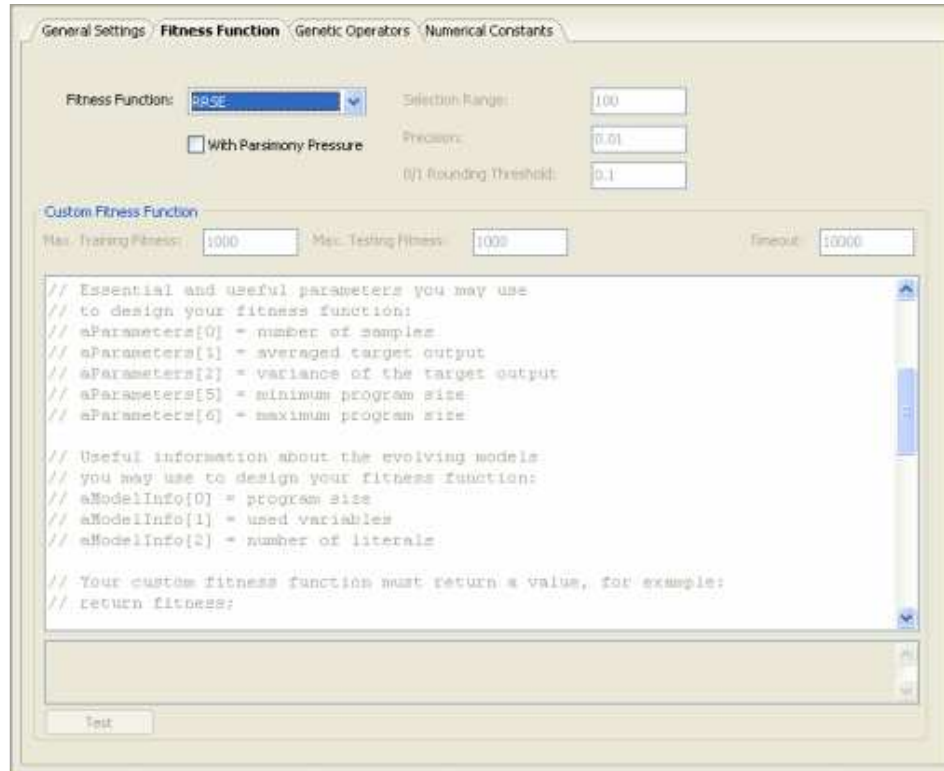
Choosing the Fitness Function

Depending on the Problem Category, you have access to different sets of built-in [fitness functions](#). Additionally, for all kinds of problems you can also design your own fitness function.



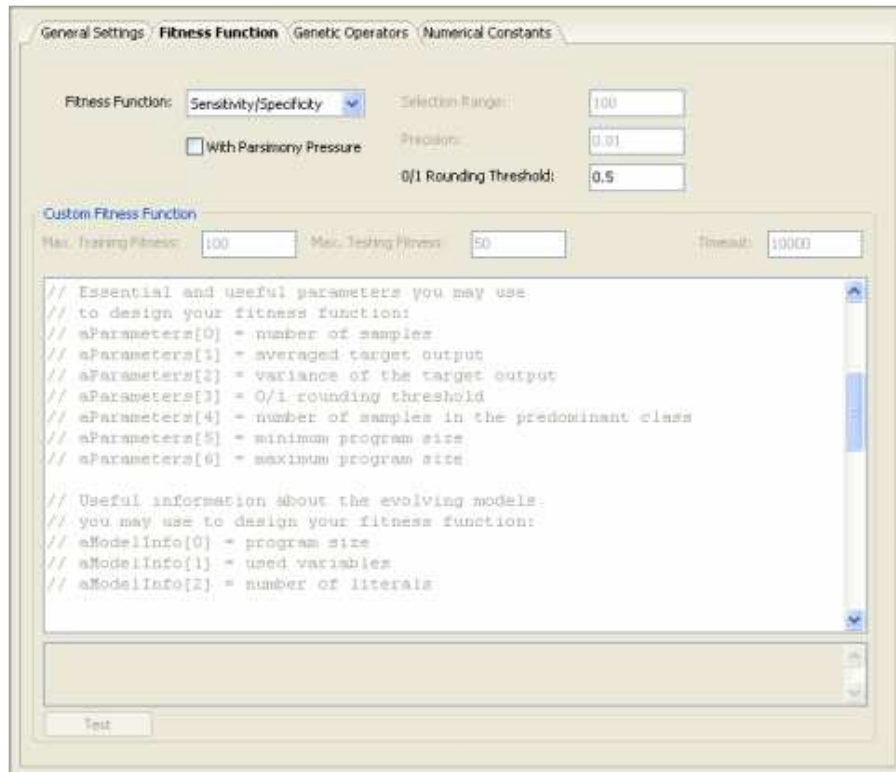
So, both for [Function Finding](#) and [Time Series Prediction](#) you can select one of the 36 built-in fitness functions:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)
- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)
- [R-square](#)
- [Correlation Coefficient](#)
- And all the above with [parsimony pressure](#)



For [Classification](#) and [Logic Synthesis](#) GeneXproTools 4.0 offers 30 different built-in fitness functions:

- [Number of Hits](#)
- [Hits with Penalty](#)
- [Accuracy](#)
- [Squared Accuracy](#)
- [Sensitivity/Specificity](#)
- [PPV/NPV](#)
- [SSPN](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [R-square](#)
- [Correlation Coefficient](#)
- And all the above with [parsimony pressure](#)



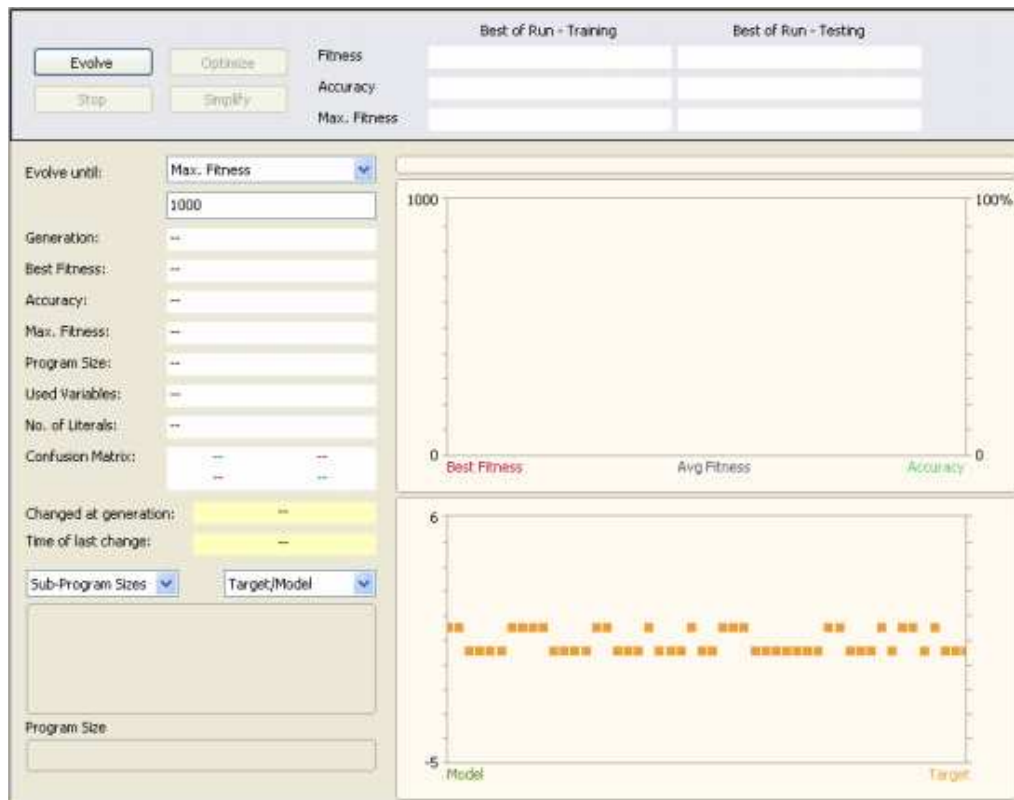
The kind of fitness function you choose will depend most probably on the statistical function you are most familiar with. And although there is nothing wrong with this, for all of them can accomplish an efficient evolution, you might want to try different fitness functions for they travel the fitness landscape differently: some of them very straightforwardly in their pursuits while others choose less obvious paths.



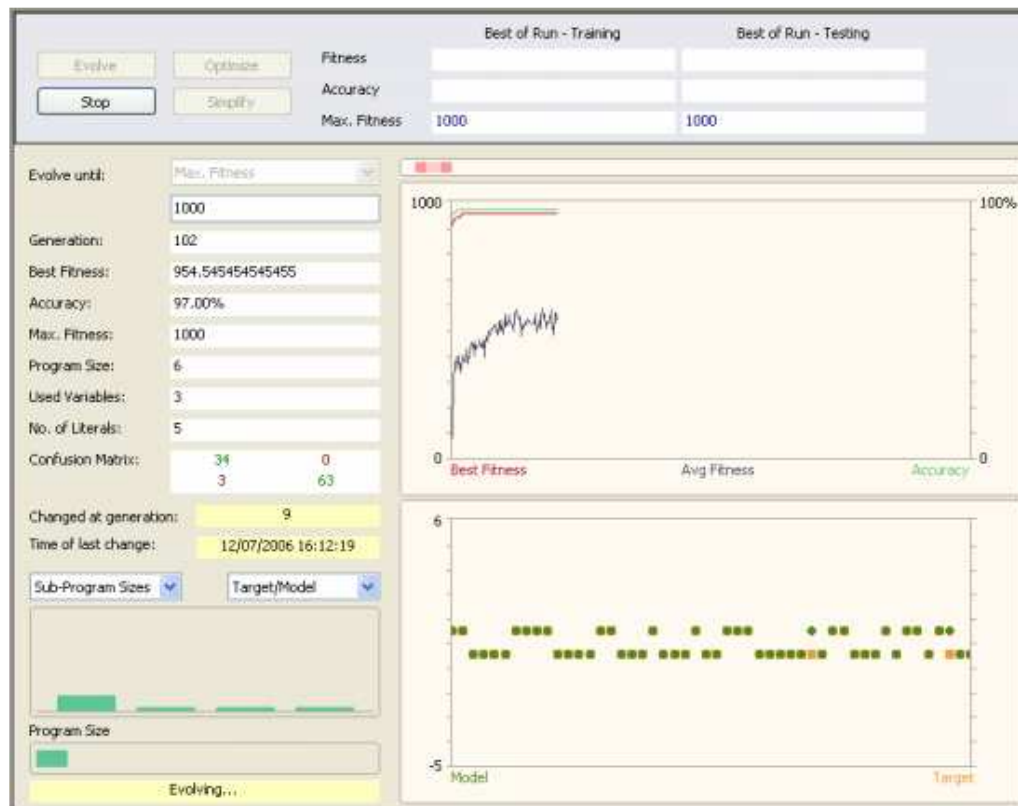
GeneXproTools Environment

Creating the Model

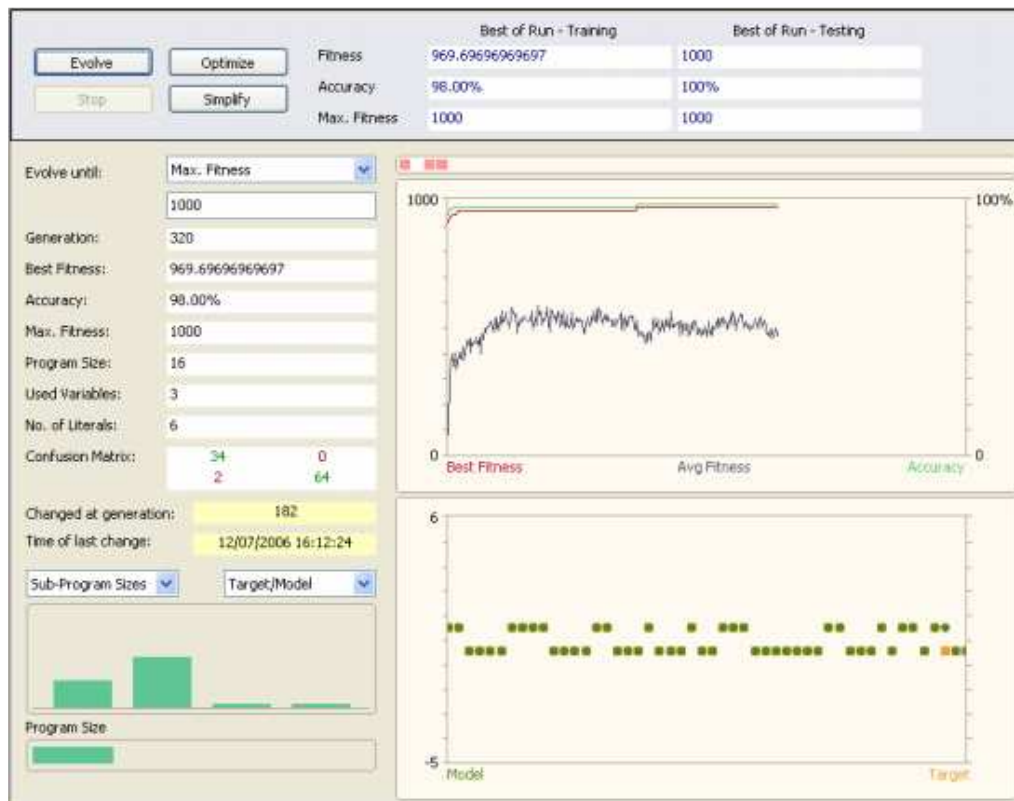
This is the easiest part on the whole modeling process. You just have to click the Evolve button in the [Run Panel](#) and let the learning algorithm of GeneXproTools do the job for you.



Depending on the Stop Criterion you chose (you select it in the box Evolve Until in the [Run Panel](#)), the modeling process will end on its own or you might stop it whenever you see fit by pressing the Stop button.



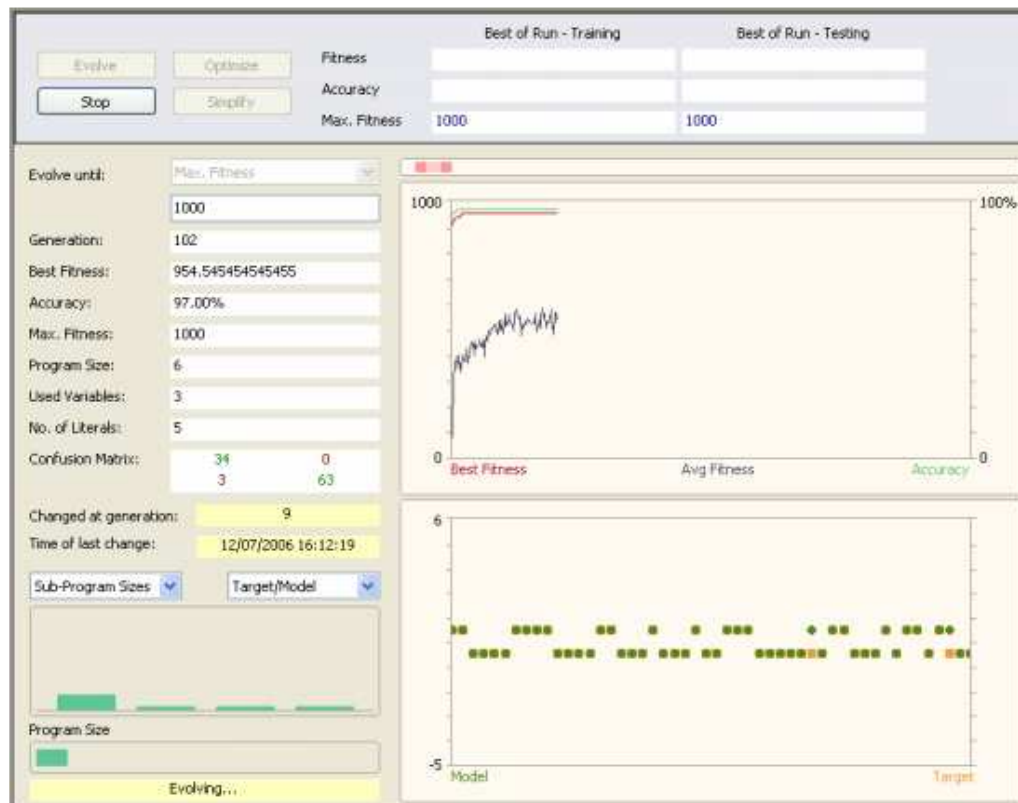
During a run you can monitor the evolution of crucial statistical indicators such as the R-square/Accuracy and Fitness of the best-of-run model that attest to the soundness of the models being discovered. And based on this visual information you might choose to halt the evolutionary process so that you can take a closer look at your model. And if you are still not satisfied with your model, there's no problem at all because you can restart exactly from where you have stopped by pressing the Optimize ([evolve with seed](#)) or Simplify (evolve with [parsimony pressure](#)) buttons.



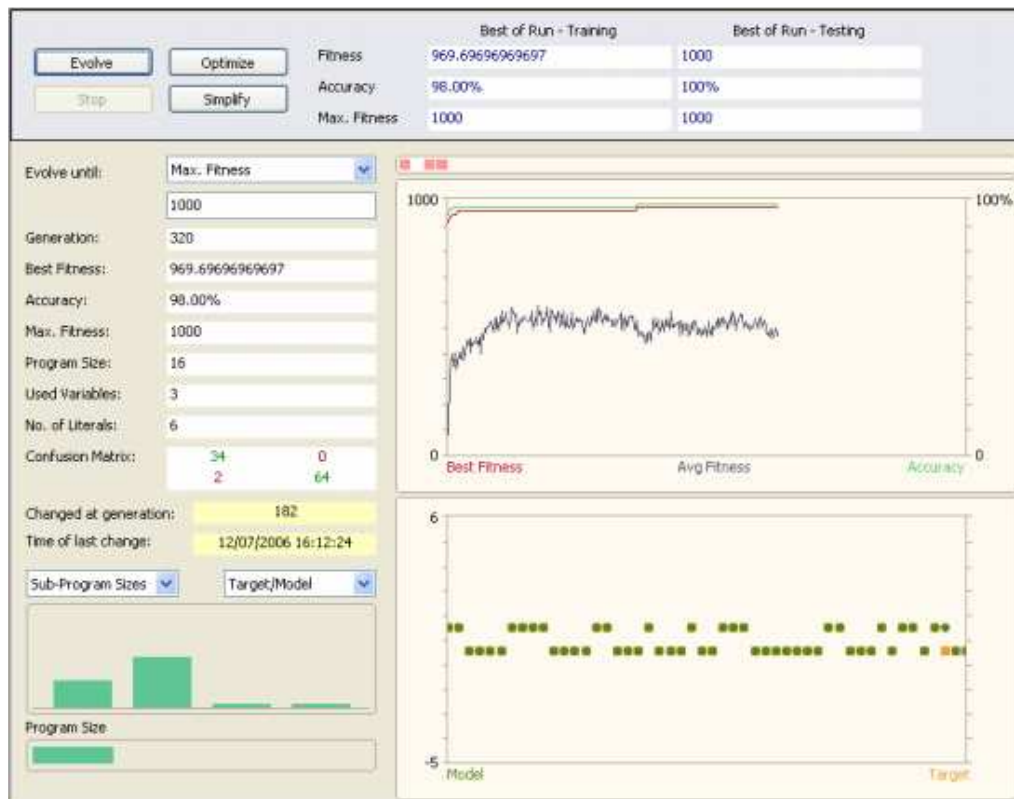
GeneXproTools Environment

Evaluating the Model

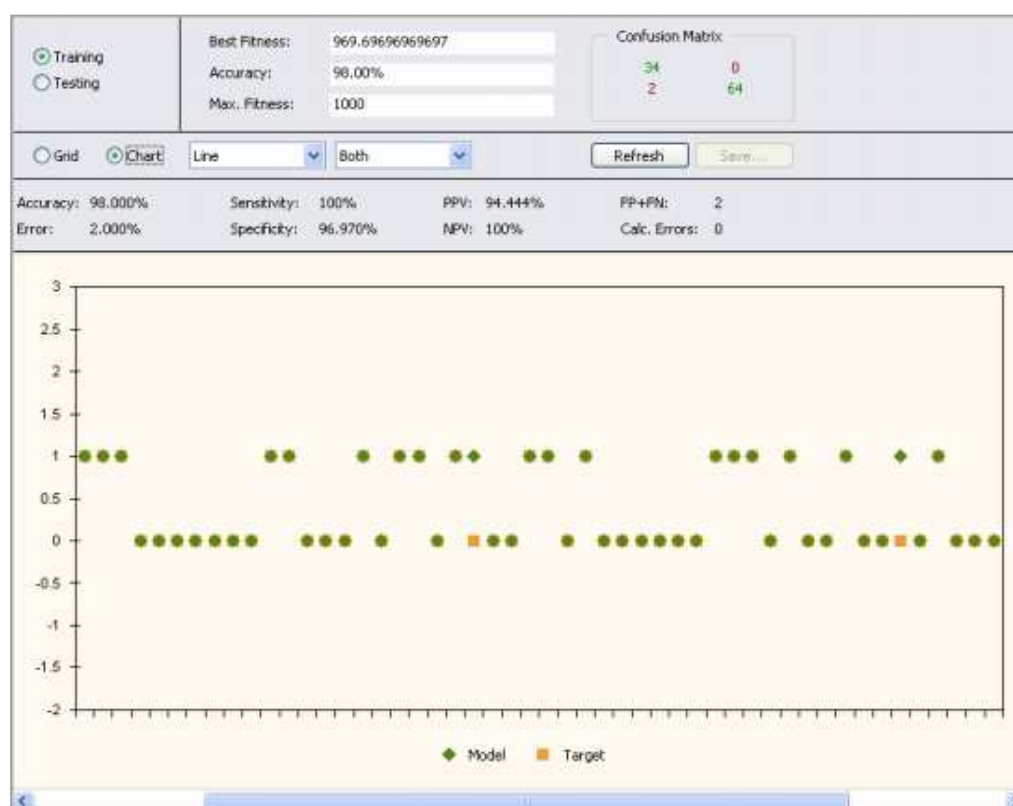
GeneXproTools 4.0 allows you to evaluate the accuracy of your models in the training set while they are still evolving. In the [Run Panel](#), during a run not only the fitness but also the [R-square](#) or [Classification Accuracy](#) of the best-of-generation model are plotted in a chart so that you can stop the learning process whenever you see fit. During evolution, you can also see how well the evolving models are fitting your data by observing the Curve Fitting plot (or Target/Model plot in Classification and Logic Synthesis problems).



And after a run, still in the [Run Panel](#), the fitness and [R-square](#) or [Classification Accuracy](#) of the best-of-run model obtained on the testing set are also shown, giving you also a quick evaluation of the generalizing capabilities of your model.



Then, in the [Results Panel](#), you can further evaluate the performance of your models both on the training and testing sets by comparing the output of your model with the target using both a spreadsheet and a curve fitting plot.



In addition, while accessing your models visually in the [Results Panel](#) you can also check the results obtained for a wide set of [statistical functions](#) both for the training and testing sets. You can save these statistical evaluations of your models and keep them in the [Report Panel](#) for future reference.

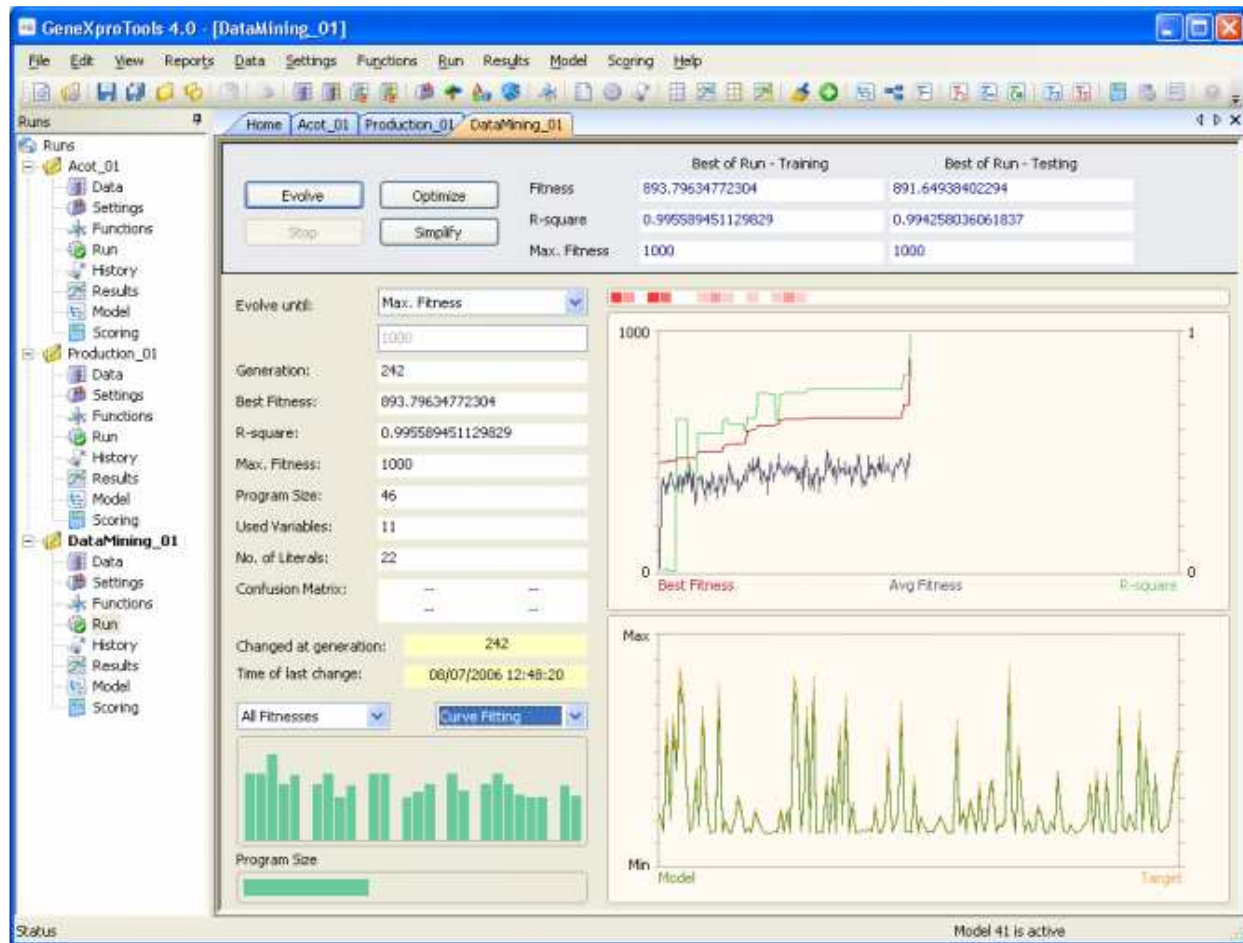
Thus, for [Function Finding](#) and [Time Series Prediction](#) problems, the values for the mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, R-square, and correlation coefficient can be immediately accessed without leaving the GeneXproTools modeling environment.

And for [Classification](#) and [Logic Synthesis](#) problems, the values for the classification error, classification accuracy, confusion matrix (true positives, true negatives, false positives, and false negatives), sensitivity, specificity, positive predictive value, negative predictive value, mean squared error, root mean squared error, mean absolute error, relative squared error, root relative squared error, relative absolute error, R-square, and correlation coefficient can also be immediately evaluated without leaving the GeneXproTools modeling environment.



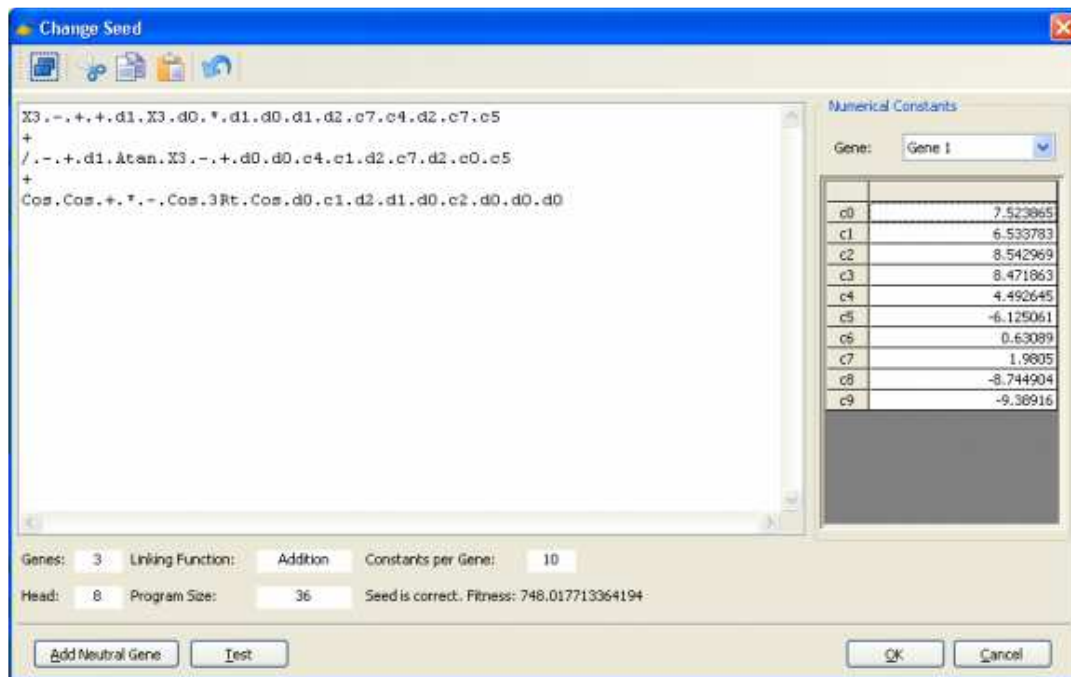
Modeling from Seed

file:///C:/Documents and Settings/candida/Local Settings/Temp/~hh4BEF.htm

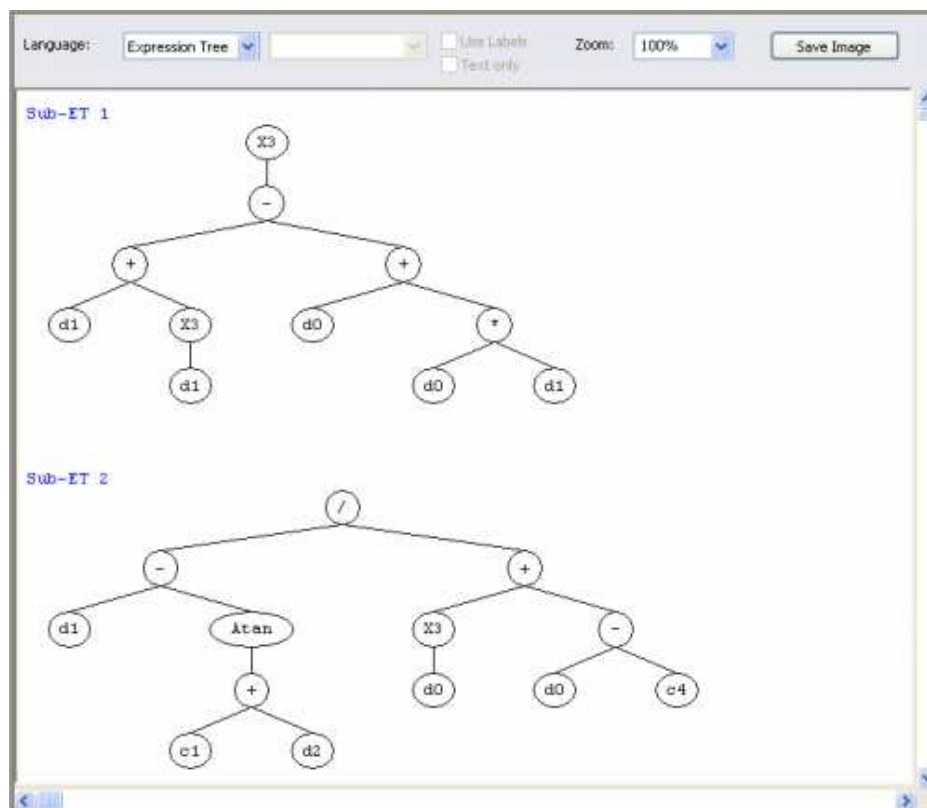


In this case, the active model of the last run (usually the best-of-run, but any model kept in the [History Panel](#) can become active) can be used immediately as seed.

In the latter case, the model used as seed must be introduced through the Change Seed window using [Karva notation](#).



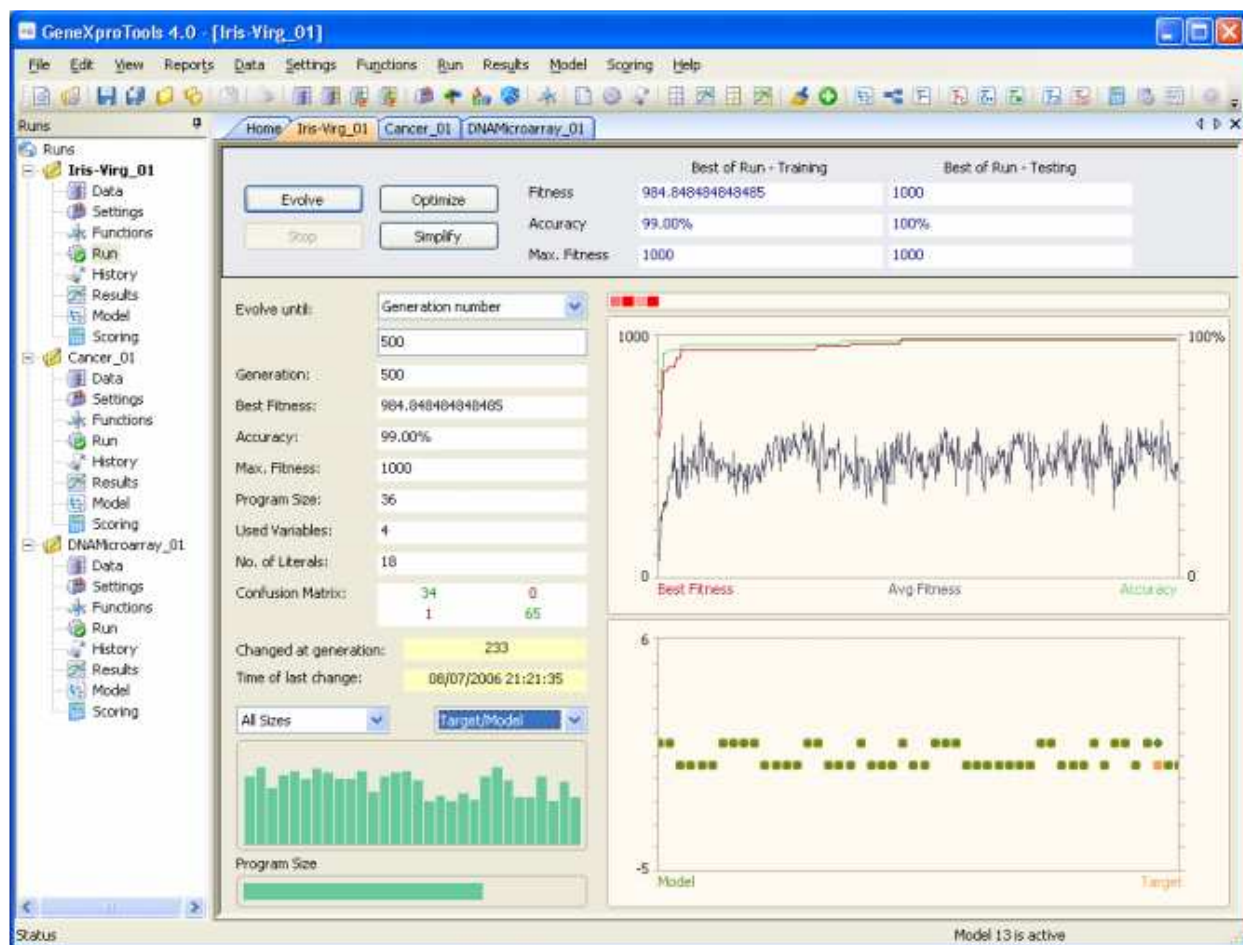
After introducing your seed model, you can test its structural soundness and evaluate its fitness by pressing the Test button on the Change Seed window. Then you can also check its translation into a more conventional representation by checking its structure in the [Model Panel](#).



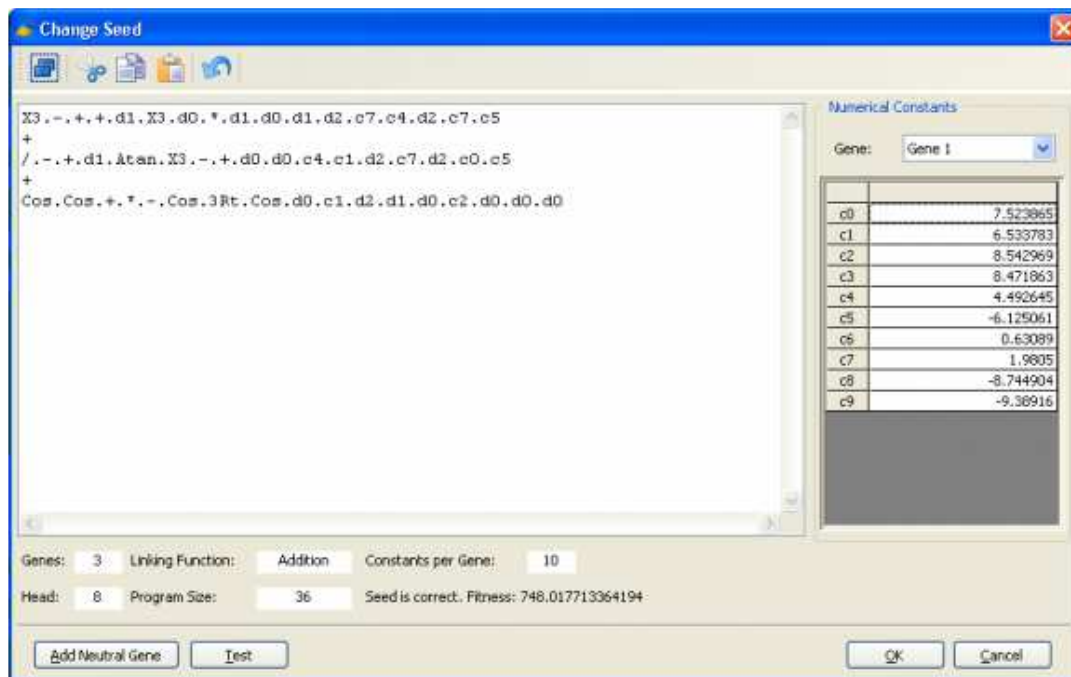
GeneXproTools Environment

Simplifying the Model

GeneXproTools 4.0 allows you to simplify an existing model (which could have been either generated by GeneXproTools or by another modeling tool) either by pressing the Simplify button on the [Run Panel](#) or by choosing a fitness function with parsimony pressure and then press Optimize or Simplify on the [Run Panel](#).

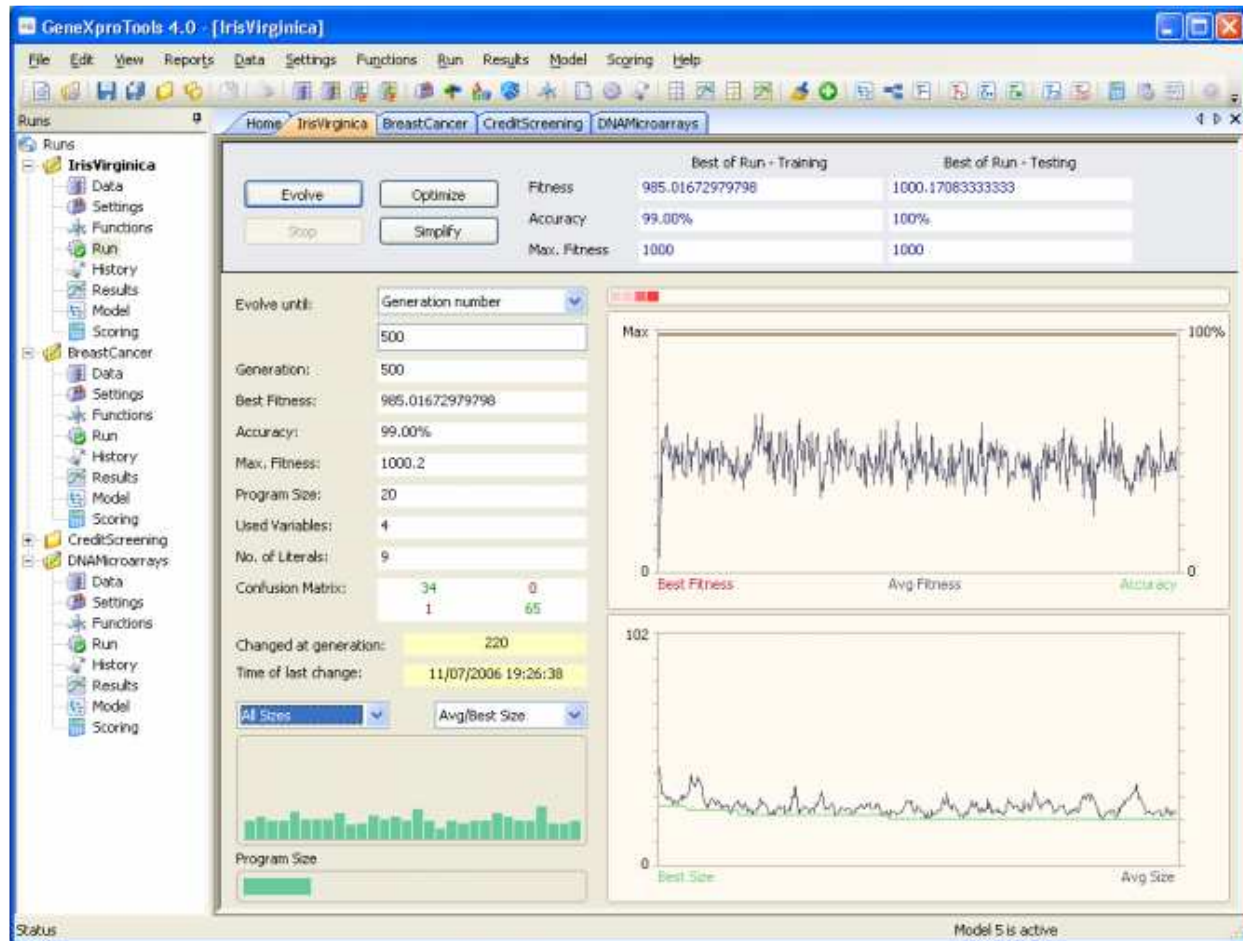


For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested.



Then, in the [Run Panel](#), by clicking the Simplify button, an evolutionary process starts in which all the subsequent models will be descendants of the model you want to simplify. Keep in mind, however, that the simplification algorithm used by GeneXproTools is an evolutionary one and, despite parsimony pressure being applied, models continue to be selected preferentially by fitness and sometimes their complexity might even increase temporarily if the gain in fitness outweighs the loss in simplicity.

For models created in the GeneXproTools 4.0 environment, you just have to select the model you want to simplify (either the best-of-run or an intermediate model picked up in the [History Panel](#)) and then click the Simplify button and let the algorithm create better descendants not only in terms of fitness but also in terms of size. This can be done for as long as you wish or until you feel the program is as parsimonious as it will get. By monitoring the plots Avg/Best Size and Sub-Program Sizes as well as the Heat Map in the [Run Panel](#), you will know exactly how the simplification process is progressing.

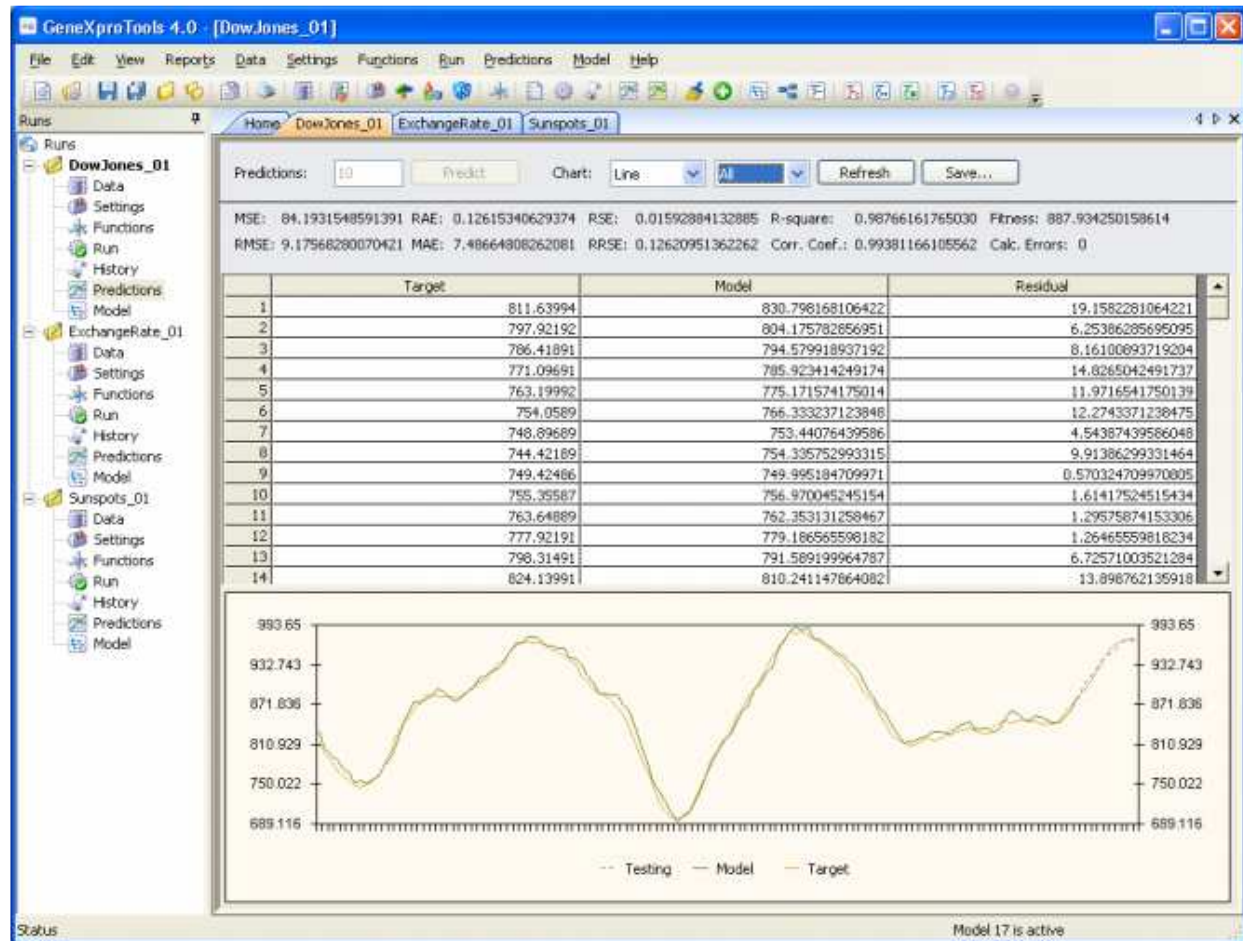


GeneXproTools Environment

Making Predictions with a Model

The **scoring engine** of GeneXproTools 4.0 allows the immediate application of the evolved models to:

- Score databases and Excel files.
- Score text files.
- Make predictions with time series models.

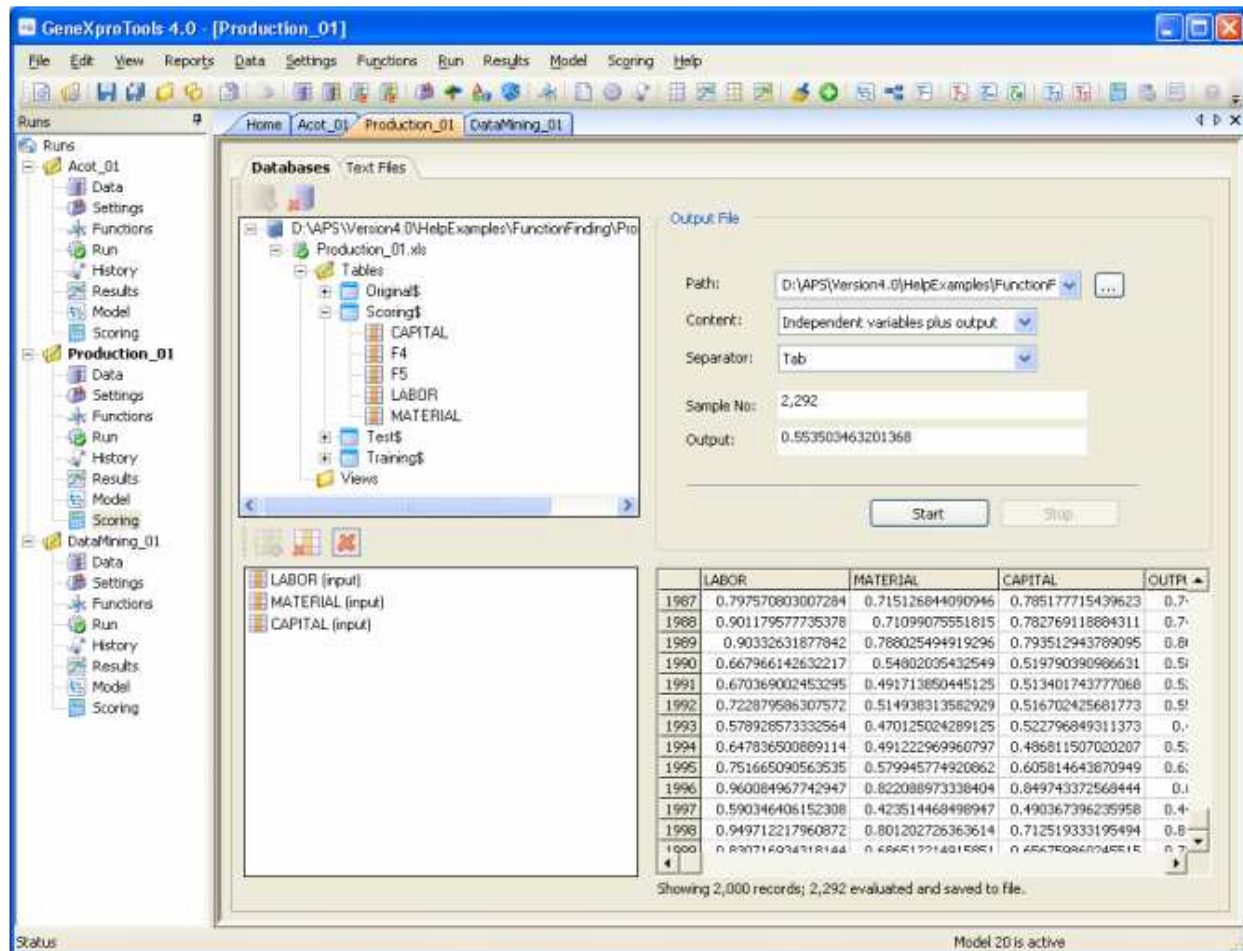


GeneXproTools Environment

Scoring from Excel and Databases

Once a model has been created it can be used hundreds or thousands of times to predict unknown behavior. The scoring engine of GeneXproTools allows you to do that without leaving the GeneXproTools modeling environment.

To score a database, on the Scoring Menu select Databases or go to the Scoring Panel and select the Databases Tab.



The scoring engine of GeneXproTools 4.0 uses the [JavaScript](#) code generated for your model to perform the computations since it already contains the code for the [UDFs](#) and [DDFs](#).

```

Language: Javascript
☐ Use Labels
☐ Text only
Zoom: 100%
Save Code

// Code generated by GeneXproTools 4.0 on 11/07/2006 23:57:39
// Training Samples: 300
// Testing Samples: 146
// Fitness Function: RRSE
// Training Fitness: 864.940137743324
// Training R-square: 0.977861370576141
// Testing Fitness: 854.354952286687
// Testing R-square: 0.974782317023882

function gepModel(d)
{
    var G1C0 = -5.373443;
    var G1C1 = 5.186767;
    var G2C0 = 4.433807;
    var G2C1 = -7.248138;
    var G3C0 = 3.755371;
    var G3C1 = 5.186767;

    var vTemp = 0.0;

    vTemp = ((G1C0+G1C1)/G1C0);
    vTemp += ((d[0]-UDF0())/DDF0(d[2],DDF0((G2C1/G2C1),d[1])));
    vTemp += d[1];

    return vTemp;
}

function DDF0(x0,x1)
{
    return (x0+x1)*(x0+x1);
}

function UDF0()
{
    var myValue = (terminals[0] + terminals[1]) / 2.0;
    return myValue;
}

```

The output is saved to a file which contains the predictions of your model for all the scores in the database. For small datasets (up to 16 variables and 2000 samples) GeneXproTools also shows the scoring results in the Scoring Panel.

Databases | Text Files

Path: D:\AP5\Version4.0\Help\Examples\FunctionF\Pro

Content: Independent variables plus output

Separator: Tab

Sample No: 2,292

Output: 0.553503463201368

Start Stop

	LABOR	MATERIAL	CAPITAL	OUTPUT
1987	0.797570803007284	0.715126844090946	0.785177715439623	0.74586
1988	0.901179577735378	0.71099075551815	0.782769118684311	0.74027
1989	0.90332631877842	0.788025494919296	0.793512943789095	0.80818
1990	0.667966142632217	0.54802035432549	0.519790390986631	0.58371
1991	0.670369002453295	0.491713850445125	0.513401743777068	0.52542
1992	0.722879586307572	0.514938313582929	0.516702425681773	0.58350
1993	0.578928573332564	0.470125024289125	0.522796849311373	0.4945
1994	0.647836500889114	0.491222969960797	0.486811507020207	0.52315
1995	0.751665090563535	0.579945774920862	0.605814643870949	0.62033
1996	0.96084967742947	0.822088973338404	0.849743372568444	0.8344
1997	0.590346406152306	0.423514468498947	0.490367396235958	0.44172
1998	0.949712217960872	0.801202726363614	0.712519333195494	0.81666
1999	0.89714074718144	0.5861771401851	0.55768847045115	0.7763

Showing 2,000 records; 2,292 evaluated and saved to file.

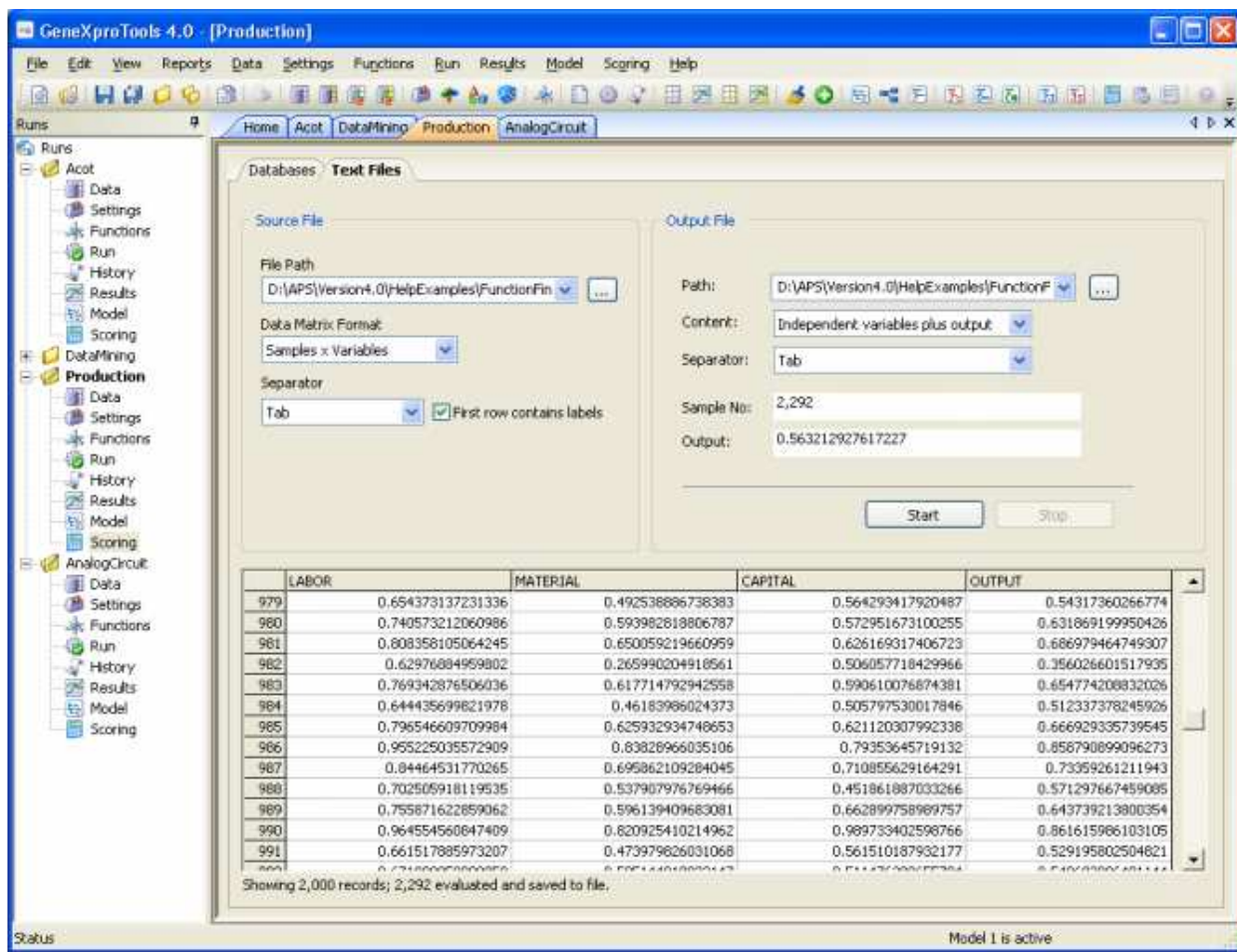


GeneXproTools Environment

Scoring from Text Files

Once a model has been created it can be used hundreds or thousands of times to predict unknown behavior. The scoring engine of GeneXproTools allows you to do that without leaving the GeneXproTools modeling environment.

For scoring data kept in text files, on the Scoring Menu select Text Files or go to the Scoring Panel and select the Text Files Tab.



The scoring engine of GeneXproTools 4.0 uses the [JavaScript](#) code of your model to perform the computations since it already contains the code for the [UDFs](#) and [DDFs](#).

Language: Javascript ☐ Use Labels ☐ Text only Zoom: 100% Save Code

```
// Code generated by GeneXproTools 4.0 on 11/07/2006 23:57:39
// Training Samples: 300
// Testing Samples: 146
// Fitness Function: RRSE
// Training Fitness: 864.940137743324
// Training R-square: 0.977861370576141
// Testing Fitness: 854.354952286687
// Testing R-square: 0.974782317023882

function gepModel(d)
{
    var G1C0 = -5.373443;
    var G1C1 = 5.186767;
    var G2C0 = 4.433807;
    var G2C1 = -7.248138;
    var G3C0 = 3.755371;
    var G3C1 = 5.186767;

    var vTemp = 0.0;

    vTemp = ((G1C0+G1C1)/G1C0);
    vTemp += ((d[0]-UDF0())/DDF0(d[2],DDF0((G2C1/G2C1),d[1])));
    vTemp += d[1];

    return vTemp;
}

function DDF0(x0,x1)
{
    return (x0+x1)*(x0+x1);
}

function UDF0()
{
    var myValue = (terminals[0] + terminals[1]) / 2.0;
    return myValue;
}
```

The output is saved to a file which contains the predictions of your model for all the scores in the text file. For small datasets (up to 16 variables and 2000 samples) GeneXproTools also shows the scoring results in the Scoring Panel.

Databases Text Files

Source File

File Path: D:\APS\Version4.0\Help\examples\FunctionF ...

Data Matrix Format: Samples x Variables

Separator: Tab ☒ First row contains labels

Output File

Path: D:\APS\Version4.0\Help\examples\FunctionF ...

Content: Independent variables plus output

Separator: Tab

Sample No: 2,292

Output: 0.553503463201368

Start Stop

	LABOR	MATERIAL	CAPITAL	OUTPUT
1987	0.797570003007284	0.715126844090946	0.785177715439623	0.745860124335517
1988	0.901179577735378	0.71099075551815	0.782769118884311	0.740272671233412
1989	0.90332631877842	0.788025494919296	0.793512943789095	0.808187245837281
1990	0.667966142632217	0.54802035432549	0.519790390986631	0.583714519916069
1991	0.670369002453295	0.491713850445125	0.513401743777068	0.525425513025935
1992	0.722879986307572	0.514938313582929	0.516702425681773	0.553503463201368
1993	0.578928573332564	0.470125024289125	0.522796849311373	0.49455977578346
1994	0.647836500889114	0.491222969960797	0.486811507020207	0.523158403098033
1995	0.751665090563535	0.579945774920862	0.605814643870949	0.620336343118778
1996	0.960084967742947	0.822088973338404	0.849743372568444	0.83446603477471
1997	0.590346406152308	0.423514468498947	0.490367396235958	0.441721196745279
1998	0.949712217960872	0.801202726363614	0.712519333195494	0.816668616029998
1999	0.830716934318144	0.688512214915851	0.656759860245515	0.720585240805507
mean	0.85486473370545	0.474110708829444	0.478989827754438	0.708089324531767

Showing 2,000 records; 2,292 evaluated and saved to file.



GeneXproTools Environment

Making Predictions with Time Series Models

The algorithms of GeneXproTools 4.0 used for Time Series Prediction allow not only the evolution of models but also the immediate use of these models to make predictions.

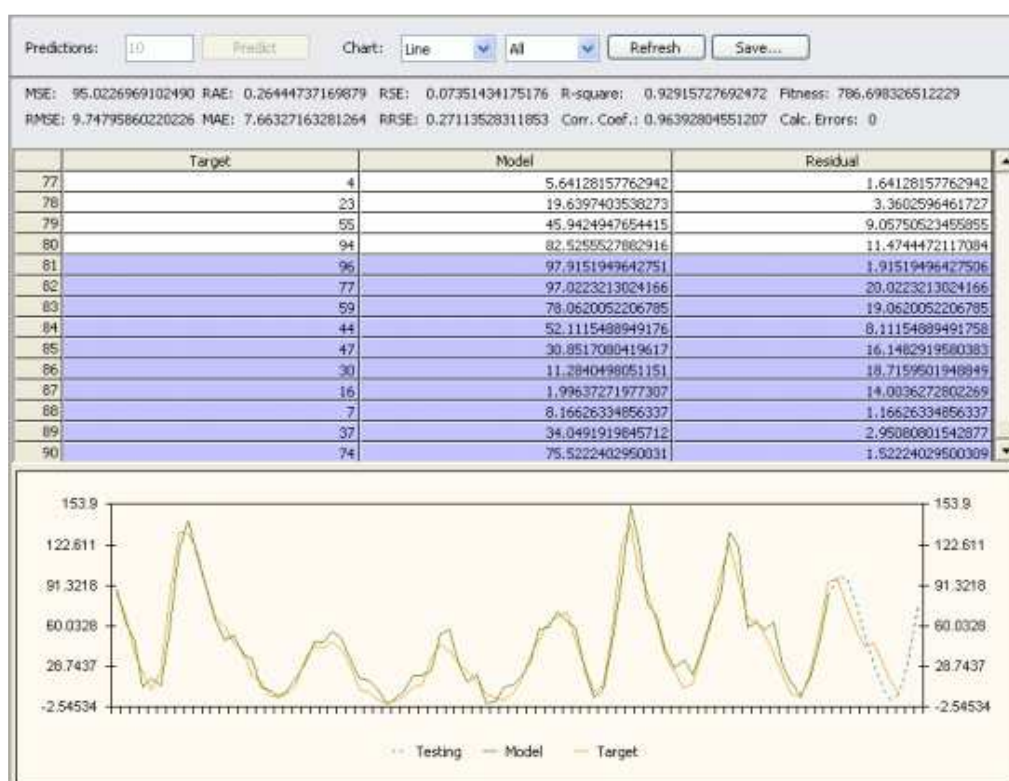
GeneXproTools 4.0 allows you to make two kinds of predictions: one for testing past known events and another for making predictions about the future. In both cases, though, predictions are made recursively, by evaluating the forecast at $t+1$, then using it to forecast $t+2$, and so on.

You must choose either one of these methods while [loading your time series data](#), as this imposes some constraints on the restructuring of the time series for training. Namely, n testing records (the n last ones) are saved for testing and sometimes a small number of records from the top must be deleted. However, you can also change these parameters later in the GeneXproTools modeling environment, by choosing General Settings in the Settings Menu.

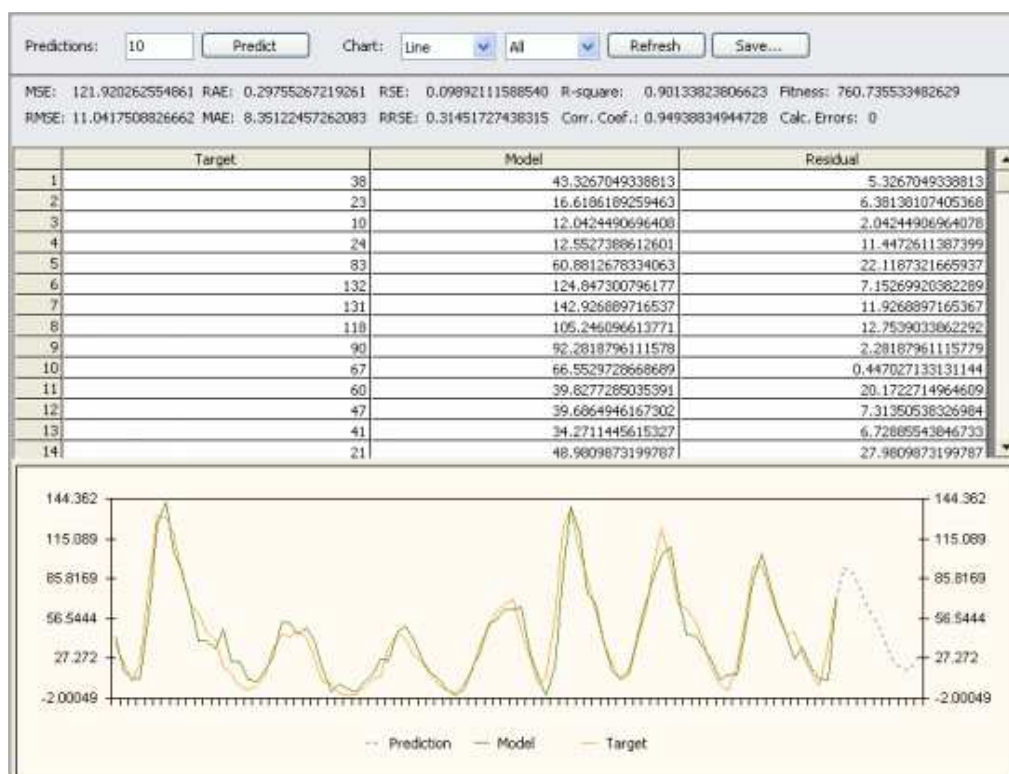
The screenshot displays the 'General Settings' tab of the GeneXproTools 4.0 interface. The settings are organized into several sections:

- Training Samples:** A text box containing '128' and a 'Reset' button.
- Records in Time Series:** A text box containing '140' and a 'Reset' button.
- Number of Chromosomes:** A spin box set to '30'.
- Head Size:** A spin box set to '8'.
- Number of Genes:** A spin box set to '5'.
- Linking Function:** A dropdown menu set to 'Addition'.
- Embedding Dimension:** A spin box set to '10'.
- Delay Time:** A spin box set to '1'.
- Prediction Mode:** A dropdown menu set to 'Testing'.
- Testing Predictions:** A spin box set to '10'.
- Enable Complexity Increase:** A checked checkbox.
- Generations without Change:** A spin box set to '2000'.
- Number of Tries:** A spin box set to '3'.
- Max. Complexity:** A spin box set to '8'.
- Save Run History:** A checked checkbox.

Thus, the first type of prediction can be used for research or pre-evaluation purposes, as it allows you to test the forecasting capabilities of your model on a set of past observations that are shown in blue in the Predictions Panel.



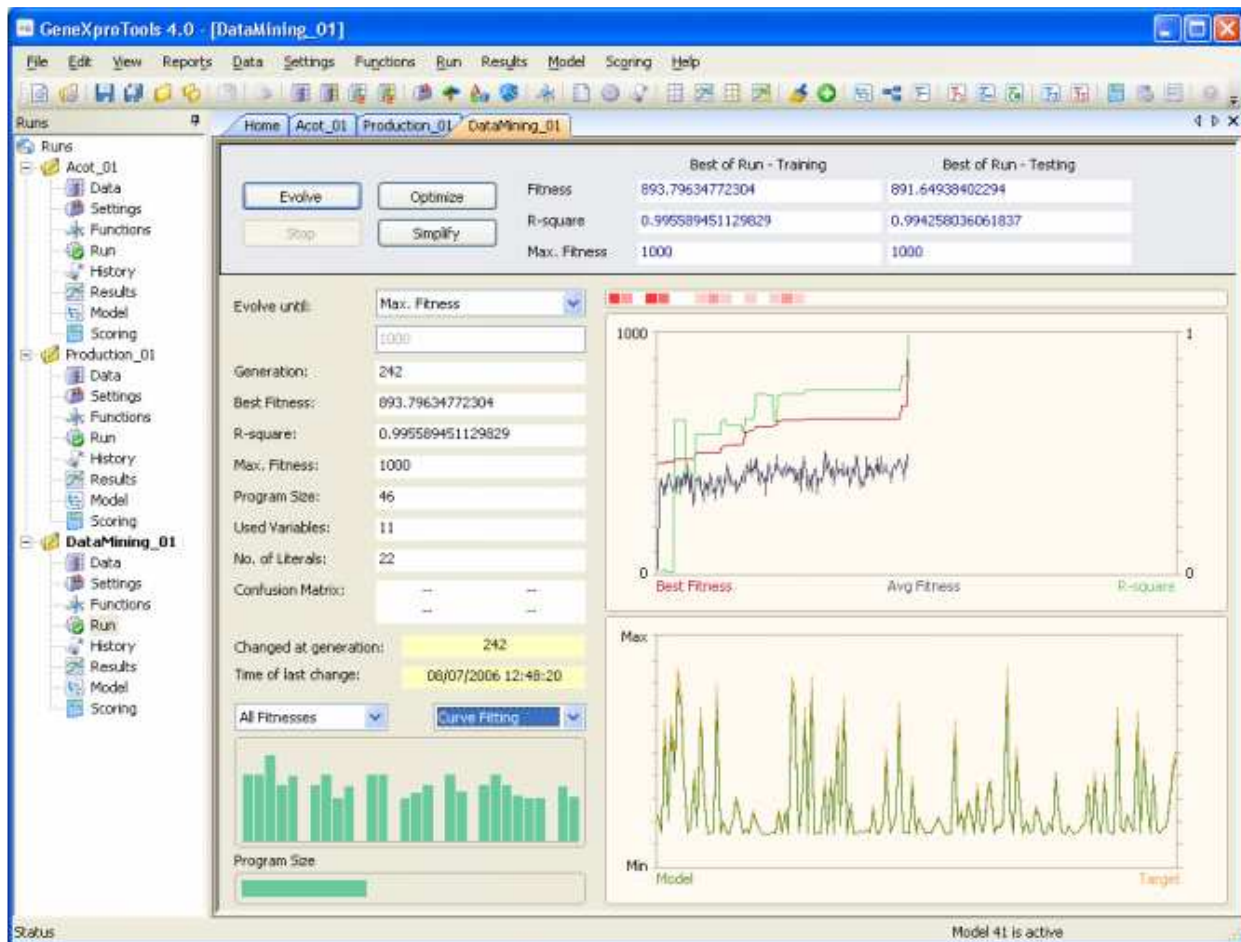
The second type is used obviously to predict unknown behavior, and GeneXproTools 4.0 allows you to venture into the future as far as you see fit, by setting the number of predictions you want to make and then click the Predict button in the Predictions Panel.



Working with Function Finding Tools

This chapter introduces the Function Finding tools of GeneXproTools and teaches you how to use them to discover complex nonlinear regression models. In this chapter, you'll learn how to:

- Load the input data for the GeneXproTools function finding algorithm.
- Explore the GeneXproTools function kit for function finding problems.
- Explore all the available architectures for building mathematical models.
- Explore the best fitness functions for function finding problems.
- Explore all the learning algorithms available for function finding problems.
- Explore essential evolutionary strategies for an efficient learning.



Working with Function Finding Tools

Loading Data

Before evolving a model with GeneXproTools 4.0 you must first load the input data for the learning algorithm. GeneXproTools 4.0 allows you to work either with databases/Excel or text files and, for text files, accepts two different **data matrix formats**.

The first is the standard **Samples x Variables** format where samples are in rows and variables in columns, with the dependent variable occupying the rightmost position. In the small example below with five samples, PRODUCTION is the dependent variable and LABOR, MATERIAL, and CAPITAL are the dependent variables:

```
LABOR MATERIAL CAPITAL PRODUCTION
0.88287491 0.70249262 0.64540872 0.73044339
0.76598265 0.59360711 0.62686264 0.64924032
0.64562062 0.56965563 0.53160265 0.58497820
0.72908206 0.58913859 0.52247179 0.62160041
0.50690655 0.23498841 0.42968111 0.36493144
```

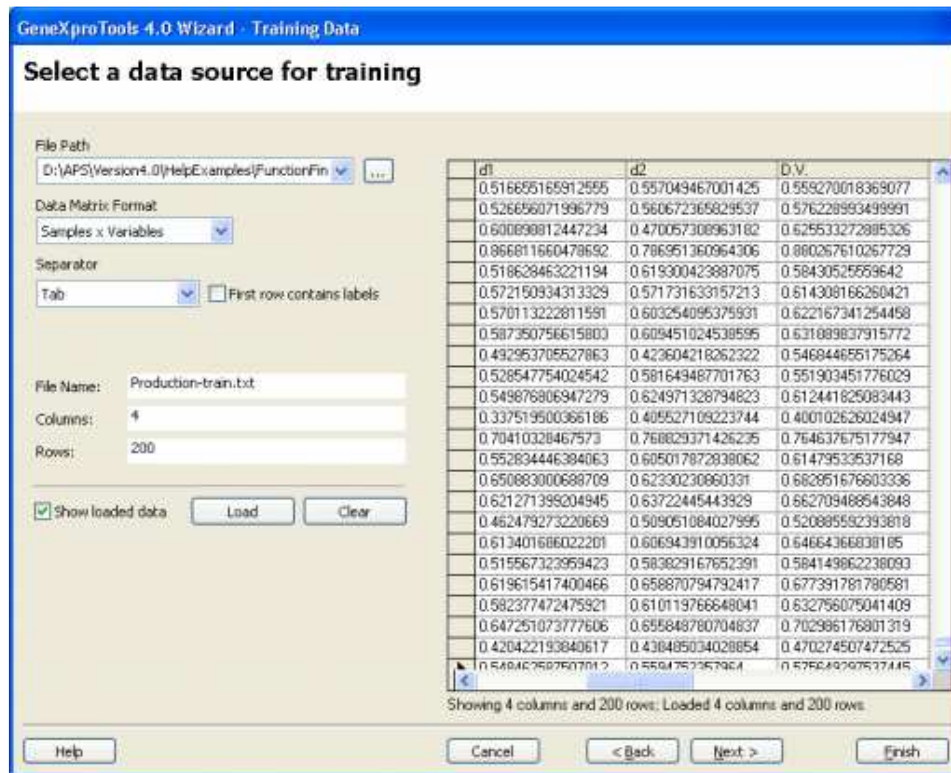
And the second, is the **Gene Expression Matrix** format commonly used in DNA microarrays studies where samples are in columns and variables in rows, with the dependent variable occupying the topmost position. For instance, in Gene Expression Matrix format, the small dataset above corresponds to:

```
PRODUCTION 0.73044339 0.64924032 0.5849782 0.62160041 0.36493144
LABOR 0.88287491 0.76598265 0.64562062 0.72908206 0.50690655
MATERIAL 0.70249262 0.59360711 0.56965563 0.58913859 0.23498841
CAPITAL 0.64540872 0.62686264 0.53160265 0.52247179 0.42968111
```

which is not visually very appealing, but of course very handy for datasets with a relatively small number of samples and thousands of variables. Note, however, that for Excel files this format is not supported and if your data is kept in this format in Excel, you must copy it to a text file so that it can be loaded into GeneXproTools.

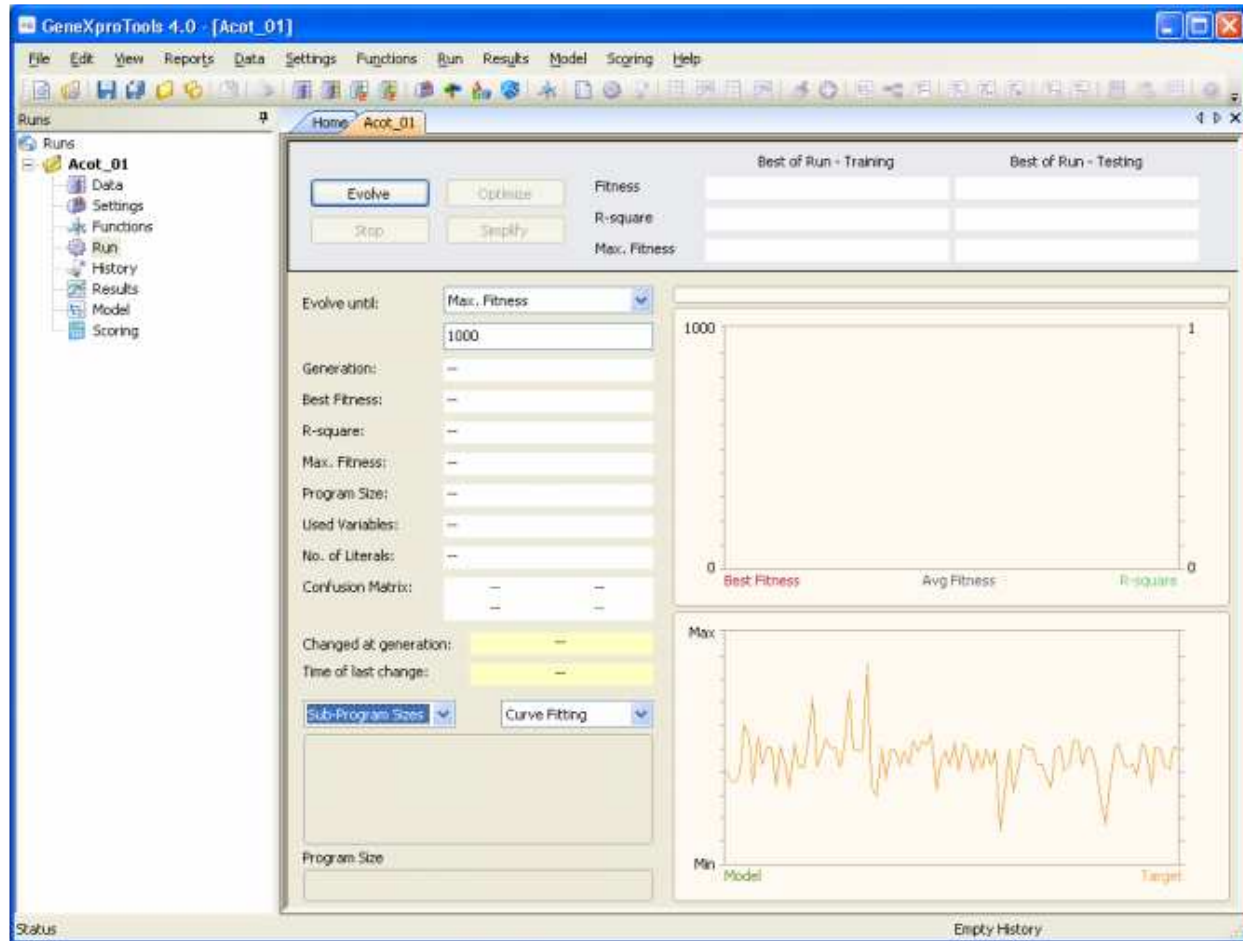
GeneXproTools uses the Samples x Variables format throughout and therefore all formats are automatically converted and shown in this format.

GeneXproTools supports the standard **separators** (space, tab, comma, semicolon, and pipe) and detects them automatically. The use of **labels** to identify your variables is optional and GeneXproTools also detects automatically whether they are present or not. If you use them, however, you will be able to generate more intelligible code where each variable is identified by its name, by checking the Use Labels box in the Model Panel.



To Load Input Data for Modeling

1. Click the File Menu and then choose New.
The New Run Wizard appears. You must give a name to your new run file (the default filename extension of GeneXproTools 4.0 run files is .gep) and then choose Function Finding in the Problem Category box and the kind of source file in the Data Source Type box. GeneXproTools 4.0 allows you to work either with Excel/databases or text files.
2. Then go to the Training Data window by clicking the Next button.
Choose the path for the training set by browsing the Open dialog box and choose the appropriate data matrix format. Irrespective of the data format used, GeneXproTools shows the loaded data in the standard Samples x Variables format, with the dependent variable occupying the rightmost position.
3. Then go to the Testing Data window by clicking the Next button.
Repeat the same steps of the previous point if you wish to use a testing set to evaluate the predictive accuracy of your model.
4. Click the Finish button to save your new run file.
The Save As dialog box appears and after choosing the directory where you want your new run file to be saved, the GeneXproTools modeling environment appears. Then you just have to click the Evolve button to create a model as GeneXproTools automatically chooses, from a gallery of templates, default settings that will enable you to evolve a model immediately.

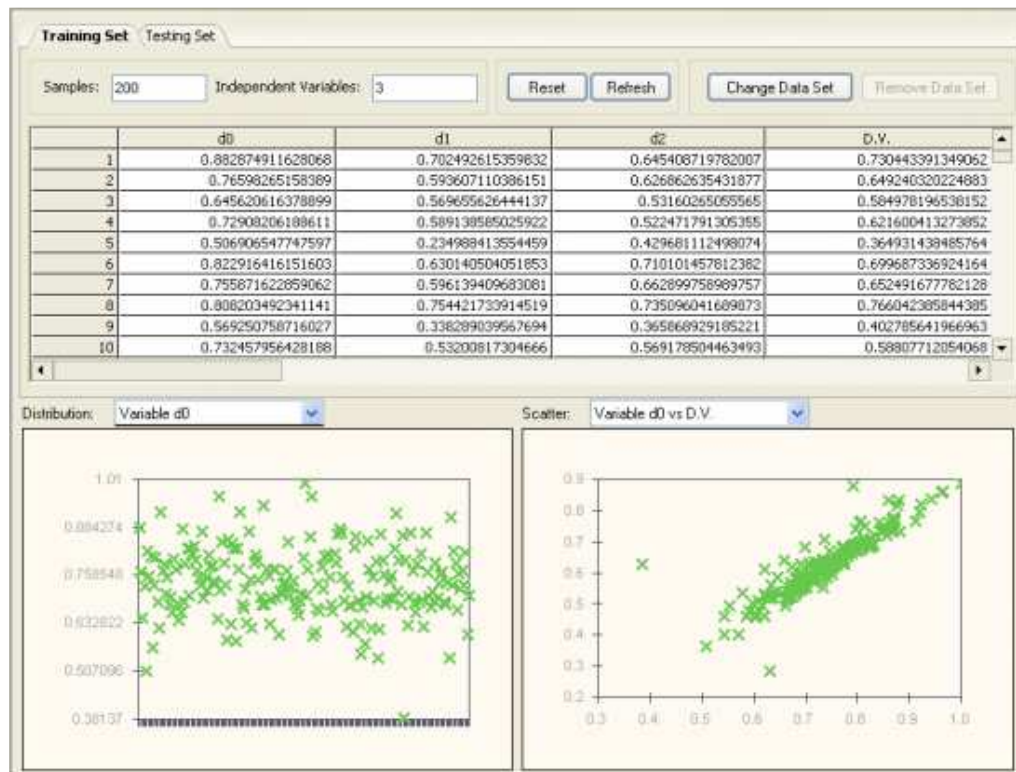


In data mining, be it performed by learning algorithms or conventional statistical methods, it really pays to take a good look at your data before embarking on a complex, usually time consuming modeling process. It's true that evolutionary algorithms are particularly well equipped to deal with noisy data, but the better the data you feed them the better the models they produce.

GeneXproTools helps you find missing and invalid (usually nominal) values in your data sets and prompts you to fix them before they are used for modeling. But the preparation of a well balanced data set should be done before loading the data into GeneXproTools, and we recommend you to particularly take care of the following:

- Avoid using duplicated samples for they can bias the modeling process considerably.
- Choose a well balanced data set.
- Choose a reasonable number of samples for training.
An excessively large data set will slow the modeling process unnecessarily. If you have access to huge data bases, it's good practice to use the surplus samples for testing instead. A good rule of thumb consists of using about 8-10 samples for each independent variable in your training data.
- Check your data sets carefully for inaccurate values. Typographical or measurement errors generally cause outliers that can be detected by graphing one variable at a time, a task that can be easily accomplished by GeneXproTools in the [Data Panel](#).

The [graphical visualization tools](#) of GeneXproTools 4.0 make it easy to identify outliers, which may well represent errors in the data files. After loading your data into GeneXproTools, in the [Data Panel](#) you can visualize the distribution of values for each variable and also plot each independent variable against the dependent variable.

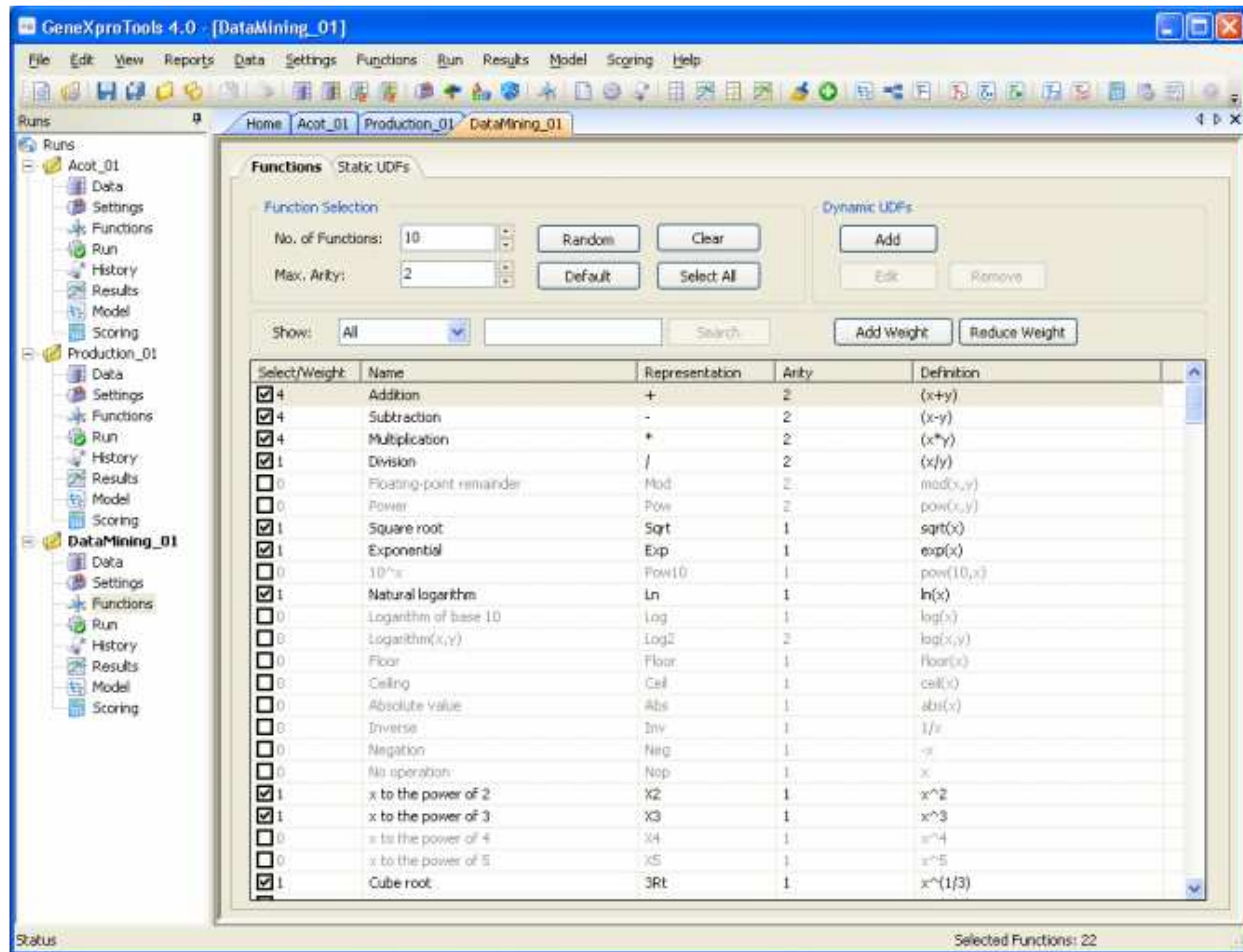


Working with Function Finding Tools

Available Functions

In the Functions Panel you can select and create:

- [Mathematical Functions](#) (including Dynamic UDFs or DDFs).
- [Static UDFs](#).

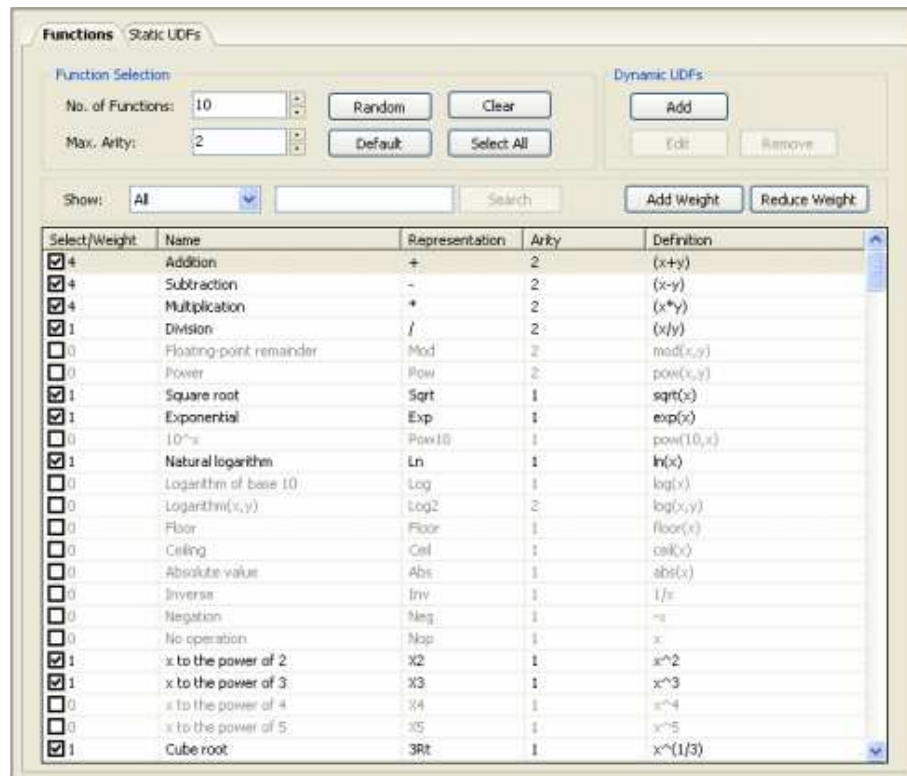


Working with Function Finding Tools

Mathematical Functions and Dynamic UDFs

GeneXproTools 4.0 offers a total of 279 **built-in mathematical functions**, including 186 different IF THEN ELSE rules, that can be used for designing nonlinear regression models. This wide set of mathematical functions allows the evolution of complex and rigorous models quickly built with the most appropriate functions.

The Function Selection Tool of GeneXproTools 4.0 helps you selecting different function sets very quickly through the combination of the Show options with the Random/Default/Clear/Select All buttons plus the Increase/Reduce Weight buttons in the [Functions Panel](#). Furthermore, GeneXproTools automatically balances your function set with the number of independent variables in your data, so now you just have to select the right functions for your problem and then choose their relative proportions by adjusting their weights.



Despite the wide set of GeneXproTools built-in mathematical functions, some users sometimes want to model with different ones. GeneXproTools 4.0 gives the user the possibility of creating custom tailored functions ([Dynamic UDFs](#) or DDFs) and evolve models with them. A note of caution though: the use of DDFs slows considerably the evolutionary process and therefore should be used with moderation.

Below are listed all the 279 **built-in mathematical functions** available in GeneXproTools 4.0, including their representation in [Karva Notation](#) (shown in red) and their mathematical definition (for more details about their implementations, see the [built-in grammars](#) of GeneXproTools):

Basic arithmetic functions:

- Addition (+): $(x+y)$
- Subtraction (-): $(x-y)$
- Multiplication (*): $(x*y)$
- Division (/): (x/y)
- Addition with 3 inputs (Add3): $(x+y+z)$
- Subtraction with 3 inputs (Sub3): $(x-y-z)$
- Multiplication with 3 inputs (Mul3): $(x*y*z)$
- Division with 3 inputs (Div3): $(x/y/z)$
- Addition with 4 inputs (Add4): $(a+b+c+d)$
- Subtraction with 4 inputs (Sub4): $(a-b-c-d)$
- Multiplication with 4 inputs (Mul4): $(a*b*c*d)$
- Division with 4 inputs (Div4): $(a/b/c/d)$

Logarithmic functions:

- Natural logarithm (Ln): $\ln(x)$
- Logarithm of base 10 (Log): $\log(x)$
- Logarithm(x,y) (Log2): $\log(x,y)$

Exponential and power functions:

- Square root (**Sqrt**): $\text{sqrt}(x)$
- Exponential (**Exp**): $\text{exp}(x)$
- Power (**Pow**): x^y
- 10^x (**Pow10**): 10^x
- x^2 (**X2**): x^2
- x^3 (**X3**): x^3
- x^4 (**X4**): x^4
- x^5 (**X5**): x^5
- Cube root (**3Rt**): $x^{(1/3)}$
- Quartic root (**4Rt**): $x^{(1/4)}$
- Quintic root (**5Rt**): $x^{(1/5)}$
- Logistic(x) (**Logi**): $\text{logistic}(x)$
- Logistic(x,y) (**Logi2**): $\text{logistic}(x,y)$
- Logistic(x,y,z) (**Logi3**): $\text{logistic}(x,y,z)$
- Logistic(a,b,c,d) (**Logi4**): $\text{logistic}(a,b,c,d)$
- Gaussian(x) (**Gau**): $\text{gaussian}(x)$
- Gaussian(x,y) (**Gau2**): $\text{gaussian}(x,y)$
- Gaussian(x,y,z) (**Gau3**): $\text{gaussian}(x,y,z)$
- Gaussian(a,b,c,d) (**Gau4**): $\text{gaussian}(a,b,c,d)$

Rounding and other simple functions:

- Floor (**Floor**): $\text{floor}(x)$
- Ceiling (**Ceil**): $\text{ceil}(x)$
- Floating-point remainder (**Mod**): $\text{mod}(x,y)$
- Absolute value (**Abs**): $\text{abs}(x)$
- Inverse (**Inv**): $1/x$
- Negation (**Neg**): $-x$
- Complement (**NOT**): $(1-x)$
- No operation (**Nop**): x

Minimum, maximum, and average functions:

- Minimum of 2 inputs (**Min2**): $\text{min}(x,y)$
- Minimum of 3 inputs (**Min3**): $\text{min}(x,y,z)$
- Minimum of 4 inputs (**Min4**): $\text{min}(a,b,c,d)$
- Maximum of 2 inputs (**Max2**): $\text{max}(x,y)$
- Maximum of 3 inputs (**Max3**): $\text{max}(x,y,z)$
- Maximum of 4 inputs (**Max4**): $\text{max}(a,b,c,d)$
- Average of 2 inputs (**Avg2**): $\text{avg}(x,y)$
- Average of 3 inputs (**Avg3**): $\text{avg}(x,y,z)$
- Average of 4 inputs (**Avg4**): $\text{avg}(a,b,c,d)$

Constant functions:

- Constant zero function (**Zero**): $0(x) = 0$
- Constant one function (**One**): $1(x) = 1$
- $0(x,y)$ (**Zero2**): $0(x,y) = 0$
- $1(x,y)$ (**One2**): $1(x,y) = 1$
- Number π (**Pi**): $\text{Pi}(x) = 3.141592653589793$
- Euler's number (**E**): $E(x) = 2.718281828459045$

Trigonometric functions (x is in radians):

- Sine (**Sin**): $\text{sin}(x)$
- Cosine (**Cos**): $\text{cos}(x)$

- Tangent (**Tan**): $\tan(x)$
- Cosecant (**Csc**): $\csc(x)$
- Secant (**Sec**): $\sec(x)$
- Cotangent (**Cot**): $\cot(x)$

Inverse trigonometric functions:

- Arcsine (**Asin**): $\arcsin(x)$
- Arccosine (**Acos**): $\arccos(x)$
- Arctangent (**Atan**): $\arctan(x)$
- Arccosecant (**Acsc**): $\operatorname{arccsc}(x)$
- Arcsecant (**Asec**): $\operatorname{arcsec}(x)$
- Arccotangent (**Acot**): $\operatorname{arccot}(x)$

Hyperbolic functions:

- Hyperbolic sine (**Sinh**): $\sinh(x)$
- Hyperbolic cosine (**Cosh**): $\cosh(x)$
- Hyperbolic tangent (**Tanh**): $\tanh(x)$
- Hyperbolic cosecant (**Csch**): $\operatorname{csch}(x)$
- Hyperbolic secant (**Sech**): $\operatorname{sech}(x)$
- Hyperbolic cotangent (**Coth**): $\operatorname{coth}(x)$

Inverse hyperbolic functions:

- Inverse hyperbolic sine (**Asinh**): $\operatorname{arcsinh}(x)$
- Inverse hyperbolic cosine (**Acosh**): $\operatorname{arccosh}(x)$
- Inverse hyperbolic tangent (**Atanh**): $\operatorname{arctanh}(x)$
- Inverse hyperbolic cosecant (**Acsch**): $\operatorname{arccsch}(x)$
- Inverse hyperbolic secant (**Asech**): $\operatorname{arcsech}(x)$
- Inverse hyperbolic cotangent (**Acoth**): $\operatorname{arccoth}(x)$

Comparison 0/1 functions of two arguments (OR series):

- OR1 (**OR1**): if $x < 0$ OR $y < 0$, then 1, else 0
- OR2 (**OR2**): if $x \geq 0$ OR $y \geq 0$, then 1, else 0
- OR3 (**OR3**): if $x \leq 0$ OR $y \leq 0$, then 1, else 0
- OR4 (**OR4**): if $x < 1$ OR $y < 1$, then 1, else 0
- OR5 (**OR5**): if $x \geq 1$ OR $y \geq 1$, then 1, else 0
- OR6 (**OR6**): if $x \leq 1$ OR $y \leq 1$, then 1, else 0

Comparison 0/1 functions of two arguments (AND series):

- AND1 (**AND1**): if $x < 0$ AND $y < 0$, then 1, else 0
- AND2 (**AND2**): if $x \geq 0$ AND $y \geq 0$, then 1, else 0
- AND3 (**AND3**): if $x \leq 0$ AND $y \leq 0$, then 1, else 0
- AND4 (**AND4**): if $x < 1$ AND $y < 1$, then 1, else 0
- AND5 (**AND5**): if $x \geq 1$ AND $y \geq 1$, then 1, else 0
- AND6 (**AND6**): if $x \leq 1$ AND $y \leq 1$, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series A):

- Less Than with 2 inputs (A) (**LT2A**): if $x < y$, then x , else y
- Greater Than with 2 inputs (A) (**GT2A**): if $x > y$, then x , else y
- Less Or Equal To with 2 inputs (A) (**LOE2A**): if $x \leq y$, then x , else y
- Greater Or Equal To with 2 inputs (A) (**GOE2A**): if $x \geq y$, then x , else y
- Equal To with 2 inputs (A) (**ET2A**): if $x = y$, then x , else y

- Not Equal To with 2 inputs (A) (**NET2A**): if $x \neq y$, then x , else y

Comparison 0/1 IF THEN ELSE functions of two arguments (series B):

- Less Than with 2 inputs (B) (**LT2B**): if $x < y$, then 1, else 0
- Greater Than with 2 inputs (B) (**GT2B**): if $x > y$, then 1, else 0
- Less Or Equal To with 2 inputs (B) (**LOE2B**): if $x \leq y$, then 1, else 0
- Greater Or Equal To with 2 inputs (B) (**GOE2B**): if $x \geq y$, then 1, else 0
- Equal To with 2 inputs (B) (**ET2B**): if $x = y$, then 1, else 0
- Not Equal To with 2 inputs (B) (**NET2B**): if $x \neq y$, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series C):

- Less Than with 2 inputs (C) (**LT2C**): if $x < y$, then $(x+y)$, else $(x-y)$
- Greater Than with 2 inputs (C) (**GT2C**): if $x > y$, then $(x+y)$, else $(x-y)$
- Less Or Equal To with 2 inputs (C) (**LOE2C**): if $x \leq y$, then $(x+y)$, else $(x-y)$
- Greater Or Equal To with 2 inputs (C) (**GOE2C**): if $x \geq y$, then $(x+y)$, else $(x-y)$
- Equal To with 2 inputs (C) (**ET2C**): if $x = y$, then $(x+y)$, else $(x-y)$
- Not Equal To with 2 inputs (C) (**NET2C**): if $x \neq y$, then $(x+y)$, else $(x-y)$

Comparison IF THEN ELSE functions of two arguments (series D):

- Less Than with 2 inputs (D) (**LT2D**): if $x < y$, then $(x*y)$, else (x/y)
- Greater Than with 2 inputs (D) (**GT2D**): if $x > y$, then $(x*y)$, else (x/y)
- Less Or Equal To with 2 inputs (D) (**LOE2D**): if $x \leq y$, then $(x*y)$, else (x/y)
- Greater Or Equal To with 2 inputs (D) (**GOE2D**): if $x \geq y$, then $(x*y)$, else (x/y)
- Equal To with 2 inputs (D) (**ET2D**): if $x = y$, then $(x*y)$, else (x/y)
- Not Equal To with 2 inputs (D) (**NET2D**): if $x \neq y$, then $(x*y)$, else (x/y)

Comparison IF THEN ELSE functions of two arguments (series E):

- Less Than with 2 inputs (E) (**LT2E**): if $x < y$, then $(x+y)$, else $(x*y)$
- Greater Than with 2 inputs (E) (**GT2E**): if $x > y$, then $(x+y)$, else $(x*y)$
- Less Or Equal To with 2 inputs (E) (**LOE2E**): if $x \leq y$, then $(x+y)$, else $(x*y)$
- Greater Or Equal To with 2 inputs (E) (**GOE2E**): if $x \geq y$, then $(x+y)$, else $(x*y)$
- Equal To with 2 inputs (E) (**ET2E**): if $x = y$, then $(x+y)$, else $(x*y)$
- Not Equal To with 2 inputs (E) (**NET2E**): if $x \neq y$, then $(x+y)$, else $(x*y)$

Comparison IF THEN ELSE functions of two arguments (series F):

- Less Than with 2 inputs (F) (**LT2F**): if $x < y$, then $(x+y)$, else $\sin(x*y)$
- Greater Than with 2 inputs (F) (**GT2F**): if $x > y$, then $(x+y)$, else $\sin(x*y)$
- Less Or Equal To with 2 inputs (F) (**LOE2F**): if $x \leq y$, then $(x+y)$, else $\sin(x*y)$
- Greater Or Equal To with 2 inputs (F) (**GOE2F**): if $x \geq y$, then $(x+y)$, else $\sin(x*y)$
- Equal To with 2 inputs (F) (**ET2F**): if $x = y$, then $(x+y)$, else $\sin(x*y)$
- Not Equal To with 2 inputs (F) (**NET2F**): if $x \neq y$, then $(x+y)$, else $\sin(x*y)$

Comparison IF THEN ELSE functions of two arguments (series G):

- Less Than with 2 inputs (G) (**LT2G**): if $x < y$, then $(x+y)$, else $\text{atan}(x*y)$
- Greater Than with 2 inputs (G) (**GT2G**): if $x > y$, then $(x+y)$, else $\text{atan}(x*y)$
- Less Or Equal To with 2 inputs (G) (**LOE2G**): if $x \leq y$, then $(x+y)$, else $\text{atan}(x*y)$
- Greater Or Equal To with 2 inputs (G) (**GOE2G**): if $x \geq y$, then $(x+y)$, else $\text{atan}(x*y)$
- Equal To with 2 inputs (G) (**ET2G**): if $x = y$, then $(x+y)$, else $\text{atan}(x*y)$
- Not Equal To with 2 inputs (G) (**NET2G**): if $x \neq y$, then $(x+y)$, else $\text{atan}(x*y)$

Comparison IF THEN ELSE functions of three arguments (series A):

- Less Than with 3 inputs (A) (**LT3A**): if $x < 0$, then y , else z
- Greater Than with 3 inputs (A) (**GT3A**): if $x > 0$, then y , else z
- Less Or Equal To with 3 inputs (A) (**LOE3A**): if $x \leq 0$, then y , else z
- Greater Or Equal To with 3 inputs (A) (**GOE3A**): if $x \geq 0$, then y , else z
- Equal To with 3 inputs (A) (**ET3A**): if $x = 0$, then y , else z
- Not Equal To with 3 inputs (A) (**NET3A**): if $x \neq 0$, then y , else z

Comparison IF THEN ELSE functions of three arguments (series B):

- Less Than with 3 inputs (B) (**LT3B**): if $(x+y) < z$, then $(x+y)$, else z
- Greater Than with 3 inputs (B) (**GT3B**): if $(x+y) > z$, then $(x+y)$, else z
- Less Or Equal To with 3 inputs (B) (**LOE3B**): if $(x+y) \leq z$, then $(x+y)$, else z
- Greater Or Equal To with 3 inputs (B) (**GOE3B**): if $(x+y) \geq z$, then $(x+y)$, else z
- Equal To with 3 inputs (B) (**ET3B**): if $(x+y) = z$, then $(x+y)$, else z
- Not Equal To with 3 inputs (B) (**NET3B**): if $(x+y) \neq z$, then $(x+y)$, else z

Comparison IF THEN ELSE functions of three arguments (series C):

- Less Than with 3 inputs (C) (**LT3C**): if $(x+y) < z$, then $(x+y)$, else $(x+z)$
- Greater Than with 3 inputs (C) (**GT3C**): if $(x+y) > z$, then $(x+y)$, else $(x+z)$
- Less Or Equal To with 3 inputs (C) (**LOE3C**): if $(x+y) \leq z$, then $(x+y)$, else $(x+z)$
- Greater Or Equal To with 3 inputs (C) (**GOE3C**): if $(x+y) \geq z$, then $(x+y)$, else $(x+z)$
- Equal To with 3 inputs (C) (**ET3C**): if $(x+y) = z$, then $(x+y)$, else $(x+z)$
- Not Equal To with 3 inputs (C) (**NET3C**): if $(x+y) \neq z$, then $(x+y)$, else $(x+z)$

Comparison IF THEN ELSE functions of three arguments (series D):

- Less Than with 3 inputs (D) (**LT3D**): if $(x+y) < z$, then $(x+y)$, else $(x-z)$
- Greater Than with 3 inputs (D) (**GT3D**): if $(x+y) > z$, then $(x+y)$, else $(x-z)$
- Less Or Equal To with 3 inputs (D) (**LOE3D**): if $(x+y) \leq z$, then $(x+y)$, else $(x-z)$
- Greater Or Equal To with 3 inputs (D) (**GOE3D**): if $(x+y) \geq z$, then $(x+y)$, else $(x-z)$
- Equal To with 3 inputs (D) (**ET3D**): if $(x+y) = z$, then $(x+y)$, else $(x-z)$
- Not Equal To with 3 inputs (D) (**NET3D**): if $(x+y) \neq z$, then $(x+y)$, else $(x-z)$

Comparison IF THEN ELSE functions of three arguments (series E):

- Less Than with 3 inputs (E) (**LT3E**): if $(x+y) < z$, then $(x+y)$, else $(x*z)$
- Greater Than with 3 inputs (E) (**GT3E**): if $(x+y) > z$, then $(x+y)$, else $(x*z)$
- Less Or Equal To with 3 inputs (E) (**LOE3E**): if $(x+y) \leq z$, then $(x+y)$, else $(x*z)$
- Greater Or Equal To with 3 inputs (E) (**GOE3E**): if $(x+y) \geq z$, then $(x+y)$, else $(x*z)$
- Equal To with 3 inputs (E) (**ET3E**): if $(x+y) = z$, then $(x+y)$, else $(x*z)$
- Not Equal To with 3 inputs (E) (**NET3E**): if $(x+y) \neq z$, then $(x+y)$, else $(x*z)$

Comparison IF THEN ELSE functions of three arguments (series F):

- Less Than with 3 inputs (F) (**LT3F**): if $(x+y) < z$, then $(x+y)$, else (x/z)
- Greater Than with 3 inputs (F) (**GT3F**): if $(x+y) > z$, then $(x+y)$, else (x/z)
- Less Or Equal To with 3 inputs (F) (**LOE3F**): if $(x+y) \leq z$, then $(x+y)$, else (x/z)
- Greater Or Equal To with 3 inputs (F) (**GOE3F**): if $(x+y) \geq z$, then $(x+y)$, else (x/z)
- Equal To with 3 inputs (F) (**ET3F**): if $(x+y) = z$, then $(x+y)$, else (x/z)
- Not Equal To with 3 inputs (F) (**NET3F**): if $(x+y) \neq z$, then $(x+y)$, else (x/z)

Comparison IF THEN ELSE functions of three arguments (series G):

- Less Than with 3 inputs (G) (**LT3G**): if $(x+y) < z$, then $(x*y)$, else $(x+z)$
- Greater Than with 3 inputs (G) (**GT3G**): if $(x+y) > z$, then $(x*y)$, else $(x+z)$
- Less Or Equal To with 3 inputs (G) (**LOE3G**): if $(x+y) \leq z$, then $(x*y)$, else $(x+z)$

- Greater Or Equal To with 3 inputs (G) (**GOE3G**): if $(x+y) \geq z$, then $(x*y)$, else $(x+z)$
- Equal To with 3 inputs (G) (**ET3G**): if $(x+y) = z$, then $(x*y)$, else $(x+z)$
- Not Equal To with 3 inputs (G) (**NET3G**): if $(x+y) \neq z$, then $(x*y)$, else $(x+z)$

Comparison IF THEN ELSE functions of three arguments (series H):

- Less Than with 3 inputs (H) (**LT3H**): if $(x+y) < z$, then $(x*y)$, else $(x-z)$
- Greater Than with 3 inputs (H) (**GT3H**): if $(x+y) > z$, then $(x*y)$, else $(x-z)$
- Less Or Equal To with 3 inputs (H) (**LOE3H**): if $(x+y) \leq z$, then $(x*y)$, else $(x-z)$
- Greater Or Equal To with 3 inputs (H) (**GOE3H**): if $(x+y) \geq z$, then $(x*y)$, else $(x-z)$
- Equal To with 3 inputs (H) (**ET3H**): if $(x+y) = z$, then $(x*y)$, else $(x-z)$
- Not Equal To with 3 inputs (H) (**NET3H**): if $(x+y) \neq z$, then $(x*y)$, else $(x-z)$

Comparison IF THEN ELSE functions of three arguments (series I):

- Less Than with 3 inputs (I) (**LT3I**): if $(x+y) < z$, then $(x*y)$, else $(x*z)$
- Greater Than with 3 inputs (I) (**GT3I**): if $(x+y) > z$, then $(x*y)$, else $(x*z)$
- Less Or Equal To with 3 inputs (I) (**LOE3I**): if $(x+y) \leq z$, then $(x*y)$, else $(x*z)$
- Greater Or Equal To with 3 inputs (I) (**GOE3I**): if $(x+y) \geq z$, then $(x*y)$, else $(x*z)$
- Equal To with 3 inputs (I) (**ET3I**): if $(x+y) = z$, then $(x*y)$, else $(x*z)$
- Not Equal To with 3 inputs (I) (**NET3I**): if $(x+y) \neq z$, then $(x*y)$, else $(x*z)$

Comparison IF THEN ELSE functions of three arguments (series J):

- Less Than with 3 inputs (J) (**LT3J**): if $(x+y) < z$, then $(x*y)$, else (x/z)
- Greater Than with 3 inputs (J) (**GT3J**): if $(x+y) > z$, then $(x*y)$, else (x/z)
- Less Or Equal To with 3 inputs (J) (**LOE3J**): if $(x+y) \leq z$, then $(x*y)$, else (x/z)
- Greater Or Equal To with 3 inputs (J) (**GOE3J**): if $(x+y) \geq z$, then $(x*y)$, else (x/z)
- Equal To with 3 inputs (J) (**ET3J**): if $(x+y) = z$, then $(x*y)$, else (x/z)
- Not Equal To with 3 inputs (J) (**NET3J**): if $(x+y) \neq z$, then $(x*y)$, else (x/z)

Comparison IF THEN ELSE functions of three arguments (series K):

- Less Than with 3 inputs (K) (**LT3K**): if $(x+y) < z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Greater Than with 3 inputs (K) (**GT3K**): if $(x+y) > z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Less Or Equal To with 3 inputs (K) (**LOE3K**): if $(x+y) \leq z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Greater Or Equal To with 3 inputs (K) (**GOE3K**): if $(x+y) \geq z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Equal To with 3 inputs (K) (**ET3K**): if $(x+y) = z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Not Equal To with 3 inputs (K) (**NET3K**): if $(x+y) \neq z$, then $(x+y+z)$, else $\sin(x*y*z)$

Comparison IF THEN ELSE functions of three arguments (series L):

- Less Than with 3 inputs (L) (**LT3L**): if $(x+y) < z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Greater Than with 3 inputs (L) (**GT3L**): if $(x+y) > z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Less Or Equal To with 3 inputs (L) (**LOE3L**): if $(x+y) \leq z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Greater Or Equal To with 3 inputs (L) (**GOE3L**): if $(x+y) \geq z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Equal To with 3 inputs (L) (**ET3L**): if $(x+y) = z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Not Equal To with 3 inputs (L) (**NET3L**): if $(x+y) \neq z$, then $(x+y+z)$, else $\tan(x*y*z)$

Comparison IF THEN ELSE functions of four arguments (series A):

- Less Than with 4 inputs (A) (**LT4A**): if $a < b$, then c , else d
- Greater Than with 4 inputs (A) (**GT4A**): if $a > b$, then c , else d
- Less Or Equal To with 4 inputs (A) (**LOE4A**): if $a \leq b$, then c , else d
- Greater Or Equal To with 4 inputs (A) (**GOE4A**): if $a \geq b$, then c , else d
- Equal To with 4 inputs (A) (**ET4A**): if $a = b$, then c , else d
- Not Equal To with 4 inputs (A) (**NET4A**): if $a \neq b$, then c , else d

Comparison IF THEN ELSE functions of four arguments (series B):

- Less Than with 4 inputs (B) (LT4B): if $(a+b) < (c+d)$, then c , else d
- Greater Than with 4 inputs (B) (GT4B): if $(a+b) > (c+d)$, then c , else d
- Less Or Equal To with 4 inputs (B) (LOE4B): if $(a+b) \leq (c+d)$, then c , else d
- Greater Or Equal To with 4 inputs (B) (GOE4B): if $(a+b) \geq (c+d)$, then c , else d
- Equal To with 4 inputs (B) (ET4B): if $(a+b) = (c+d)$, then c , else d
- Not Equal To with 4 inputs (B) (NET4B): if $(a+b) \neq (c+d)$, then c , else d

Comparison IF THEN ELSE functions of four arguments (series C):

- Less Than with 4 inputs (C) (LT4C): if $(a+b) < (c+d)$, then $(a+b)$, else $(c+d)$
- Greater Than with 4 inputs (C) (GT4C): if $(a+b) > (c+d)$, then $(a+b)$, else $(c+d)$
- Less Or Equal To with 4 inputs (C) (LOE4C): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c+d)$
- Greater Or Equal To with 4 inputs (C) (GOE4C): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c+d)$
- Equal To with 4 inputs (C) (ET4C): if $(a+b) = (c+d)$, then $(a+b)$, else $(c+d)$
- Not Equal To with 4 inputs (C) (NET4C): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c+d)$

Comparison IF THEN ELSE functions of four arguments (series D):

- Less Than with 4 inputs (D) (LT4D): if $(a+b) < (c+d)$, then $(a+b)$, else $(c-d)$
- Greater Than with 4 inputs (D) (GT4D): if $(a+b) > (c+d)$, then $(a+b)$, else $(c-d)$
- Less Or Equal To with 4 inputs (D) (LOE4D): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c-d)$
- Greater Or Equal To with 4 inputs (D) (GOE4D): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c-d)$
- Equal To with 4 inputs (D) (ET4D): if $(a+b) = (c+d)$, then $(a+b)$, else $(c-d)$
- Not Equal To with 4 inputs (D) (NET4D): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c-d)$

Comparison IF THEN ELSE functions of four arguments (series E):

- Less Than with 4 inputs (E) (LT4E): if $(a+b) < (c+d)$, then $(a+b)$, else $(c*d)$
- Greater Than with 4 inputs (E) (GT4E): if $(a+b) > (c+d)$, then $(a+b)$, else $(c*d)$
- Less Or Equal To with 4 inputs (E) (LOE4E): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c*d)$
- Greater Or Equal To with 4 inputs (E) (GOE4E): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c*d)$
- Equal To with 4 inputs (E) (ET4E): if $(a+b) = (c+d)$, then $(a+b)$, else $(c*d)$
- Not Equal To with 4 inputs (E) (NET4E): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c*d)$

Comparison IF THEN ELSE functions of four arguments (series F):

- Less Than with 4 inputs (F) (LT4F): if $(a+b) < (c+d)$, then $(a+b)$, else (c/d)
- Greater Than with 4 inputs (F) (GT4F): if $(a+b) > (c+d)$, then $(a+b)$, else (c/d)
- Less Or Equal To with 4 inputs (F) (LOE4F): if $(a+b) \leq (c+d)$, then $(a+b)$, else (c/d)
- Greater Or Equal To with 4 inputs (F) (GOE4F): if $(a+b) \geq (c+d)$, then $(a+b)$, else (c/d)
- Equal To with 4 inputs (F) (ET4F): if $(a+b) = (c+d)$, then $(a+b)$, else (c/d)
- Not Equal To with 4 inputs (F) (NET4F): if $(a+b) \neq (c+d)$, then $(a+b)$, else (c/d)

Comparison IF THEN ELSE functions of four arguments (series G):

- Less Than with 4 inputs (G) (LT4G): if $(a+b) < (c+d)$, then $(a*b)$, else $(c+d)$
- Greater Than with 4 inputs (G) (GT4G): if $(a+b) > (c+d)$, then $(a*b)$, else $(c+d)$
- Less Or Equal To with 4 inputs (G) (LOE4G): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c+d)$
- Greater Or Equal To with 4 inputs (G) (GOE4G): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c+d)$
- Equal To with 4 inputs (G) (ET4G): if $(a+b) = (c+d)$, then $(a*b)$, else $(c+d)$
- Not Equal To with 4 inputs (G) (NET4G): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c+d)$

Comparison IF THEN ELSE functions of four arguments (series H):

- Less Than with 4 inputs (H) (LT4H): if $(a+b) < (c+d)$, then $(a*b)$, else $(c-d)$

- Greater Than with 4 inputs (H) (**GT4H**): if $(a+b) > (c+d)$, then $(a*b)$, else $(c-d)$
- Less Or Equal To with 4 inputs (H) (**LOE4H**): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c-d)$
- Greater Or Equal To with 4 inputs (H) (**GOE4H**): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c-d)$
- Equal To with 4 inputs (H) (**ET4H**): if $(a+b) = (c+d)$, then $(a*b)$, else $(c-d)$
- Not Equal To with 4 inputs (H) (**NET4H**): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c-d)$

Comparison IF THEN ELSE functions of four arguments (series I):

- Less Than with 4 inputs (I) (**LT4I**): if $(a+b) < (c+d)$, then $(a*b)$, else $(c*d)$
- Greater Than with 4 inputs (I) (**GT4I**): if $(a+b) > (c+d)$, then $(a*b)$, else $(c*d)$
- Less Or Equal To with 4 inputs (I) (**LOE4I**): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c*d)$
- Greater Or Equal To with 4 inputs (I) (**GOE4I**): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c*d)$
- Equal To with 4 inputs (I) (**ET4I**): if $(a+b) = (c+d)$, then $(a*b)$, else $(c*d)$
- Not Equal To with 4 inputs (I) (**NET4I**): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c*d)$

Comparison IF THEN ELSE functions of four arguments (series J):

- Less Than with 4 inputs (J) (**LT4J**): if $(a+b) < (c+d)$, then $(a*b)$, else (c/d)
- Greater Than with 4 inputs (J) (**GT4J**): if $(a+b) > (c+d)$, then $(a*b)$, else (c/d)
- Less Or Equal To with 4 inputs (J) (**LOE4J**): if $(a+b) \leq (c+d)$, then $(a*b)$, else (c/d)
- Greater Or Equal To with 4 inputs (J) (**GOE4J**): if $(a+b) \geq (c+d)$, then $(a*b)$, else (c/d)
- Equal To with 4 inputs (J) (**ET4J**): if $(a+b) = (c+d)$, then $(a*b)$, else (c/d)
- Not Equal To with 4 inputs (J) (**NET4J**): if $(a+b) \neq (c+d)$, then $(a*b)$, else (c/d)

Comparison IF THEN ELSE functions of four arguments (series K):

- Less Than with 4 inputs (K) (**LT4K**): if $(a+b) < (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Greater Than with 4 inputs (K) (**GT4K**): if $(a+b) > (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Less Or Equal To with 4 inputs (K) (**LOE4K**): if $(a+b) \leq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Greater Or Equal To with 4 inputs (K) (**GOE4K**): if $(a+b) \geq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Equal To with 4 inputs (K) (**ET4K**): if $(a+b) = (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Not Equal To with 4 inputs (K) (**NET4K**): if $(a+b) \neq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$

Comparison IF THEN ELSE functions of four arguments (series L):

- Less Than with 4 inputs (L) (**LT4L**): if $(a+b) < (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Greater Than with 4 inputs (L) (**GT4L**): if $(a+b) > (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Less Or Equal To with 4 inputs (L) (**LOE4L**): if $(a+b) \leq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Greater Or Equal To with 4 inputs (L) (**GOE4L**): if $(a+b) \geq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Equal To with 4 inputs (L) (**ET4L**): if $(a+b) = (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Not Equal To with 4 inputs (L) (**NET4L**): if $(a+b) \neq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$

By selecting the Functions Tab in the [Functions Panel](#), you have full access to the 279 built-in mathematical functions of GeneXproTools 4.0. Here is also the place where you can add **Dynamic UDFs** to your modeling kit.

Functions Static UDFs

Function Selection

No. of Functions: 10 Random Clear

Max. Arity: 1 Default Select All

Dynamic UDFs

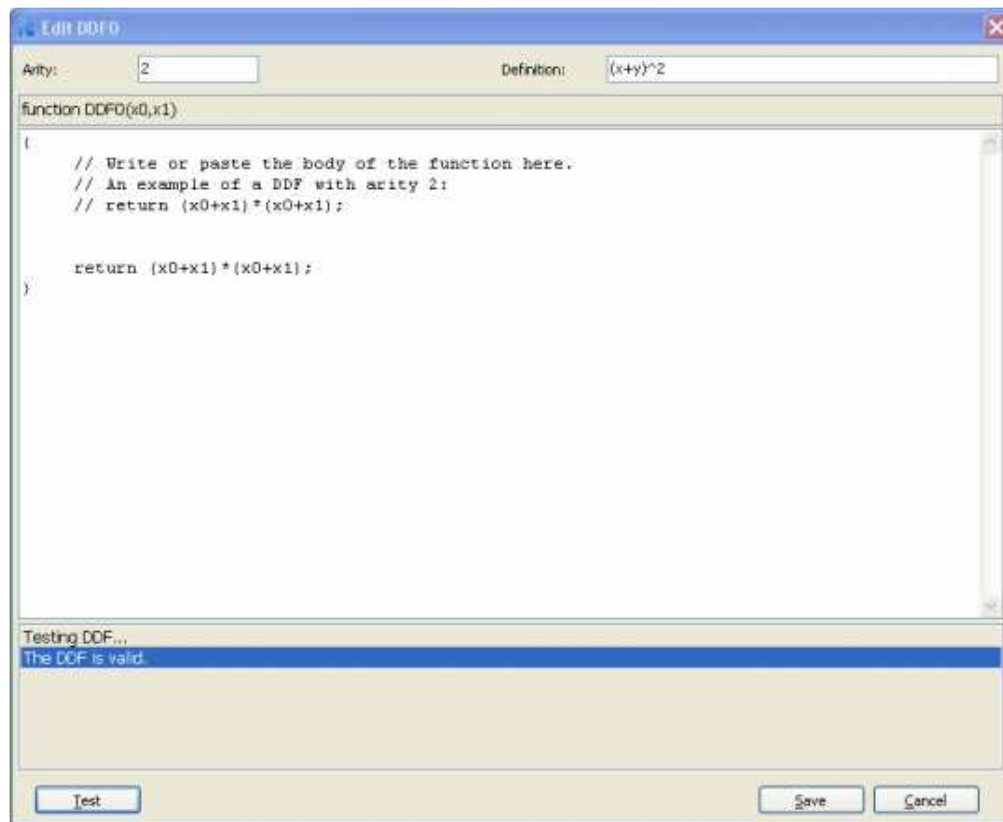
Add Edit Remove

Show: Selected Only Search Add Weight Reduce Weight

Select/Weight	Name	Representation	Arity	Definition
☒ 4	Addition	+	2	(x+y)
☒ 4	Subtraction	-	2	(x-y)
☒ 4	Multiplication	*	2	(x*y)
☒ 1	Division	/	2	(x/y)
☒ 1	Power	Pow	2	pow(x,y)
☒ 1	Square root	Sqrt	1	sqrt(x)
☒ 2	Exponential	Exp	1	exp(x)
☒ 1	Natural logarithm	Ln	1	ln(x)
☒ 1	x to the power of 2	X2	1	x^2
☒ 2	x to the power of 3	X3	1	x^3
☒ 1	x to the power of 5	X5	1	x^5
☒ 1	Cube root	3Rt	1	x^(1/3)
☒ 1	Quartic root	4Rt	1	x^(1/4)
☒ 1	Constant zero function	Zero	1	0(x) = 0
☒ 1	Sine	Sin	1	sin(x)
☒ 2	Cosine	Cos	1	cos(x)
☒ 1	Cotangent	Cot	1	cot(x)
☒ 1	Arctangent	Atan	1	arctan(x)
☒ 1	Arccosecant	Acsc	1	arcsc(x)
☒ 1	Arcsecant	Asec	1	arcsec(x)
☒ 1	Hyperbolic cosine	Cosh	1	cosh(x)
☒ 1	Dynamic UDF	DDF	2	My DDF

To select a function, just check the box on the left. By default, the weight of each function is 1, but you can increase the probability of a function being included in your models by increasing its weight in the Select/Weight column. GeneXproTools automatically balances your function set with the number of independent variables in your data, so now you just have to select the right functions for your problem and then choose their relative proportions by choosing their weights.

To add a **DDF** to your modeling kit, just click the Add button on the Dynamic UDFs frame and the **DDF Editor** appears.



By choosing the arity (minimum is 1 and maximum is 4) in the Arity box, the function header appears in the code window. Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be conveniently tested for compiling errors by pressing the Test button.

In the Definition box, you can write a brief description of the function for your future reference. The text you write there will appear in the Definition column.

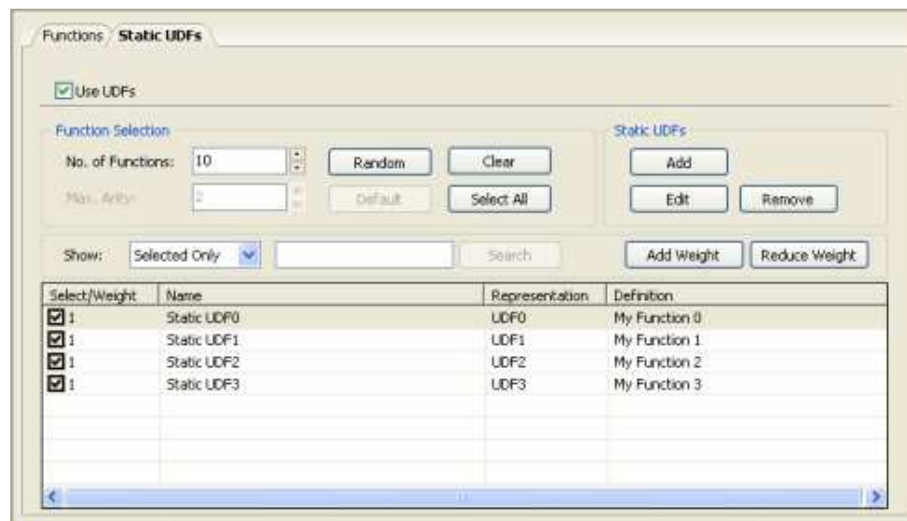
Dynamic UDFs are extremely powerful and interesting tools as they are treated exactly like the built-in functions of GeneXproTools and therefore can be used to model all kinds of relationships between variables or complex expressions. For instance, you can design a DDF so that it will model the sum of four expressions, that is, $DDF = (\text{expression 1}) + (\text{expression 2}) + (\text{expression 3}) + (\text{expression 4})$, where the value of each expression will depend on the context of the DDF in the expression tree. A note of caution, though, although extremely interesting, DDFs decrease considerably the speed of the algorithm and therefore we advise you to choose your functions from the wide set of GeneXproTools 4.0 built-in functions.



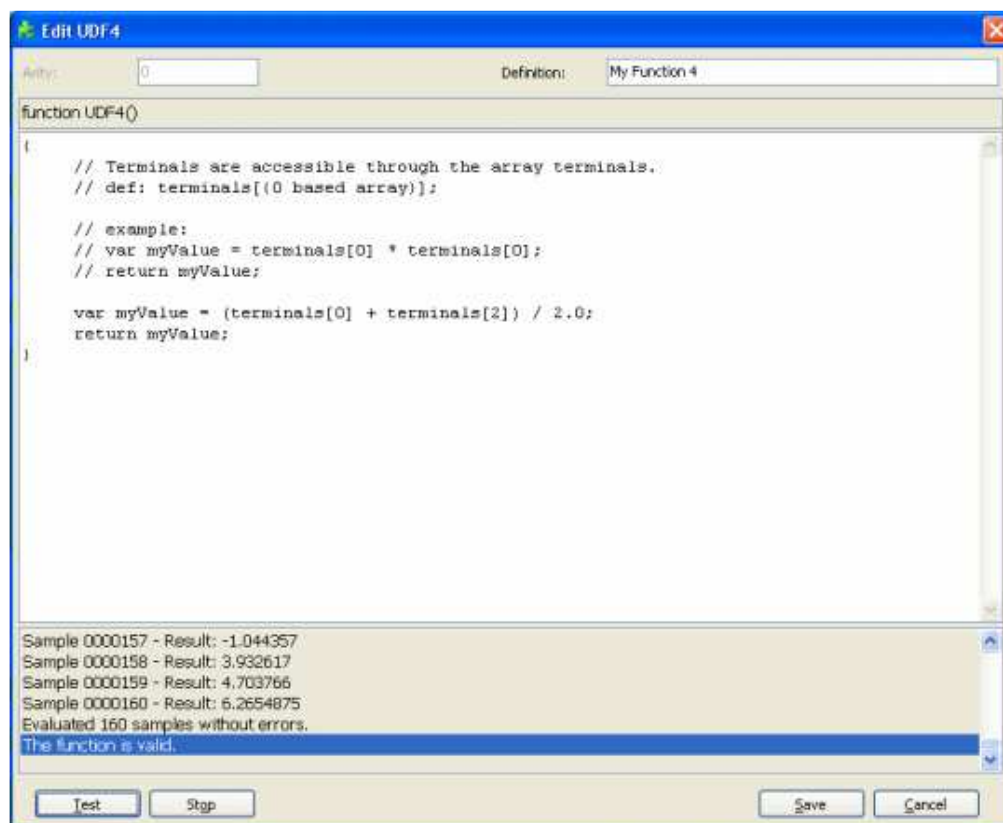
Working with Function Finding Tools

Static UDFs

In the [Functions Panel](#), by selecting the [Static UDFs Tab](#) and checking the Use UDFs box, the Static UDFs frame lights up and you can now add your own user defined functions.

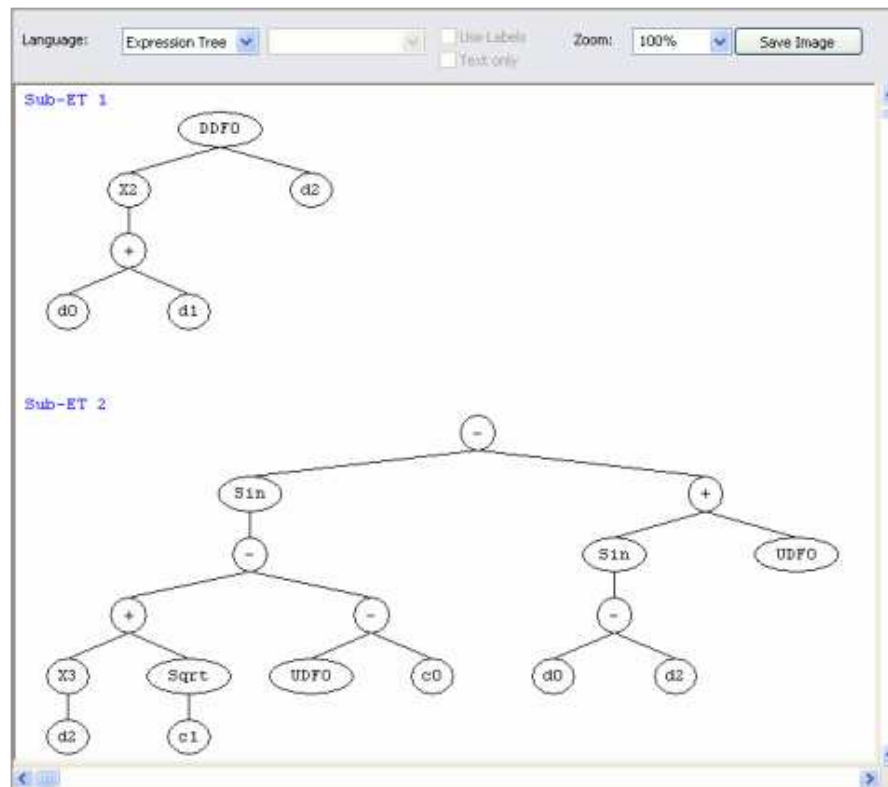


To add a **Static UDF** (UDF for short) to your modeling kit, just click the Add button on the Static UDFs frame and the **UDF Editor** appears with the function header already in place.



Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be rapidly tested for compiling errors by pressing the Test button.

You will notice that the Arity box is disabled and set to zero as all UDFs will occupy leaf nodes in the trees, that is, they behave exactly as terminals.



In the Definition box, you can write a brief description of the UDF for your future reference. The text you write there will appear in the Definition column of the Static UDFs Tab.

The use of UDFs is a very powerful tool as they allow you to explore inside knowledge about relatively simple relationships that exist between certain variables in your data. For instance, suppose you wanted to explore the fact that the sum of six of your variables is relevant, say, $d0 + d1 + d2 + d3 + d4 + d5$. So you design a UDF to express this relationship, for instance, $UDF1 = d0 + d1 + d2 + d3 + d4 + d5$, and use this new UDF as an extra building block in your modeling kit. The advantage of well designed UDFs is that they make the discovery of more complex models composed of several simpler models much easier.



Working with Function Finding Tools

Available Chromosomal Architectures

The [chromosome architecture](#) of your models include the **head size**, the **number of genes** and the **linking function**. You choose these parameters in the Settings Panel -> General Settings Tab.

General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 200 [Reset]

Testing Samples: 246 [Reset]

Number of Chromosomes: 30

Head Size: 7

Number of Genes: 3

Linking Function: Addition

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 5

☒ Save Run History

The Head Size determines the complexity of each term in your model. In the [heads of genes](#), the GeneXproTools learning algorithm tries out different arrangements of functions and terminals (variables and constants) in order to model your data. The plasticity of this architecture allows the discovery of a virtually infinite number of models of different sizes and shapes which are afterwards tested and selected during the learning process. The heads of genes are shown in blue in the compact, linear representation of your models in the Model Panel.

Language: Karva | Genes: | ☐ Use Labels | Zoom: 100% | ☐ Text only | Save Code

```
Exp. - .Aten. + / + .d2 / .Exp. + * + .d1 .d2 .c0 .d0 .d2 .c1 .d0 .d2 .c1 .d2 .d1 .d1 .d1
+
* .Cos .d1 / Exp. - - - Cos .c0 .c0 * .c0 .d1 .d0 .d0 .d0 .d1 .d0 .d2 .c1 .c0 .c0 .c1 .c1
+
* .Cos / c0 - X3 Cos. - + .c1 * + .c1 .d1 .c1 .d1 .d1 .d1 .d2 .c0 .c1 .d2 .d0 .c1 .c1
```

Numerical Constants:

Gene 1
c0 = 3.410004
c1 = 0.216186

Gene 2
c0 = 9.978333
c1 = 3.325409

Gene 3
c0 = 2.533783
c1 = 3.325409

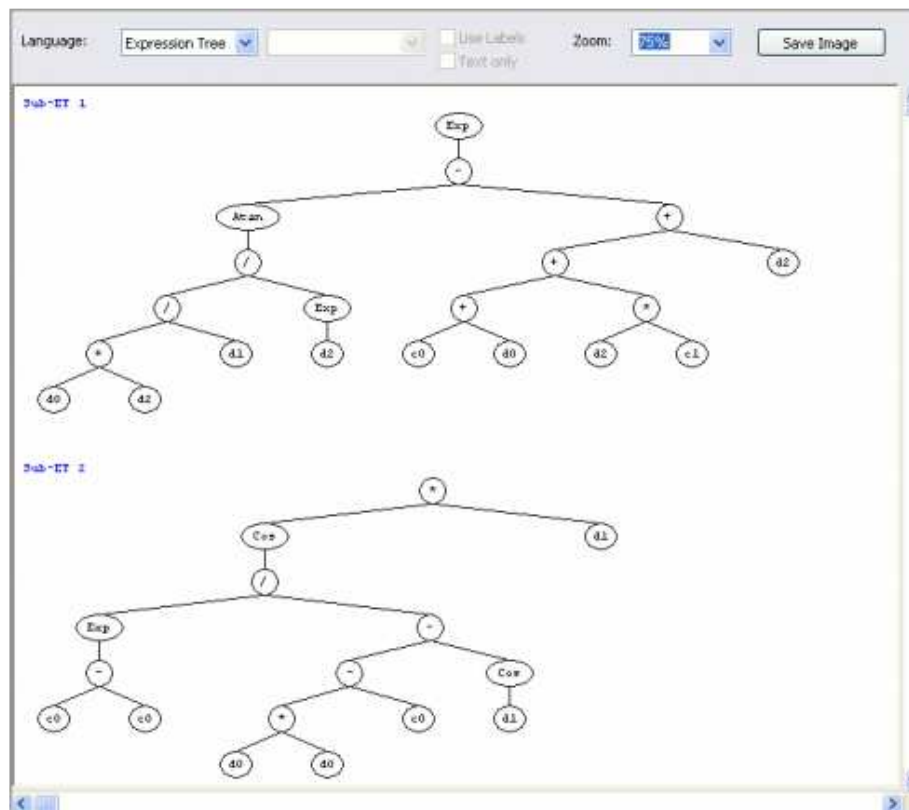
More specifically, the **head size** h of each gene determines the maximum width w and maximum depth d of the [sub-expression trees](#) encoded in the gene, which are given by the formulas:

$$w = (n - 1) * h + 1$$

$$d = ((h + 1) / m) * ((m + 1) / 2)$$

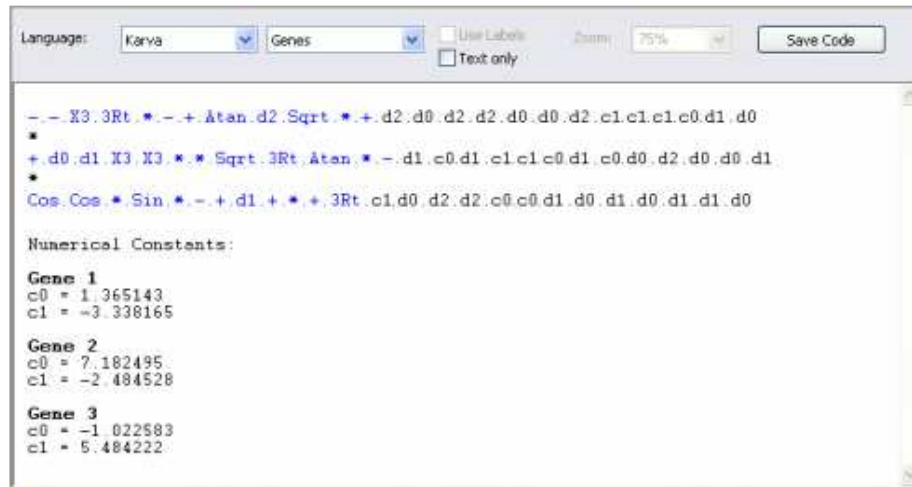
where m is minimum arity and n is maximum arity.

Thus, the GeneXproTools [learning algorithm](#) selects its models between these extreme cases, fine-tuning the ideal size and shape during the evolutionary process without human intervention.



The [number of genes](#) per chromosome is also an important parameter. It will determine the number of (complex) terms in your model as each gene codes for a different [parse tree](#) (sub-expression tree or sub-ET). Theoretically, one could just use a huge single gene in order to evolve very complex models. But the partition of the chromosome into simpler, more manageable units gives an edge to the learning process and more efficient and elegant models can be discovered using [multigenic chromosomes](#).

Whenever the number of genes is greater than one, you must also choose a suitable [linking function](#) for linking the mathematical terms encoded in each gene. GeneXproTools 4.0 allows you to choose **addition**, **subtraction**, **multiplication**, or **division** to link the sub-ETs. As expected, addition (and obviously subtraction and And) works very well for virtually all problems but sometimes one of the other linkers could be useful for searching different solution spaces and finding a very good, albeit unexpected model.



Working with Function Finding Tools

Fitness Functions

For Function Finding problems, in the Fitness Function Tab of the [Settings Panel](#) you have access to 36 [built-in fitness functions](#). Additionally, you can also design your own [custom fitness function](#) and explore the solution space with it. By choosing Custom in the Fitness Function box, the custom fitness editor is activated.

General Settings **Fitness Function** Genetic Operators Numerical Constants

Fitness Functions: **MSE** Selection Range: 100

☐ With Parsimony Pressure Precision: 0.01

Rounding Threshold: 0.1

Custom Fitness Function

Max. Training Fitness: 1000 Max. Testing Fitness: 1000 Timeout: 10000

```
// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals

// Your custom fitness function must return a value, for example:
// return fitness;
```

Test

You can design your own [custom fitness function](#) using the Custom Fitness Function window to write the code of your fitness function. The code for the custom fitness function must be in JavaScript and can be tested before evolving a model with it by pressing the Test button.

General Settings **Fitness Function** Genetic Operators Numerical Constants

Fitness Functions: **Custom** Selection Range: 100

☐ With Parsimony Pressure Precision: 0.01

Rounding Threshold: 0.1

Custom Fitness Function

Max. Training Fitness: 1000 Max. Testing Fitness: 1000 Timeout: 10000

```
// Below is an example of a correct fitness function:
// the Mean Squared Error (MSE) fitness function of GeneXproTools,
// for which the maximum fitness is equal to 1000:

// MEAN SQUARED ERROR
var nSamples = aParameters[0];
var fitness = 0.0;
var modelMinusTargetSquared = 0.0;
var MSE = 0.0;
for (var i=0; i<nSamples; i++)
{
    var temp1 = 0.0;
    temp1 = aOutputModel[i] - aOutputTarget[i];
    temp1 *= temp1;
    modelMinusTargetSquared += temp1;
}

return fitness;
```

Correct function.

Test

The 36 built-in fitness functions of GeneXproTools 4.0 for Function Finding:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)
- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)
- [R-square](#)
- [Correlation Coefficient](#)
- And all the above with [parsimony pressure](#)

The kind of fitness function you choose will depend most probably on the statistical function you are most familiar with. And although there is nothing wrong with this for all of them can accomplish an efficient evolution, you might want to try different fitness functions for they travel the fitness landscape differently: some of them very straightforwardly in their pursuits while others choose less obvious paths.



Working with Function Finding Tools

Available Algorithms

GeneXproTools 4.0 uses two different [learning algorithms](#) for Function Finding problems. The first – the basic gene expression algorithm or simply [Gene Expression Programming](#) (GEP) – does not support the direct manipulation of random numerical constants, whereas the second – [GEP with Random Numerical Constants](#) or [GEP-RNC](#) for short – has a facility for handling them directly. So, these two algorithms search the solution landscape differently and therefore you might wish to try them both on your problems.

The kinds of models these algorithms produce are quite different and, when both of them perform equally well on the problem at hand, you still might prefer one to the other. But there are cases, however, where numerical constants are crucial for an efficient modeling and, therefore, the second algorithm is the default in GeneXproTools 4.0. You activate this algorithm in the Settings Panel -> Numerical Constants by checking the Use Random Numerical Constants box.

The [GEP-RNC algorithm](#) is slightly more complex than [GEP](#) as it uses an additional gene domain (Dc) for encoding the random numerical constants. Consequently, this algorithm comes equipped with an additional set of genetic operators ([RNC mutation](#), [Dc mutation](#), [Dc inversion](#), and [Dc IS transposition](#)) especially developed for handling random numerical constants (if you are not familiar with these operators, please use the default values by clicking the Default button for they work very well in all cases).

And last but not least since these parameters are crucial if you are handling numerical constants directly, you must also choose and adjust the range and type of numerical constants that will be used by the [GEP-RNC algorithm](#) during the learning process. As for the Number of Constants per Gene parameter, a good rule of thumb consists of using a small set of 10 different constants per gene as this seems to provide enough diversity for most problems without inflating the structural complexity much.



Working with Function Finding Tools

Evolutionary Strategies

Predicting unknown behavior efficiently is of course the foremost goal in modeling. But extracting knowledge from the blindly designed models is becoming more and more crucial as this knowledge can be used not only to enlighten further the modeling process but also to understand the complex relationships between variables.

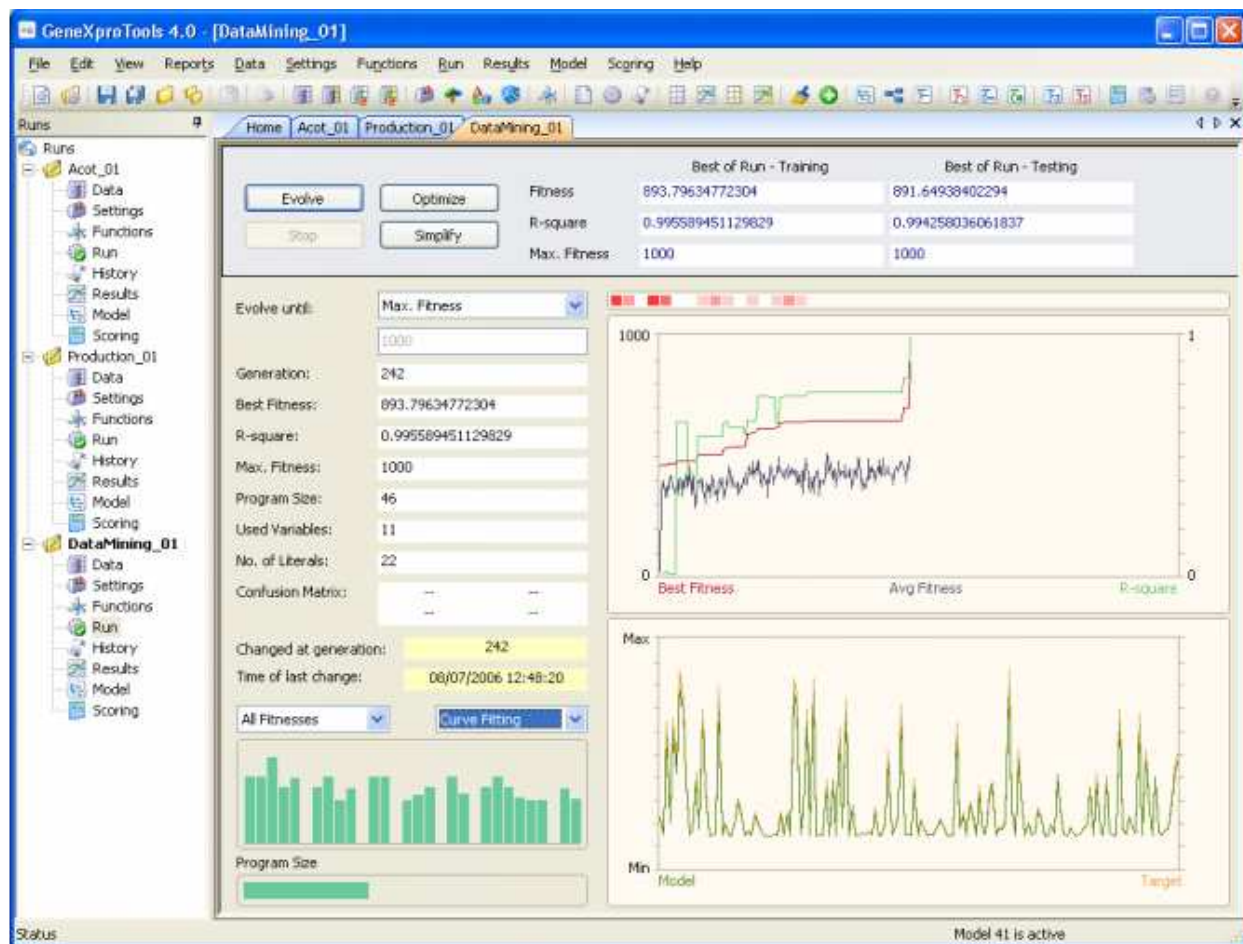
So, the evolutionary strategies we recommend in the GeneXproTools [templates](#) for Function Finding reflect these two main concerns: efficiency and simplicity. Basically, we recommend starting the modeling process with the [GEP-RNC algorithm](#) and a function set well adjusted to the complexity of the problem.

GeneXproTools 4.0 chooses the appropriate template for your problem according to the number of variables in your data. This kind of template is a good starting point that allows you to start the modeling process immediately with just a mouse click. Indeed, even if you are not familiar with evolutionary computation in general and [Gene Expression Programming](#) in particular, you will be able to design complex nonlinear models immediately thanks to the templates of GeneXproTools 4.0. In these templates, all the adjustable parameters of the learning algorithm are already set and, for

instance, you don't have to know how to create genetic diversity, how to set the appropriate population size, the chromosome architecture, the fitness function, how to increase the complexity of your models, and so forth. Then, as you learn more about GeneXproTools, you will be able to explore all its modeling tools and create quickly and efficiently very good models that will allow you to understand your data like never before.

There is, however, a very important setting in GeneXproTools that is not controlled by GeneXproTools 4.0 templates and must be wisely chosen by you: the **number of training samples**. Theoretically, if your data is well balanced and in good condition, evolutionarily speaking, the more samples the better. But there's a catch, obviously: the larger the training set the slower evolution or, in other words, the more time will be needed for generations to go by. So, you must compromise here and choose a training set with the appropriate size. A good rule of thumb consists of choosing 8-10 training samples for each independent variable in your data; all the remaining samples could be used for testing the generalizing capability of the evolved models.

So, after creating a new run you just have to click the Evolve button in the [Run Panel](#) in order to design a model. Then you observe carefully the evolutionary process, especially the curve fitting plot. Then, whenever you see fit, you can stop the run without fear of stopping the evolutionary process prematurely as GeneXproTools 4.0 allows you to continue the evolutionary process at a later time by using the best-of-run model as the starting point ([evolve with seed](#)). For that you just have to click on the Optimize button in the [Run Panel](#).



This strategy has enormous advantages as you might choose to stop the run at any time and then take a closer look at the evolved model. For instance, you can analyze its mathematical representation, its performance in the testing set, evaluate a wide set of statistical functions for a quick and rigorous assessment of its accuracy, see how it performs on a different testing set, and so on. Then you might

choose to adjust a few parameters, say, choose a different fitness function, expand the function set, add a neutral gene, apply parsimony pressure, change the training set for model refreshing, and so forth, and then explore this new evolutionary path. You can repeat this process for as long as you want or until you are completely satisfied with the evolved model.

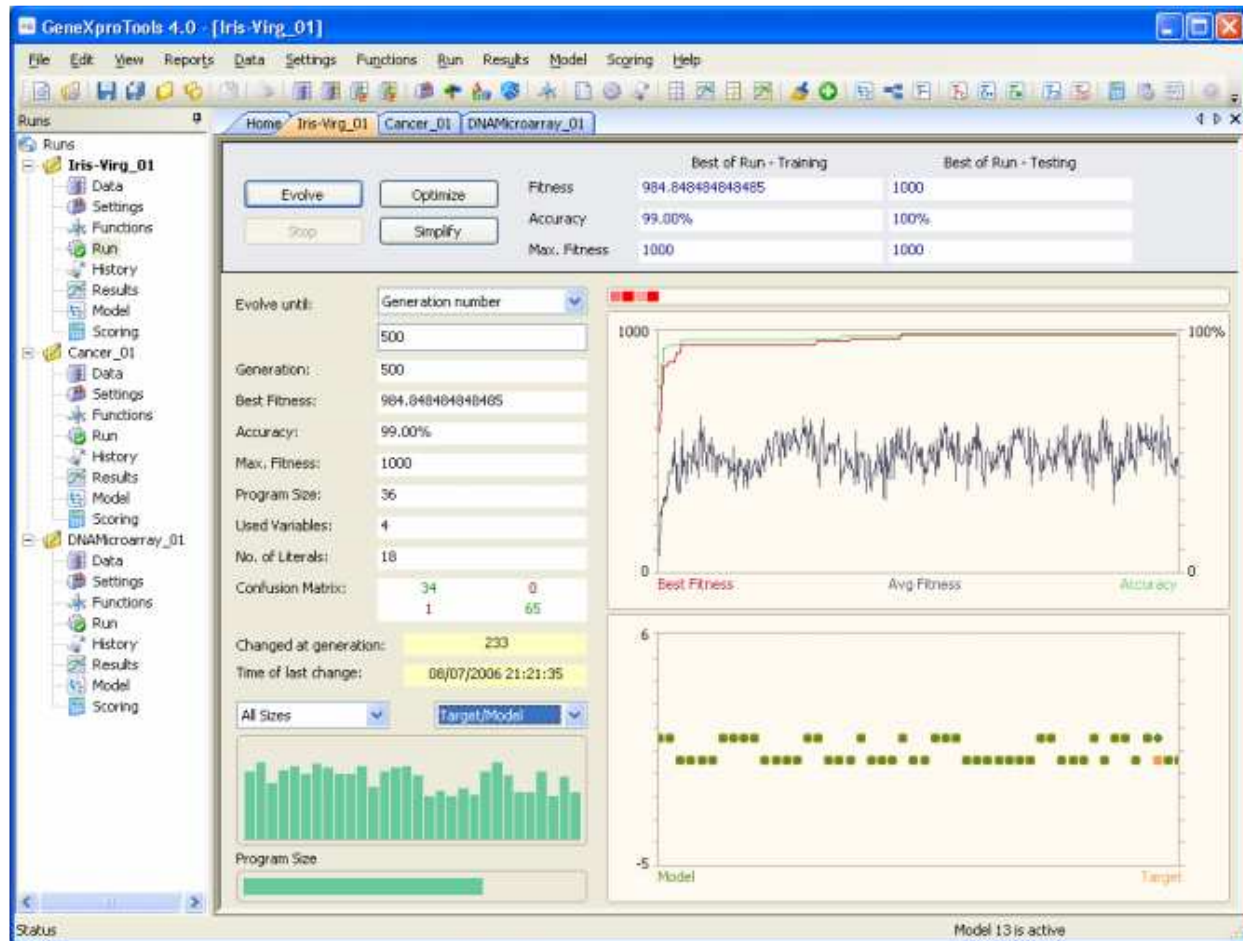
In addition, if you wish for GeneXproTools 4.0 to increase the complexity of your models automatically, you just have to activate the [Complexity Increase Engine](#) and then click the Evolve button and GeneXproTools will evolve better and better models composed of an increasing number of terms until no increase in best fitness takes place for a certain period of time.



Working with Classification Tools

This chapter introduces the Classification tools of GeneXproTools and teaches you how to use them to discover complex nonlinear classification models. In this chapter, you'll learn how to:

- [Load the input data for the GeneXproTools classification algorithm.](#)
- [Explore the GeneXproTools function kit for classification problems.](#)
- [Explore all the available architectures for building classification models.](#)
- [Explore the best fitness functions for classification problems.](#)
- [Explore all the learning algorithms available for classification problems.](#)
- [Explore essential evolutionary strategies for an efficient learning.](#)



Working with Classification Tools

Loading Data

The [learning algorithm](#) GeneXproTools 4.0 uses for classification, classifies your input data into two classes: class "0" and class "1". Obviously, the dependent variable in your training and testing sets can only have two distinct values: 0 or 1.

GeneXproTools 4.0 classifies the value returned by the evolved model as "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Classification problems with more than two classes are also easily solved with GeneXproTools 4.0. When you are classifying data into more than two classes, say, n distinct classes, you must decompose your problem into n separate 0/1 classification tasks as follows:

- C_1 versus Not C_1
- C_2 versus Not C_2

...
 C_n versus Not C_n

Then evolve n different models separately and combine the different models to make the final classification model.

But before evolving a model with GeneXproTools 4.0 you must first load the input data for the learning algorithm. GeneXproTools 4.0 allows you to work either with databases/Excel or text files and, for text files, accepts two different **data matrix formats**.

The first is the standard **Samples x Variables** format where samples are in rows and variables in columns, with the dependent variable occupying the rightmost position. In the small example below with 10 samples, IRIS_PLANT is the class and SEPAL_LENGTH, SEPAL_WIDTH, PETAL_LENGTH, and PETAL_WIDTH are the dependent variables or attributes:

```
SEPAL_LENGTH SEPAL_WIDTH PETAL_LENGTH PETAL_WIDTH IRIS_PLANT
5.4 3.4 1.7 0.2 0
6.1 3.0 4.6 1.4 0
5.0 3.4 1.6 0.4 0
5.2 3.5 1.5 0.2 0
5.1 3.7 1.5 0.4 0
5.5 2.4 3.7 1.0 0
7.2 3.2 6.0 1.8 1
6.3 2.7 4.9 1.8 1
7.7 3.8 6.7 2.2 1
4.8 3.4 1.9 0.2 0
```

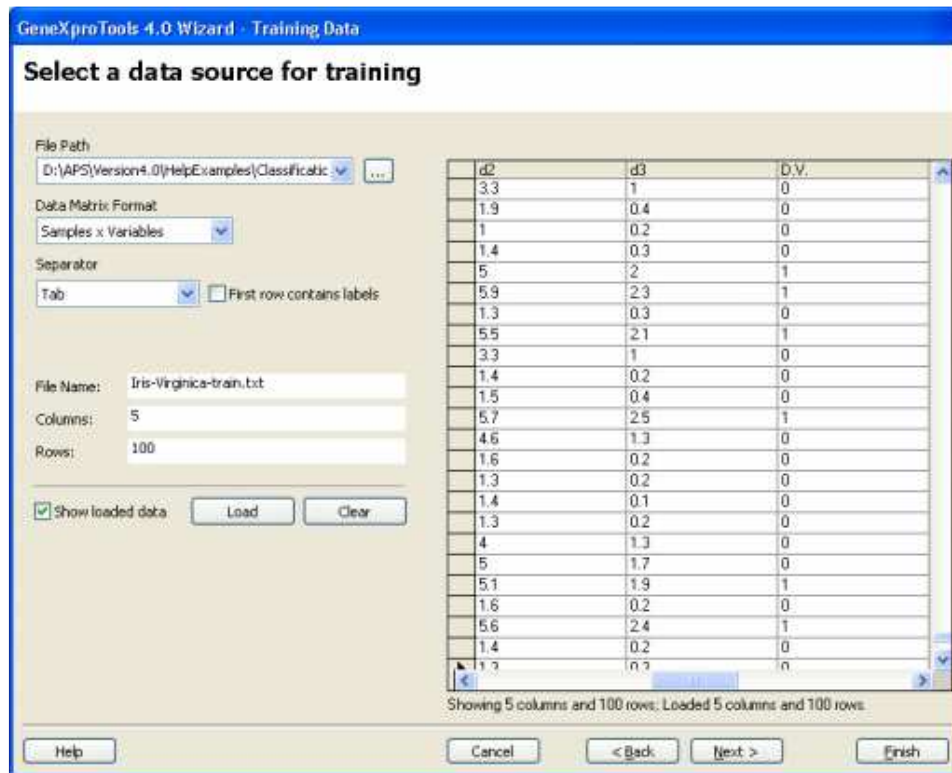
And the second, is the **Gene Expression Matrix** format commonly used in DNA microarrays studies where samples are in columns and variables in rows, with the class occupying the topmost position. For instance, in Gene Expression Matrix format, the small dataset above corresponds to:

```
IRIS_PLANT 0 0 0 0 0 0 1 1 1 0
SEPAL_LENGTH 5.4 6.1 5.0 5.2 5.1 5.5 7.2 6.3 7.7 4.8
SEPAL_WIDTH 3.4 3.0 3.4 3.5 3.7 2.4 3.2 2.7 3.8 3.4
PETAL_LENGTH 1.7 4.6 1.6 1.5 1.5 3.7 6.0 4.9 6.7 1.9
PETAL_WIDTH 0.2 1.4 0.4 0.2 0.4 1.0 1.8 1.8 2.2 0.2
```

which is very handy for datasets with a relatively small number of samples and thousands of variables. Note, however, that for Excel files this format is not supported and if your data is kept in this format in Excel, you must copy it to a text file so that it can be loaded into GeneXproTools.

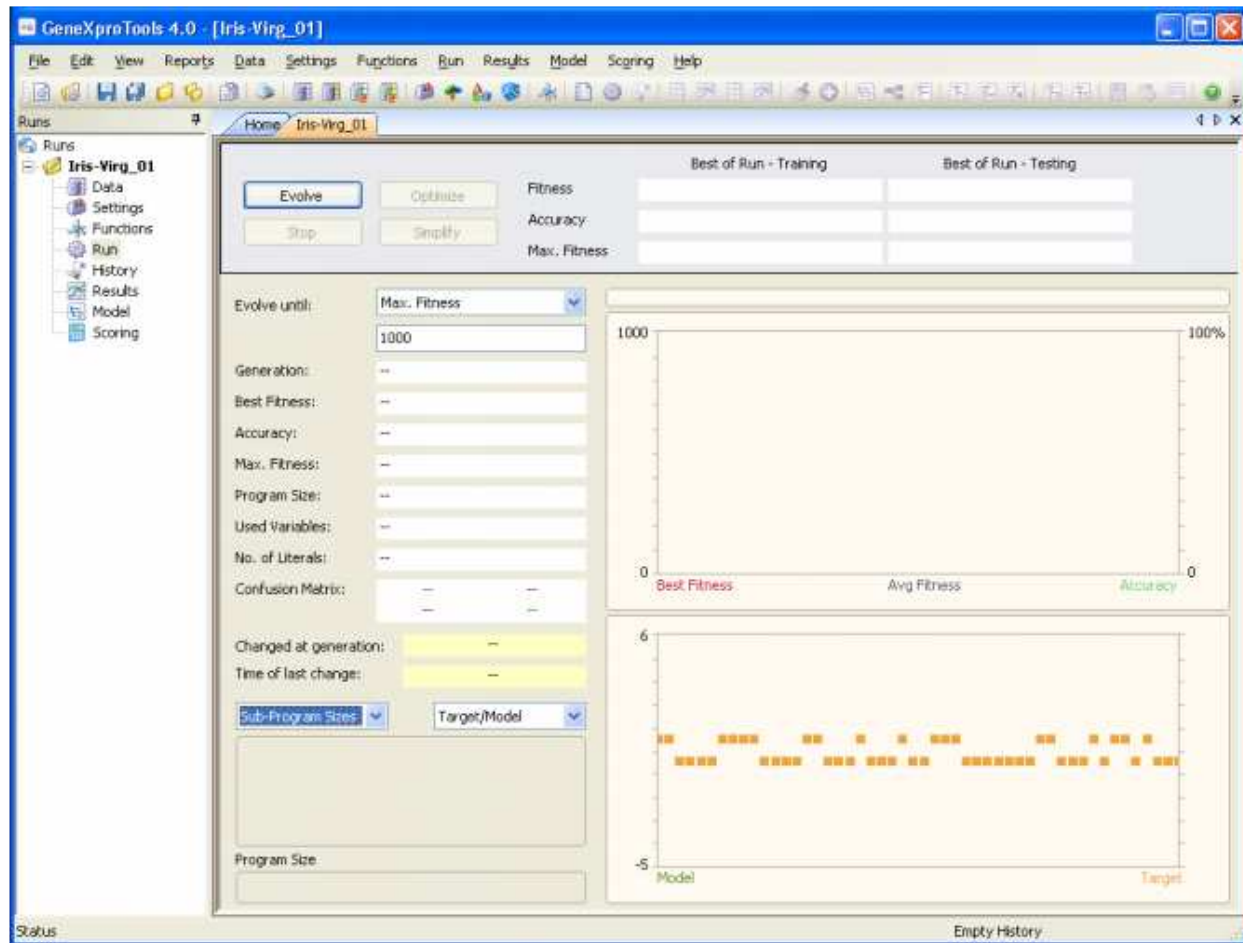
GeneXproTools uses the Samples x Variables format throughout and therefore all formats are automatically converted and shown in this format.

GeneXproTools supports the standard **separators** (space, tab, comma, semicolon, and pipe) and detects them automatically. The use of **labels** to identify your variables is optional and GeneXproTools also detects automatically whether they are present or not. If you use them, however, you will be able to generate more intelligible code where each variable is identified by its name, by checking the Use Labels box in the Model Panel.



To Load Input Data for Modeling

1. Click the File Menu and then choose New.
The New Run Wizard appears. You must give a name to your new run file (the default filename extension of GeneXproTools 4.0 run files is .gep) and then choose Classification in the Problem Category box and the kind of source file in the Data Source Type box. GeneXproTools 4.0 allows you to work either with Excel/databases or text files.
2. Then go to the Training Data window by clicking the Next button.
Choose the path for the training set by browsing the Open dialog box and choose the appropriate data matrix format. Irrespective of the data format used, GeneXproTools shows the loaded data in the standard Samples x Variables format, with the class or dependent variable occupying the rightmost position.
3. Then go to the Testing Data window by clicking the Next button.
Repeat the same steps of the previous point if you wish to use a testing set to evaluate the predictive accuracy of your model.
4. Click the Finish button to save your new run file.
The Save As dialog box appears and after choosing the directory where you want your new run file to be saved, the GeneXproTools modeling environment appears. Then you just have to click the Evolve button to create a model as GeneXproTools automatically chooses, from a gallery of templates, default settings that will enable you to evolve a model immediately.

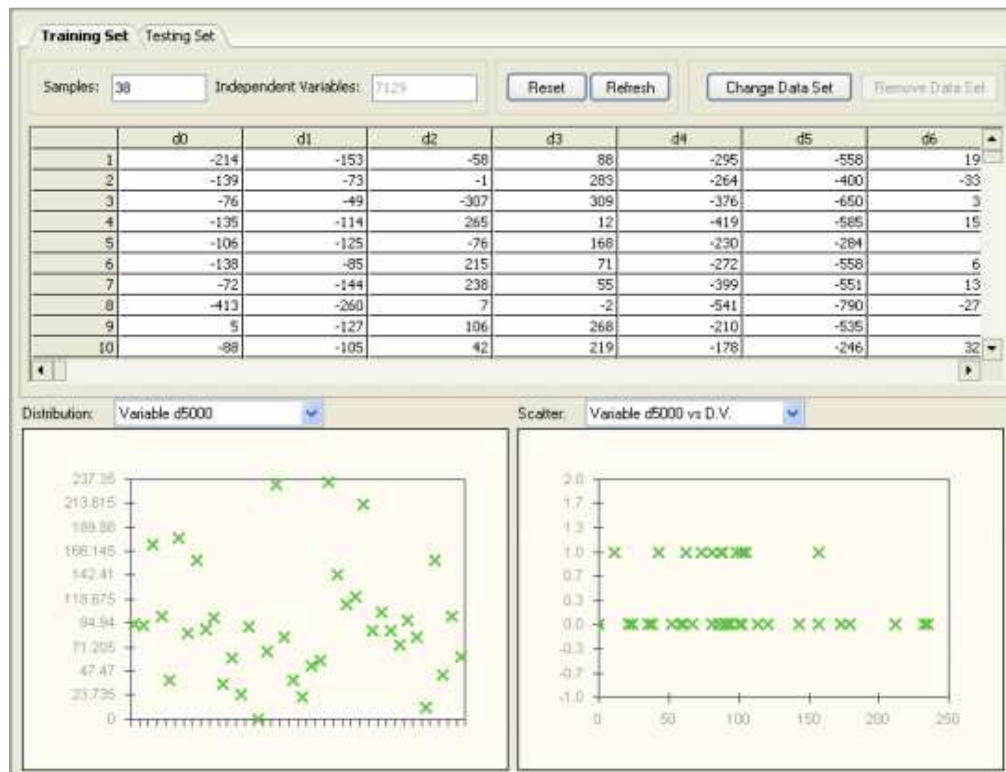


In data mining, be it performed by learning algorithms or conventional statistical methods, it really pays to take a good look at your data before embarking on a complex, usually time consuming modeling process. It's true that evolutionary algorithms are particularly well equipped to deal with noisy data, but the better the data you feed them the better the models they produce.

GeneXproTools helps you find missing and invalid (usually nominal) values in your datasets and prompts you to fix them before they are used for modeling. But the preparation of a well balanced dataset should be done before loading the data into GeneXproTools, and we recommend you to particularly take care of the following:

- Avoid using duplicated samples for they can bias the modeling process considerably.
- Choose a well balanced dataset.
- Choose a reasonable number of samples for training.
An excessively large dataset will slow the modeling process unnecessarily. If you have access to huge databases it's good practice to use the surplus samples for testing instead. A good rule of thumb consists of using about 8-10 samples for each independent variable in your training data.
- Check your datasets carefully for inaccurate values. Typographical or measurement errors generally cause outliers that can be detected by graphing one variable at a time, a task that can be easily accomplished by GeneXproTools in the [Data Panel](#).

The [graphical visualization tools](#) of GeneXproTools 4.0 make it easy to identify outliers, which may well represent errors in the data files. After loading your data into GeneXproTools, in the [Data Panel](#) you can visualize the distribution of values for each variable and also plot each independent variable against the dependent variable.



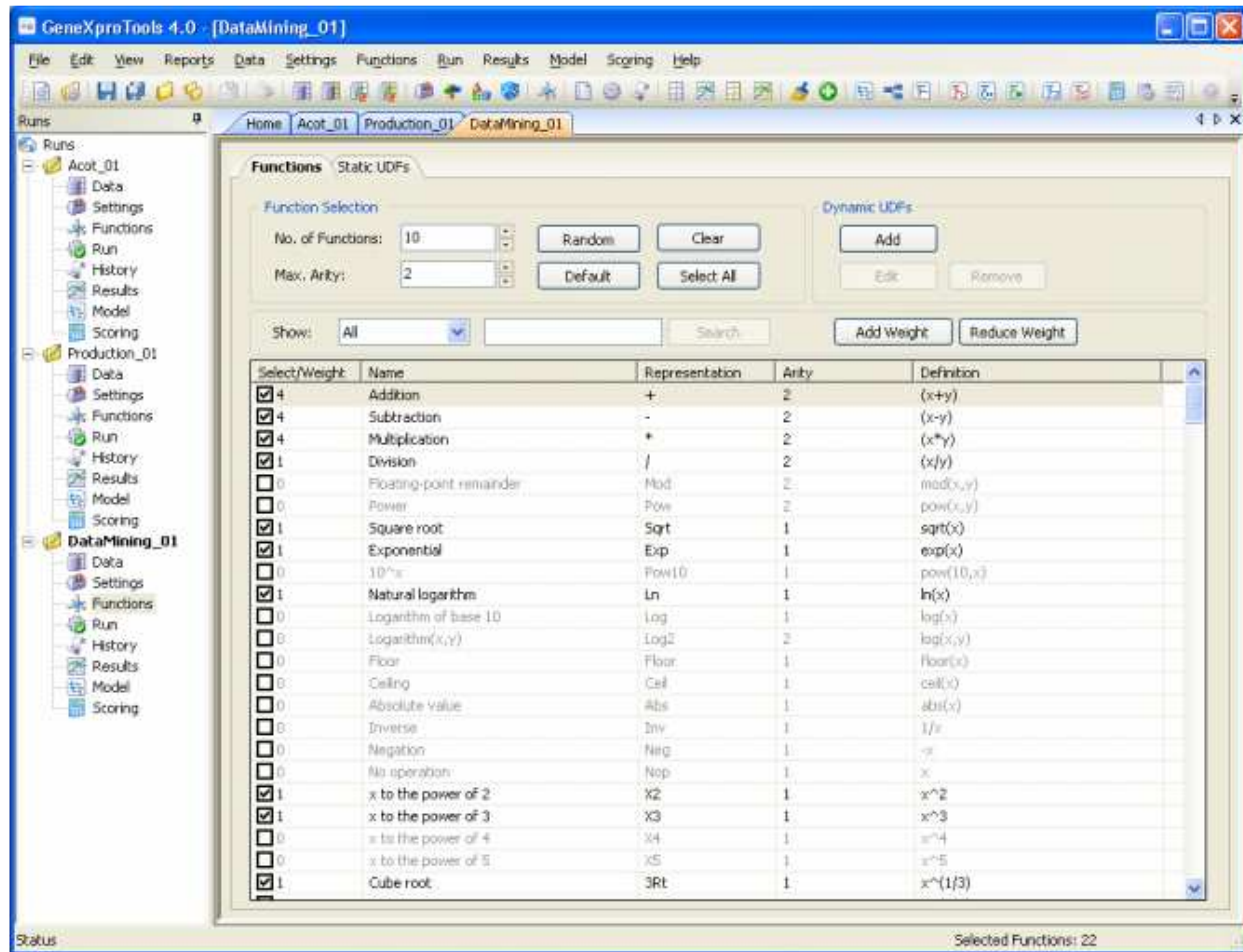
◀ | ▶

Working with Classification Tools

Available Functions

In the Functions Panel you can select and create:

- [Mathematical Functions](#) (including Dynamic UDFs or DDFs).
- [Static UDFs](#).

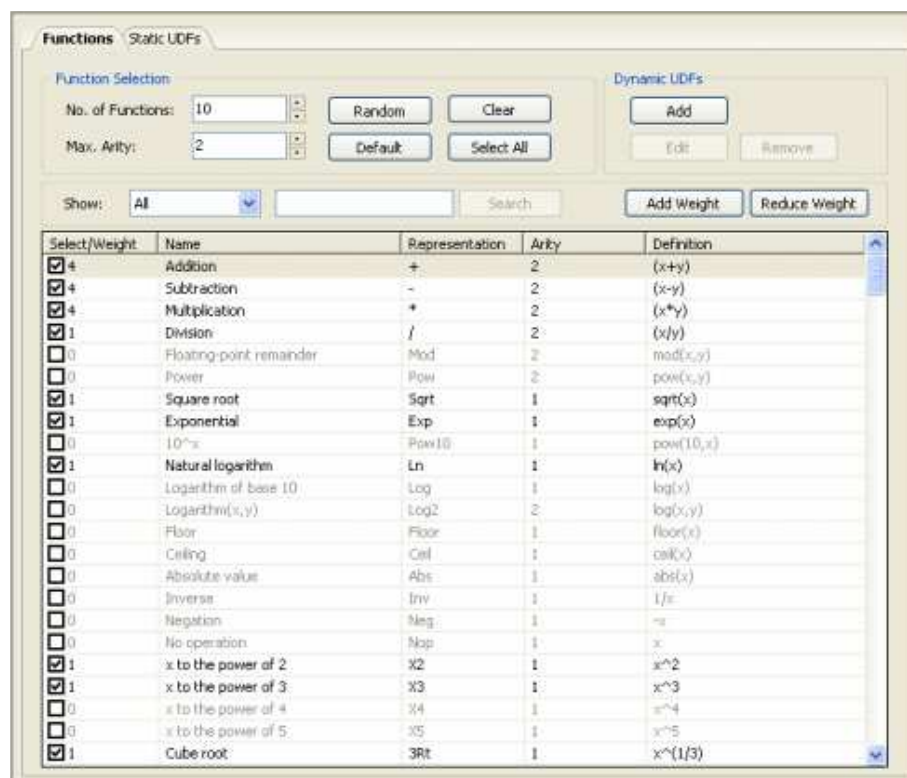


Working with Classification Tools

Mathematical Functions and Dynamic UDFs

GeneXproTools 4.0 offers a total of 279 **built-in mathematical functions**, including 186 IF THEN ELSE rules, six OR operators, and six AND operators, that can be used for designing nonlinear classification models. This wide set of mathematical functions allows the evolution of complex and rigorous models quickly built with the most appropriate functions and rules.

The Function Selection Tool of GeneXproTools 4.0 helps you selecting different function sets very quickly through the combination of the Show options with the Random/Default/Clear/Select All buttons plus the Increase/Reduce Weight buttons in the [Functions Panel](#). Furthermore, GeneXproTools automatically balances your function set with the number of independent variables in your data, so now you just have to select the right functions for your problem and then choose their relative proportions by adjusting their weights.



Despite the wide set of GeneXproTools built-in mathematical functions, some users sometimes want to model with different ones. GeneXproTools 4.0 gives the user the possibility of creating custom tailored functions ([Dynamic UDFs](#) or DDFs) and evolve models with them. A note of caution though: the use of DDFs slows considerably the evolutionary process and therefore should be used with moderation.

Below are listed all the 279 **built-in mathematical functions** available in GeneXproTools 4.0, including their representation in [Karva Notation](#) (shown in red) and their mathematical definition (for more details about their implementations, see the [built-in grammars](#) of GeneXproTools):

Basic arithmetic functions:

- Addition (**+**): $(x+y)$
- Subtraction (**-**): $(x-y)$
- Multiplication (*****): $(x*y)$
- Division (**/**): (x/y)
- Addition with 3 inputs (**Add3**): $(x+y+z)$
- Subtraction with 3 inputs (**Sub3**): $(x-y-z)$
- Multiplication with 3 inputs (**Mul3**): $(x*y*z)$
- Division with 3 inputs (**Div3**): $(x/y/z)$
- Addition with 4 inputs (**Add4**): $(a+b+c+d)$
- Subtraction with 4 inputs (**Sub4**): $(a-b-c-d)$
- Multiplication with 4 inputs (**Mul4**): $(a*b*c*d)$
- Division with 4 inputs (**Div4**): $(a/b/c/d)$

Logarithmic functions:

- Natural logarithm (**Ln**): $\ln(x)$
- Logarithm of base 10 (**Log**): $\log(x)$
- Logarithm(x,y) (**Log2**): $\log(x,y)$

Exponential and power functions:

- Square root (**Sqrt**): $\text{sqrt}(x)$
- Exponential (**Exp**): $\text{exp}(x)$
- Power (**Pow**): x^y
- 10^x (**Pow10**): 10^x
- x^2 (**X2**): x^2
- x^3 (**X3**): x^3
- x^4 (**X4**): x^4
- x^5 (**X5**): x^5
- Cube root (**3Rt**): $x^{(1/3)}$
- Quartic root (**4Rt**): $x^{(1/4)}$
- Quintic root (**5Rt**): $x^{(1/5)}$
- Logistic(x) (**Logi**): $\text{logistic}(x)$
- Logistic(x,y) (**Logi2**): $\text{logistic}(x,y)$
- Logistic(x,y,z) (**Logi3**): $\text{logistic}(x,y,z)$
- Logistic(a,b,c,d) (**Logi4**): $\text{logistic}(a,b,c,d)$
- Gaussian(x) (**Gau**): $\text{gaussian}(x)$
- Gaussian(x,y) (**Gau2**): $\text{gaussian}(x,y)$
- Gaussian(x,y,z) (**Gau3**): $\text{gaussian}(x,y,z)$
- Gaussian(a,b,c,d) (**Gau4**): $\text{gaussian}(a,b,c,d)$

Rounding and other simple functions:

- Floor (**Floor**): $\text{floor}(x)$
- Ceiling (**Ceil**): $\text{ceil}(x)$
- Floating-point remainder (**Mod**): $\text{mod}(x,y)$
- Absolute value (**Abs**): $\text{abs}(x)$
- Inverse (**Inv**): $1/x$
- Negation (**Neg**): $-x$
- Complement (**NOT**): $(1-x)$
- No operation (**Nop**): x

Minimum, maximum, and average functions:

- Minimum of 2 inputs (**Min2**): $\text{min}(x,y)$
- Minimum of 3 inputs (**Min3**): $\text{min}(x,y,z)$
- Minimum of 4 inputs (**Min4**): $\text{min}(a,b,c,d)$
- Maximum of 2 inputs (**Max2**): $\text{max}(x,y)$
- Maximum of 3 inputs (**Max3**): $\text{max}(x,y,z)$
- Maximum of 4 inputs (**Max4**): $\text{max}(a,b,c,d)$
- Average of 2 inputs (**Avg2**): $\text{avg}(x,y)$
- Average of 3 inputs (**Avg3**): $\text{avg}(x,y,z)$
- Average of 4 inputs (**Avg4**): $\text{avg}(a,b,c,d)$

Constant functions:

- Constant zero function (**Zero**): $0(x) = 0$
- Constant one function (**One**): $1(x) = 1$
- $0(x,y)$ (**Zero2**): $0(x,y) = 0$
- $1(x,y)$ (**One2**): $1(x,y) = 1$
- Number π (**Pi**): $\text{Pi}(x) = 3.141592653589793$
- Euler's number (**E**): $E(x) = 2.718281828459045$

Trigonometric functions (x is in radians):

- Sine (**Sin**): $\text{sin}(x)$
- Cosine (**Cos**): $\text{cos}(x)$

- Tangent (**Tan**): $\tan(x)$
- Cosecant (**Csc**): $\csc(x)$
- Secant (**Sec**): $\sec(x)$
- Cotangent (**Cot**): $\cot(x)$

Inverse trigonometric functions:

- Arcsine (**Asin**): $\arcsin(x)$
- Arccosine (**Acos**): $\arccos(x)$
- Arctangent (**Atan**): $\arctan(x)$
- Arccosecant (**Acsc**): $\operatorname{arccsc}(x)$
- Arcsecant (**Asec**): $\operatorname{arcsec}(x)$
- Arccotangent (**Acot**): $\operatorname{arccot}(x)$

Hyperbolic functions:

- Hyperbolic sine (**Sinh**): $\sinh(x)$
- Hyperbolic cosine (**Cosh**): $\cosh(x)$
- Hyperbolic tangent (**Tanh**): $\tanh(x)$
- Hyperbolic cosecant (**Csch**): $\operatorname{csch}(x)$
- Hyperbolic secant (**Sech**): $\operatorname{sech}(x)$
- Hyperbolic cotangent (**Coth**): $\operatorname{coth}(x)$

Inverse hyperbolic functions:

- Inverse hyperbolic sine (**Asinh**): $\operatorname{arcsinh}(x)$
- Inverse hyperbolic cosine (**Acosh**): $\operatorname{arccosh}(x)$
- Inverse hyperbolic tangent (**Atanh**): $\operatorname{arctanh}(x)$
- Inverse hyperbolic cosecant (**Acsch**): $\operatorname{arccsch}(x)$
- Inverse hyperbolic secant (**Asech**): $\operatorname{arcsech}(x)$
- Inverse hyperbolic cotangent (**Acoth**): $\operatorname{arccoth}(x)$

Comparison 0/1 functions of two arguments (OR series):

- OR1 (**OR1**): if $x < 0$ OR $y < 0$, then 1, else 0
- OR2 (**OR2**): if $x \geq 0$ OR $y \geq 0$, then 1, else 0
- OR3 (**OR3**): if $x \leq 0$ OR $y \leq 0$, then 1, else 0
- OR4 (**OR4**): if $x < 1$ OR $y < 1$, then 1, else 0
- OR5 (**OR5**): if $x \geq 1$ OR $y \geq 1$, then 1, else 0
- OR6 (**OR6**): if $x \leq 1$ OR $y \leq 1$, then 1, else 0

Comparison 0/1 functions of two arguments (AND series):

- AND1 (**AND1**): if $x < 0$ AND $y < 0$, then 1, else 0
- AND2 (**AND2**): if $x \geq 0$ AND $y \geq 0$, then 1, else 0
- AND3 (**AND3**): if $x \leq 0$ AND $y \leq 0$, then 1, else 0
- AND4 (**AND4**): if $x < 1$ AND $y < 1$, then 1, else 0
- AND5 (**AND5**): if $x \geq 1$ AND $y \geq 1$, then 1, else 0
- AND6 (**AND6**): if $x \leq 1$ AND $y \leq 1$, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series A):

- Less Than with 2 inputs (A) (**LT2A**): if $x < y$, then x , else y
- Greater Than with 2 inputs (A) (**GT2A**): if $x > y$, then x , else y
- Less Or Equal To with 2 inputs (A) (**LOE2A**): if $x \leq y$, then x , else y
- Greater Or Equal To with 2 inputs (A) (**GOE2A**): if $x \geq y$, then x , else y
- Equal To with 2 inputs (A) (**ET2A**): if $x = y$, then x , else y

- Not Equal To with 2 inputs (A) (**NET2A**): if $x \neq y$, then x , else y

Comparison 0/1 IF THEN ELSE functions of two arguments (series B):

- Less Than with 2 inputs (B) (**LT2B**): if $x < y$, then 1, else 0
- Greater Than with 2 inputs (B) (**GT2B**): if $x > y$, then 1, else 0
- Less Or Equal To with 2 inputs (B) (**LOE2B**): if $x \leq y$, then 1, else 0
- Greater Or Equal To with 2 inputs (B) (**GOE2B**): if $x \geq y$, then 1, else 0
- Equal To with 2 inputs (B) (**ET2B**): if $x = y$, then 1, else 0
- Not Equal To with 2 inputs (B) (**NET2B**): if $x \neq y$, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series C):

- Less Than with 2 inputs (C) (**LT2C**): if $x < y$, then $(x+y)$, else $(x-y)$
- Greater Than with 2 inputs (C) (**GT2C**): if $x > y$, then $(x+y)$, else $(x-y)$
- Less Or Equal To with 2 inputs (C) (**LOE2C**): if $x \leq y$, then $(x+y)$, else $(x-y)$
- Greater Or Equal To with 2 inputs (C) (**GOE2C**): if $x \geq y$, then $(x+y)$, else $(x-y)$
- Equal To with 2 inputs (C) (**ET2C**): if $x = y$, then $(x+y)$, else $(x-y)$
- Not Equal To with 2 inputs (C) (**NET2C**): if $x \neq y$, then $(x+y)$, else $(x-y)$

Comparison IF THEN ELSE functions of two arguments (series D):

- Less Than with 2 inputs (D) (**LT2D**): if $x < y$, then $(x*y)$, else (x/y)
- Greater Than with 2 inputs (D) (**GT2D**): if $x > y$, then $(x*y)$, else (x/y)
- Less Or Equal To with 2 inputs (D) (**LOE2D**): if $x \leq y$, then $(x*y)$, else (x/y)
- Greater Or Equal To with 2 inputs (D) (**GOE2D**): if $x \geq y$, then $(x*y)$, else (x/y)
- Equal To with 2 inputs (D) (**ET2D**): if $x = y$, then $(x*y)$, else (x/y)
- Not Equal To with 2 inputs (D) (**NET2D**): if $x \neq y$, then $(x*y)$, else (x/y)

Comparison IF THEN ELSE functions of two arguments (series E):

- Less Than with 2 inputs (E) (**LT2E**): if $x < y$, then $(x+y)$, else $(x*y)$
- Greater Than with 2 inputs (E) (**GT2E**): if $x > y$, then $(x+y)$, else $(x*y)$
- Less Or Equal To with 2 inputs (E) (**LOE2E**): if $x \leq y$, then $(x+y)$, else $(x*y)$
- Greater Or Equal To with 2 inputs (E) (**GOE2E**): if $x \geq y$, then $(x+y)$, else $(x*y)$
- Equal To with 2 inputs (E) (**ET2E**): if $x = y$, then $(x+y)$, else $(x*y)$
- Not Equal To with 2 inputs (E) (**NET2E**): if $x \neq y$, then $(x+y)$, else $(x*y)$

Comparison IF THEN ELSE functions of two arguments (series F):

- Less Than with 2 inputs (F) (**LT2F**): if $x < y$, then $(x+y)$, else $\sin(x*y)$
- Greater Than with 2 inputs (F) (**GT2F**): if $x > y$, then $(x+y)$, else $\sin(x*y)$
- Less Or Equal To with 2 inputs (F) (**LOE2F**): if $x \leq y$, then $(x+y)$, else $\sin(x*y)$
- Greater Or Equal To with 2 inputs (F) (**GOE2F**): if $x \geq y$, then $(x+y)$, else $\sin(x*y)$
- Equal To with 2 inputs (F) (**ET2F**): if $x = y$, then $(x+y)$, else $\sin(x*y)$
- Not Equal To with 2 inputs (F) (**NET2F**): if $x \neq y$, then $(x+y)$, else $\sin(x*y)$

Comparison IF THEN ELSE functions of two arguments (series G):

- Less Than with 2 inputs (G) (**LT2G**): if $x < y$, then $(x+y)$, else $\text{atan}(x*y)$
- Greater Than with 2 inputs (G) (**GT2G**): if $x > y$, then $(x+y)$, else $\text{atan}(x*y)$
- Less Or Equal To with 2 inputs (G) (**LOE2G**): if $x \leq y$, then $(x+y)$, else $\text{atan}(x*y)$
- Greater Or Equal To with 2 inputs (G) (**GOE2G**): if $x \geq y$, then $(x+y)$, else $\text{atan}(x*y)$
- Equal To with 2 inputs (G) (**ET2G**): if $x = y$, then $(x+y)$, else $\text{atan}(x*y)$
- Not Equal To with 2 inputs (G) (**NET2G**): if $x \neq y$, then $(x+y)$, else $\text{atan}(x*y)$

Comparison IF THEN ELSE functions of three arguments (series A):

- Less Than with 3 inputs (A) (**LT3A**): if $x < 0$, then y , else z
- Greater Than with 3 inputs (A) (**GT3A**): if $x > 0$, then y , else z
- Less Or Equal To with 3 inputs (A) (**LOE3A**): if $x \leq 0$, then y , else z
- Greater Or Equal To with 3 inputs (A) (**GOE3A**): if $x \geq 0$, then y , else z
- Equal To with 3 inputs (A) (**ET3A**): if $x = 0$, then y , else z
- Not Equal To with 3 inputs (A) (**NET3A**): if $x \neq 0$, then y , else z

Comparison IF THEN ELSE functions of three arguments (series B):

- Less Than with 3 inputs (B) (**LT3B**): if $(x+y) < z$, then $(x+y)$, else z
- Greater Than with 3 inputs (B) (**GT3B**): if $(x+y) > z$, then $(x+y)$, else z
- Less Or Equal To with 3 inputs (B) (**LOE3B**): if $(x+y) \leq z$, then $(x+y)$, else z
- Greater Or Equal To with 3 inputs (B) (**GOE3B**): if $(x+y) \geq z$, then $(x+y)$, else z
- Equal To with 3 inputs (B) (**ET3B**): if $(x+y) = z$, then $(x+y)$, else z
- Not Equal To with 3 inputs (B) (**NET3B**): if $(x+y) \neq z$, then $(x+y)$, else z

Comparison IF THEN ELSE functions of three arguments (series C):

- Less Than with 3 inputs (C) (**LT3C**): if $(x+y) < z$, then $(x+y)$, else $(x+z)$
- Greater Than with 3 inputs (C) (**GT3C**): if $(x+y) > z$, then $(x+y)$, else $(x+z)$
- Less Or Equal To with 3 inputs (C) (**LOE3C**): if $(x+y) \leq z$, then $(x+y)$, else $(x+z)$
- Greater Or Equal To with 3 inputs (C) (**GOE3C**): if $(x+y) \geq z$, then $(x+y)$, else $(x+z)$
- Equal To with 3 inputs (C) (**ET3C**): if $(x+y) = z$, then $(x+y)$, else $(x+z)$
- Not Equal To with 3 inputs (C) (**NET3C**): if $(x+y) \neq z$, then $(x+y)$, else $(x+z)$

Comparison IF THEN ELSE functions of three arguments (series D):

- Less Than with 3 inputs (D) (**LT3D**): if $(x+y) < z$, then $(x+y)$, else $(x-z)$
- Greater Than with 3 inputs (D) (**GT3D**): if $(x+y) > z$, then $(x+y)$, else $(x-z)$
- Less Or Equal To with 3 inputs (D) (**LOE3D**): if $(x+y) \leq z$, then $(x+y)$, else $(x-z)$
- Greater Or Equal To with 3 inputs (D) (**GOE3D**): if $(x+y) \geq z$, then $(x+y)$, else $(x-z)$
- Equal To with 3 inputs (D) (**ET3D**): if $(x+y) = z$, then $(x+y)$, else $(x-z)$
- Not Equal To with 3 inputs (D) (**NET3D**): if $(x+y) \neq z$, then $(x+y)$, else $(x-z)$

Comparison IF THEN ELSE functions of three arguments (series E):

- Less Than with 3 inputs (E) (**LT3E**): if $(x+y) < z$, then $(x+y)$, else $(x*z)$
- Greater Than with 3 inputs (E) (**GT3E**): if $(x+y) > z$, then $(x+y)$, else $(x*z)$
- Less Or Equal To with 3 inputs (E) (**LOE3E**): if $(x+y) \leq z$, then $(x+y)$, else $(x*z)$
- Greater Or Equal To with 3 inputs (E) (**GOE3E**): if $(x+y) \geq z$, then $(x+y)$, else $(x*z)$
- Equal To with 3 inputs (E) (**ET3E**): if $(x+y) = z$, then $(x+y)$, else $(x*z)$
- Not Equal To with 3 inputs (E) (**NET3E**): if $(x+y) \neq z$, then $(x+y)$, else $(x*z)$

Comparison IF THEN ELSE functions of three arguments (series F):

- Less Than with 3 inputs (F) (**LT3F**): if $(x+y) < z$, then $(x+y)$, else (x/z)
- Greater Than with 3 inputs (F) (**GT3F**): if $(x+y) > z$, then $(x+y)$, else (x/z)
- Less Or Equal To with 3 inputs (F) (**LOE3F**): if $(x+y) \leq z$, then $(x+y)$, else (x/z)
- Greater Or Equal To with 3 inputs (F) (**GOE3F**): if $(x+y) \geq z$, then $(x+y)$, else (x/z)
- Equal To with 3 inputs (F) (**ET3F**): if $(x+y) = z$, then $(x+y)$, else (x/z)
- Not Equal To with 3 inputs (F) (**NET3F**): if $(x+y) \neq z$, then $(x+y)$, else (x/z)

Comparison IF THEN ELSE functions of three arguments (series G):

- Less Than with 3 inputs (G) (**LT3G**): if $(x+y) < z$, then $(x*y)$, else $(x+z)$
- Greater Than with 3 inputs (G) (**GT3G**): if $(x+y) > z$, then $(x*y)$, else $(x+z)$
- Less Or Equal To with 3 inputs (G) (**LOE3G**): if $(x+y) \leq z$, then $(x*y)$, else $(x+z)$

- Greater Or Equal To with 3 inputs (G) (**GOE3G**): if $(x+y) \geq z$, then $(x*y)$, else $(x+z)$
- Equal To with 3 inputs (G) (**ET3G**): if $(x+y) = z$, then $(x*y)$, else $(x+z)$
- Not Equal To with 3 inputs (G) (**NET3G**): if $(x+y) \neq z$, then $(x*y)$, else $(x+z)$

Comparison IF THEN ELSE functions of three arguments (series H):

- Less Than with 3 inputs (H) (**LT3H**): if $(x+y) < z$, then $(x*y)$, else $(x-z)$
- Greater Than with 3 inputs (H) (**GT3H**): if $(x+y) > z$, then $(x*y)$, else $(x-z)$
- Less Or Equal To with 3 inputs (H) (**LOE3H**): if $(x+y) \leq z$, then $(x*y)$, else $(x-z)$
- Greater Or Equal To with 3 inputs (H) (**GOE3H**): if $(x+y) \geq z$, then $(x*y)$, else $(x-z)$
- Equal To with 3 inputs (H) (**ET3H**): if $(x+y) = z$, then $(x*y)$, else $(x-z)$
- Not Equal To with 3 inputs (H) (**NET3H**): if $(x+y) \neq z$, then $(x*y)$, else $(x-z)$

Comparison IF THEN ELSE functions of three arguments (series I):

- Less Than with 3 inputs (I) (**LT3I**): if $(x+y) < z$, then $(x*y)$, else $(x*z)$
- Greater Than with 3 inputs (I) (**GT3I**): if $(x+y) > z$, then $(x*y)$, else $(x*z)$
- Less Or Equal To with 3 inputs (I) (**LOE3I**): if $(x+y) \leq z$, then $(x*y)$, else $(x*z)$
- Greater Or Equal To with 3 inputs (I) (**GOE3I**): if $(x+y) \geq z$, then $(x*y)$, else $(x*z)$
- Equal To with 3 inputs (I) (**ET3I**): if $(x+y) = z$, then $(x*y)$, else $(x*z)$
- Not Equal To with 3 inputs (I) (**NET3I**): if $(x+y) \neq z$, then $(x*y)$, else $(x*z)$

Comparison IF THEN ELSE functions of three arguments (series J):

- Less Than with 3 inputs (J) (**LT3J**): if $(x+y) < z$, then $(x*y)$, else (x/z)
- Greater Than with 3 inputs (J) (**GT3J**): if $(x+y) > z$, then $(x*y)$, else (x/z)
- Less Or Equal To with 3 inputs (J) (**LOE3J**): if $(x+y) \leq z$, then $(x*y)$, else (x/z)
- Greater Or Equal To with 3 inputs (J) (**GOE3J**): if $(x+y) \geq z$, then $(x*y)$, else (x/z)
- Equal To with 3 inputs (J) (**ET3J**): if $(x+y) = z$, then $(x*y)$, else (x/z)
- Not Equal To with 3 inputs (J) (**NET3J**): if $(x+y) \neq z$, then $(x*y)$, else (x/z)

Comparison IF THEN ELSE functions of three arguments (series K):

- Less Than with 3 inputs (K) (**LT3K**): if $(x+y) < z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Greater Than with 3 inputs (K) (**GT3K**): if $(x+y) > z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Less Or Equal To with 3 inputs (K) (**LOE3K**): if $(x+y) \leq z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Greater Or Equal To with 3 inputs (K) (**GOE3K**): if $(x+y) \geq z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Equal To with 3 inputs (K) (**ET3K**): if $(x+y) = z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Not Equal To with 3 inputs (K) (**NET3K**): if $(x+y) \neq z$, then $(x+y+z)$, else $\sin(x*y*z)$

Comparison IF THEN ELSE functions of three arguments (series L):

- Less Than with 3 inputs (L) (**LT3L**): if $(x+y) < z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- Greater Than with 3 inputs (L) (**GT3L**): if $(x+y) > z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- Less Or Equal To with 3 inputs (L) (**LOE3L**): if $(x+y) \leq z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- Greater Or Equal To with 3 inputs (L) (**GOE3L**): if $(x+y) \geq z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- Equal To with 3 inputs (L) (**ET3L**): if $(x+y) = z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- Not Equal To with 3 inputs (L) (**NET3L**): if $(x+y) \neq z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$

Comparison IF THEN ELSE functions of four arguments (series A):

- Less Than with 4 inputs (A) (**LT4A**): if $a < b$, then c , else d
- Greater Than with 4 inputs (A) (**GT4A**): if $a > b$, then c , else d
- Less Or Equal To with 4 inputs (A) (**LOE4A**): if $a \leq b$, then c , else d
- Greater Or Equal To with 4 inputs (A) (**GOE4A**): if $a \geq b$, then c , else d
- Equal To with 4 inputs (A) (**ET4A**): if $a = b$, then c , else d
- Not Equal To with 4 inputs (A) (**NET4A**): if $a \neq b$, then c , else d

Comparison IF THEN ELSE functions of four arguments (series B):

- Less Than with 4 inputs (B) (LT4B): if $(a+b) < (c+d)$, then c , else d
- Greater Than with 4 inputs (B) (GT4B): if $(a+b) > (c+d)$, then c , else d
- Less Or Equal To with 4 inputs (B) (LOE4B): if $(a+b) \leq (c+d)$, then c , else d
- Greater Or Equal To with 4 inputs (B) (GOE4B): if $(a+b) \geq (c+d)$, then c , else d
- Equal To with 4 inputs (B) (ET4B): if $(a+b) = (c+d)$, then c , else d
- Not Equal To with 4 inputs (B) (NET4B): if $(a+b) \neq (c+d)$, then c , else d

Comparison IF THEN ELSE functions of four arguments (series C):

- Less Than with 4 inputs (C) (LT4C): if $(a+b) < (c+d)$, then $(a+b)$, else $(c+d)$
- Greater Than with 4 inputs (C) (GT4C): if $(a+b) > (c+d)$, then $(a+b)$, else $(c+d)$
- Less Or Equal To with 4 inputs (C) (LOE4C): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c+d)$
- Greater Or Equal To with 4 inputs (C) (GOE4C): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c+d)$
- Equal To with 4 inputs (C) (ET4C): if $(a+b) = (c+d)$, then $(a+b)$, else $(c+d)$
- Not Equal To with 4 inputs (C) (NET4C): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c+d)$

Comparison IF THEN ELSE functions of four arguments (series D):

- Less Than with 4 inputs (D) (LT4D): if $(a+b) < (c+d)$, then $(a+b)$, else $(c-d)$
- Greater Than with 4 inputs (D) (GT4D): if $(a+b) > (c+d)$, then $(a+b)$, else $(c-d)$
- Less Or Equal To with 4 inputs (D) (LOE4D): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c-d)$
- Greater Or Equal To with 4 inputs (D) (GOE4D): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c-d)$
- Equal To with 4 inputs (D) (ET4D): if $(a+b) = (c+d)$, then $(a+b)$, else $(c-d)$
- Not Equal To with 4 inputs (D) (NET4D): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c-d)$

Comparison IF THEN ELSE functions of four arguments (series E):

- Less Than with 4 inputs (E) (LT4E): if $(a+b) < (c+d)$, then $(a+b)$, else $(c*d)$
- Greater Than with 4 inputs (E) (GT4E): if $(a+b) > (c+d)$, then $(a+b)$, else $(c*d)$
- Less Or Equal To with 4 inputs (E) (LOE4E): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c*d)$
- Greater Or Equal To with 4 inputs (E) (GOE4E): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c*d)$
- Equal To with 4 inputs (E) (ET4E): if $(a+b) = (c+d)$, then $(a+b)$, else $(c*d)$
- Not Equal To with 4 inputs (E) (NET4E): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c*d)$

Comparison IF THEN ELSE functions of four arguments (series F):

- Less Than with 4 inputs (F) (LT4F): if $(a+b) < (c+d)$, then $(a+b)$, else (c/d)
- Greater Than with 4 inputs (F) (GT4F): if $(a+b) > (c+d)$, then $(a+b)$, else (c/d)
- Less Or Equal To with 4 inputs (F) (LOE4F): if $(a+b) \leq (c+d)$, then $(a+b)$, else (c/d)
- Greater Or Equal To with 4 inputs (F) (GOE4F): if $(a+b) \geq (c+d)$, then $(a+b)$, else (c/d)
- Equal To with 4 inputs (F) (ET4F): if $(a+b) = (c+d)$, then $(a+b)$, else (c/d)
- Not Equal To with 4 inputs (F) (NET4F): if $(a+b) \neq (c+d)$, then $(a+b)$, else (c/d)

Comparison IF THEN ELSE functions of four arguments (series G):

- Less Than with 4 inputs (G) (LT4G): if $(a+b) < (c+d)$, then $(a*b)$, else $(c+d)$
- Greater Than with 4 inputs (G) (GT4G): if $(a+b) > (c+d)$, then $(a*b)$, else $(c+d)$
- Less Or Equal To with 4 inputs (G) (LOE4G): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c+d)$
- Greater Or Equal To with 4 inputs (G) (GOE4G): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c+d)$
- Equal To with 4 inputs (G) (ET4G): if $(a+b) = (c+d)$, then $(a*b)$, else $(c+d)$
- Not Equal To with 4 inputs (G) (NET4G): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c+d)$

Comparison IF THEN ELSE functions of four arguments (series H):

- Less Than with 4 inputs (H) (LT4H): if $(a+b) < (c+d)$, then $(a*b)$, else $(c-d)$

- Greater Than with 4 inputs (H) (**GT4H**): if $(a+b) > (c+d)$, then $(a*b)$, else $(c-d)$
- Less Or Equal To with 4 inputs (H) (**LOE4H**): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c-d)$
- Greater Or Equal To with 4 inputs (H) (**GOE4H**): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c-d)$
- Equal To with 4 inputs (H) (**ET4H**): if $(a+b) = (c+d)$, then $(a*b)$, else $(c-d)$
- Not Equal To with 4 inputs (H) (**NET4H**): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c-d)$

Comparison IF THEN ELSE functions of four arguments (series I):

- Less Than with 4 inputs (I) (**LT4I**): if $(a+b) < (c+d)$, then $(a*b)$, else $(c*d)$
- Greater Than with 4 inputs (I) (**GT4I**): if $(a+b) > (c+d)$, then $(a*b)$, else $(c*d)$
- Less Or Equal To with 4 inputs (I) (**LOE4I**): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c*d)$
- Greater Or Equal To with 4 inputs (I) (**GOE4I**): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c*d)$
- Equal To with 4 inputs (I) (**ET4I**): if $(a+b) = (c+d)$, then $(a*b)$, else $(c*d)$
- Not Equal To with 4 inputs (I) (**NET4I**): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c*d)$

Comparison IF THEN ELSE functions of four arguments (series J):

- Less Than with 4 inputs (J) (**LT4J**): if $(a+b) < (c+d)$, then $(a*b)$, else (c/d)
- Greater Than with 4 inputs (J) (**GT4J**): if $(a+b) > (c+d)$, then $(a*b)$, else (c/d)
- Less Or Equal To with 4 inputs (J) (**LOE4J**): if $(a+b) \leq (c+d)$, then $(a*b)$, else (c/d)
- Greater Or Equal To with 4 inputs (J) (**GOE4J**): if $(a+b) \geq (c+d)$, then $(a*b)$, else (c/d)
- Equal To with 4 inputs (J) (**ET4J**): if $(a+b) = (c+d)$, then $(a*b)$, else (c/d)
- Not Equal To with 4 inputs (J) (**NET4J**): if $(a+b) \neq (c+d)$, then $(a*b)$, else (c/d)

Comparison IF THEN ELSE functions of four arguments (series K):

- Less Than with 4 inputs (K) (**LT4K**): if $(a+b) < (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Greater Than with 4 inputs (K) (**GT4K**): if $(a+b) > (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Less Or Equal To with 4 inputs (K) (**LOE4K**): if $(a+b) \leq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Greater Or Equal To with 4 inputs (K) (**GOE4K**): if $(a+b) \geq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Equal To with 4 inputs (K) (**ET4K**): if $(a+b) = (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Not Equal To with 4 inputs (K) (**NET4K**): if $(a+b) \neq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$

Comparison IF THEN ELSE functions of four arguments (series L):

- Less Than with 4 inputs (L) (**LT4L**): if $(a+b) < (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Greater Than with 4 inputs (L) (**GT4L**): if $(a+b) > (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Less Or Equal To with 4 inputs (L) (**LOE4L**): if $(a+b) \leq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Greater Or Equal To with 4 inputs (L) (**GOE4L**): if $(a+b) \geq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Equal To with 4 inputs (L) (**ET4L**): if $(a+b) = (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Not Equal To with 4 inputs (L) (**NET4L**): if $(a+b) \neq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$

By selecting the Functions Tab in the [Functions Panel](#), you have full access to the 279 built-in mathematical functions of GeneXproTools 4.0. Here is also the place where you can add **Dynamic UDFs** to your modeling kit.

Functions Static UDFs

Function Selection

No. of Functions: 10 Random Clear

Max. Arity: 1 Default Select All

Dynamic UDFs

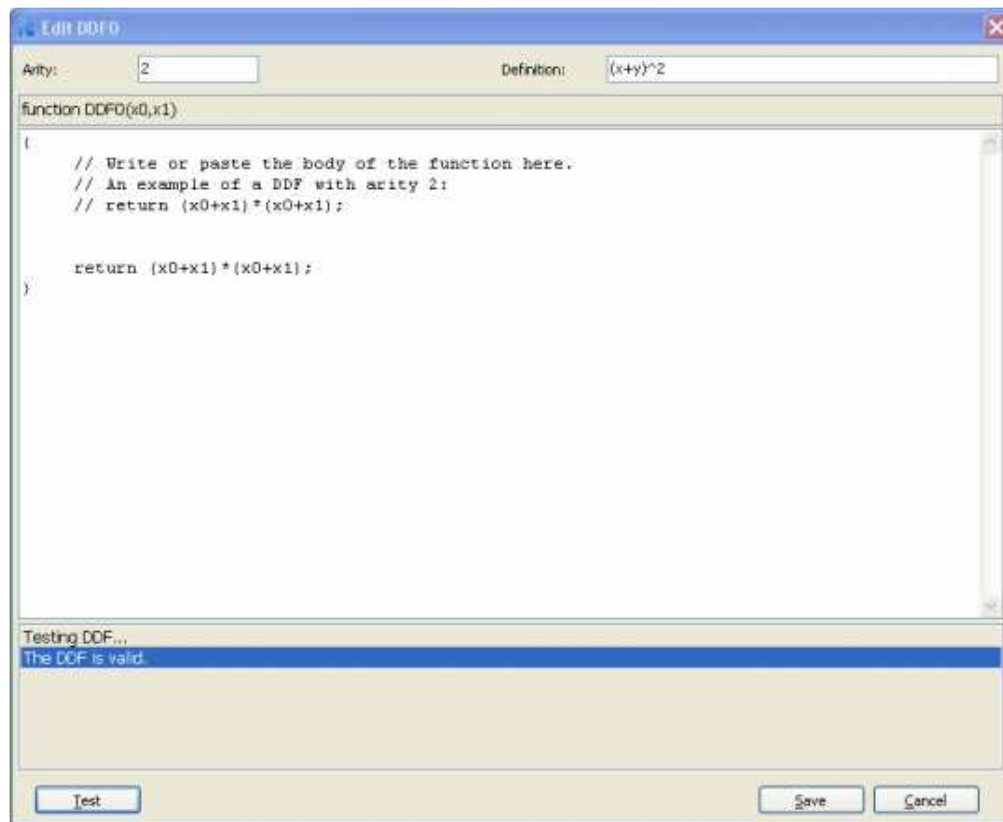
Add Edit Remove

Show: Selected Only Search Add Weight Reduce Weight

Select/Weight	Name	Representation	Arity	Definition
☒ 4	Addition	+	2	(x+y)
☒ 4	Subtraction	-	2	(x-y)
☒ 4	Multiplication	*	2	(x*y)
☒ 1	Division	/	2	(x/y)
☒ 1	Power	Pow	2	pow(x,y)
☒ 1	Square root	Sqrt	1	sqrt(x)
☒ 2	Exponential	Exp	1	exp(x)
☒ 1	Natural logarithm	Ln	1	ln(x)
☒ 1	x to the power of 2	X2	1	x^2
☒ 2	x to the power of 3	X3	1	x^3
☒ 1	x to the power of 5	X5	1	x^5
☒ 1	Cube root	3Rt	1	x^(1/3)
☒ 1	Quartic root	4Rt	1	x^(1/4)
☒ 1	Constant zero function	Zero	1	0(x) = 0
☒ 1	Sine	Sin	1	sin(x)
☒ 2	Cosine	Cos	1	cos(x)
☒ 1	Cotangent	Cot	1	cot(x)
☒ 1	Arctangent	Atan	1	arctan(x)
☒ 1	Arccosecant	Acsc	1	arcsc(x)
☒ 1	Arcsecant	Asec	1	arcsec(x)
☒ 1	Hyperbolic cosine	Cosh	1	cosh(x)
☒ 1	Dynamic UDF	DDF	2	My DDF

To select a function, just check the box on the left. By default, the weight of each function is 1, but you can increase the probability of a function being included in your models by increasing its weight in the Select/Weight column. GeneXproTools automatically balances your function set with the number of independent variables in your data, so now you just have to select the right functions for your problem and then choose their relative proportions by choosing their weights.

To add a **DDF** to your modeling kit, just click the Add button on the Dynamic UDFs frame and the **DDF Editor** appears.



By choosing the arity (minimum is 1 and maximum is 4) in the Arity box, the function header appears below. Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be conveniently tested for compiling errors by pressing the Test button.

In the Definition box, you can write a brief description of the function for your future reference. The text you write there will appear in the Definition column.

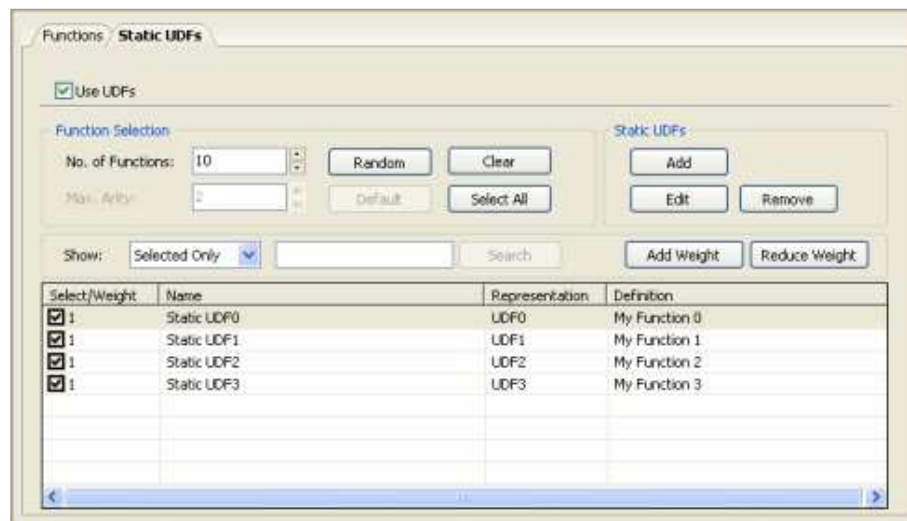
Dynamic UDFs are extremely powerful and interesting tools as they are treated exactly like the built-in functions of GeneXproTools and therefore can be used to model all kinds of relationships between variables or complex expressions. For instance, you can design a DDF so that it will model the sum of four expressions, that is, $DDF = (\text{expression 1}) + (\text{expression 2}) + (\text{expression 3}) + (\text{expression 4})$, where the value of each expression will depend on the context of the DDF in the expression tree. A note of caution, though, although extremely interesting, DDFs decrease considerably the speed of the algorithm and therefore we advise you to choose your functions from the wide set of GeneXproTools 4.0 built-in functions.



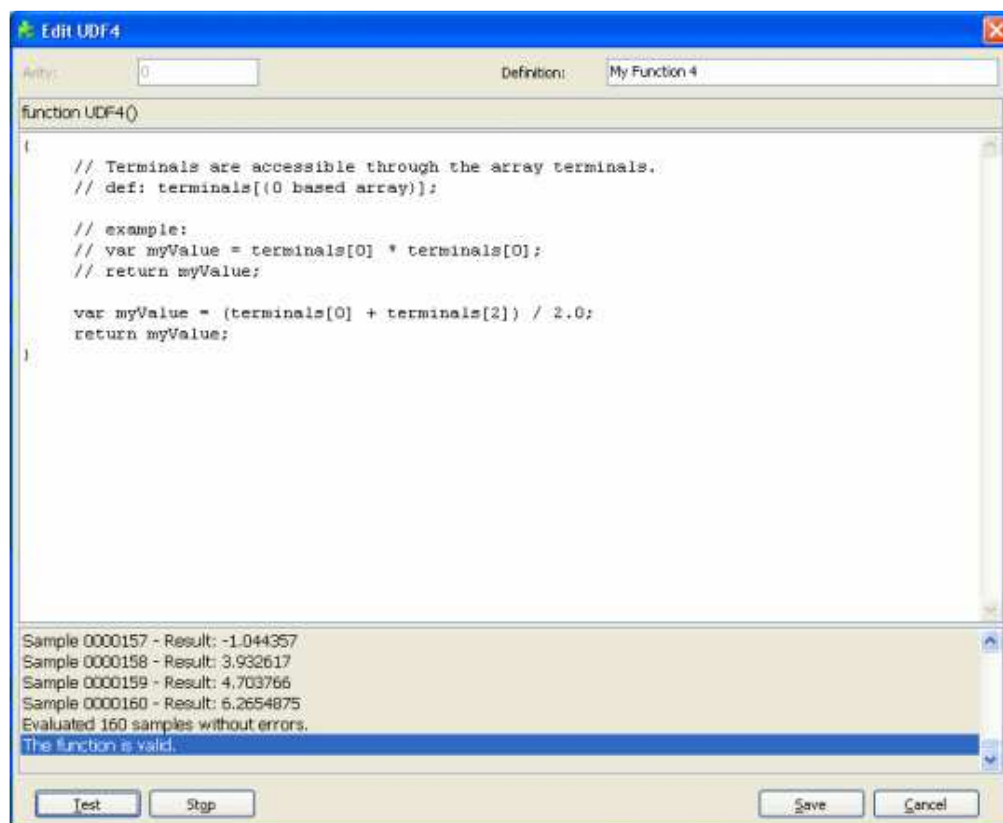
Working with Classification Tools

Static UDFs

In the [Functions Panel](#), by selecting the [Static UDFs Tab](#) and checking the Use UDFs box, the Static UDFs frame lights up and you can now add your own user defined functions.

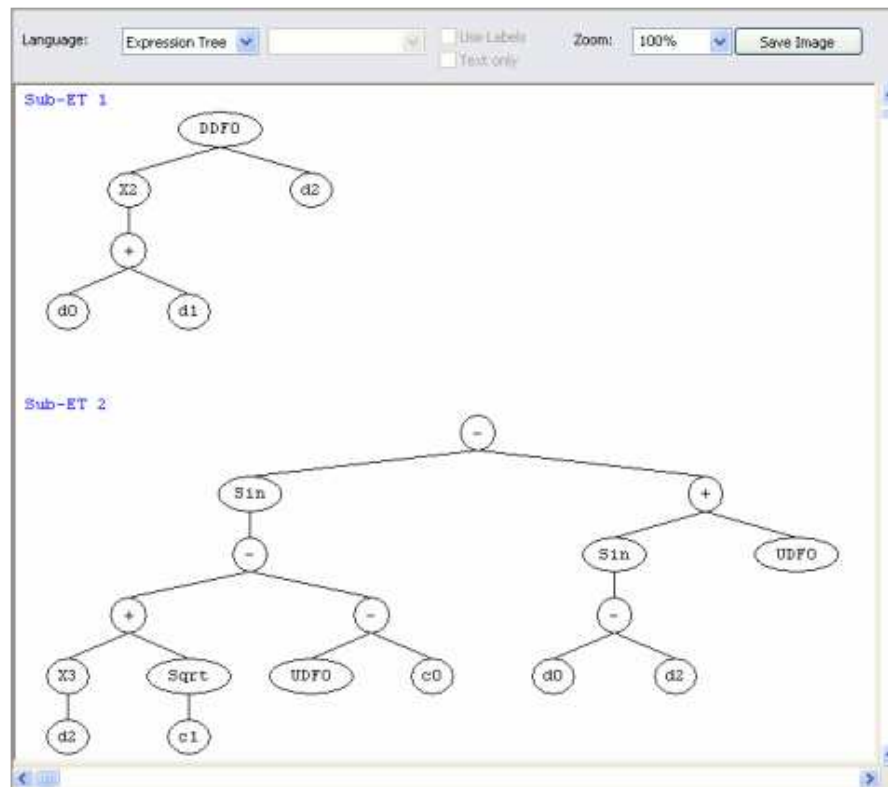


To add a **Static UDF** (UDF for short) to your modeling kit, just click the Add button on the Static UDFs frame and the **UDF Editor** appears with the function header already in place.



Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be rapidly tested for compiling errors by pressing the Test button.

You will notice that the Arity box is disabled and set to zero as all UDFs will occupy leaf nodes in the trees, that is, they behave exactly as terminals.



In the Definition box, you can write a brief description of the function for your future reference. The text you write here will appear in the Definition column of the Static UDFs Tab.

The use of UDFs is a very powerful tool as they allow you to explore inside knowledge about relatively simple relationships that exist between certain variables in your data. For instance, suppose you wanted to explore the fact that the sum of six of your variables is relevant, say, $d0 + d1 + d2 + d3 + d4 + d5$. So you design a UDF to express this relationship, for instance, $UDF1 = d0 + d1 + d2 + d3 + d4 + d5$, and use this new UDF as an extra building block in your modeling kit. The advantage of well designed UDFs is that they make the discovery of more complex models composed of several simpler models much easier.



Working with Classification Tools

Available Chromosomal Architectures

The [chromosome architecture](#) of your models include the **head size**, the **number of genes** and the **linking function**. You choose these parameters in the Settings Panel -> General Settings.

General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 200 [Reset]

Testing Samples: 246 [Reset]

Number of Chromosomes: 30

Head Size: 7

Number of Genes: 3

Linking Function: Addition

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 5

☒ Save Run History

The Head Size determines the complexity of each term in your model. In the [heads of genes](#), the GeneXproTools learning algorithm tries out different arrangements of functions and terminals (variables and constants) in order to model your data. The plasticity of this architecture allows the discovery of a virtually infinite number of models of different sizes and shapes which are afterwards tested and selected during the learning process. The heads of genes are shown in blue in the compact, linear representation of your models in the Model Panel.

Language: Karva Genes: [Use Labels] [Text only] Zoom: 100% [Save Code]

```
Exp. - .Aten. + / + .d2 / .Exp. + * + .d1 .d2 .c0 .d0 .d2 .c1 .d0 .d2 .c1 .d2 .d1 .d1 .d1
+
* .Cos .d1 / Exp. - - - Cos .c0 .c0 * .c0 .d1 .d0 .d0 .d0 .d1 .d0 .d2 .c1 .c0 .c0 .c1 .c1
+
* .Cos / c0 - X3 Cos. - + .c1 * + .c1 .d1 .c1 .d1 .d1 .d1 .d2 .c0 .c1 .d2 .d0 .c1 .c1
```

Numerical Constants:

Gene 1
c0 = 3.410004
c1 = 0.216186

Gene 2
c0 = 9.978333
c1 = 3.325409

Gene 3
c0 = 2.533783
c1 = 3.325409

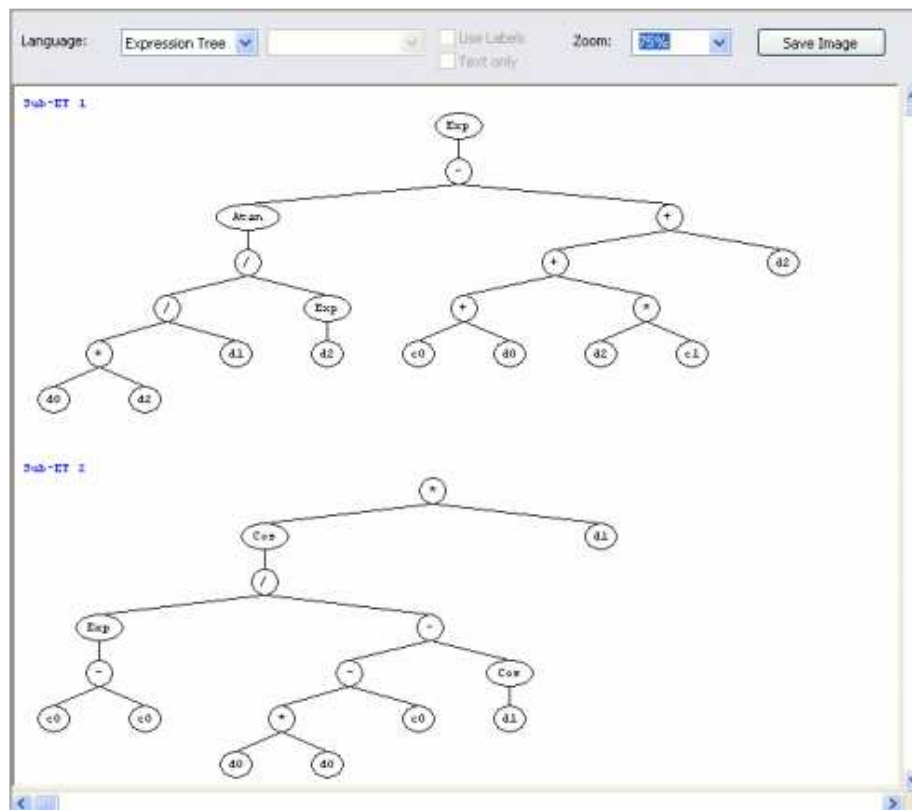
More specifically, the **head size** h of each gene determines the maximum width w and maximum depth d of the [sub-expression trees](#) encoded in the gene, which are given by the formulas:

$$w = (n - 1) * h + 1$$

$$d = ((h + 1) / m) * ((m + 1) / 2)$$

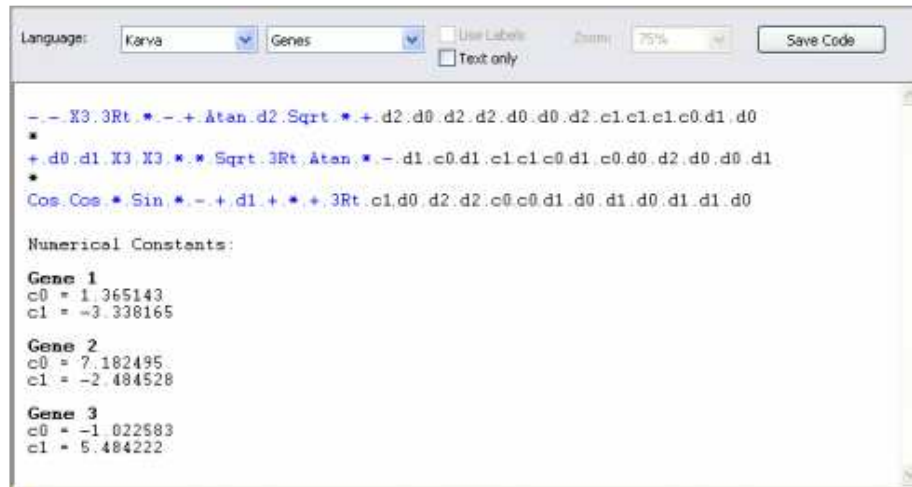
where m is minimum arity and n is maximum arity.

Thus, the GeneXproTools [learning algorithm](#) selects its models between these extreme cases, fine-tuning the ideal size and shape during the evolutionary process without human intervention.



The [number of genes](#) per chromosome is also an important parameter. It will determine the number of (complex) terms in your model as each gene codes for a different [parse tree](#) (sub-expression tree or sub-ET). Theoretically, one could just use a huge single gene in order to evolve very complex models. But the partition of the chromosome into simpler, more manageable units gives an edge to the learning process and more efficient and elegant models can be discovered using [multigenic chromosomes](#).

Whenever the number of genes is greater than one, you must also choose a suitable [linking function](#) for linking the mathematical terms encoded in each gene. GeneXproTools 4.0 allows you to choose **addition**, **subtraction**, **multiplication**, or **division** to link the sub-ETs. As expected, addition (and obviously subtraction) works very well for virtually all problems but sometimes one of the other linkers could be useful for searching different solution spaces and finding a very good, albeit unexpected model.



Working with Classification Tools

Fitness Functions and the Rounding Threshold

The [learning algorithm](#) that GeneXproTools 4.0 uses for classification, classifies your input data into two classes: class "0" and class "1". Obviously, the dependent variable or class in your training and testing sets can only have two distinct values: 0 or 1.

GeneXproTools 4.0 classifies the value returned by the evolved model as "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Classification problems with more than two classes are also easily solved with GeneXproTools 4.0. When you are classifying data into more than two classes, say, n distinct classes, you must decompose your problem into n separate 0/1 classification tasks as follows:

C_1 versus Not C_1
 C_2 versus Not C_2
 ...
 C_n versus Not C_n

Then evolve n different models separately and combine the different models to make the final classification model.

For Classification problems, in the Fitness Function Tab of the [Settings Panel](#) you have access to 30 [built-in fitness functions](#). Additionally, you can also design your own [custom fitness function](#) and explore the solution space with it. In all cases, though, you must always take into consideration the 0/1 rounding threshold.

The screenshot shows the 'Fitness Function' tab in the GeneXproTools interface. The 'Fitness Function' dropdown is set to 'Sensitivity/Specificity'. Below it, there is a checkbox for 'With Parsimony Pressure' which is unchecked. To the right, there are input fields for 'Selection Range' (100), 'Precision' (0.01), and '0/1 Rounding Threshold' (0.5). The 'Custom Fitness Function' section has input fields for 'Max. Training Fitness' (100), 'Max. Testing Fitness' (50), and 'Timeout' (10000). A large text area contains the following JavaScript code:

```
// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[3] = 0/1 rounding threshold
// aParameters[4] = number of samples in the predominant class
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models:
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals
```

A 'Test' button is located at the bottom left of the window.

You can design your own [custom fitness function](#) using the Custom Fitness Function window to write the code of your fitness function. The code for the custom fitness function must be in JavaScript and can be tested before evolving a model with it by pressing the Test button.

The screenshot shows the 'Fitness Function' tab with the 'Fitness Function' dropdown set to 'Custom'. The 'With Parsimony Pressure' checkbox is unchecked. The 'Selection Range' is 100, 'Precision' is 0.01, and '0/1 Rounding Threshold' is 0.1. The 'Custom Fitness Function' section has 'Max. Training Fitness' set to 350, 'Max. Testing Fitness' set to 174, and 'Timeout' set to 10000. The text area contains the following JavaScript code:

```
// Below is an example of a correct fitness function, more specifically,
// the Number of Hits fitness function of GeneXproTools, for which
// the maximum fitness is equal to the number of sample cases:

// NUMBER OF HITS
var nSamples = aParameters[0];
var roundingThreshold = aParameters[3];
var fitness = 0.0;
for (var i=0; i<nSamples; i++)
{
    // Convert the value returned by the model into 1 or 0
    if (aOutputModel[i] >= roundingThreshold)
    {
        aOutputModel[i] = 1;
    }
    else
    {
        aOutputModel[i] = 0;
    }
    fitness += aOutputModel[i];
}

return fitness;
```

A 'Test' button is located at the bottom left of the window.

The 30 **built-in fitness functions** of GeneXproTools 4.0 for Classification are based on the:

- [Number of Hits](#)
- [Hits with Penalty](#)
- [Accuracy](#)
- [Squared Accuracy](#)
- [Sensitivity/Specificity](#)
- [PPV/NPV](#)
- [SSPN](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [R-square](#)
- [Correlation Coefficient](#)
- And all the above with [parsimony pressure](#)

The kind of fitness function you choose will depend most probably on the indicator you are most familiar with. And although there is nothing wrong with this, for all of them can accomplish an efficient evolution, you might want to try different fitness functions for they travel the fitness landscape differently: some of them very straightforwardly in their pursuits while others choose less obvious paths.



Working with Classification Tools

Available Algorithms

GeneXproTools 4.0 uses two different [learning algorithms](#) for Classification problems. The first – the basic gene expression algorithm or simply [Gene Expression Programming](#) (GEP) – does not support the direct manipulation of random numerical constants, whereas the second – [GEP with Random Numerical Constants](#) or [GEP-RNC](#) for short – has a facility for handling them directly. So, these two algorithms search the solution landscape differently and therefore you might wish to try them both on your problems.

The kinds of models these algorithms produce are quite different and, when both of them perform equally well on the problem at hand, you still might prefer one to the other. But there are cases, however, where numerical constants are crucial for an efficient modeling and, therefore, the second algorithm is the default in GeneXproTools 4.0. You activate this algorithm in the Settings Panel -> Numerical Constants by checking the Use Random Numerical Constants box.

The [GEP-RNC algorithm](#) is slightly more complex than [GEP](#) as it uses an additional gene domain (Dc) for encoding the random numerical constants. Consequently, this algorithm comes equipped with an additional set of genetic operators ([RNC mutation](#), [Dc mutation](#), [Dc inversion](#), and [Dc IS transposition](#)) especially developed for handling random numerical constants (if you are not familiar with these operators, please use the default values by clicking the Default button for they work very well in all cases).

And last but not least since these parameters are crucial if you are handling numerical constants directly, you must also choose and adjust the range and type of numerical constants that will be used by the [GEP-RNC algorithm](#) during the learning process. As for the Number of Constants per Gene parameter, a good rule of thumb consists of using a small set of 10 different constants per gene as this seems to provide enough diversity for most problems without inflating the structural complexity much.



Working with Classification Tools

Evolutionary Strategies

The [learning algorithm](#) that GeneXproTools 4.0 uses for Classification, classifies your input data into two classes: class "0" and class "1". Obviously, the class or dependent variable in your training and testing sets can only have two distinct values: 0 or 1.

GeneXproTools 4.0 classifies the value returned by the evolved model as "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Classification problems with more than two classes are also easily solved with GeneXproTools 4.0. When you are classifying data into more than two classes, say n distinct classes, you must decompose your problem into n separate 0/1 classification tasks as follows:

C_1 versus Not C_1
 C_2 versus Not C_2
 ...

C_n versus Not C_n

Then evolve n different models separately and combine the output of the different models to make the final classification.

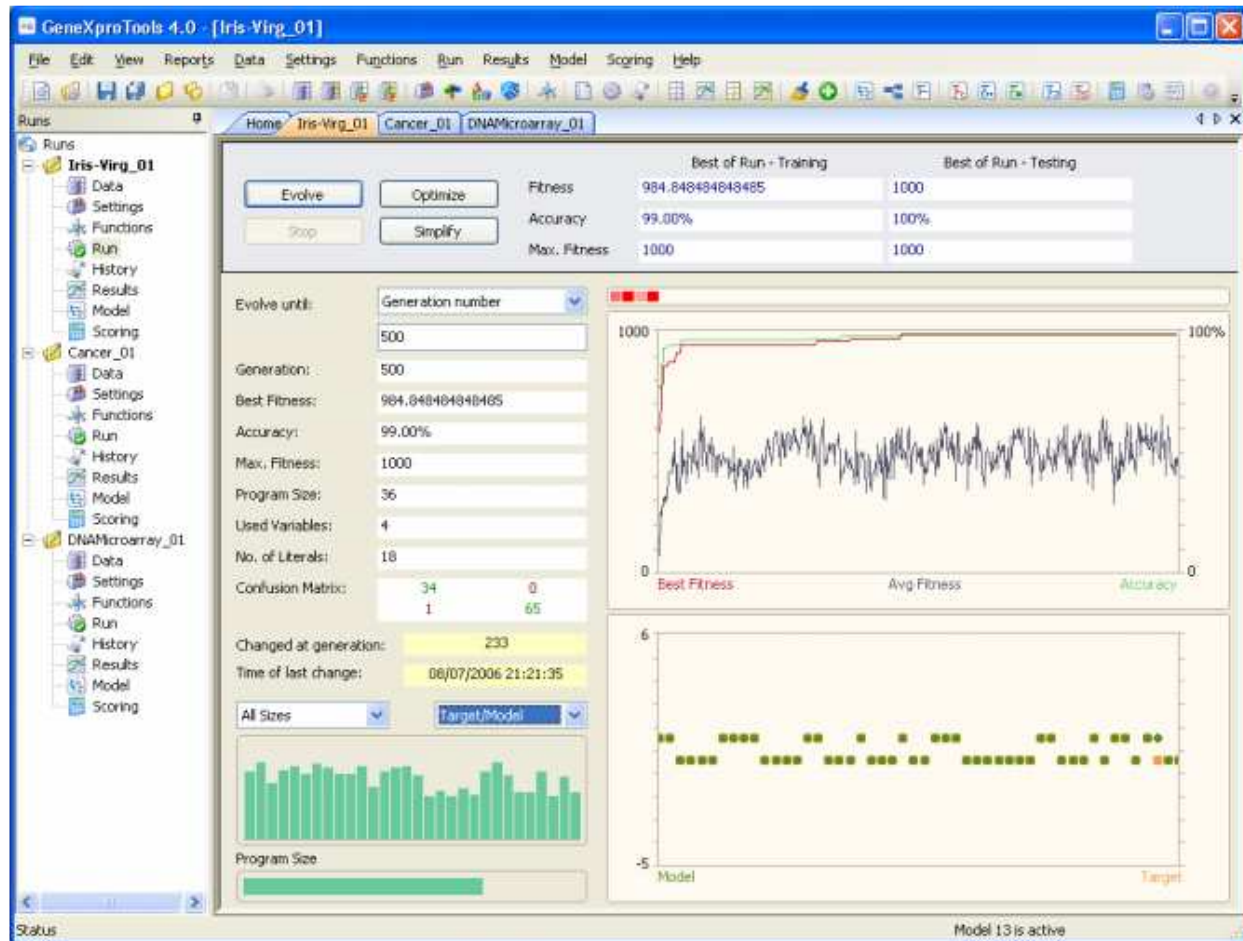
Predicting unknown behavior efficiently is of course the foremost goal in modeling. But extracting knowledge from the blindly designed models is becoming more and more crucial as this knowledge can be used not only to enlighten further the modeling process but also to understand the complex relationships between variables.

So, the evolutionary strategies we recommend in the GeneXproTools [templates](#) for Classification reflect these two main concerns: efficiency and simplicity. Basically, we recommend starting the modeling process with the [GEP-RNC algorithm](#) and a function set well adjusted to the complexity of the problem.

GeneXproTools 4.0 chooses the appropriate template for your problem according to the number of variables in your data. This kind of template is a good starting point that allows you to start the modeling process immediately with just a mouse click. Indeed, even if you are not familiar with evolutionary computation in general and [Gene Expression Programming](#) in particular, you will be able to design complex nonlinear models immediately thanks to the templates of GeneXproTools 4.0. In these templates, all the adjustable parameters of the learning algorithm are already set and, for instance, you don't have to know how to create genetic diversity, how to set the appropriate population size, the chromosome architecture, the fitness function, how to increase the complexity of your models, and so forth. Then, as you learn more about GeneXproTools, you will be able to explore all its modeling tools and create quickly and efficiently very good models that will allow you to understand your data like never before.

There is, however, a very important setting in GeneXproTools that is not controlled by GeneXproTools 4.0 templates and must be wisely chosen by you: the **number of training samples**. Theoretically, if your data is well balanced and in good condition, evolutionarily speaking, the more samples the better. But there's a catch, obviously: the larger the training set the slower evolution or, in other words, the more time will be needed for generations to go by. So, you must compromise here and choose a training set with the appropriate size. A good rule of thumb consists of choosing 8-10 training samples for each attribute or independent variable in your data; all the remaining samples could be used for testing the generalizing capability of the evolved models.

So, after creating a new run you just have to click the Evolve button in the [Run Panel](#) in order to design a model. Then you observe carefully the evolutionary process, especially the Target/Model comparison plot. Then, whenever you see fit, you can stop the run without fear of stopping the evolutionary process prematurely as GeneXproTools 4.0 allows you to continue the evolutionary process at a later time by using the best-of-run model as the starting point ([evolve with seed](#)). For that you just have to click on the Optimize button in the [Run Panel](#).



This strategy has enormous advantages as you might choose to stop the run at any time and then take a closer look at the evolved model. For instance, you can analyze its mathematical representation, its performance in the testing set, evaluate a wide set of statistical functions for a quick and rigorous assessment of its accuracy, see how it performs on a different testing set, and so on. Then you might choose to adjust a few parameters, say, choose a different fitness function, expand the function set, add a neutral gene, apply parsimony pressure, change the training set for model refreshing, choose a different 0/1 rounding threshold, and so forth, and then explore this new evolutionary path. You can repeat this process for as long as you want or until you are completely satisfied with the evolved model.

In addition, if you wish for GeneXproTools 4.0 to increase the complexity of your models automatically, you just have to activate the [Complexity Increase Engine](#) and then click the Evolve button and GeneXproTools will evolve better and better models composed of an increasing number of terms until no increase in best fitness takes place for a certain period of time.

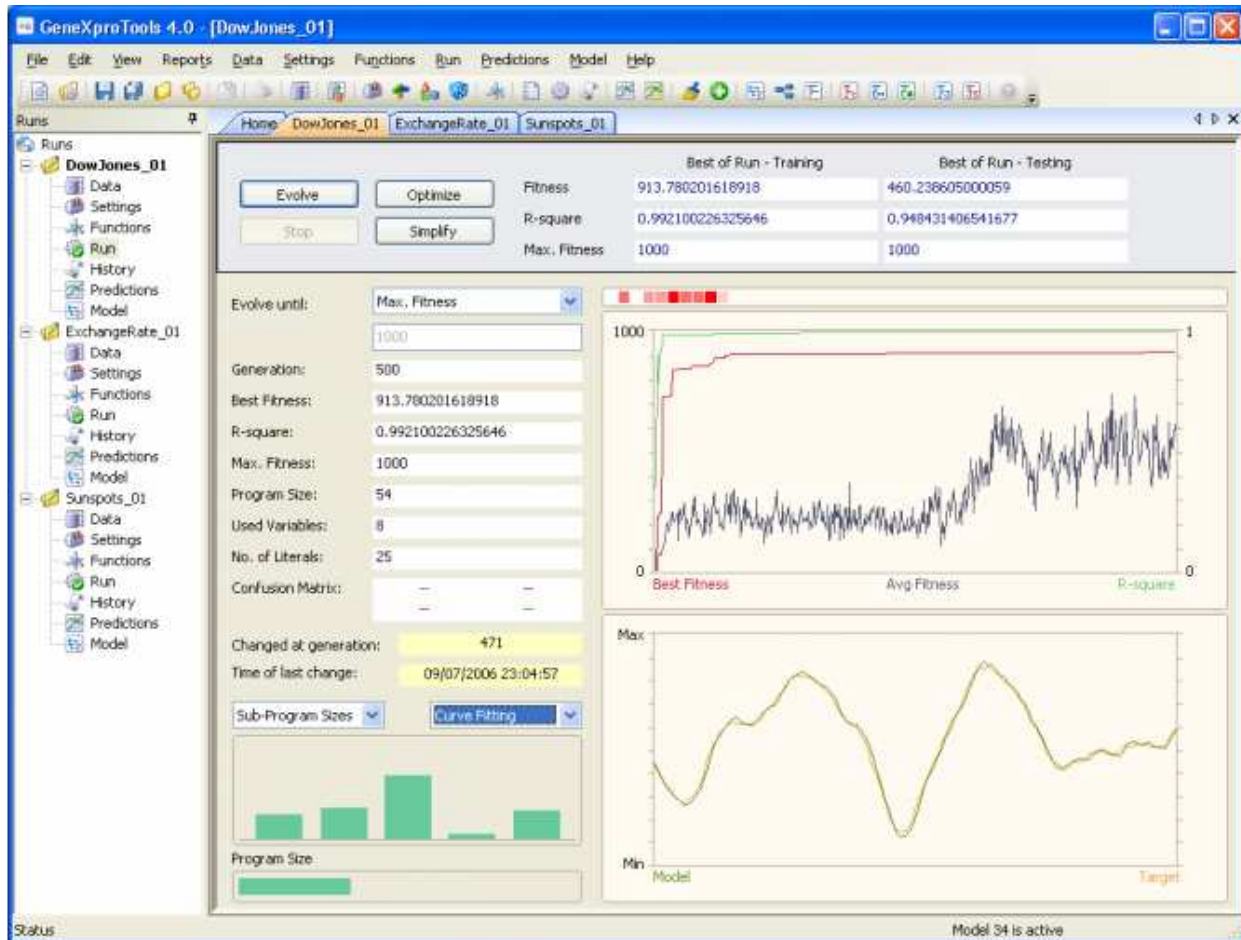


Working with Time Series Prediction Tools

This chapter introduces the Time Series Prediction tools of GeneXproTools and teaches you how to use

them to discover complex nonlinear prediction models. In this chapter, you'll learn how to:

- Load the input data for the GeneXproTools time series prediction algorithm.
- Explore the GeneXproTools function kit for time series prediction problems.
- Explore all the available architectures for building prediction models.
- Explore the best fitness functions for time series prediction problems.
- Explore all the learning algorithms available for time series prediction problems.
- Explore essential evolutionary strategies for an efficient learning.



Working with Time Series Prediction Tools

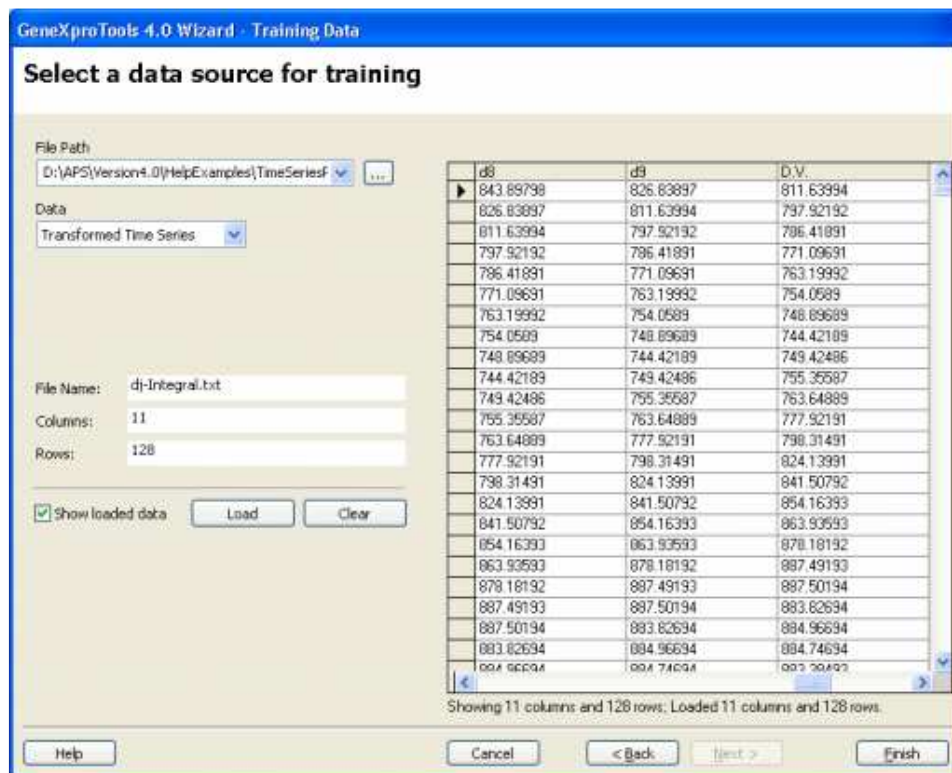
Loading Data

Before evolving a model with GeneXproTools 4.0 you must first load and transform the input time series for the learning algorithm. GeneXproTools 4.0 allows you to work either with databases/Excel or text files. In both cases, though, the time series must be in a single column:

82
66
35
31
7
20
92
154
125
85
68
38
23
10
24
83
132
131
118

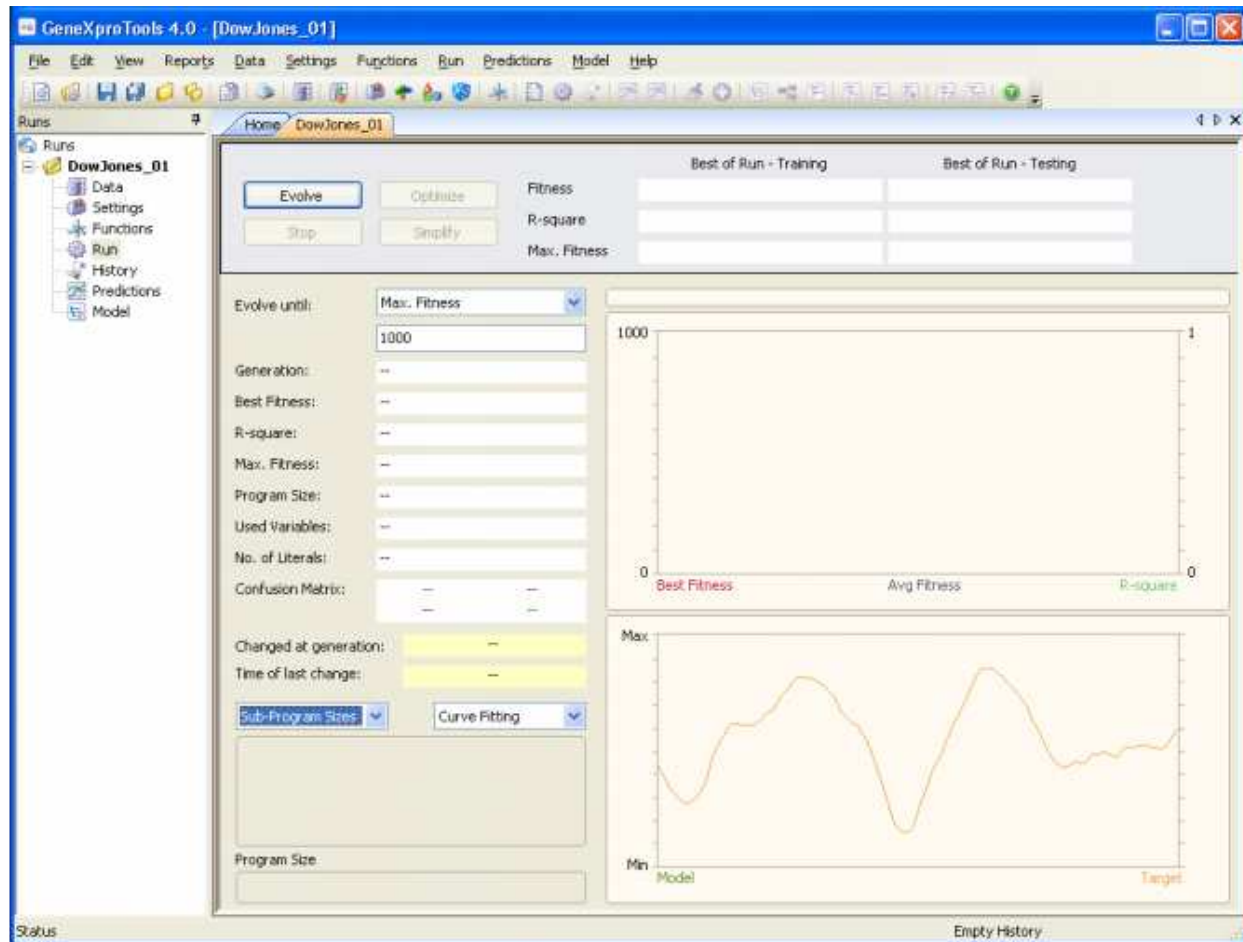
Then GeneXproTools automatically transforms the time series according to your specifications (the embedding dimension, delay time, and prediction mode you chose for your analysis). For instance, for the small time series of 20 observations above, an embedding dimension of 5 and a delay time of 1 is automatically transformed into the following training data:

101 82 66 35 31 7
82 66 35 31 7 20
66 35 31 7 20 92
35 31 7 20 92 154
31 7 20 92 154 125
7 20 92 154 125 85
20 92 154 125 85 68
92 154 125 85 68 38
154 125 85 68 38 23
125 85 68 38 23 10
85 68 38 23 10 24
68 38 23 10 24 83
38 23 10 24 83 132
23 10 24 83 132 131
10 24 83 132 131 118



To Load the Input Time Series for Modeling

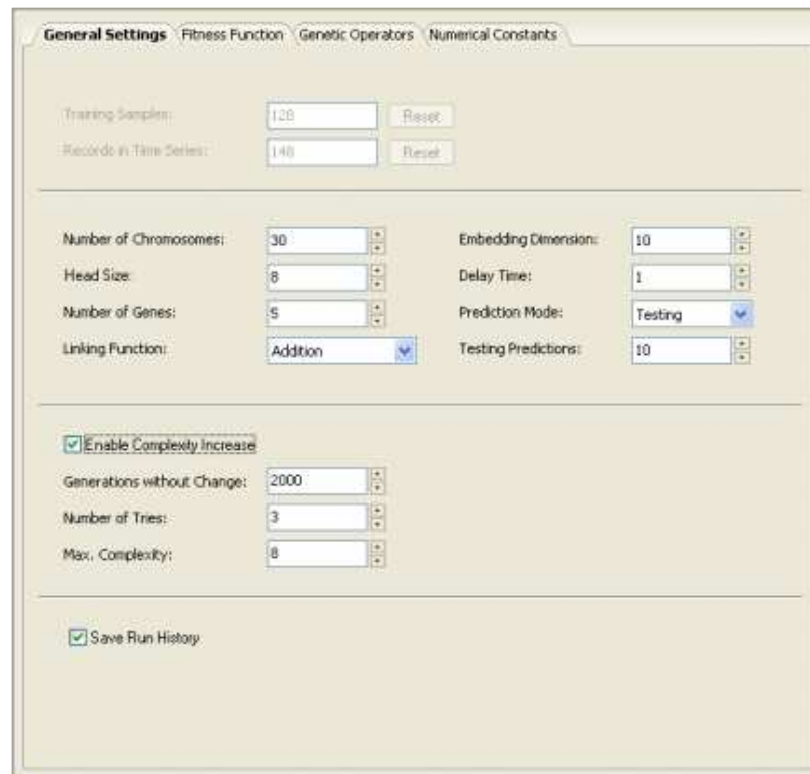
1. Click the File Menu and then choose New.
The New Run wizard appears. You must give a name to your new run file (the default filename extension of GeneXproTools 4.0 run files is .gep) and then choose Time Series Prediction in the Problem Category box. The frame with the parameters used for transforming the time series lights up and you must set the appropriate Embedding Dimension and Delay Time for transforming the time series. Then, depending on whether you wish to make predictions about the future or test the predictive power of the evolved models on known behavior, choose either Testing or Prediction in the Prediction Mode box.
Here you must also choose the kind of source file in the Data Source Type box and choose either Excel & Databases or Text Files.
2. Then go to the Training Data window by clicking the Next button.
Choose the path for the time series by browsing the Open dialog box. The time series is automatically transformed and you can monitor both the original and transformed time series by selecting the appropriate option in the Data box.
3. Click Finish to save your new run file.
The Save As dialog box appears and after choosing the directory where you want your new run file to be saved, the GeneXproTools modeling environment appears. Then you just have to click the Evolve button to create a model as GeneXproTools automatically chooses, from a gallery of templates, default settings that will enable you to create a model immediately.



In time series modeling, be it performed by learning algorithms or conventional statistical methods, it is extremely important the way one chooses to transform the time series before embarking on a complex, usually time consuming modeling process.

First of all, there is the size of the time series to consider and how you choose to transform it, that is, which values will be chosen for the Embedding Dimension and the Delay Time, because all these factors together will determine the number of Training Samples actually used for modeling.

The time series transformation engine of GeneXproTools 4.0 is operational every time you create a new run or every time you change either the Embedding Dimension or the Delay Time, the Prediction Mode or the number of Testing Predictions during a run in the General Settings Tab.



General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 128 [Reset]

Records in Time Series: 148 [Reset]

Number of Chromosomes: 30 [v]

Embedding Dimension: 10 [v]

Head Size: 8 [v]

Delay Time: 1 [v]

Number of Genes: 5 [v]

Prediction Mode: Testing [v]

Linking Function: Addition [v]

Testing Predictions: 10 [v]

☒ Enable Complexity Increase

Generations without Change: 2000 [v]

Number of Tries: 3 [v]

Max. Complexity: 8 [v]

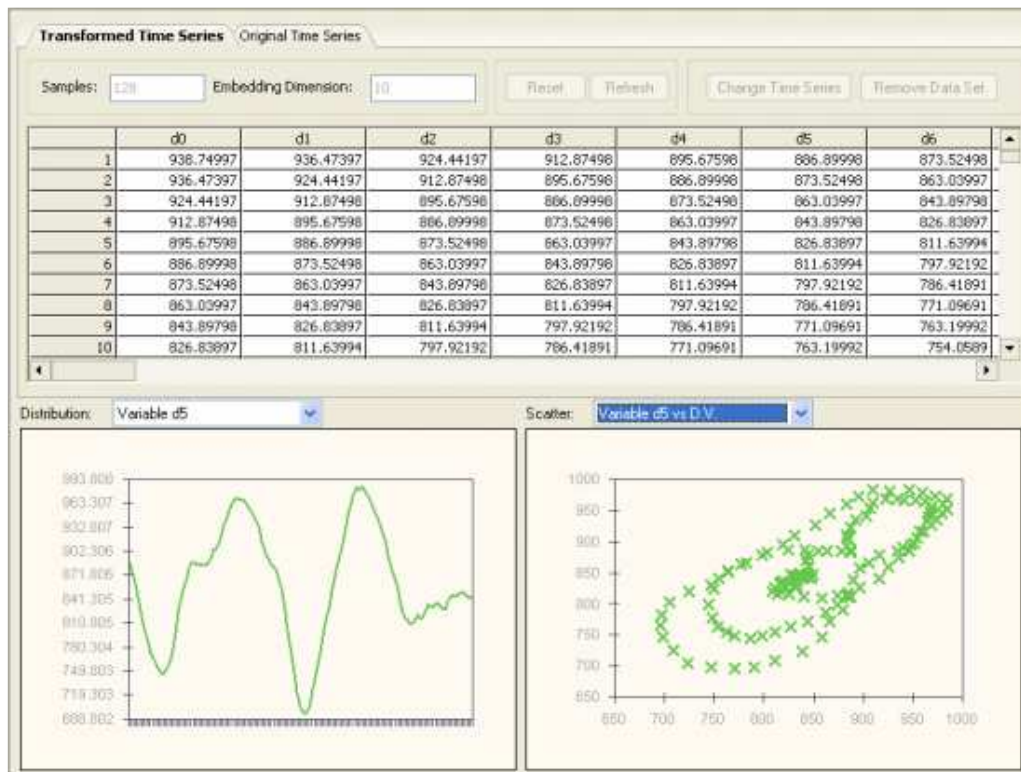
☒ Save Run History

In practice, the embedding dimension corresponds to the number of independent variables or terminals after your time series has been transformed and, therefore, will have a strong impact on the complexity of the problem.

The delay time τ determines how data are processed, that is, continuously if $\tau = 1$ or at τ intervals.

These two parameters, together with the size of the time series and the prediction mode, will determine the final number of training samples after the transformation of the time series.

The data are then ready for evolving prediction models with them and can be visualized in the [Data Panel](#) before evolving a new model.



And, as for Function Finding or Classification problems, it is important to choose a reasonable number of samples for training: an excessively large number of samples will slow the modeling process unnecessarily and will make it harder for the algorithm to find useful patterns. A good rule of thumb consists of using about 8-10 samples for each independent variable in your training data. For instance, for a time series with 100 observations, an embedding dimension of 10 and a delay time of one, will result in a training set with 90 samples which consists of an extremely well balanced training set.

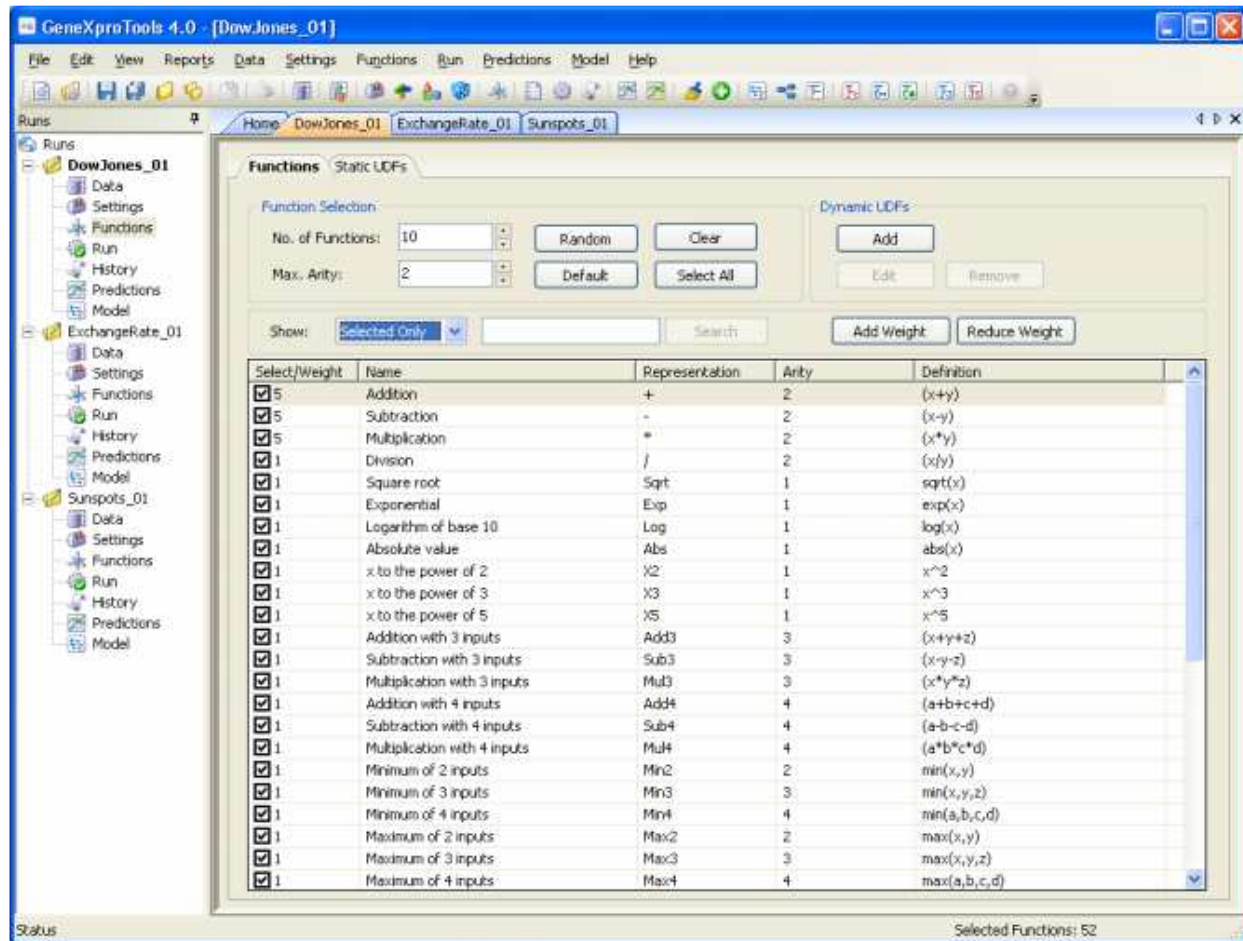


Working with Time Series Prediction Tools

Available Functions

In the Functions Panel you can select and create:

- [Mathematical Functions](#) (including Dynamic UDFs or DDFs).
- [Static UDFs](#).

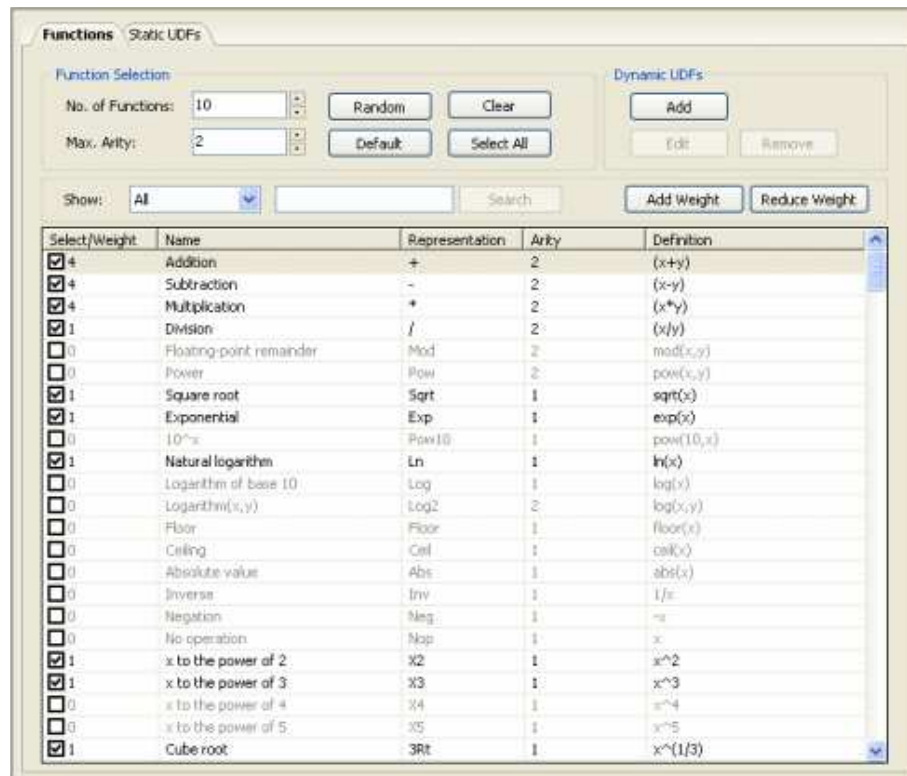


Working with Time Series Prediction Tools

Mathematical Functions and Dynamic UDFs

GeneXproTools 4.0 offers a total of 279 **built-in mathematical functions**, including 186 different IF THEN ELSE rules, that can be used for designing nonlinear regression models. This wide set of mathematical functions allows the evolution of complex and rigorous models quickly built with the most appropriate functions.

The Function Selection Tool of GeneXproTools 4.0 helps you selecting different function sets very quickly through the combination of the Show options with the Random/Default/Clear/Select All buttons plus the Increase/Reduce Weight buttons in the [Functions Panel](#). Furthermore, GeneXproTools automatically balances your function set with the number of independent variables in your data, so now you just have to select the right functions for your problem and then choose their relative proportions by adjusting their weights.



Despite the wide set of GeneXproTools built-in mathematical functions, some users sometimes want to model with different ones. GeneXproTools 4.0 gives the user the possibility of creating custom tailored functions ([Dynamic UDFs](#) or DDFs) and evolve models with them. A note of caution though: the use of DDFs slows considerably the evolutionary process and therefore should be used with moderation.

Below are listed all the 279 **built-in mathematical functions** available in GeneXproTools 4.0, including their representation in [Karva Notation](#) (shown in red) and their mathematical definition (for more details about their implementations, see the [built-in grammars](#) of GeneXproTools):

Basic arithmetic functions:

- Addition (+): $(x+y)$
- Subtraction (-): $(x-y)$
- Multiplication (*): $(x*y)$
- Division (/): (x/y)
- Addition with 3 inputs (Add3): $(x+y+z)$
- Subtraction with 3 inputs (Sub3): $(x-y-z)$
- Multiplication with 3 inputs (Mul3): $(x*y*z)$
- Division with 3 inputs (Div3): $(x/y/z)$
- Addition with 4 inputs (Add4): $(a+b+c+d)$
- Subtraction with 4 inputs (Sub4): $(a-b-c-d)$
- Multiplication with 4 inputs (Mul4): $(a*b*c*d)$
- Division with 4 inputs (Div4): $(a/b/c/d)$

Logarithmic functions:

- Natural logarithm (Ln): $\ln(x)$
- Logarithm of base 10 (Log): $\log(x)$
- Logarithm(x,y) (Log2): $\log(x,y)$

Exponential and power functions:

- Square root (**Sqrt**): $\text{sqrt}(x)$
- Exponential (**Exp**): $\text{exp}(x)$
- Power (**Pow**): x^y
- 10^x (**Pow10**): 10^x
- x^2 (**X2**): x^2
- x^3 (**X3**): x^3
- x^4 (**X4**): x^4
- x^5 (**X5**): x^5
- Cube root (**3Rt**): $x^{(1/3)}$
- Quartic root (**4Rt**): $x^{(1/4)}$
- Quintic root (**5Rt**): $x^{(1/5)}$
- Logistic(x) (**Logi**): $\text{logistic}(x)$
- Logistic(x,y) (**Logi2**): $\text{logistic}(x,y)$
- Logistic(x,y,z) (**Logi3**): $\text{logistic}(x,y,z)$
- Logistic(a,b,c,d) (**Logi4**): $\text{logistic}(a,b,c,d)$
- Gaussian(x) (**Gau**): $\text{gaussian}(x)$
- Gaussian(x,y) (**Gau2**): $\text{gaussian}(x,y)$
- Gaussian(x,y,z) (**Gau3**): $\text{gaussian}(x,y,z)$
- Gaussian(a,b,c,d) (**Gau4**): $\text{gaussian}(a,b,c,d)$

Rounding and other simple functions:

- Floor (**Floor**): $\text{floor}(x)$
- Ceiling (**Ceil**): $\text{ceil}(x)$
- Floating-point remainder (**Mod**): $\text{mod}(x,y)$
- Absolute value (**Abs**): $\text{abs}(x)$
- Inverse (**Inv**): $1/x$
- Negation (**Neg**): $-x$
- Complement (**NOT**): $(1-x)$
- No operation (**Nop**): x

Minimum, maximum, and average functions:

- Minimum of 2 inputs (**Min2**): $\text{min}(x,y)$
- Minimum of 3 inputs (**Min3**): $\text{min}(x,y,z)$
- Minimum of 4 inputs (**Min4**): $\text{min}(a,b,c,d)$
- Maximum of 2 inputs (**Max2**): $\text{max}(x,y)$
- Maximum of 3 inputs (**Max3**): $\text{max}(x,y,z)$
- Maximum of 4 inputs (**Max4**): $\text{max}(a,b,c,d)$
- Average of 2 inputs (**Avg2**): $\text{avg}(x,y)$
- Average of 3 inputs (**Avg3**): $\text{avg}(x,y,z)$
- Average of 4 inputs (**Avg4**): $\text{avg}(a,b,c,d)$

Constant functions:

- Constant zero function (**Zero**): $0(x) = 0$
- Constant one function (**One**): $1(x) = 1$
- $0(x,y)$ (**Zero2**): $0(x,y) = 0$
- $1(x,y)$ (**One2**): $1(x,y) = 1$
- Number π (**Pi**): $\text{Pi}(x) = 3.141592653589793$
- Euler's number (**E**): $E(x) = 2.718281828459045$

Trigonometric functions (x is in radians):

- Sine (**Sin**): $\text{sin}(x)$
- Cosine (**Cos**): $\text{cos}(x)$

- Tangent (**Tan**): $\tan(x)$
- Cosecant (**Csc**): $\csc(x)$
- Secant (**Sec**): $\sec(x)$
- Cotangent (**Cot**): $\cot(x)$

Inverse trigonometric functions:

- Arcsine (**Asin**): $\arcsin(x)$
- Arccosine (**Acos**): $\arccos(x)$
- Arctangent (**Atan**): $\arctan(x)$
- Arccosecant (**Acsc**): $\operatorname{arccsc}(x)$
- Arcsecant (**Asec**): $\operatorname{arcsec}(x)$
- Arccotangent (**Acot**): $\operatorname{arccot}(x)$

Hyperbolic functions:

- Hyperbolic sine (**Sinh**): $\sinh(x)$
- Hyperbolic cosine (**Cosh**): $\cosh(x)$
- Hyperbolic tangent (**Tanh**): $\tanh(x)$
- Hyperbolic cosecant (**Csch**): $\operatorname{csch}(x)$
- Hyperbolic secant (**Sech**): $\operatorname{sech}(x)$
- Hyperbolic cotangent (**Coth**): $\operatorname{coth}(x)$

Inverse hyperbolic functions:

- Inverse hyperbolic sine (**Asinh**): $\operatorname{arcsinh}(x)$
- Inverse hyperbolic cosine (**Acosh**): $\operatorname{arccosh}(x)$
- Inverse hyperbolic tangent (**Atanh**): $\operatorname{arctanh}(x)$
- Inverse hyperbolic cosecant (**Acsch**): $\operatorname{arccsch}(x)$
- Inverse hyperbolic secant (**Asech**): $\operatorname{arcsech}(x)$
- Inverse hyperbolic cotangent (**Acoth**): $\operatorname{arccoth}(x)$

Comparison 0/1 functions of two arguments (OR series):

- OR1 (**OR1**): if $x < 0$ OR $y < 0$, then 1, else 0
- OR2 (**OR2**): if $x \geq 0$ OR $y \geq 0$, then 1, else 0
- OR3 (**OR3**): if $x \leq 0$ OR $y \leq 0$, then 1, else 0
- OR4 (**OR4**): if $x < 1$ OR $y < 1$, then 1, else 0
- OR5 (**OR5**): if $x \geq 1$ OR $y \geq 1$, then 1, else 0
- OR6 (**OR6**): if $x \leq 1$ OR $y \leq 1$, then 1, else 0

Comparison 0/1 functions of two arguments (AND series):

- AND1 (**AND1**): if $x < 0$ AND $y < 0$, then 1, else 0
- AND2 (**AND2**): if $x \geq 0$ AND $y \geq 0$, then 1, else 0
- AND3 (**AND3**): if $x \leq 0$ AND $y \leq 0$, then 1, else 0
- AND4 (**AND4**): if $x < 1$ AND $y < 1$, then 1, else 0
- AND5 (**AND5**): if $x \geq 1$ AND $y \geq 1$, then 1, else 0
- AND6 (**AND6**): if $x \leq 1$ AND $y \leq 1$, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series A):

- Less Than with 2 inputs (A) (**LT2A**): if $x < y$, then x , else y
- Greater Than with 2 inputs (A) (**GT2A**): if $x > y$, then x , else y
- Less Or Equal To with 2 inputs (A) (**LOE2A**): if $x \leq y$, then x , else y
- Greater Or Equal To with 2 inputs (A) (**GOE2A**): if $x \geq y$, then x , else y
- Equal To with 2 inputs (A) (**ET2A**): if $x = y$, then x , else y

- Not Equal To with 2 inputs (A) (**NET2A**): if $x \neq y$, then x , else y

Comparison 0/1 IF THEN ELSE functions of two arguments (series B):

- Less Than with 2 inputs (B) (**LT2B**): if $x < y$, then 1, else 0
- Greater Than with 2 inputs (B) (**GT2B**): if $x > y$, then 1, else 0
- Less Or Equal To with 2 inputs (B) (**LOE2B**): if $x \leq y$, then 1, else 0
- Greater Or Equal To with 2 inputs (B) (**GOE2B**): if $x \geq y$, then 1, else 0
- Equal To with 2 inputs (B) (**ET2B**): if $x = y$, then 1, else 0
- Not Equal To with 2 inputs (B) (**NET2B**): if $x \neq y$, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series C):

- Less Than with 2 inputs (C) (**LT2C**): if $x < y$, then $(x+y)$, else $(x-y)$
- Greater Than with 2 inputs (C) (**GT2C**): if $x > y$, then $(x+y)$, else $(x-y)$
- Less Or Equal To with 2 inputs (C) (**LOE2C**): if $x \leq y$, then $(x+y)$, else $(x-y)$
- Greater Or Equal To with 2 inputs (C) (**GOE2C**): if $x \geq y$, then $(x+y)$, else $(x-y)$
- Equal To with 2 inputs (C) (**ET2C**): if $x = y$, then $(x+y)$, else $(x-y)$
- Not Equal To with 2 inputs (C) (**NET2C**): if $x \neq y$, then $(x+y)$, else $(x-y)$

Comparison IF THEN ELSE functions of two arguments (series D):

- Less Than with 2 inputs (D) (**LT2D**): if $x < y$, then $(x*y)$, else (x/y)
- Greater Than with 2 inputs (D) (**GT2D**): if $x > y$, then $(x*y)$, else (x/y)
- Less Or Equal To with 2 inputs (D) (**LOE2D**): if $x \leq y$, then $(x*y)$, else (x/y)
- Greater Or Equal To with 2 inputs (D) (**GOE2D**): if $x \geq y$, then $(x*y)$, else (x/y)
- Equal To with 2 inputs (D) (**ET2D**): if $x = y$, then $(x*y)$, else (x/y)
- Not Equal To with 2 inputs (D) (**NET2D**): if $x \neq y$, then $(x*y)$, else (x/y)

Comparison IF THEN ELSE functions of two arguments (series E):

- Less Than with 2 inputs (E) (**LT2E**): if $x < y$, then $(x+y)$, else $(x*y)$
- Greater Than with 2 inputs (E) (**GT2E**): if $x > y$, then $(x+y)$, else $(x*y)$
- Less Or Equal To with 2 inputs (E) (**LOE2E**): if $x \leq y$, then $(x+y)$, else $(x*y)$
- Greater Or Equal To with 2 inputs (E) (**GOE2E**): if $x \geq y$, then $(x+y)$, else $(x*y)$
- Equal To with 2 inputs (E) (**ET2E**): if $x = y$, then $(x+y)$, else $(x*y)$
- Not Equal To with 2 inputs (E) (**NET2E**): if $x \neq y$, then $(x+y)$, else $(x*y)$

Comparison IF THEN ELSE functions of two arguments (series F):

- Less Than with 2 inputs (F) (**LT2F**): if $x < y$, then $(x+y)$, else $\sin(x*y)$
- Greater Than with 2 inputs (F) (**GT2F**): if $x > y$, then $(x+y)$, else $\sin(x*y)$
- Less Or Equal To with 2 inputs (F) (**LOE2F**): if $x \leq y$, then $(x+y)$, else $\sin(x*y)$
- Greater Or Equal To with 2 inputs (F) (**GOE2F**): if $x \geq y$, then $(x+y)$, else $\sin(x*y)$
- Equal To with 2 inputs (F) (**ET2F**): if $x = y$, then $(x+y)$, else $\sin(x*y)$
- Not Equal To with 2 inputs (F) (**NET2F**): if $x \neq y$, then $(x+y)$, else $\sin(x*y)$

Comparison IF THEN ELSE functions of two arguments (series G):

- Less Than with 2 inputs (G) (**LT2G**): if $x < y$, then $(x+y)$, else $\text{atan}(x*y)$
- Greater Than with 2 inputs (G) (**GT2G**): if $x > y$, then $(x+y)$, else $\text{atan}(x*y)$
- Less Or Equal To with 2 inputs (G) (**LOE2G**): if $x \leq y$, then $(x+y)$, else $\text{atan}(x*y)$
- Greater Or Equal To with 2 inputs (G) (**GOE2G**): if $x \geq y$, then $(x+y)$, else $\text{atan}(x*y)$
- Equal To with 2 inputs (G) (**ET2G**): if $x = y$, then $(x+y)$, else $\text{atan}(x*y)$
- Not Equal To with 2 inputs (G) (**NET2G**): if $x \neq y$, then $(x+y)$, else $\text{atan}(x*y)$

Comparison IF THEN ELSE functions of three arguments (series A):

- Less Than with 3 inputs (A) (**LT3A**): if $x < 0$, then y , else z
- Greater Than with 3 inputs (A) (**GT3A**): if $x > 0$, then y , else z
- Less Or Equal To with 3 inputs (A) (**LOE3A**): if $x \leq 0$, then y , else z
- Greater Or Equal To with 3 inputs (A) (**GOE3A**): if $x \geq 0$, then y , else z
- Equal To with 3 inputs (A) (**ET3A**): if $x = 0$, then y , else z
- Not Equal To with 3 inputs (A) (**NET3A**): if $x \neq 0$, then y , else z

Comparison IF THEN ELSE functions of three arguments (series B):

- Less Than with 3 inputs (B) (**LT3B**): if $(x+y) < z$, then $(x+y)$, else z
- Greater Than with 3 inputs (B) (**GT3B**): if $(x+y) > z$, then $(x+y)$, else z
- Less Or Equal To with 3 inputs (B) (**LOE3B**): if $(x+y) \leq z$, then $(x+y)$, else z
- Greater Or Equal To with 3 inputs (B) (**GOE3B**): if $(x+y) \geq z$, then $(x+y)$, else z
- Equal To with 3 inputs (B) (**ET3B**): if $(x+y) = z$, then $(x+y)$, else z
- Not Equal To with 3 inputs (B) (**NET3B**): if $(x+y) \neq z$, then $(x+y)$, else z

Comparison IF THEN ELSE functions of three arguments (series C):

- Less Than with 3 inputs (C) (**LT3C**): if $(x+y) < z$, then $(x+y)$, else $(x+z)$
- Greater Than with 3 inputs (C) (**GT3C**): if $(x+y) > z$, then $(x+y)$, else $(x+z)$
- Less Or Equal To with 3 inputs (C) (**LOE3C**): if $(x+y) \leq z$, then $(x+y)$, else $(x+z)$
- Greater Or Equal To with 3 inputs (C) (**GOE3C**): if $(x+y) \geq z$, then $(x+y)$, else $(x+z)$
- Equal To with 3 inputs (C) (**ET3C**): if $(x+y) = z$, then $(x+y)$, else $(x+z)$
- Not Equal To with 3 inputs (C) (**NET3C**): if $(x+y) \neq z$, then $(x+y)$, else $(x+z)$

Comparison IF THEN ELSE functions of three arguments (series D):

- Less Than with 3 inputs (D) (**LT3D**): if $(x+y) < z$, then $(x+y)$, else $(x-z)$
- Greater Than with 3 inputs (D) (**GT3D**): if $(x+y) > z$, then $(x+y)$, else $(x-z)$
- Less Or Equal To with 3 inputs (D) (**LOE3D**): if $(x+y) \leq z$, then $(x+y)$, else $(x-z)$
- Greater Or Equal To with 3 inputs (D) (**GOE3D**): if $(x+y) \geq z$, then $(x+y)$, else $(x-z)$
- Equal To with 3 inputs (D) (**ET3D**): if $(x+y) = z$, then $(x+y)$, else $(x-z)$
- Not Equal To with 3 inputs (D) (**NET3D**): if $(x+y) \neq z$, then $(x+y)$, else $(x-z)$

Comparison IF THEN ELSE functions of three arguments (series E):

- Less Than with 3 inputs (E) (**LT3E**): if $(x+y) < z$, then $(x+y)$, else $(x*z)$
- Greater Than with 3 inputs (E) (**GT3E**): if $(x+y) > z$, then $(x+y)$, else $(x*z)$
- Less Or Equal To with 3 inputs (E) (**LOE3E**): if $(x+y) \leq z$, then $(x+y)$, else $(x*z)$
- Greater Or Equal To with 3 inputs (E) (**GOE3E**): if $(x+y) \geq z$, then $(x+y)$, else $(x*z)$
- Equal To with 3 inputs (E) (**ET3E**): if $(x+y) = z$, then $(x+y)$, else $(x*z)$
- Not Equal To with 3 inputs (E) (**NET3E**): if $(x+y) \neq z$, then $(x+y)$, else $(x*z)$

Comparison IF THEN ELSE functions of three arguments (series F):

- Less Than with 3 inputs (F) (**LT3F**): if $(x+y) < z$, then $(x+y)$, else (x/z)
- Greater Than with 3 inputs (F) (**GT3F**): if $(x+y) > z$, then $(x+y)$, else (x/z)
- Less Or Equal To with 3 inputs (F) (**LOE3F**): if $(x+y) \leq z$, then $(x+y)$, else (x/z)
- Greater Or Equal To with 3 inputs (F) (**GOE3F**): if $(x+y) \geq z$, then $(x+y)$, else (x/z)
- Equal To with 3 inputs (F) (**ET3F**): if $(x+y) = z$, then $(x+y)$, else (x/z)
- Not Equal To with 3 inputs (F) (**NET3F**): if $(x+y) \neq z$, then $(x+y)$, else (x/z)

Comparison IF THEN ELSE functions of three arguments (series G):

- Less Than with 3 inputs (G) (**LT3G**): if $(x+y) < z$, then $(x*y)$, else $(x+z)$
- Greater Than with 3 inputs (G) (**GT3G**): if $(x+y) > z$, then $(x*y)$, else $(x+z)$
- Less Or Equal To with 3 inputs (G) (**LOE3G**): if $(x+y) \leq z$, then $(x*y)$, else $(x+z)$

- Greater Or Equal To with 3 inputs (G) (**GOE3G**): if $(x+y) \geq z$, then $(x*y)$, else $(x+z)$
- Equal To with 3 inputs (G) (**ET3G**): if $(x+y) = z$, then $(x*y)$, else $(x+z)$
- Not Equal To with 3 inputs (G) (**NET3G**): if $(x+y) \neq z$, then $(x*y)$, else $(x+z)$

Comparison IF THEN ELSE functions of three arguments (series H):

- Less Than with 3 inputs (H) (**LT3H**): if $(x+y) < z$, then $(x*y)$, else $(x-z)$
- Greater Than with 3 inputs (H) (**GT3H**): if $(x+y) > z$, then $(x*y)$, else $(x-z)$
- Less Or Equal To with 3 inputs (H) (**LOE3H**): if $(x+y) \leq z$, then $(x*y)$, else $(x-z)$
- Greater Or Equal To with 3 inputs (H) (**GOE3H**): if $(x+y) \geq z$, then $(x*y)$, else $(x-z)$
- Equal To with 3 inputs (H) (**ET3H**): if $(x+y) = z$, then $(x*y)$, else $(x-z)$
- Not Equal To with 3 inputs (H) (**NET3H**): if $(x+y) \neq z$, then $(x*y)$, else $(x-z)$

Comparison IF THEN ELSE functions of three arguments (series I):

- Less Than with 3 inputs (I) (**LT3I**): if $(x+y) < z$, then $(x*y)$, else $(x*z)$
- Greater Than with 3 inputs (I) (**GT3I**): if $(x+y) > z$, then $(x*y)$, else $(x*z)$
- Less Or Equal To with 3 inputs (I) (**LOE3I**): if $(x+y) \leq z$, then $(x*y)$, else $(x*z)$
- Greater Or Equal To with 3 inputs (I) (**GOE3I**): if $(x+y) \geq z$, then $(x*y)$, else $(x*z)$
- Equal To with 3 inputs (I) (**ET3I**): if $(x+y) = z$, then $(x*y)$, else $(x*z)$
- Not Equal To with 3 inputs (I) (**NET3I**): if $(x+y) \neq z$, then $(x*y)$, else $(x*z)$

Comparison IF THEN ELSE functions of three arguments (series J):

- Less Than with 3 inputs (J) (**LT3J**): if $(x+y) < z$, then $(x*y)$, else (x/z)
- Greater Than with 3 inputs (J) (**GT3J**): if $(x+y) > z$, then $(x*y)$, else (x/z)
- Less Or Equal To with 3 inputs (J) (**LOE3J**): if $(x+y) \leq z$, then $(x*y)$, else (x/z)
- Greater Or Equal To with 3 inputs (J) (**GOE3J**): if $(x+y) \geq z$, then $(x*y)$, else (x/z)
- Equal To with 3 inputs (J) (**ET3J**): if $(x+y) = z$, then $(x*y)$, else (x/z)
- Not Equal To with 3 inputs (J) (**NET3J**): if $(x+y) \neq z$, then $(x*y)$, else (x/z)

Comparison IF THEN ELSE functions of three arguments (series K):

- Less Than with 3 inputs (K) (**LT3K**): if $(x+y) < z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Greater Than with 3 inputs (K) (**GT3K**): if $(x+y) > z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Less Or Equal To with 3 inputs (K) (**LOE3K**): if $(x+y) \leq z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Greater Or Equal To with 3 inputs (K) (**GOE3K**): if $(x+y) \geq z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Equal To with 3 inputs (K) (**ET3K**): if $(x+y) = z$, then $(x+y+z)$, else $\sin(x*y*z)$
- Not Equal To with 3 inputs (K) (**NET3K**): if $(x+y) \neq z$, then $(x+y+z)$, else $\sin(x*y*z)$

Comparison IF THEN ELSE functions of three arguments (series L):

- Less Than with 3 inputs (L) (**LT3L**): if $(x+y) < z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Greater Than with 3 inputs (L) (**GT3L**): if $(x+y) > z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Less Or Equal To with 3 inputs (L) (**LOE3L**): if $(x+y) \leq z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Greater Or Equal To with 3 inputs (L) (**GOE3L**): if $(x+y) \geq z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Equal To with 3 inputs (L) (**ET3L**): if $(x+y) = z$, then $(x+y+z)$, else $\tan(x*y*z)$
- Not Equal To with 3 inputs (L) (**NET3L**): if $(x+y) \neq z$, then $(x+y+z)$, else $\tan(x*y*z)$

Comparison IF THEN ELSE functions of four arguments (series A):

- Less Than with 4 inputs (A) (**LT4A**): if $a < b$, then c , else d
- Greater Than with 4 inputs (A) (**GT4A**): if $a > b$, then c , else d
- Less Or Equal To with 4 inputs (A) (**LOE4A**): if $a \leq b$, then c , else d
- Greater Or Equal To with 4 inputs (A) (**GOE4A**): if $a \geq b$, then c , else d
- Equal To with 4 inputs (A) (**ET4A**): if $a = b$, then c , else d
- Not Equal To with 4 inputs (A) (**NET4A**): if $a \neq b$, then c , else d

Comparison IF THEN ELSE functions of four arguments (series B):

- Less Than with 4 inputs (B) (LT4B): if $(a+b) < (c+d)$, then c , else d
- Greater Than with 4 inputs (B) (GT4B): if $(a+b) > (c+d)$, then c , else d
- Less Or Equal To with 4 inputs (B) (LOE4B): if $(a+b) \leq (c+d)$, then c , else d
- Greater Or Equal To with 4 inputs (B) (GOE4B): if $(a+b) \geq (c+d)$, then c , else d
- Equal To with 4 inputs (B) (ET4B): if $(a+b) = (c+d)$, then c , else d
- Not Equal To with 4 inputs (B) (NET4B): if $(a+b) \neq (c+d)$, then c , else d

Comparison IF THEN ELSE functions of four arguments (series C):

- Less Than with 4 inputs (C) (LT4C): if $(a+b) < (c+d)$, then $(a+b)$, else $(c+d)$
- Greater Than with 4 inputs (C) (GT4C): if $(a+b) > (c+d)$, then $(a+b)$, else $(c+d)$
- Less Or Equal To with 4 inputs (C) (LOE4C): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c+d)$
- Greater Or Equal To with 4 inputs (C) (GOE4C): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c+d)$
- Equal To with 4 inputs (C) (ET4C): if $(a+b) = (c+d)$, then $(a+b)$, else $(c+d)$
- Not Equal To with 4 inputs (C) (NET4C): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c+d)$

Comparison IF THEN ELSE functions of four arguments (series D):

- Less Than with 4 inputs (D) (LT4D): if $(a+b) < (c+d)$, then $(a+b)$, else $(c-d)$
- Greater Than with 4 inputs (D) (GT4D): if $(a+b) > (c+d)$, then $(a+b)$, else $(c-d)$
- Less Or Equal To with 4 inputs (D) (LOE4D): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c-d)$
- Greater Or Equal To with 4 inputs (D) (GOE4D): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c-d)$
- Equal To with 4 inputs (D) (ET4D): if $(a+b) = (c+d)$, then $(a+b)$, else $(c-d)$
- Not Equal To with 4 inputs (D) (NET4D): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c-d)$

Comparison IF THEN ELSE functions of four arguments (series E):

- Less Than with 4 inputs (E) (LT4E): if $(a+b) < (c+d)$, then $(a+b)$, else $(c*d)$
- Greater Than with 4 inputs (E) (GT4E): if $(a+b) > (c+d)$, then $(a+b)$, else $(c*d)$
- Less Or Equal To with 4 inputs (E) (LOE4E): if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c*d)$
- Greater Or Equal To with 4 inputs (E) (GOE4E): if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c*d)$
- Equal To with 4 inputs (E) (ET4E): if $(a+b) = (c+d)$, then $(a+b)$, else $(c*d)$
- Not Equal To with 4 inputs (E) (NET4E): if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c*d)$

Comparison IF THEN ELSE functions of four arguments (series F):

- Less Than with 4 inputs (F) (LT4F): if $(a+b) < (c+d)$, then $(a+b)$, else (c/d)
- Greater Than with 4 inputs (F) (GT4F): if $(a+b) > (c+d)$, then $(a+b)$, else (c/d)
- Less Or Equal To with 4 inputs (F) (LOE4F): if $(a+b) \leq (c+d)$, then $(a+b)$, else (c/d)
- Greater Or Equal To with 4 inputs (F) (GOE4F): if $(a+b) \geq (c+d)$, then $(a+b)$, else (c/d)
- Equal To with 4 inputs (F) (ET4F): if $(a+b) = (c+d)$, then $(a+b)$, else (c/d)
- Not Equal To with 4 inputs (F) (NET4F): if $(a+b) \neq (c+d)$, then $(a+b)$, else (c/d)

Comparison IF THEN ELSE functions of four arguments (series G):

- Less Than with 4 inputs (G) (LT4G): if $(a+b) < (c+d)$, then $(a*b)$, else $(c+d)$
- Greater Than with 4 inputs (G) (GT4G): if $(a+b) > (c+d)$, then $(a*b)$, else $(c+d)$
- Less Or Equal To with 4 inputs (G) (LOE4G): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c+d)$
- Greater Or Equal To with 4 inputs (G) (GOE4G): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c+d)$
- Equal To with 4 inputs (G) (ET4G): if $(a+b) = (c+d)$, then $(a*b)$, else $(c+d)$
- Not Equal To with 4 inputs (G) (NET4G): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c+d)$

Comparison IF THEN ELSE functions of four arguments (series H):

- Less Than with 4 inputs (H) (LT4H): if $(a+b) < (c+d)$, then $(a*b)$, else $(c-d)$

- Greater Than with 4 inputs (H) (**GT4H**): if $(a+b) > (c+d)$, then $(a*b)$, else $(c-d)$
- Less Or Equal To with 4 inputs (H) (**LOE4H**): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c-d)$
- Greater Or Equal To with 4 inputs (H) (**GOE4H**): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c-d)$
- Equal To with 4 inputs (H) (**ET4H**): if $(a+b) = (c+d)$, then $(a*b)$, else $(c-d)$
- Not Equal To with 4 inputs (H) (**NET4H**): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c-d)$

Comparison IF THEN ELSE functions of four arguments (series I):

- Less Than with 4 inputs (I) (**LT4I**): if $(a+b) < (c+d)$, then $(a*b)$, else $(c*d)$
- Greater Than with 4 inputs (I) (**GT4I**): if $(a+b) > (c+d)$, then $(a*b)$, else $(c*d)$
- Less Or Equal To with 4 inputs (I) (**LOE4I**): if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c*d)$
- Greater Or Equal To with 4 inputs (I) (**GOE4I**): if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c*d)$
- Equal To with 4 inputs (I) (**ET4I**): if $(a+b) = (c+d)$, then $(a*b)$, else $(c*d)$
- Not Equal To with 4 inputs (I) (**NET4I**): if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c*d)$

Comparison IF THEN ELSE functions of four arguments (series J):

- Less Than with 4 inputs (J) (**LT4J**): if $(a+b) < (c+d)$, then $(a*b)$, else (c/d)
- Greater Than with 4 inputs (J) (**GT4J**): if $(a+b) > (c+d)$, then $(a*b)$, else (c/d)
- Less Or Equal To with 4 inputs (J) (**LOE4J**): if $(a+b) \leq (c+d)$, then $(a*b)$, else (c/d)
- Greater Or Equal To with 4 inputs (J) (**GOE4J**): if $(a+b) \geq (c+d)$, then $(a*b)$, else (c/d)
- Equal To with 4 inputs (J) (**ET4J**): if $(a+b) = (c+d)$, then $(a*b)$, else (c/d)
- Not Equal To with 4 inputs (J) (**NET4J**): if $(a+b) \neq (c+d)$, then $(a*b)$, else (c/d)

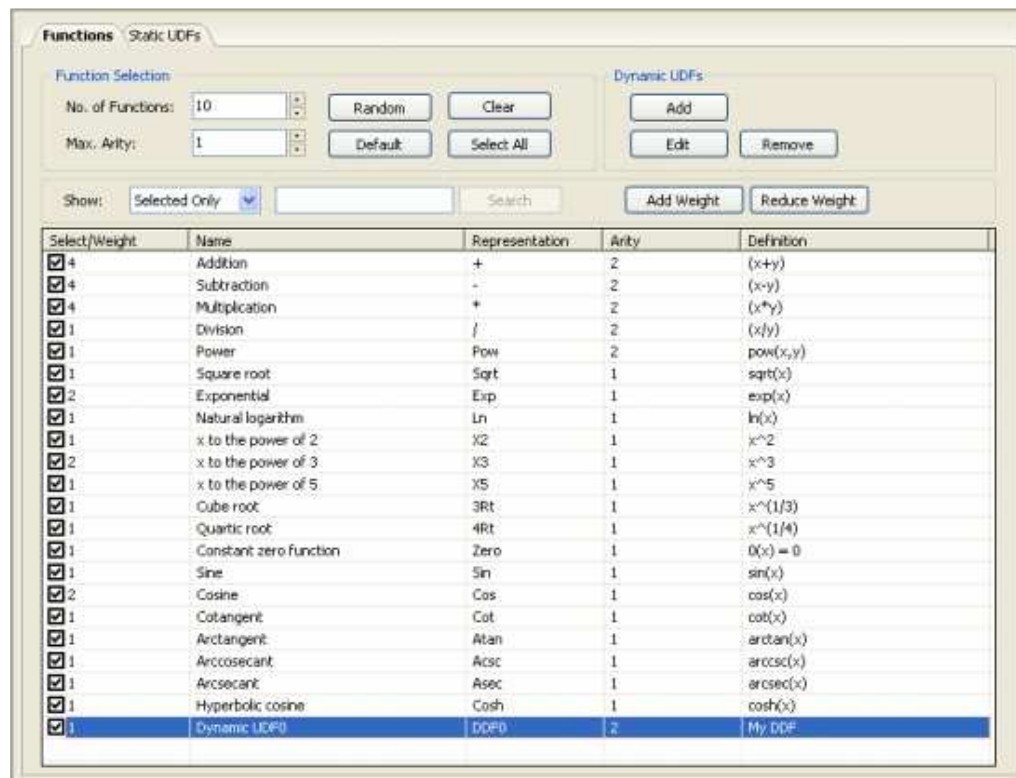
Comparison IF THEN ELSE functions of four arguments (series K):

- Less Than with 4 inputs (K) (**LT4K**): if $(a+b) < (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Greater Than with 4 inputs (K) (**GT4K**): if $(a+b) > (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Less Or Equal To with 4 inputs (K) (**LOE4K**): if $(a+b) \leq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Greater Or Equal To with 4 inputs (K) (**GOE4K**): if $(a+b) \geq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Equal To with 4 inputs (K) (**ET4K**): if $(a+b) = (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- Not Equal To with 4 inputs (K) (**NET4K**): if $(a+b) \neq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$

Comparison IF THEN ELSE functions of four arguments (series L):

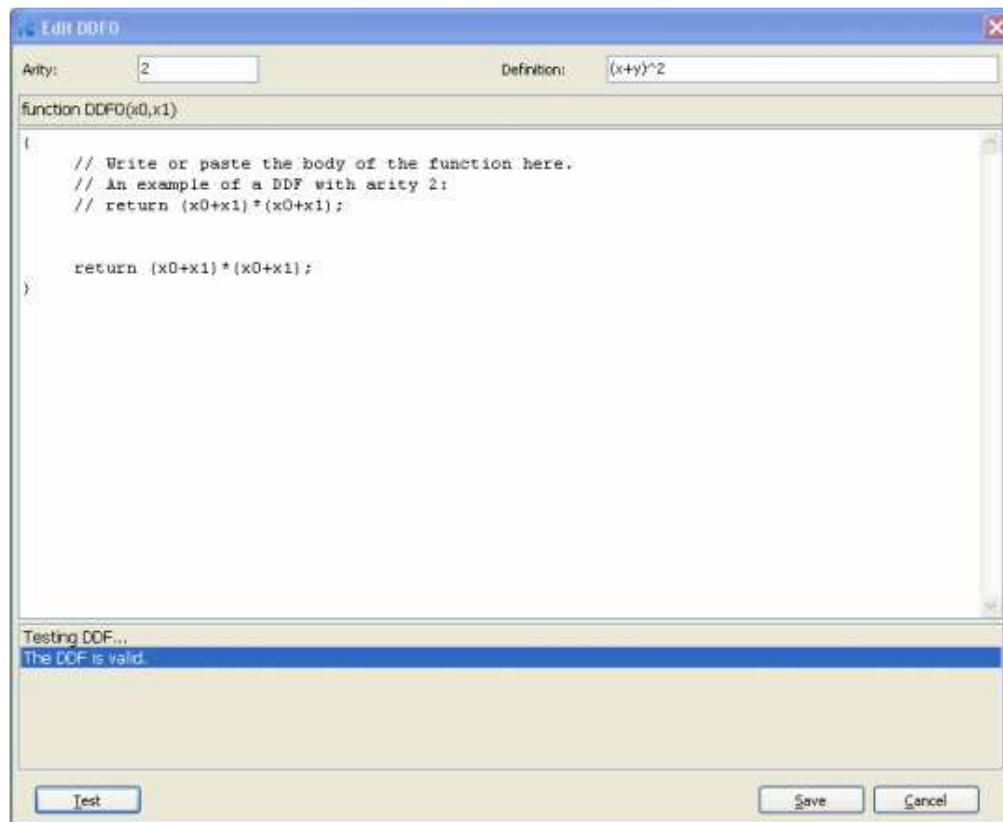
- Less Than with 4 inputs (L) (**LT4L**): if $(a+b) < (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Greater Than with 4 inputs (L) (**GT4L**): if $(a+b) > (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Less Or Equal To with 4 inputs (L) (**LOE4L**): if $(a+b) \leq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Greater Or Equal To with 4 inputs (L) (**GOE4L**): if $(a+b) \geq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Equal To with 4 inputs (L) (**ET4L**): if $(a+b) = (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- Not Equal To with 4 inputs (L) (**NET4L**): if $(a+b) \neq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$

By selecting the Functions Tab in the [Functions Panel](#), you have full access to the 279 built-in mathematical functions of GeneXproTools 4.0. Here is also the place where you can add **Dynamic UDFs** to your modeling kit.



To select a function, just check the box on the left. By default, the weight of each function is 1, but you can increase the probability of a function being included in your models by increasing its weight in the Select/Weight column. GeneXproTools automatically balances your function set with the number of independent variables in your data (that is, the embedding dimension you chose to transform your time series), so now you just have to select the right functions for your problem and then choose their relative proportions by choosing their weights.

To add a **DDF** to your modeling kit, just click the Add button on the Dynamic UDFs frame and the [DDF Editor](#) appears.



By choosing the arity (minimum is 1 and maximum is 4) in the Arity box, the function header appears below. Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be conveniently tested for compiling errors by pressing the Test button.

In the Definition box, you can write a brief description of the function for your future reference. The text you write there will appear in the Definition column.

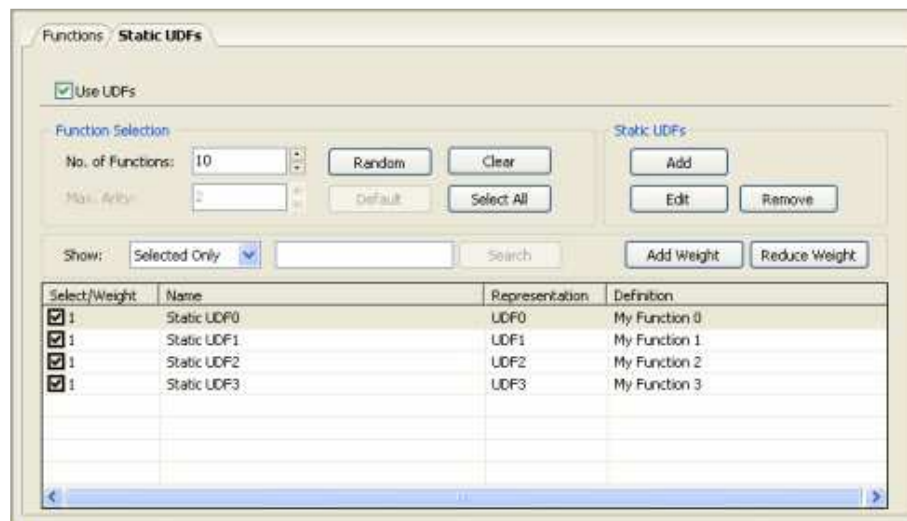
Dynamic UDFs are extremely powerful and interesting tools as they behave exactly like the built-in functions of GeneXproTools and therefore can be used to model all kinds of relationships between variables or complex expressions. For instance, you can design a DDF so that it will model the sum of four expressions, that is, $DDF = (\text{expression 1}) + (\text{expression 2}) + (\text{expression 3}) + (\text{expression 4})$, where the value of each expression will depend on the context of the DDF in the expression tree. A note of caution, though, although extremely interesting, DDFs decrease considerably the speed of the algorithm and therefore we advise you to choose your functions from the wide set of GeneXproTools 4.0 built-in functions.



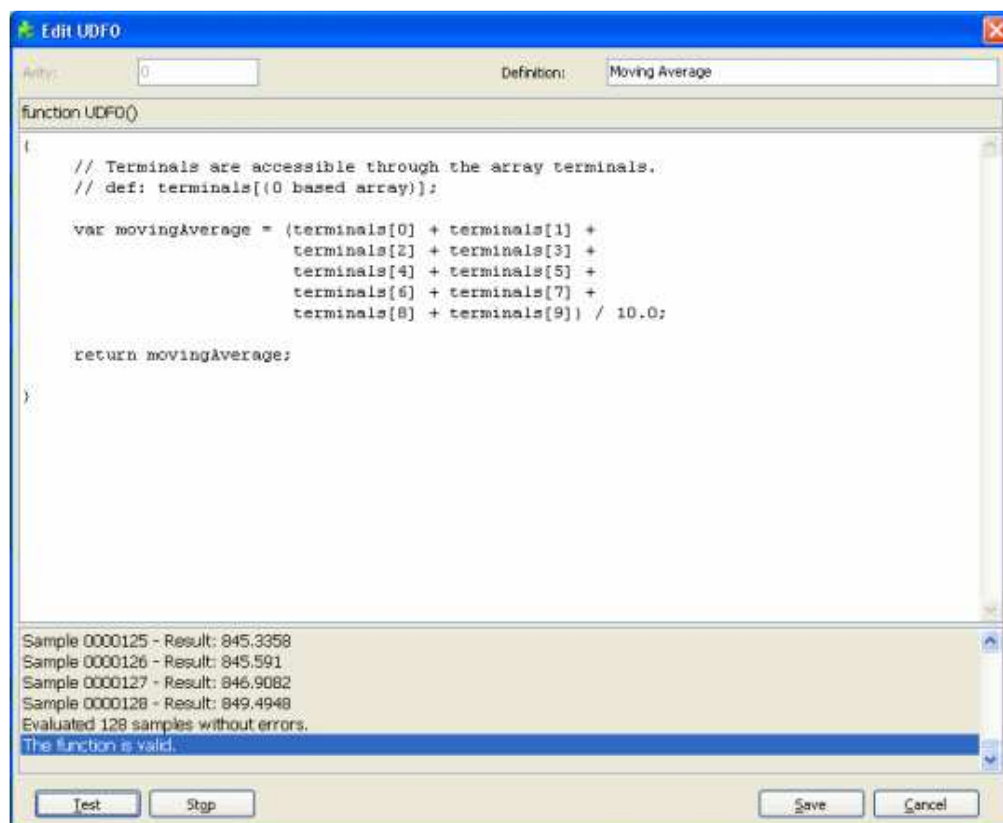
Working with Time Series Prediction Tools

Static UDFs

In the [Functions Panel](#), by selecting the [Static UDFs Tab](#) and checking the Use UDFs box, the Static UDFs frame lights up and you can now add your own user defined functions.

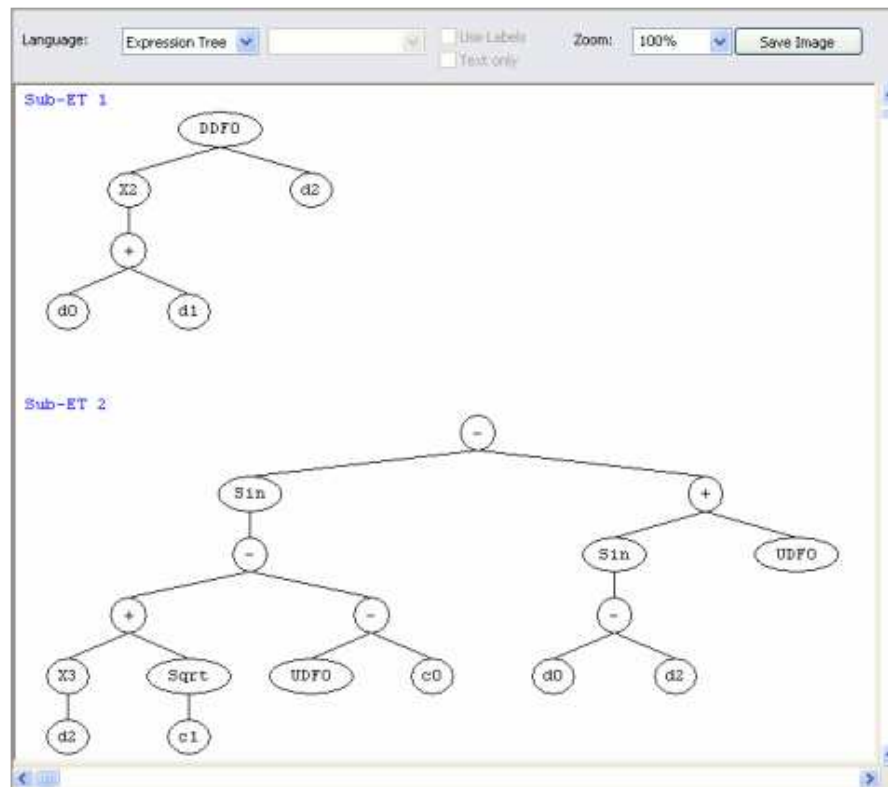


To add a **Static UDF** (UDF for short) to your modeling kit, just click the Add button on the Static UDFs frame and the **UDF Editor** appears with the function header already in place.



Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be rapidly tested for compiling errors by pressing the Test button.

You will notice that the Arity box is disabled and set to zero as all UDFs will occupy leaf nodes in the trees, that is, they behave exactly as terminals.



In the Definition box, you can write a brief description of the function for your future reference. The text you write here will appear in the Definition column of the Static UDFs Tab.

The use of UDFs is a very powerful tool as they allow you to explore inside knowledge about relatively simple relationships that exist between certain variables in your data. For instance, suppose you wanted to try out the impact of a moving average in the modeling process. Then you could design a UDF to express a moving average, for instance, $UDF1 = (d0 + d1 + d2 + d3 + d5 + d6 + d7 + d8 + d9)/10$, and use this new UDF as an extra building block in your modeling kit. The advantage of well designed UDFs is that they make the discovery of more complex models composed of several simpler models much easier.



Working with Time Series Prediction Tools

Available Chromosomal Architectures

The [chromosome architecture](#) of your models include the **head size**, the **number of genes** and the **linking function**. You choose these parameters in the Settings Panel -> General Settings.

The Head Size determines the complexity of each term in your model. In the [heads of genes](#), the GeneXproTools learning algorithm tries out different arrangements of functions and terminals (variables and constants) in order to model your data. The plasticity of this architecture allows the discovery of a virtually infinite number of models of different sizes and shapes which are afterwards tested and selected during the learning process. The heads of genes are shown in blue in the compact, linear representation of your models in the Model Panel.

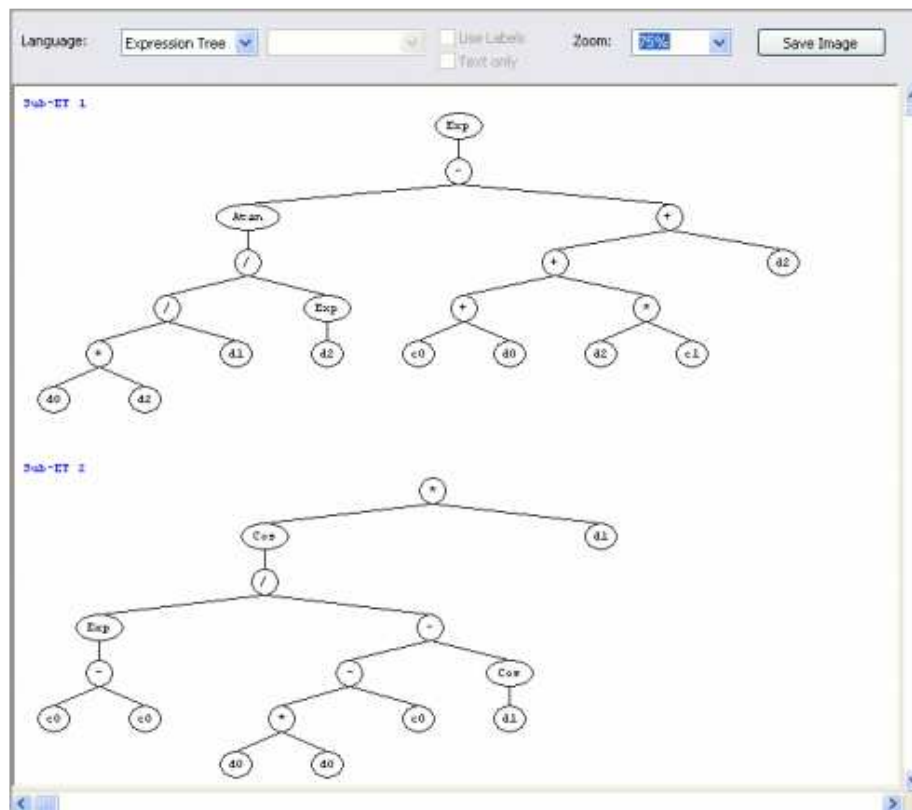
More specifically, the **head size** h of each gene determines the maximum width w and maximum depth d of the [sub-expression trees](#) encoded in the gene, which are given by the formulas:

$$w = (n - 1) * h + 1$$

$$d = ((h + 1) / m) * ((m + 1) / 2)$$

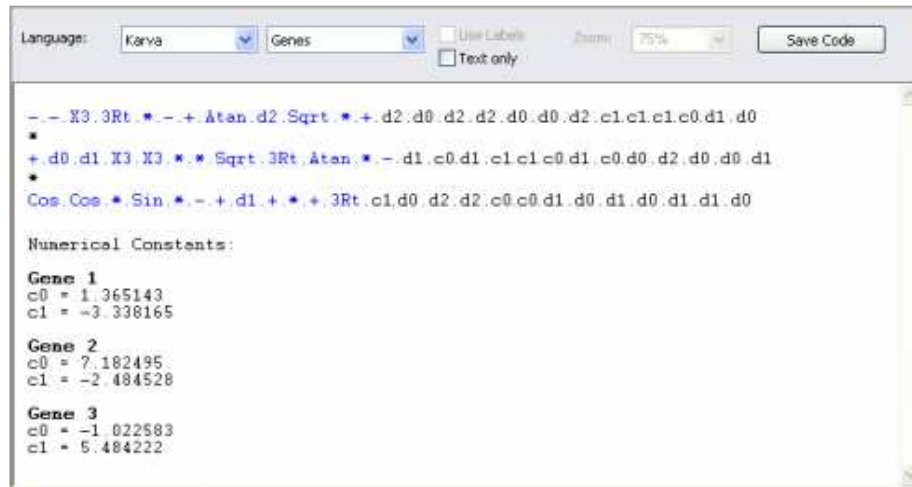
where m is minimum arity and n is maximum arity.

Thus, the GeneXproTools [learning algorithm](#) selects its models between these extreme cases, fine-tuning the ideal size and shape during the evolutionary process without human intervention.



The [number of genes](#) per chromosome is also an important parameter. It will determine the number of (complex) terms in your model as each gene codes for a different [parse tree](#) (sub-expression tree or sub-ET). Theoretically, one could just use a huge single gene in order to evolve very complex models. But the partition of the chromosome into simpler, more manageable units gives an edge to the learning process and more efficient and elegant models can be discovered using [multigenic chromosomes](#).

Whenever the number of genes is greater than one, you must also choose a suitable [linking function](#) for linking the mathematical terms encoded in each gene. GeneXproTools 4.0 allows you to choose **addition**, **subtraction**, **multiplication**, or **division** to link the sub-ETs. As expected, addition (and obviously subtraction) works very well for virtually all problems but sometimes one of the other linkers could be useful for searching different solution spaces and finding a very good, albeit unexpected model.



Working with Time Series Prediction Tools

Fitness Functions

For Time Series Prediction problems, in the Fitness Function Tab of the [Settings Panel](#) you have access to 36 [built-in fitness functions](#). Additionally, you can also design your own [custom fitness function](#) and explore the solution space with it. By choosing Custom in the Fitness Function box, the custom fitness editor is activated.

General Settings **Fitness Function** Genetic Operators Numerical Constants

Fitness Functions: **MSE** Selection Range: 100

☐ With Parsimony Pressure Precision: 0.01

Rounding Threshold: 0.1

Custom Fitness Function

Max. Training Fitness: 1000 Max. Testing Fitness: 1000 Timeout: 10000

```
// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals

// Your custom fitness function must return a value, for example:
// return fitness;
```

Test

You can design your own [custom fitness function](#) using the Custom Fitness Function window to write the code of your fitness function. The code for the custom fitness function must be in JavaScript and can be tested before evolving a model with it by pressing the Test button.

General Settings **Fitness Function** Genetic Operators Numerical Constants

Fitness Functions: **Custom** Selection Range: 100

☐ With Parsimony Pressure Precision: 0.01

Rounding Threshold: 0.1

Custom Fitness Function

Max. Training Fitness: 1000 Max. Testing Fitness: 1000 Timeout: 10000

```
// Below is an example of a correct fitness function:
// the Mean Squared Error (MSE) fitness function of GeneXproTools,
// for which the maximum fitness is equal to 1000:

// MEAN SQUARED ERROR
var nSamples = aParameters[0];
var fitness = 0.0;
var modelMinusTargetSquared = 0.0;
var MSE = 0.0;
for (var i=0; i<nSamples; i++)
{
    var temp1 = 0.0;
    temp1 = aOutputModel[i] - aOutputTarget[i];
    temp1 *= temp1;
    modelMinusTargetSquared += temp1;
}

Correct function.
```

Test

The 36 built-in fitness functions of GeneXproTools 4.0 for Time Series Prediction:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)
- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)
- [R-square](#)
- [Correlation Coefficient](#)
- And all the above with [parsimony pressure](#)

The kind of fitness function you choose will depend most probably on the statistical function you are most familiar with. And although there is nothing wrong with this for all of them can accomplish an efficient evolution, you might want to try different fitness functions for they travel the fitness landscape differently: some of them very straightforwardly in their pursuits while others choose less obvious paths.



Working with Time Series Prediction Tools

Available Algorithms

GeneXproTools 4.0 uses two different [learning algorithms](#) for Time Series Prediction problems. The first – the basic gene expression algorithm or simply [Gene Expression Programming](#) (GEP) – does not support the direct manipulation of random numerical constants, whereas the second – [GEP with Random Numerical Constants](#) or [GEP-RNC](#) for short – has a facility for handling them directly. So, these two algorithms search the solution landscape differently and therefore you might wish to try them both on your problems.

The kinds of models these algorithms produce are quite different and, when both of them perform equally well on the problem at hand, you might still prefer one to the other. But there are cases, however, where numerical constants are crucial for an efficient modeling and, therefore, the second algorithm is the default in GeneXproTools 4.0. You activate this algorithm in the Settings Panel -> Numerical Constants by checking the Use Random Numerical Constants box.

The [GEP-RNC algorithm](#) is slightly more complex than [GEP](#) as it uses an additional gene domain (Dc) for encoding the random numerical constants. Consequently, this algorithm comes equipped with an additional set of genetic operators ([RNC mutation](#), [Dc mutation](#), [Dc inversion](#), and [Dc IS transposition](#)) especially developed for handling random numerical constants (if you are not familiar with these operators, please use the default values by clicking the Default button for they work very well in all cases).

And last but not least since these parameters are crucial if you are handling numerical constants directly, you must also choose and adjust the range and type of numerical constants that will be used by the [GEP-RNC algorithm](#) during the learning process. As for the Number of Constants per Gene parameter, a good rule of thumb consists of using a small set of 10 different constants per gene as this seems to provide enough diversity for most problems without inflating the structural complexity much.

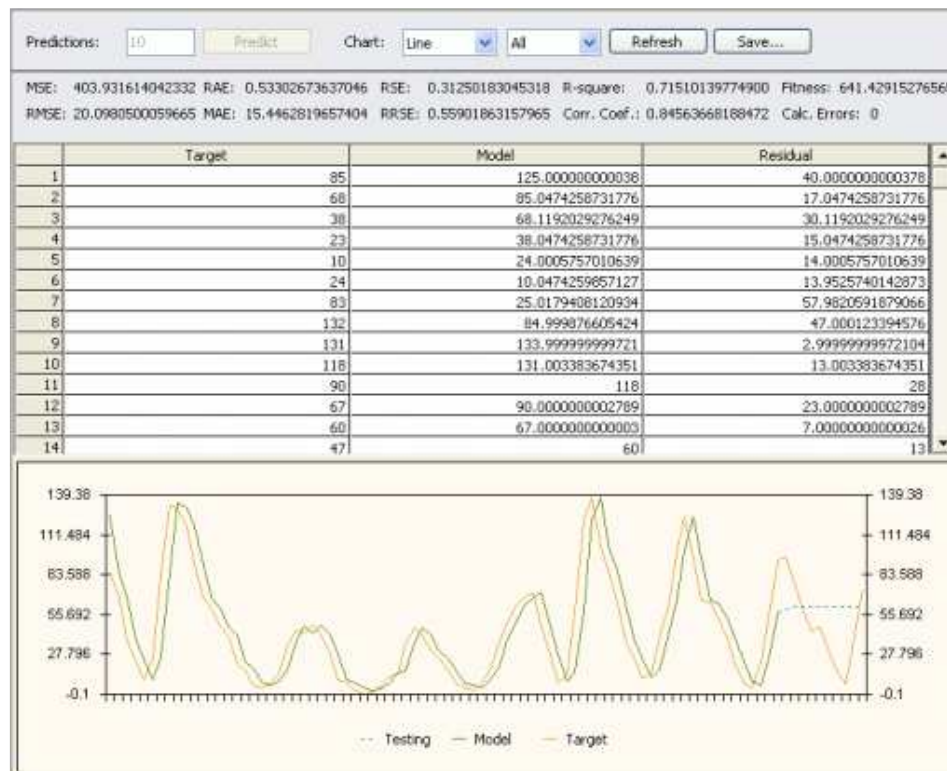


Working with Time Series Prediction Tools

Evolutionary Strategies

Making accurate predictions about the future is of course the foremost goal in time series prediction. But extracting knowledge from the blindly designed models is also important as this knowledge can be used not only to enlighten further the modeling process but also to understand the past.

The greatest challenge in time series predictive modeling consists of avoiding local optima in the solution landscape. And for most time series there is a prevalent local optimum that makes it very difficult to design good predictive models as all modeling treks seem to progress insidiously in that direction. As you most probably guessed, this strong attractor consists of using the present state for predicting the next, giving rise to models with fabulous statistics but nil predictive power. These models can be easily identified as they produce the well-known shadow plots in time series analysis.



So, for time series prediction, the art of good modeling consists basically in finding ways around this strong attractor. And the [learning algorithms](#) of GeneXproTools 4.0 provide you with the necessary tools and strategies for finding good time series predictive models. The most important are outlined below:

- Choose a varied function set by making use of the Function Selection Tool of GeneXproTools 4.0, especially the Random button. Function sets that work extremely well for most time series prediction problems involve a combination of the basic arithmetic operators conveniently weighted (say, four times each) with about thirty randomly chosen functions of one argument.
- Start the modeling process with a relatively complex architecture, say, 4-5 genes with a head size of 7-10, in order to make it difficult for the algorithm to stumble upon this strong attractor.
- Or you can start with a simpler architecture, say, one gene with a head size of 7-10, but then stop the evolutionary process before the attractor is found (or you can stop the evolutionary process when the attractor is found, as GeneXproTools 4.0 allows you to select any of the best-of-generation models of a run and use it as seed). Then you can use one of these models to genetically modify the seed by introducing an additional neutral gene. This neutral gene paves the way to more complex models. Theoretically, by repeating this process, you can approximate any continuous function to an arbitrary precision if there is a sufficient number of terms.
- GeneXproTools 4.0 also allows you to increase the complexity of your model automatically by activating the Complexity Increase Engine in the Settings Panel -> General Settings Tab. Then you must fill the Generations Without Change box to choose the period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction or a neutral gene (an extra term) is automatically added to the evolving models; the Number of Tries box corresponds to the number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models; in the Maximum Complexity box you write the maximum number of terms (genes) you'll allow in your model and no other terms will be introduced beyond this threshold.

General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 128 [Reset]

Records in Time Series: 148 [Reset]

Number of Chromosomes: 30

Embedding Dimension: 10

Head Size: 8

Delay Time: 1

Number of Genes: 5

Prediction Mode: Testing

Linking Function: Addition

Testing Predictions: 10

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 8

☒ Save Run History

So, the evolutionary strategies we recommend in the GeneXproTools [templates](#) for Time Series Prediction are all oriented toward the creation of models with good predictive power. This means that the models created using the templates will be highly sophisticated and will require a considerable amount of time to design. However, if you wish to create less complex models you can stop the evolutionary process any time you see fit and check its capabilities at making predictions about the future in the [Predictions Panel](#). Then you can try to design a better model by continuing the evolutionary process with your current model as [seed](#).

GeneXproTools 4.0 chooses the appropriate template for your problem according to the number of variables in your data. This kind of template is a good starting point that allows you to start the modeling process immediately with just a mouse click. Indeed, even if you are not familiar with evolutionary computation in general and [Gene Expression Programming](#) in particular, you will be able to design complex nonlinear models immediately thanks to the templates of GeneXproTools 4.0. In the provided templates, all the adjustable parameters of the learning algorithm are already set and, for instance, you don't have to know how to create genetic diversity, how to set the appropriate population size, the chromosome architecture, the fitness function, how to increase the complexity of your models, and so forth. Then, as you learn more about GeneXproTools, you will be able to explore all its modeling tools and create quickly and efficiently very good models that will allow you to make accurate predictions about the future.

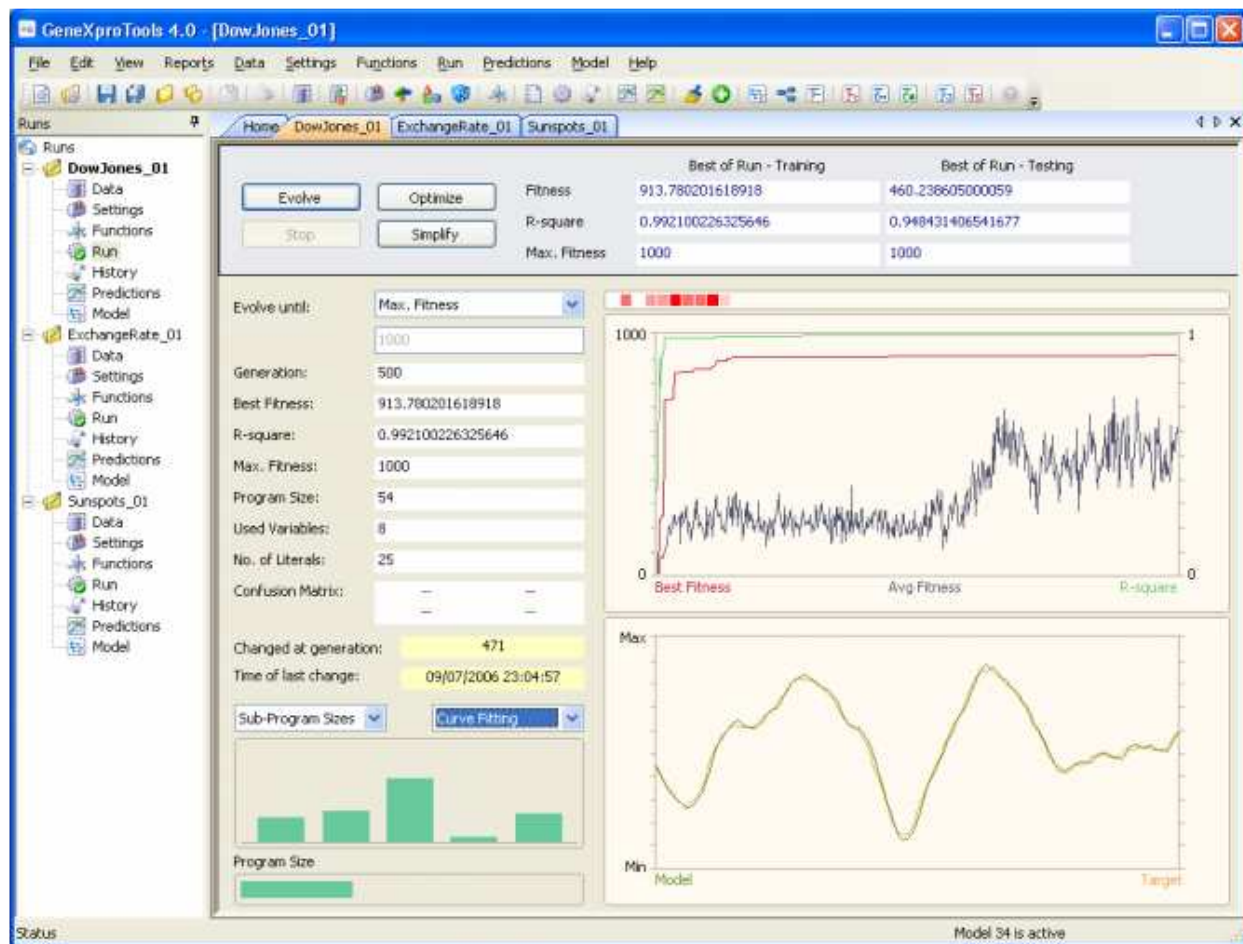
There is, however, a very important setting in GeneXproTools that is not controlled by GeneXproTools 4.0 templates and must be wisely chosen by you: the number of Training Samples which, in Time Series Prediction, depends not only on the size of the time series but also on the Embedding Dimension, the Delay Time, and the Prediction Mode.

Thus, in practice, the embedding dimension corresponds to the number of independent variables or terminals after your time series has been transformed and, therefore, will have a strong impact on the complexity of the problem. The delay time τ determines how data are processed, that is, continuously if $\tau = 1$ or at τ intervals.

As stated for Function Finding or Classification problems, the larger the training set the slower

evolution or, in other words, the more time will be needed for generations to go by. So, you must compromise here and choose a training set with the appropriate size. A good rule of thumb consists of choosing 8-10 training samples for each independent variable in your data.

So, after creating a new run you just have to click the Evolve button in the [Run Panel](#) in order to design a model. Then you observe carefully the evolutionary process, especially the Curve Fitting plot. Then, whenever you see fit, you can stop the run without fear of stopping the evolutionary process prematurely as GeneXproTools 4.0 allows you to continue the evolutionary process at a later time by using the best-of-run model as the starting point ([evolve with seed](#)). For that you just have to click on the Optimize button in the [Run Panel](#).



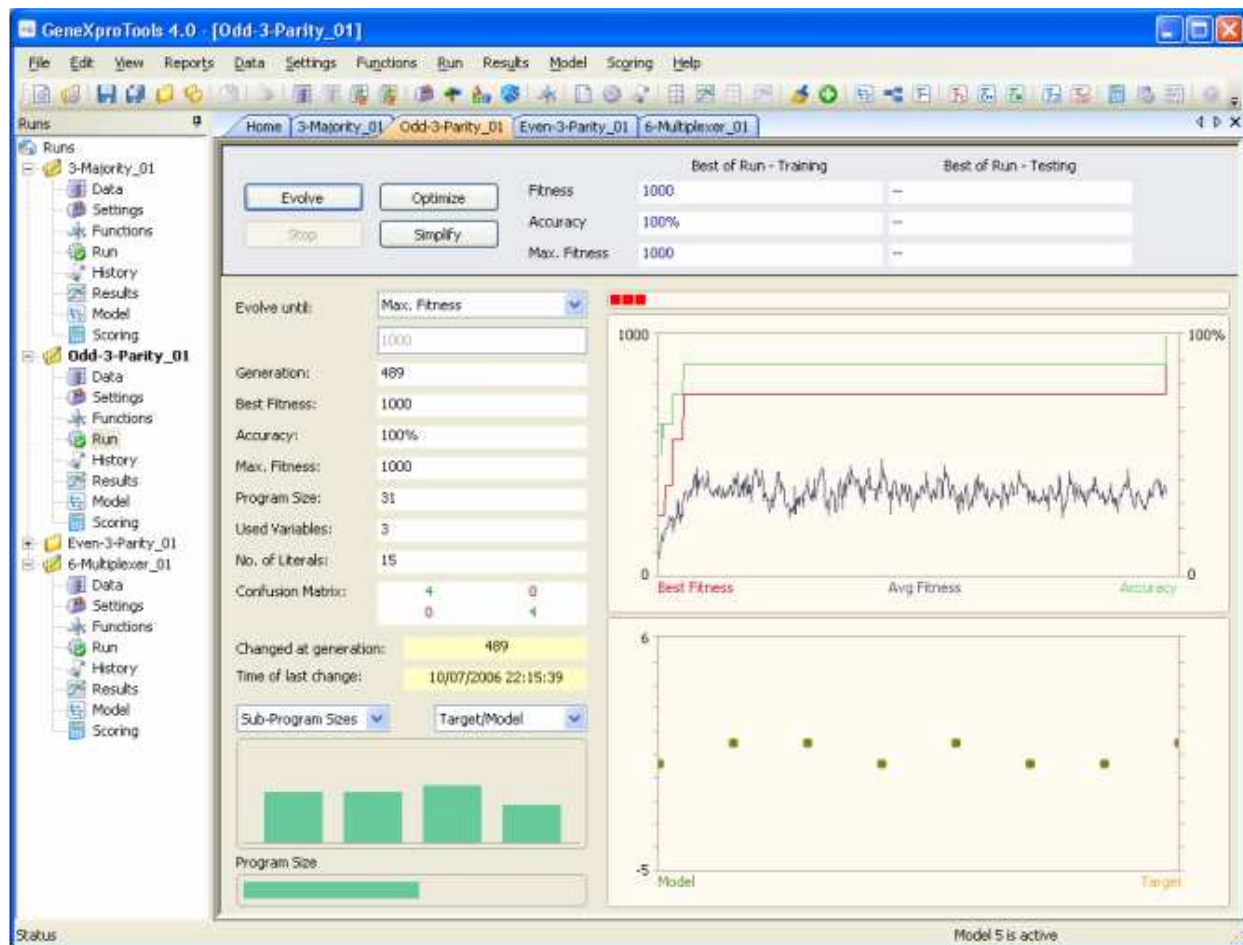
This strategy has enormous advantages as you might choose to stop the run at any time and then take a closer look at the evolved model. For instance, you can analyze its mathematical representation, evaluate its performance in the testing predictions, check a wide set of statistical functions for a quick and rigorous assessment of its accuracy, make a few predictions with it, and so on. Then you might choose to adjust a few parameters, say, choose a different fitness function, expand the function set, add a neutral gene, apply parsimony pressure, and so forth, and then explore this new evolutionary path. You can repeat this process for as long as you want or until you are completely satisfied with the evolved model.

In addition, if you wish for GeneXproTools 4.0 to increase the complexity of your models automatically, you just have to activate the [Complexity Increase Engine](#) and then click the Evolve button and GeneXproTools will evolve better and better models composed of an increasing number of terms until no increase in best fitness takes place for a certain period of time.

Working with Logic Synthesis Tools

This chapter introduces the Logic Synthesis tools of GeneXproTools and teaches you how to use them to design parsimonious logic circuits. In this chapter, you'll learn how to:

- Load the truth tables for the GeneXproTools logic synthesis algorithm.
- Explore the GeneXproTools function kit for logic synthesis problems.
- Explore all the available architectures for building logic circuits.
- Explore the best fitness functions for logic synthesis problems.
- Explore all the learning algorithms available for logic synthesis problems.
- Explore essential evolutionary strategies for an efficient learning.



Working with Logic Synthesis Tools

Loading Truth Tables

Before designing a circuit with GeneXproTools 4.0 you must first load the truth tables for the learning algorithm. The data screening engine of GeneXproTools 4.0 checks the validity of all the truth tables and is operative every time you load data sets either for training or testing. Only the values of False and True (case insensitive) or "0" and "1" are valid as input. Note, however, that GeneXproTools uses 0's and 1's internally and, therefore, False and True are respectively converted into "0" and "1" and shown in this format. Missing or invalid values can be detected by the data screening engine so that you can identify these faulty samples and correct them.

GeneXproTools 4.0 allows you to work either with databases/Excel or text files and, for text files, accepts two different **data matrix formats**.

The first is the standard **Samples x Variables** format where samples are in rows and variables in columns, with the dependent variable or function output occupying the rightmost position. In the small example below with eight samples, OUTPUT is the function output and A, B, and C are the dependent variables:

```
A B C OUTPUT
0 0 0 0
0 0 1 0
0 1 0 0
0 1 1 1
1 0 0 0
1 0 1 1
1 1 0 1
1 1 1 1
```

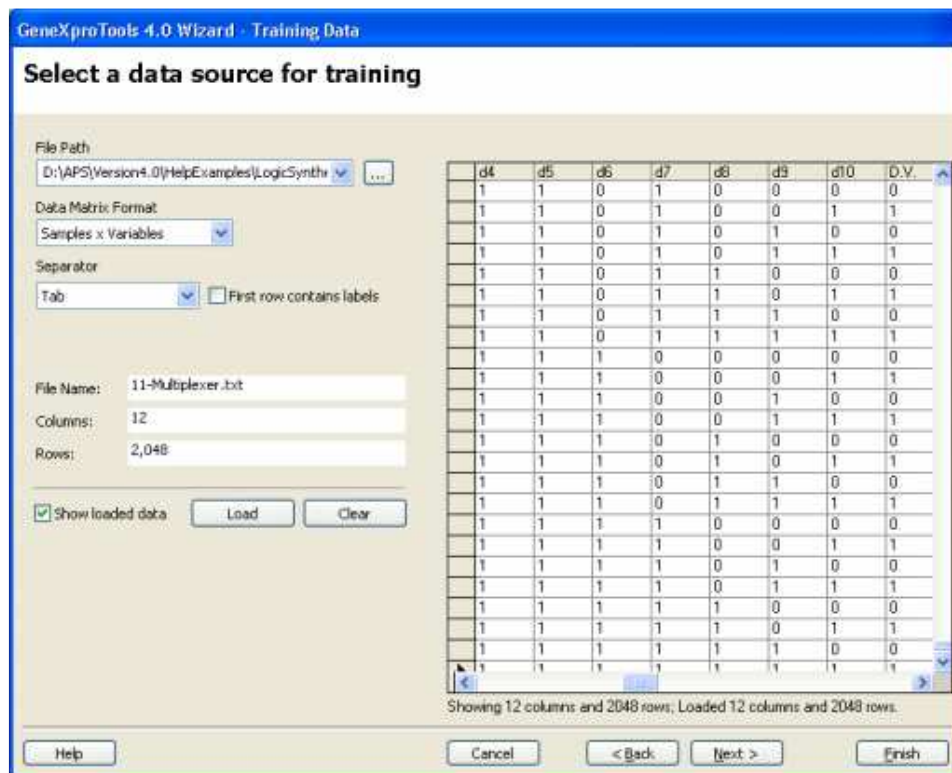
And the second, is the **Gene Expression Matrix** format commonly used in DNA microarrays studies where samples are in columns and variables in rows, with the function output occupying the topmost position. For instance, in Gene Expression Matrix format, the truth table above corresponds to:

```
OUTPUT 0 0 0 1 0 1 1 1
A 0 0 0 0 1 1 1 1
B 0 0 1 1 0 0 1 1
C 0 1 0 1 0 1 0 1
```

which is very handy for datasets with a relatively small number of samples and thousands of variables. Note, however, that for Excel files this format is not supported and if your data is kept in this format in Excel, you must copy it to a text file so that it can be loaded into GeneXproTools.

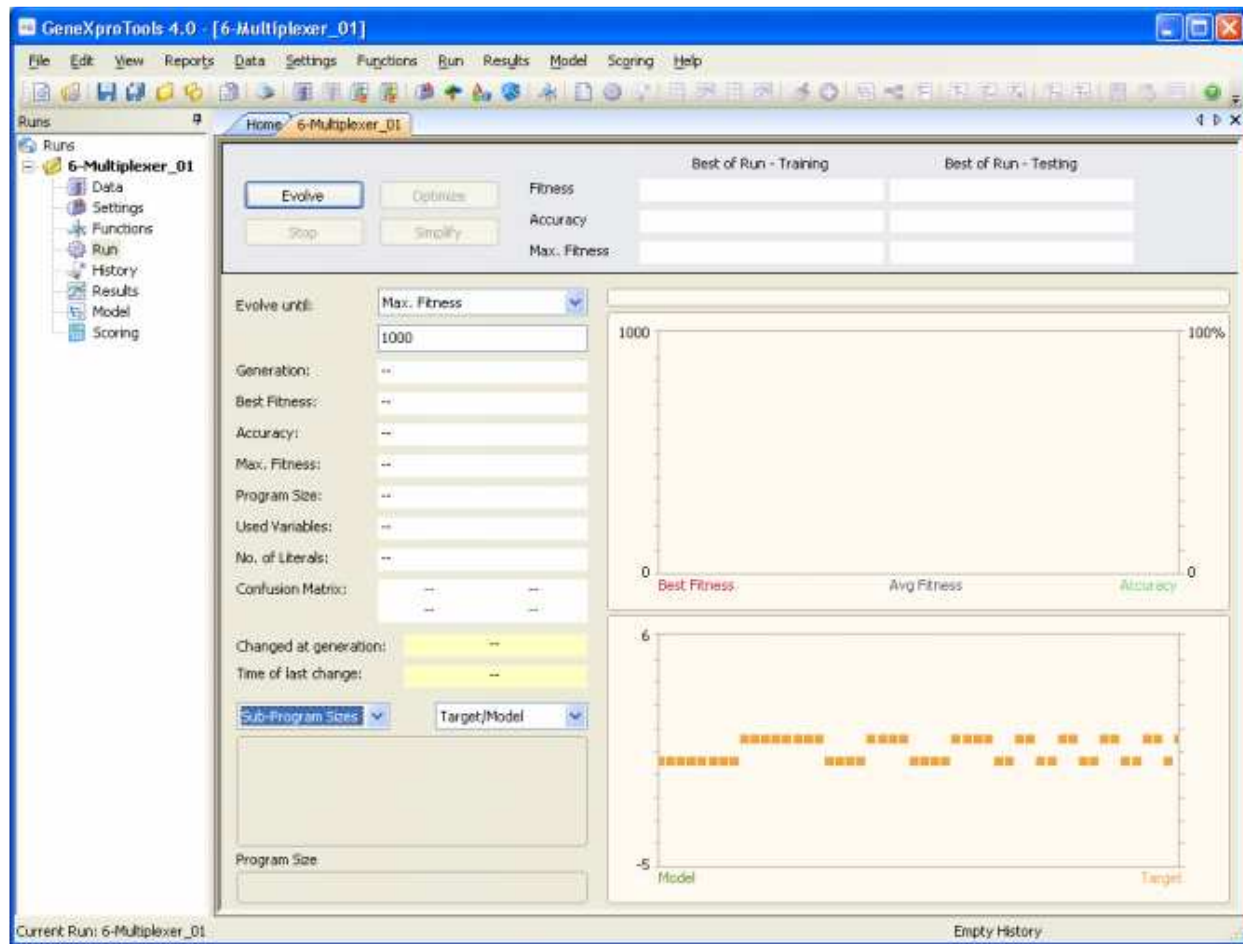
GeneXproTools uses the Samples x Variables format throughout and therefore all formats are automatically converted and shown in this format.

GeneXproTools supports the standard **separators** (space, tab, comma, semicolon, and pipe) and detects them automatically. The use of **labels** to identify your variables is optional and GeneXproTools also detects automatically whether they are present or not. If you use them, however, you will be able to generate more intelligible code where each variable is identified by its name, by checking the Use Labels box in the Model Panel.

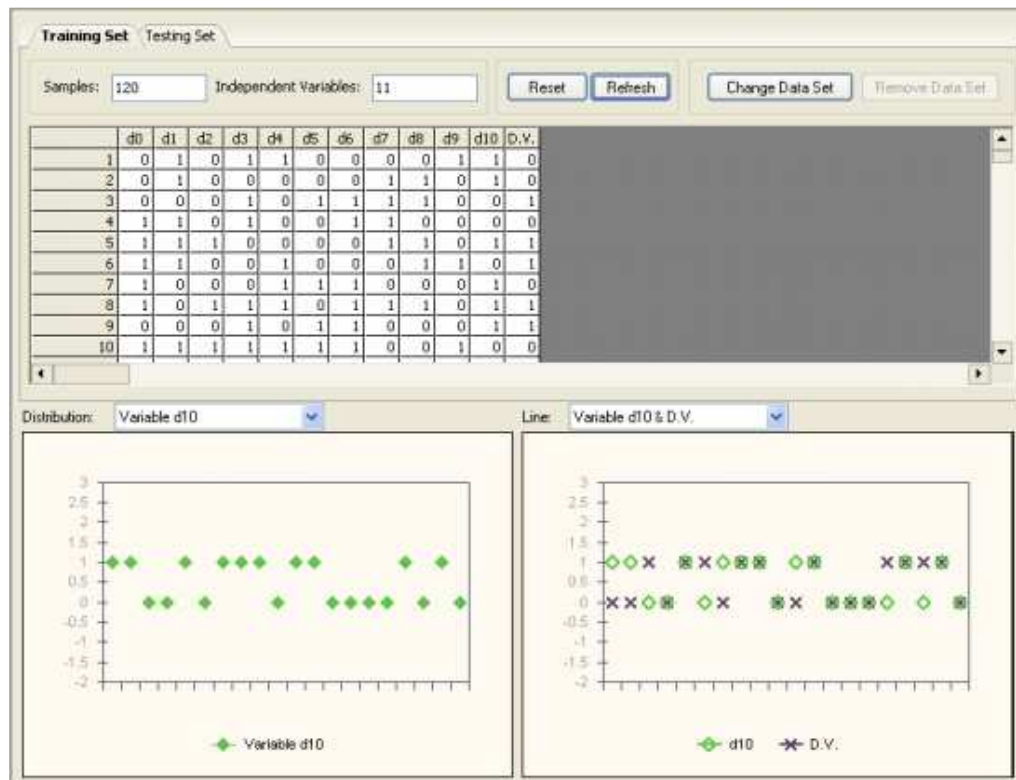


To Load Truth Tables for Modeling

1. Click the File Menu and then choose New.
The New Run Wizard appears. You must give a name to your new run file (the default filename extension of GeneXproTools 4.0 run files is .gep) and then choose Logic Synthesis in the Problem Category box and the kind of source file in the Data Source Type box. GeneXproTools 4.0 allows you to work either with Excel/databases or text files.
2. Then go to the Training Data window by clicking the Next button.
Choose the path for the training set by browsing the Open dialog box and choose the appropriate data matrix format. Irrespective of the data format used, GeneXproTools shows the loaded data in the standard Samples x Variables format, with the function output or dependent variable occupying the rightmost position.
3. Then go to the Testing Data window by clicking the Next button.
The use of a testing set is not common practice in Logic Synthesis as usually the entire truth table is used to design the logic circuit, but repeat the same steps of the previous point if you wish to use a testing set to evaluate the predictive accuracy of your circuit.
4. Click the Finish button to save your new run file.
The Save As dialog box appears and after choosing the directory where you want your new run file to be saved, the GeneXproTools modeling environment appears. Then you just have to click the Evolve button to design a logic circuit as GeneXproTools automatically chooses, from a gallery of templates, default settings that will enable you to design one immediately.



After loading your data into GeneXproTools, in the [Data Panel](#) you can visualize the distribution of values for each variable and also plot each independent variable against the function output or dependent variable.



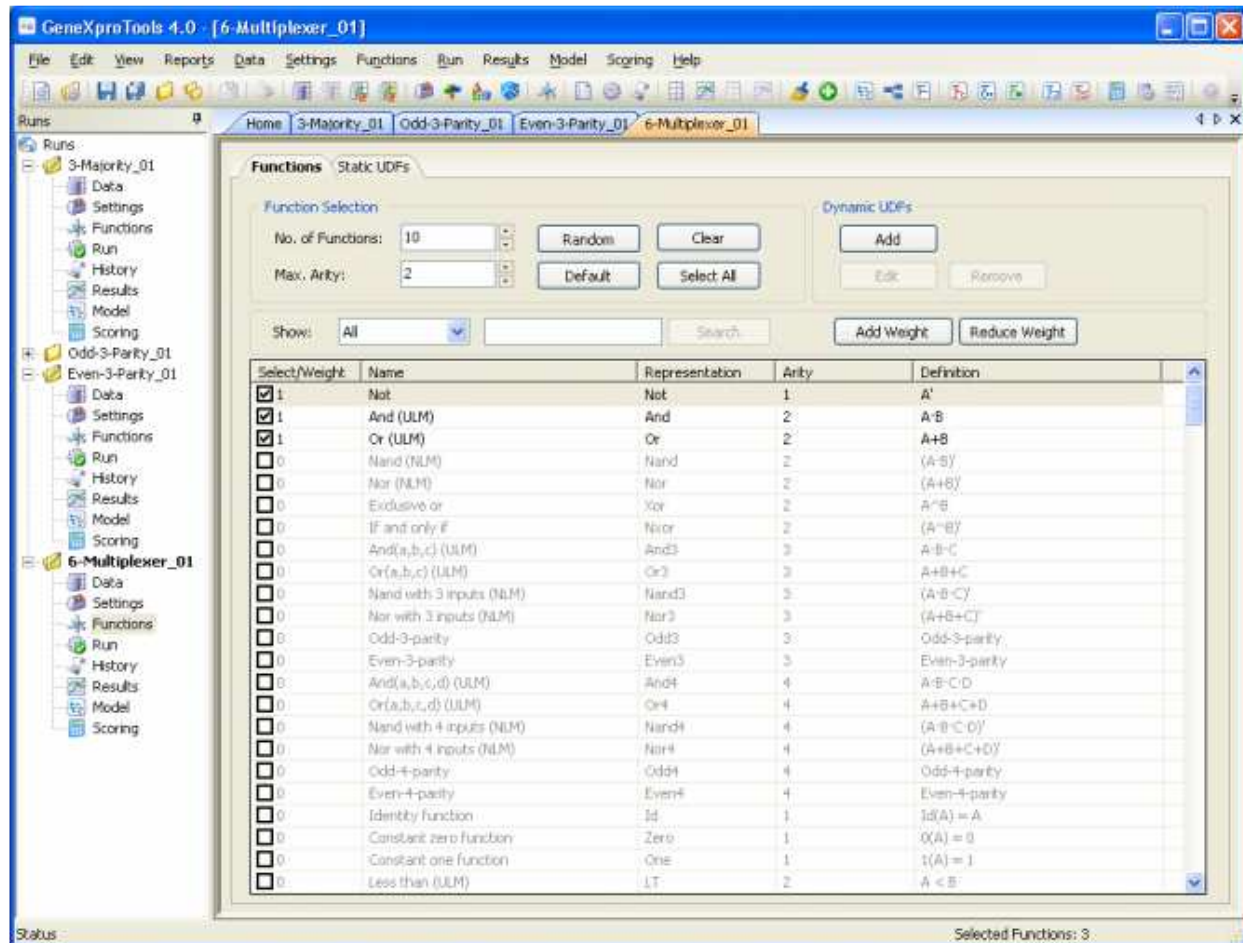
◀ | ▶

Working with Logic Synthesis Tools

Available Functions

In the Functions Panel you can select and create:

- [Logical Functions](#) (including Dynamic UDFs or DDFs).
- [Static UDFs](#).

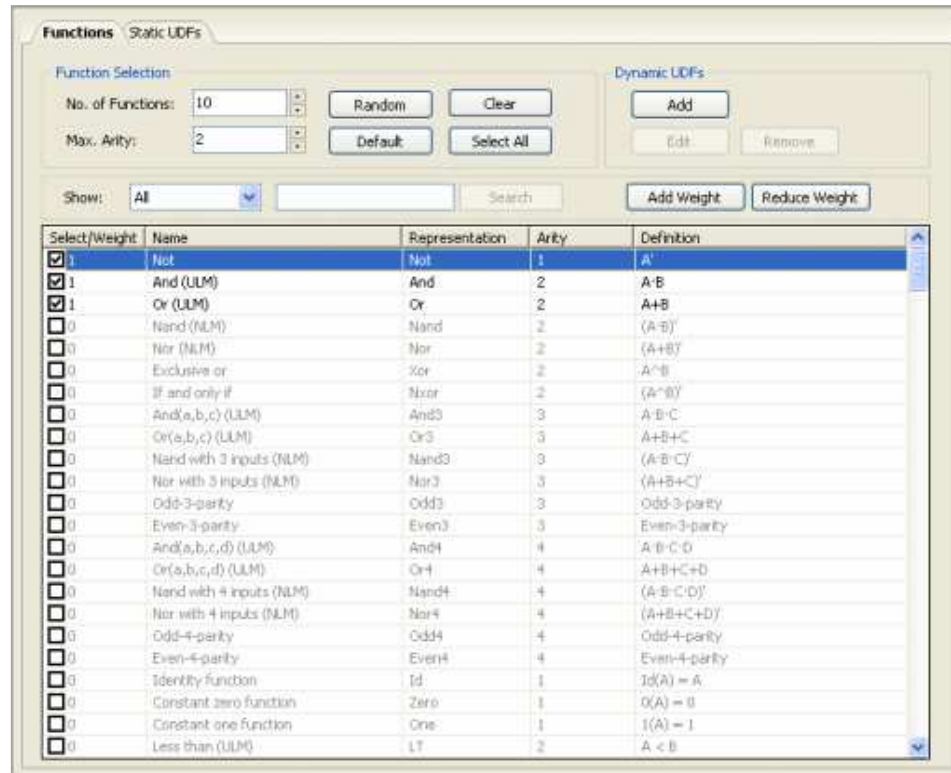


Working with Logic Synthesis Tools

Logical Functions and Dynamic UDFs

GeneXproTools 4.0 offers a total of 258 **built-in logical functions**, including all the Boolean functions with one and two inputs, 92 functions with three inputs, and 146 functions with four inputs, that can be used for designing logic circuits. This wide set of logical functions allows the evolution of sophisticated logic circuits that can be easily simplified and immediately ported to different logical systems.

The Function Selection Tool of GeneXproTools 4.0 helps you selecting different function sets very quickly through the combination of the Show options with the Random/Default/Clear/Select All buttons plus the Increase/Reduce Weight buttons in the [Functions Panel](#). Furthermore, GeneXproTools automatically balances your function set with the number of independent variables in your data, which means that you just have to select the right functions for your problem and then choose their relative proportions by adjusting their weights.



Despite the wide set of GeneXproTools built-in logical functions, some users sometimes want to model with different ones. GeneXproTools 4.0 gives the user the possibility of creating custom tailored functions ([Dynamic UDFs](#) or DDFs) and evolve models with them. A note of caution though: the use of DDFs slows considerably the evolutionary process and therefore should be used with moderation.

Below are listed all the 258 **built-in logical functions** available in GeneXproTools 4.0, including their class (universal logical modules are identified by ULM, whereas universal NAND-like modules are identified by NLM), their representation in [Karva Notation](#) (shown in red) and their logical definition (for more details about their implementations, see the [built-in grammars](#) of GeneXproTools):

Basic logical functions with 1 and 2 inputs:

- Not (**Not**): A'
- And (ULM) (**And**): $A \cdot B$
- Or (ULM) (**Or**): $A + B$
- Nand (NLM) (**Nand**): $(A \cdot B)'$
- Nor (NLM) (**Nor**): $(A + B)'$
- Exclusive or (**Xor**): $A \oplus B$
- If and only if (**Nxor**): $(A \oplus B)'$

Basic logical functions with 3 inputs:

- And(a,b,c) (ULM) (**And3**): $A \cdot B \cdot C$
- Or(a,b,c) (ULM) (**Or3**): $A + B + C$
- Nand with 3 inputs (NLM) (**Nand3**): $(A \cdot B \cdot C)'$
- Nor with 3 inputs (NLM) (**Nor3**): $(A + B + C)'$
- Odd-3-parity (**Odd3**): Odd-3-parity
- Even-3-parity (**Even3**): Even-3-parity

Basic logical functions with 4 inputs:

- And(a,b,c,d) (ULM) (**And4**): $A \cdot B \cdot C \cdot D$
- Or(a,b,c,d) (ULM) (**Or4**): $A + B + C + D$
- Nand with 4 inputs (NLM) (**Nand4**): $(A \cdot B \cdot C \cdot D)'$
- Nor with 4 inputs (NLM) (**Nor4**): $(A + B + C + D)'$
- Odd-4-parity (**Odd4**): Odd-4-parity
- Even-4-parity (**Even4**): Even-4-parity

Additional logical functions with 1 input:

- Identity function (**Id**): $\text{Id}(A) = A$
- Constant zero function (**Zero**): $0(A) = 0$
- Constant one function (**One**): $1(A) = 1$

Additional logical functions with 2 inputs:

- Less than (ULM) (**LT**): $A < B$
- Greater than (ULM) (**GT**): $A > B$
- Less or equal (ULM) (**LOE**): $A \leq B$
- Greater or equal (ULM) (**GOE**): $A \geq B$
- Not A (**NotA**): $\text{NOT}(A, B) = \text{NOT } A$
- Not B (**NotB**): $\text{NOT}(A, B) = \text{NOT } B$
- Input A (**IdA**): $\text{IdA}(A, B) = A$
- Input B (**IdB**): $\text{IdB}(A, B) = B$
- 0(a,b) (**Zero2**): $0(A, B) = 0$
- 1(a,b) (**One2**): $1(A, B) = 1$

Derived logical functions with 3 inputs:

- LT(a,b,c) (ULM) (**LT3**): $A < B < C$
- GT(a,b,c) (ULM) (**GT3**): $A > B > C$
- LOE(a,b,c) (ULM) (**LOE3**): $A \leq B \leq C$
- GOE(a,b,c) (ULM) (**GOE3**): $A \geq B \geq C$

Common logical functions with 3 inputs:

- 3-Multiplexer (ULM) (**Mux**): 3-Multiplexer
- If(a,b,c) (ULM) (**If**): If $A = 1$, then B , else C
- Majority(a,b,c) (ULM) (**Maj**): Majority(A, B, C)
- Minority(a,b,c) (ULM) (**Min**): Minority(A, B, C)
- Exactly two off (ULM) (**2Off**): Exactly two off
- Exactly two on (ULM) (**2On**): Exactly two on

Universal logical modules with 3 inputs (series A):

- ULM with 3 inputs (A1) (**LM3A1**): $AC' + BC$
- ULM with 3 inputs (A2) (**LM3A2**): $AC' + B'C$
- ULM with 3 inputs (A3) (**LM3A3**): $A'C + BC$
- NLM with 3 inputs (A4) (**LM3A4**): $A'C + B'C$

Universal logical modules with 3 inputs (series B):

- ULM with 3 inputs (B1) (**LM3B1**): $(A + C') \cdot (B + C)$
- ULM with 3 inputs (B2) (**LM3B2**): $(A + C') \cdot (B' + C)$
- ULM with 3 inputs (B3) (**LM3B3**): $(A' + C') \cdot (B + C)$
- NLM with 3 inputs (B4) (**LM3B4**): $(A' + C') \cdot (B' + C)$

Universal logical modules with 3 inputs (series C):

- ULM with 3 inputs (C1) (LM3C1): $AB' + BC$
- ULM with 3 inputs (C2) (LM3C2): $AB' + BC'$
- ULM with 3 inputs (C3) (LM3C3): $A'B + BC$
- NLM with 3 inputs (C4) (LM3C4): $A'B + BC'$

Universal logical modules with 3 inputs (series D):

- ULM with 3 inputs (D1) (LM3D1): $(A+B') \cdot (B+C)$
- ULM with 3 inputs (D2) (LM3D2): $(A+B') \cdot (B+C')$
- ULM with 3 inputs (D3) (LM3D3): $(A'+B') \cdot (B+C)$
- NLM with 3 inputs (D4) (LM3D4): $(A'+B') \cdot (B+C')$

Universal logical modules with 3 inputs (series E):

- ULM with 3 inputs (E1) (LM3E1): $A'C + AB'$
- ULM with 3 inputs (E2) (LM3E2): $A'C' + AB$
- NLM with 3 inputs (E3) (LM3E3): $A'C + AB'$

Universal logical modules with 3 inputs (series F):

- ULM with 3 inputs (F1) (LM3F1): $(A'+C) \cdot (A+B')$
- ULM with 3 inputs (F2) (LM3F2): $(A'+C') \cdot (A+B)$
- NLM with 3 inputs (F3) (LM3F3): $(A'+C') \cdot (A+B')$

Universal logical modules with 3 inputs (series G):

- ULM with 3 inputs (G1) (LM3G1): $(A \wedge C') \cdot (B \wedge C)$
- ULM with 3 inputs (G2) (LM3G2): $(A \wedge C') \cdot (B' \wedge C)$
- ULM with 3 inputs (G3) (LM3G3): $(A' \wedge C) \cdot (B \wedge C)$
- ULM with 3 inputs (G4) (LM3G4): $(A' \wedge C) \cdot (B' \wedge C)$

Universal logical modules with 3 inputs (series H):

- ULM with 3 inputs (H1) (LM3H1): $((A \cdot B)' \cdot C)'$
- ULM with 3 inputs (H2) (LM3H2): $(A \cdot (B \cdot C))'$
- ULM with 3 inputs (H3) (LM3H3): $((A+B)' + C)'$
- ULM with 3 inputs (H4) (LM3H4): $(A + (B+C))'$

Comparison IF THEN ELSE functions with 3 inputs (series A):

- Less Than with 3 inputs (A) (NLM) (LT3A): If $A < B$, then $(A \cdot C)$, else $(B \cdot C)'$
- Greater Than with 3 inputs (A) (NLM) (GT3A): If $A > B$, then $(A \cdot C)$, else $(B \cdot C)'$
- Less Or Equal To with 3 inputs (A) (ULM) (LOE3A): If $A \leq B$, then $(A \cdot C)$, else $(B \cdot C)'$
- Greater Or Equal To with 3 inputs (A) (ULM) (GOE3A): If $A \geq B$, then $(A \cdot C)$, else $(B \cdot C)'$
- Equal To with 3 inputs (A) (ULM) (ET3A): If $A = B$, then $(A \cdot C)$, else $(B \cdot C)'$
- Not Equal To with 3 inputs (A) (NLM) (NET3A): If $A \neq B$, then $(A \cdot C)$, else $(B \cdot C)'$

Comparison IF THEN ELSE functions with 3 inputs (series B):

- Less Than with 3 inputs (B) (ULM) (LT3B): If $A < B$, then $(A \cdot C)'$, else $(A \cdot C)$
- Greater Than with 3 inputs (B) (ULM) (GT3B): If $A > B$, then $(A \cdot C)'$, else $(A \cdot C)$
- Less Or Equal To with 3 inputs (B) (NLM) (LOE3B): If $A \leq B$, then $(A \cdot C)'$, else $(A \cdot C)$
- Greater Or Equal To with 3 inputs (B) (NLM) (GOE3B): If $A \geq B$, then $(A \cdot C)'$, else $(A \cdot C)$
- Equal To with 3 inputs (B) (NLM) (ET3B): If $A = B$, then $(A \cdot C)'$, else $(A \cdot C)$
- Not Equal To with 3 inputs (B) (ULM) (NET3B): If $A \neq B$, then $(A \cdot C)'$, else $(A \cdot C)$

Comparison IF THEN ELSE functions with 3 inputs (series C):

- Less Than with 3 inputs (C) (ULM) (**LT3C**): If $A < B$, then $(A+C)'$, else $(B+C)$
- Greater Than with 3 inputs (C) (ULM) (**GT3C**): If $A > B$, then $(A+C)'$, else $(B+C)$
- Less Or Equal To with 3 inputs (C) (NLM) (**LOE3C**): If $A \leq B$, then $(A+C)'$, else $(B+C)$
- Greater Or Equal To with 3 inputs (C) (NLM) (**GOE3C**): If $A \geq B$, then $(A+C)'$, else $(B+C)$
- Equal To with 3 inputs (C) (NLM) (**ET3C**): If $A = B$, then $(A+C)'$, else $(B+C)$
- Not Equal To with 3 inputs (C) (ULM) (**NET3C**): If $A \neq B$, then $(A+C)'$, else $(B+C)$

Additional universal logical modules with 3 inputs:

- Rule 4 (ULM) (**T004**): 00000100
- Rule 8 (ULM) (**T008**): 00001000
- Rule 9 (NLM) (**T009**): 00001001
- Rule 32 (ULM) (**T032**): 00100000
- Rule 33 (NLM) (**T033**): 00100001
- Rule 41 (NLM) (**T041**): 00101001
- Rule 55 (NLM) (**T055**): 00110111
- Rule 57 (NLM) (**T057**): 00111001
- Rule 64 (ULM) (**T064**): 01000000
- Rule 65 (NLM) (**T065**): 01000001
- Rule 69 (NLM) (**T069**): 01000101
- Rule 73 (NLM) (**T073**): 01001001
- Rule 81 (NLM) (**T081**): 01010001
- Rule 89 (NLM) (**T089**): 01011001
- Rule 93 (NLM) (**T093**): 01011101
- Rule 96 (ULM) (**T096**): 01100000
- Rule 101 (NLM) (**T101**): 01100101
- Rule 109 (NLM) (**T109**): 01101101
- Rule 111 (NLM) (**T111**): 01101111
- Rule 121 (NLM) (**T121**): 01111001
- Rule 123 (NLM) (**T123**): 01111011
- Rule 125 (NLM) (**T125**): 01111101
- Rule 154 (ULM) (**T154**): 10011010
- Rule 223 (ULM) (**T223**): 11011111
- Rule 239 (ULM) (**T239**): 11101111
- Rule 249 (ULM) (**T249**): 11111001
- Rule 251 (ULM) (**T251**): 11111011
- Rule 253 (ULM) (**T253**): 11111101

Derived logical functions with 4 inputs:

- LT(a,b,c,d) (ULM) (**LT4**): $A < B < C < D$
- GT(a,b,c,d) (ULM) (**GT4**): $A > B > C > D$
- LOE(a,b,c,d) (ULM) (**LOE4**): $A \leq B \leq C \leq D$
- GOE(a,b,c,d) (ULM) (**GOE4**): $A \geq B \geq C \geq D$

Common logical functions with 4 inputs:

- Tie (ULM) (**Tie**): Tie
- Not tie (ULM) (**Ntie**): Not tie
- Exactly three off (ULM) (**3Off**): Exactly three off
- Exactly three on (ULM) (**3On**): Exactly three on

Universal logical modules with 4 inputs (series A):

- ULM with 4 inputs (A1) (**LM4A1**): $AD' + BD + CD$

- ULM with 4 inputs (A2) (LM4A2): $AD' + B'D + CD$
- ULM with 4 inputs (A3) (LM4A3): $AD' + BD + C'D$
- ULM with 4 inputs (A4) (LM4A4): $AD' + B'D + C'D$
- ULM with 4 inputs (A5) (LM4A5): $A'D' + BD + CD$
- ULM with 4 inputs (A6) (LM4A6): $A'D' + B'D + CD$
- ULM with 4 inputs (A7) (LM4A7): $A'D' + BD + C'D$
- NLM with 4 inputs (A8) (LM4A8): $A'D' + B'D + C'D$

Universal logical modules with 4 inputs (series B):

- ULM with 4 inputs (B1) (LM4B1): $(A+D') \cdot (B+D) \cdot (C+D)$
- ULM with 4 inputs (B2) (LM4B2): $(A+D') \cdot (B'+D) \cdot (C+D)$
- ULM with 4 inputs (B3) (LM4B3): $(A+D') \cdot (B+D) \cdot (C'+D)$
- ULM with 4 inputs (B4) (LM4B4): $(A+D') \cdot (B'+D) \cdot (C'+D)$
- ULM with 4 inputs (B5) (LM4B5): $(A'+D') \cdot (B+D) \cdot (C+D)$
- ULM with 4 inputs (B6) (LM4B6): $(A'+D') \cdot (B'+D) \cdot (C+D)$
- ULM with 4 inputs (B7) (LM4B7): $(A'+D') \cdot (B+D) \cdot (C'+D)$
- NLM with 4 inputs (B8) (LM4B8): $(A'+D') \cdot (B'+D) \cdot (C'+D)$

Universal logical modules with 4 inputs (series C):

- ULM with 4 inputs (C1) (LM4C1): $AB' + BC + BD$
- ULM with 4 inputs (C2) (LM4C2): $AB' + B'C + BD$
- ULM with 4 inputs (C3) (LM4C3): $AB' + BC + B'D$
- ULM with 4 inputs (C4) (LM4C4): $AB' + B'C + B'D$
- ULM with 4 inputs (C5) (LM4C5): $A'B' + BC + BD$
- ULM with 4 inputs (C6) (LM4C6): $A'B' + B'C + BD$
- ULM with 4 inputs (C7) (LM4C7): $A'B' + BC + B'D$
- NLM with 4 inputs (C8) (LM4C8): $A'B' + B'C + B'D$

Universal logical modules with 4 inputs (series D):

- ULM with 4 inputs (D1) (LM4D1): $(A+B') \cdot (B+C) \cdot (B+D)$
- ULM with 4 inputs (D2) (LM4D2): $(A+B') \cdot (B'+C) \cdot (B+D)$
- ULM with 4 inputs (D3) (LM4D3): $(A+B') \cdot (B+C) \cdot (B'+D)$
- ULM with 4 inputs (D4) (LM4D4): $(A+B') \cdot (B'+C) \cdot (B'+D)$
- ULM with 4 inputs (D5) (LM4D5): $(A'+B') \cdot (B+C) \cdot (B+D)$
- ULM with 4 inputs (D6) (LM4D6): $(A'+B') \cdot (B'+C) \cdot (B+D)$
- ULM with 4 inputs (D7) (LM4D7): $(A'+B') \cdot (B+C) \cdot (B'+D)$
- NLM with 4 inputs (D8) (LM4D8): $(A'+B') \cdot (B'+C) \cdot (B'+D)$

Universal logical modules with 4 inputs (series E):

- ULM with 4 inputs (E1) (LM4E1): $AC' + BC + CD$
- ULM with 4 inputs (E2) (LM4E2): $AC' + B'C + CD$
- ULM with 4 inputs (E3) (LM4E3): $AC' + BC + C'D$
- ULM with 4 inputs (E4) (LM4E4): $AC' + B'C + C'D$
- ULM with 4 inputs (E5) (LM4E5): $A'C' + BC + CD$
- ULM with 4 inputs (E6) (LM4E6): $A'C' + B'C + CD$
- ULM with 4 inputs (E7) (LM4E7): $A'C' + BC + C'D$
- NLM with 4 inputs (E8) (LM4E8): $A'C' + B'C + C'D$

Universal logical modules with 4 inputs (series F):

- ULM with 4 inputs (F1) (LM4F1): $(A+C') \cdot (B+C) \cdot (C+D)$
- ULM with 4 inputs (F2) (LM4F2): $(A+C') \cdot (B'+C) \cdot (C+D)$
- ULM with 4 inputs (F3) (LM4F3): $(A+C') \cdot (B+C) \cdot (C'+D)$

- ULM with 4 inputs (F4) (LM4F4): $(A+C') \cdot (B'+C) \cdot (C'+D)$
- ULM with 4 inputs (F5) (LM4F5): $(A'+C') \cdot (B+C) \cdot (C+D)$
- ULM with 4 inputs (F6) (LM4F6): $(A'+C') \cdot (B'+C) \cdot (C+D)$
- ULM with 4 inputs (F7) (LM4F7): $(A'+C') \cdot (B+C) \cdot (C'+D)$
- NLM with 4 inputs (F8) (LM4F8): $(A'+C') \cdot (B'+C) \cdot (C'+D)$

Universal logical modules with 4 inputs (series G):

- ULM with 4 inputs (G1) (LM4G1): $A'D+AB+AC$
- ULM with 4 inputs (G2) (LM4G2): $A'D+AB'+AC$
- ULM with 4 inputs (G3) (LM4G3): $A'D+AB+AC'$
- ULM with 4 inputs (G4) (LM4G4): $A'D+AB'+AC'$
- ULM with 4 inputs (G5) (LM4G5): $A'D'+AB+AC$
- ULM with 4 inputs (G6) (LM4G6): $A'D'+AB'+AC$
- ULM with 4 inputs (G7) (LM4G7): $A'D'+AB+AC'$
- NLM with 4 inputs (G8) (LM4G8): $A'D'+AB'+AC'$

Universal logical modules with 4 inputs (series H):

- ULM with 4 inputs (H1) (LM4H1): $(A'+D) \cdot (A+B) \cdot (A+C)$
- ULM with 4 inputs (H2) (LM4H2): $(A'+D) \cdot (A+B') \cdot (A+C)$
- ULM with 4 inputs (H3) (LM4H3): $(A'+D) \cdot (A+B) \cdot (A+C')$
- ULM with 4 inputs (H4) (LM4H4): $(A'+D) \cdot (A+B') \cdot (A+C')$
- ULM with 4 inputs (H5) (LM4H5): $(A'+D') \cdot (A+B) \cdot (A+C)$
- ULM with 4 inputs (H6) (LM4H6): $(A'+D') \cdot (A+B') \cdot (A+C)$
- ULM with 4 inputs (H7) (LM4H7): $(A'+D') \cdot (A+B) \cdot (A+C')$
- NLM with 4 inputs (H8) (LM4H8): $(A'+D') \cdot (A+B') \cdot (A+C')$

Universal logical modules with 4 inputs (series I):

- NLM with 4 inputs (I1) (LM4I1): $((A \cdot B)' \cdot C)' \cdot D'$
- NLM with 4 inputs (I2) (LM4I2): $(A \cdot (B \cdot (C \cdot D)))'$
- NLM with 4 inputs (I3) (LM4I3): $((A \cdot (B \cdot C))' \cdot D)'$
- NLM with 4 inputs (I4) (LM4I4): $(A \cdot ((B \cdot C)' \cdot D))'$
- NLM with 4 inputs (I5) (LM4I5): $((A+B)' + C)' + D'$
- NLM with 4 inputs (I6) (LM4I6): $(A + (B + (C + D)))'$
- NLM with 4 inputs (I7) (LM4I7): $((A + (B + C))' + D)'$
- NLM with 4 inputs (I8) (LM4I8): $(A + ((B + C)' + D))'$

Comparison IF THEN ELSE functions with 4 inputs (series A):

- Less Than with 4 inputs (A) (ULM) (LT4A): If $A < B$, then C , else D
- Greater Than with 4 inputs (A) (ULM) (GT4A): If $A > B$, then C , else D
- Less Or Equal To with 4 inputs (A) (ULM) (LOE4A): If $A \leq B$, then C , else D
- Greater Or Equal To with 4 inputs (A) (ULM) (GOE4A): If $A \geq B$, then C , else D
- Equal To with 4 inputs (A) (ULM) (ET4A): If $A = B$, then C , else D
- Not Equal To with 4 inputs (A) (ULM) (NET4A): If $A \neq B$, then C , else D

Comparison IF THEN ELSE functions with 4 inputs (series B):

- Less Than with 4 inputs (B) (NLM) (LT4B): If $A < B$, then $(C \cdot D)$, else D'
- Greater Than with 4 inputs (B) (NLM) (GT4B): If $A > B$, then $(C \cdot D)$, else D'
- Less Or Equal To with 4 inputs (B) (ULM) (LOE4B): If $A \leq B$, then $(C \cdot D)$, else D'
- Greater Or Equal To with 4 inputs (B) (ULM) (GOE4B): If $A \geq B$, then $(C \cdot D)$, else D'
- Equal To with 4 inputs (B) (ULM) (ET4B): If $A = B$, then $(C \cdot D)$, else D'
- Not Equal To with 4 inputs (B) (NLM) (NET4B): If $A \neq B$, then $(C \cdot D)$, else D'

Comparison IF THEN ELSE functions with 4 inputs (series C):

- Less Than with 4 inputs (C) (NLM) (**LT4C**): If $A < B$, then $(C+D)$, else D'
- Greater Than with 4 inputs (C) (NLM) (**GT4C**): If $A > B$, then $(C+D)$, else D'
- Less Or Equal To with 4 inputs (C) (ULM) (**LOE4C**): If $A \leq B$, then $(C+D)$, else D'
- Greater Or Equal To with 4 inputs (C) (ULM) (**GOE4C**): If $A \geq B$, then $(C+D)$, else D'
- Equal To with 4 inputs (C) (ULM) (**ET4C**): If $A = B$, then $(C+D)$, else D'
- Not Equal To with 4 inputs (C) (NLM) (**NET4C**): If $A \neq B$, then $(C+D)$, else D'

Comparison IF THEN ELSE functions with 4 inputs (series D):

- Less Than with 4 inputs (D) (NLM) (**LT4D**): If $A < B$, then $(A \cdot D)$, else $(C \cdot D)'$
- Greater Than with 4 inputs (D) (NLM) (**GT4D**): If $A > B$, then $(A \cdot D)$, else $(C \cdot D)'$
- Less Or Equal To with 4 inputs (D) (ULM) (**LOE4D**): If $A \leq B$, then $(A \cdot D)$, else $(C \cdot D)'$
- Greater Or Equal To with 4 inputs (D) (ULM) (**GOE4D**): If $A \geq B$, then $(A \cdot D)$, else $(C \cdot D)'$
- Equal To with 4 inputs (D) (ULM) (**ET4D**): If $A = B$, then $(A \cdot D)$, else $(C \cdot D)'$
- Not Equal To with 4 inputs (D) (NLM) (**NET4D**): If $A \neq B$, then $(A \cdot D)$, else $(C \cdot D)'$

Comparison IF THEN ELSE functions with 4 inputs (series E):

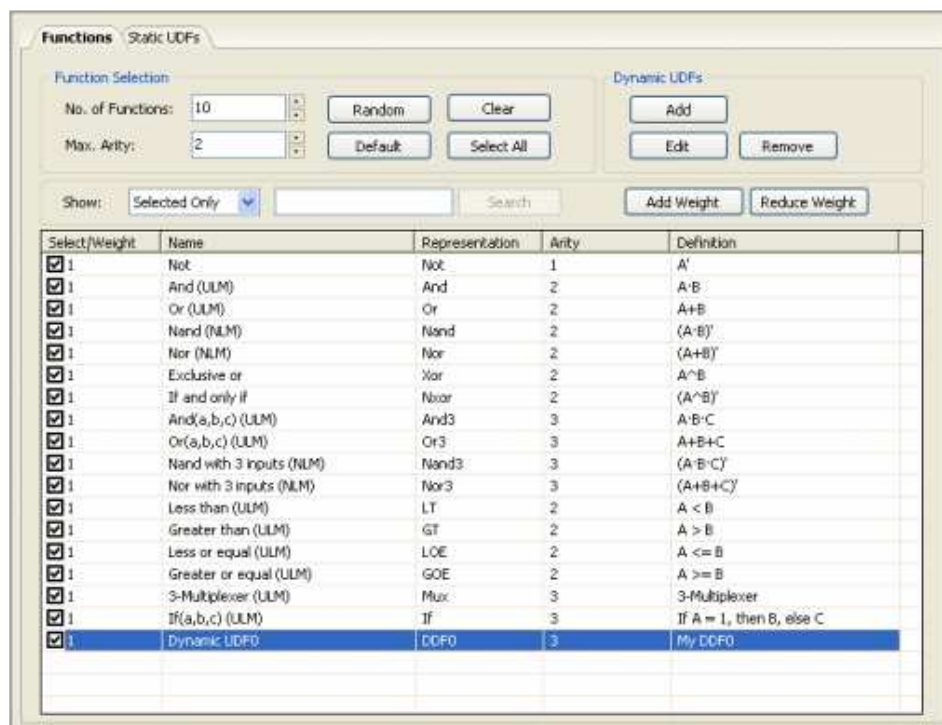
- Less Than with 4 inputs (E) (ULM) (**LT4E**): If $A < B$, then $(A \cdot D)'$, else $(A \cdot C)$
- Greater Than with 4 inputs (E) (ULM) (**GT4E**): If $A > B$, then $(A \cdot D)'$, else $(A \cdot C)$
- Less Or Equal To with 4 inputs (E) (NLM) (**LOE4E**): If $A \leq B$, then $(A \cdot D)'$, else $(A \cdot C)$
- Greater Or Equal To with 4 inputs (E) (NLM) (**GOE4E**): If $A \geq B$, then $(A \cdot D)'$, else $(A \cdot C)$
- Equal To with 4 inputs (E) (NLM) (**ET4E**): If $A = B$, then $(A \cdot D)'$, else $(A \cdot C)$
- Not Equal To with 4 inputs (E) (ULM) (**NET4E**): If $A \neq B$, then $(A \cdot D)'$, else $(A \cdot C)$

Additional universal logical modules with 4 inputs:

- Rule 00002 (ULM) (**Q0002**): 0000000000000010
- Rule 00028 (ULM) (**Q001C**): 0000000000011100
- Rule 00072 (ULM) (**Q0048**): 0000000001001000
- Rule 02048 (ULM) (**Q0800**): 0000100000000000
- Rule 13176 (ULM) (**Q3378**): 0011001101111000
- Rule 13429 (NLM) (**Q3475**): 0011010001110101
- Rule 15536 (ULM) (**Q3CB0**): 0011110010110000
- Rule 15855 (NLM) (**Q3DEF**): 0011110111101111
- Rule 15871 (NLM) (**Q3DFF**): 0011110111111111
- Rule 16896 (ULM) (**Q4200**): 0100001000000000
- Rule 19473 (NLM) (**Q4C11**): 0100110000010001
- Rule 20736 (ULM) (**Q5100**): 0101000100000000
- Rule 24303 (NLM) (**Q5EEF**): 0101111011101111
- Rule 24319 (NLM) (**Q5EFF**): 0101111011111111
- Rule 27245 (NLM) (**Q6A6D**): 0110101001101101
- Rule 28533 (NLM) (**Q6F75**): 011011101110101
- Rule 29892 (ULM) (**Q74C4**): 0111010011000100
- Rule 32163 (NLM) (**Q7DA3**): 0111110110100011
- Rule 33540 (ULM) (**Q8304**): 1000001100000100
- Rule 33840 (ULM) (**Q8430**): 1000010000110000
- Rule 34115 (ULM) (**Q8543**): 1000010101000011
- Rule 40320 (ULM) (**Q9D80**): 1001110110000000
- Rule 41106 (ULM) (**QA092**): 1010000010010010
- Rule 45930 (ULM) (**QB36A**): 1011001101101010
- Rule 52175 (ULM) (**QCBCF**): 1100101111001111
- Rule 61105 (ULM) (**QEEB1**): 1110111010110001
- Rule 61439 (ULM) (**QEFFF**): 1110111111111111
- Rule 65403 (ULM) (**QFF7B**): 1111111101111011

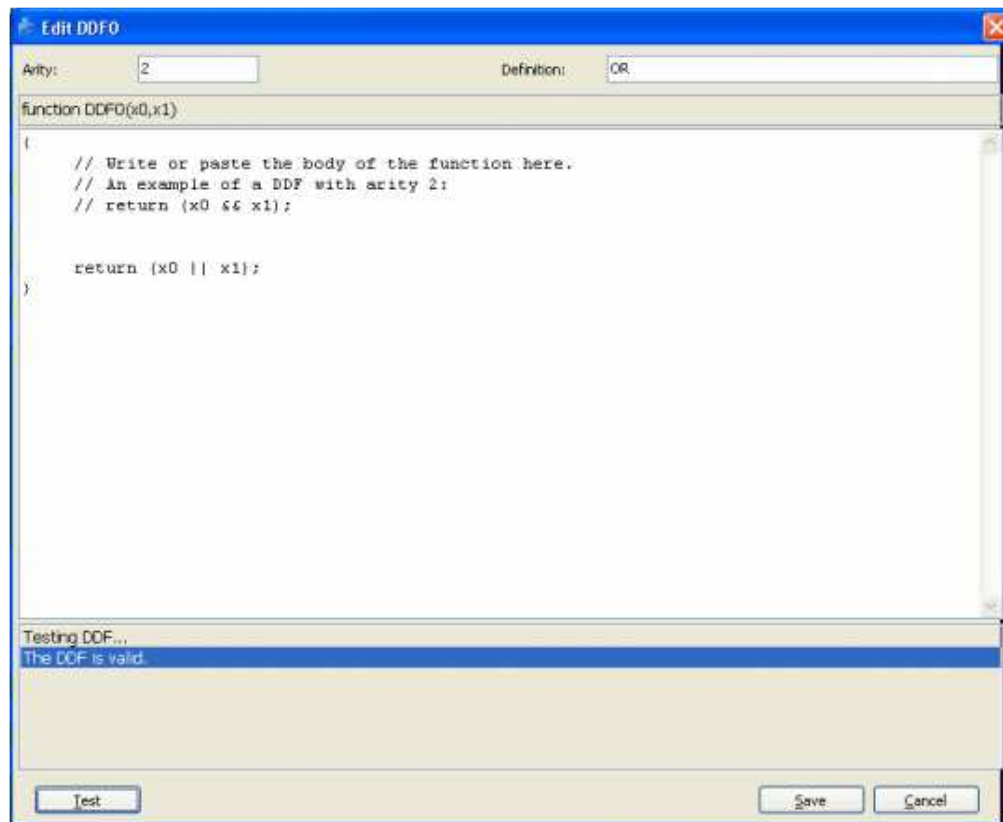
- Rule 65526 (ULM) (QFFF6): 111111111110110
- Rule 65531 (ULM) (QFFFB): 111111111111011

By selecting the Functions Tab in the [Functions Panel](#), you have full access to the 258 built-in logical functions of GeneXproTools 4.0. Here is also the place where you can add [Dynamic UDFs](#) to your modeling kit.



To select a function, just check the box on the left. By default, the weight of each function is 1, but you can increase the probability of a function being included in your logic circuits by increasing its weight in the Select/Weight column. GeneXproTools automatically balances your function set with the number of independent variables in your data, so now you just have to select the right functions for your problem and then choose their relative proportions by choosing their weights.

To add a **DDF** to your modeling kit, just click the Add button on the Dynamic UDFs frame and the [DDF Editor](#) appears.



By choosing the arity (minimum is 1 and maximum is 4) in the Arity box, the function header appears below. Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be conveniently tested for compiling errors by pressing the Test button.

In the Definition box, you can write a brief description of the function for your future reference. The text you write there will appear in the Definition column.

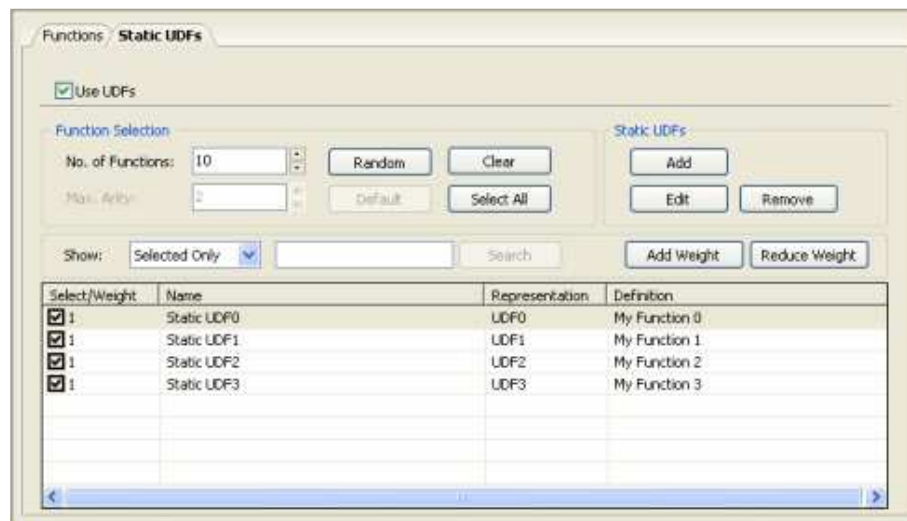
Dynamic UDFs are extremely powerful and interesting tools as they are treated exactly like the built-in functions of GeneXproTools and therefore can be used to model all kinds of relationships between variables or complex expressions. For instance, you can design a DDF so that it will model four expressions OR-ed together, that is, $DDF = (expression\ 1) \text{ OR } (expression\ 2) \text{ OR } (expression\ 3) \text{ OR } (expression\ 4)$, where the value of each expression will depend on the context of the DDF in the expression tree. A note of caution, though, although extremely interesting, DDFs decrease considerably the speed of the algorithm and therefore we advise you to choose your functions from the wide set of GeneXproTools 4.0 built-in logical functions.



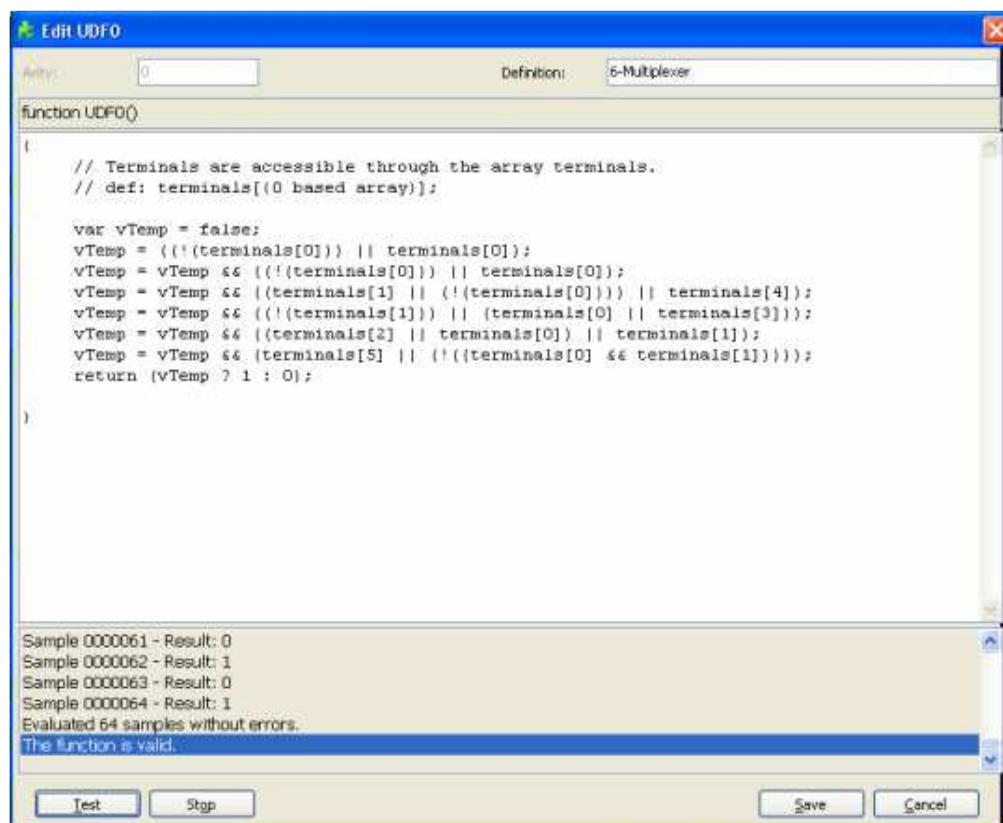
Working with Logic Synthesis Tools

Static UDFs

In the [Functions Panel](#), by selecting the [Static UDFs Tab](#) and checking the Use UDFs box, the Static UDFs frame lights up and you can now add your own user defined functions.

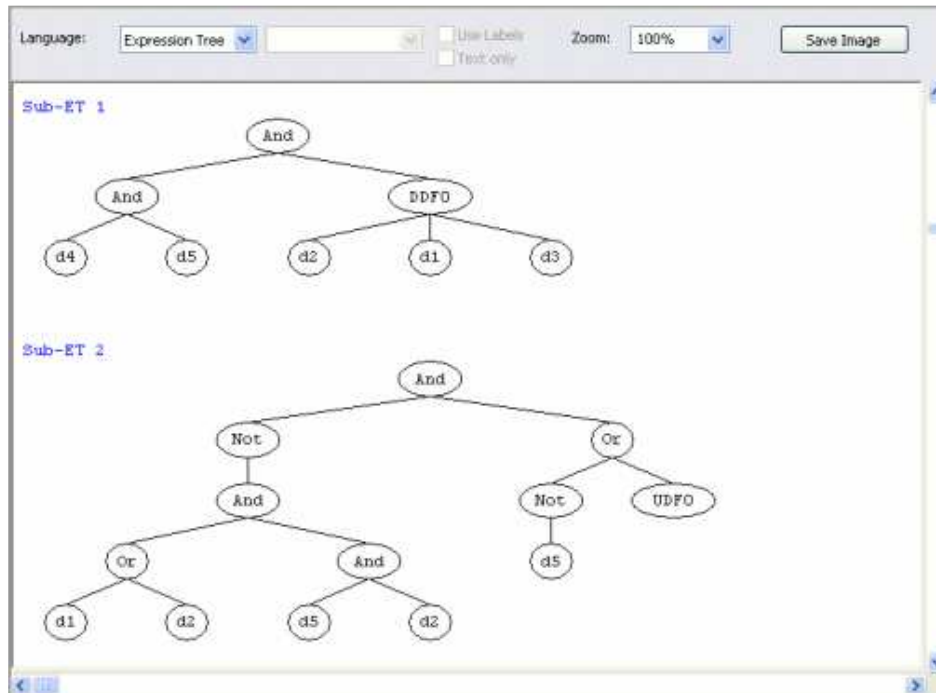


To add a **Static UDF** (UDF for short) to your modeling kit, just click the Add button on the Static UDFs frame and the **UDF Editor** appears with the function header already in place.



Then you just have to write the body of the function in the code editor. The code must be in JavaScript and can be rapidly tested for compiling errors by pressing the Test button.

You will notice that the Arity box is disabled and set to zero as all UDFs will occupy leaf nodes in the trees, that is, they behave exactly as terminals.



In the Definition box, you can write a brief description of the function for your future reference. The text you write there will appear in the Definition column of the Static UDFs Tab.

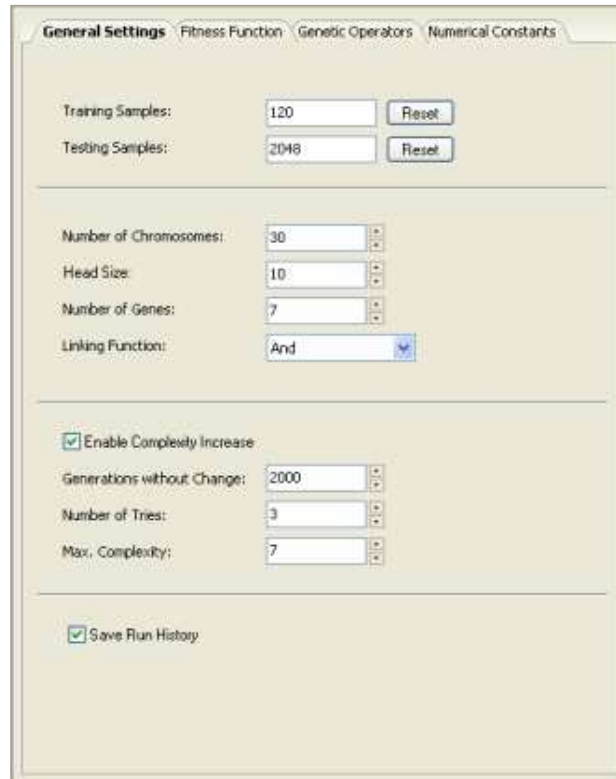
The use of UDFs is a very powerful tool as they allow you to explore inside knowledge about your problem and design useful building blocks that are essential to design more complex logic circuits. For instance, suppose you wanted to design a logic circuit for the 11-multiplexer using the 6-multiplexer as a basic building block. So you design a UDF for the 6-multiplexer and then use this new UDF as an extra building block in your modeling kit. The advantage of well designed UDFs is that they make the discovery of more complex circuits composed of several simpler units much easier.



Working with Logic Synthesis Tools

Available Chromosomal Architectures

The [chromosome architecture](#) of your logic circuits include the **head size**, the **number of genes** and the **linking function**. You choose these parameters in the Settings Panel -> General Settings.



The Head Size determines the complexity of each term in your logic circuit. In the [heads of genes](#), the GeneXproTools learning algorithm tries out different arrangements of functions and terminals (variables and constants) in order to model your data. The plasticity of this architecture allows the discovery of a virtually infinite number of logic circuits of different sizes and shapes which are afterwards tested and selected during the learning process. The heads of genes are shown in blue in the compact, linear representation of your models in the Model Panel.



```

Or .d5 .Or .Not .Not .Or .And .Not .d1 .d0 .d1 .d2 .d0 .d5 .d5 .d1 .d1
And
Or .Or .Or .d0 .d2 .d0 .Or .And .d1 .d2 .d3 .d2 .d2 .d5 .d0 .d2
And
Or .Not .Not .Not .And .Or .d1 .And .d3 .d5 .d1 .d0 .d3 .d5 .d5 .d0 .d5
And
Or .d0 .Or .d3 .Not .d1 .d1 .d5 .d0 .d0 .d2 .d1 .d3 .d2 .d5 .d3 .d4
And
Or .Not .Or .d0 .Or .d4 .d4 .d1 .d4 .d1 .d4 .d4 .d3 .d2 .d4 .d4 .d5
  
```

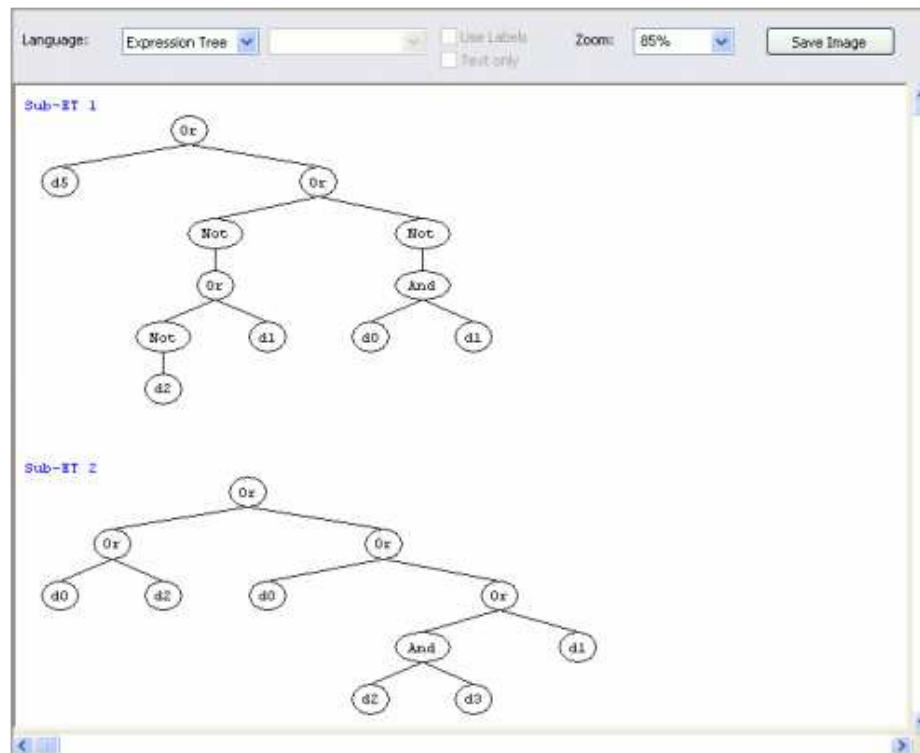
More specifically, the **head size** h of each gene determines the maximum width w and maximum depth d of the [sub-expression trees](#) encoded in the gene, which are given by the formulas:

$$w = (n - 1) * h + 1$$

$$d = ((h + 1) / m) * ((m + 1) / 2)$$

where m is minimum arity and n is maximum arity.

Thus, the GeneXproTools [learning algorithm](#) selects its logic circuits between these extreme cases, fine-tuning the ideal size and shape during the evolutionary process without human intervention.

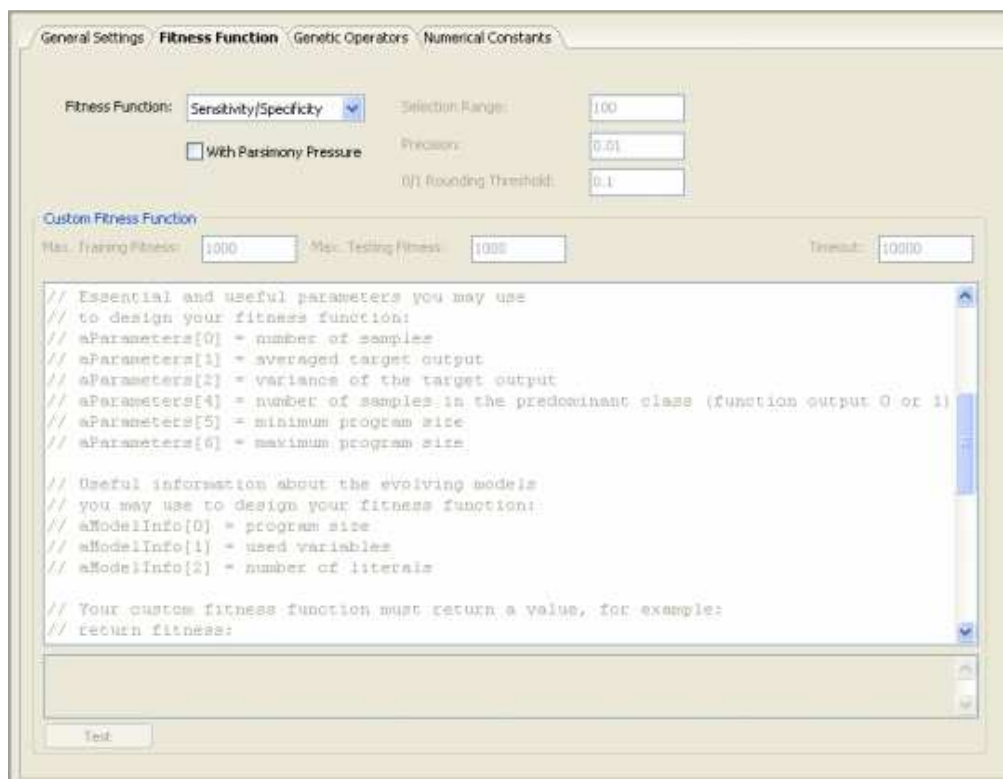


The [number of genes](#) per chromosome is also an important parameter. It will determine the number of (complex) terms in your logic circuits as each gene codes for a different [parse tree](#) (sub-expression tree or sub-ET). Theoretically, one could just use a huge single gene in order to evolve very complex circuits. But the partition of the chromosome into simpler, more manageable units gives an edge to the learning process and more efficient and elegant circuits can be discovered using [multigenic chromosomes](#).

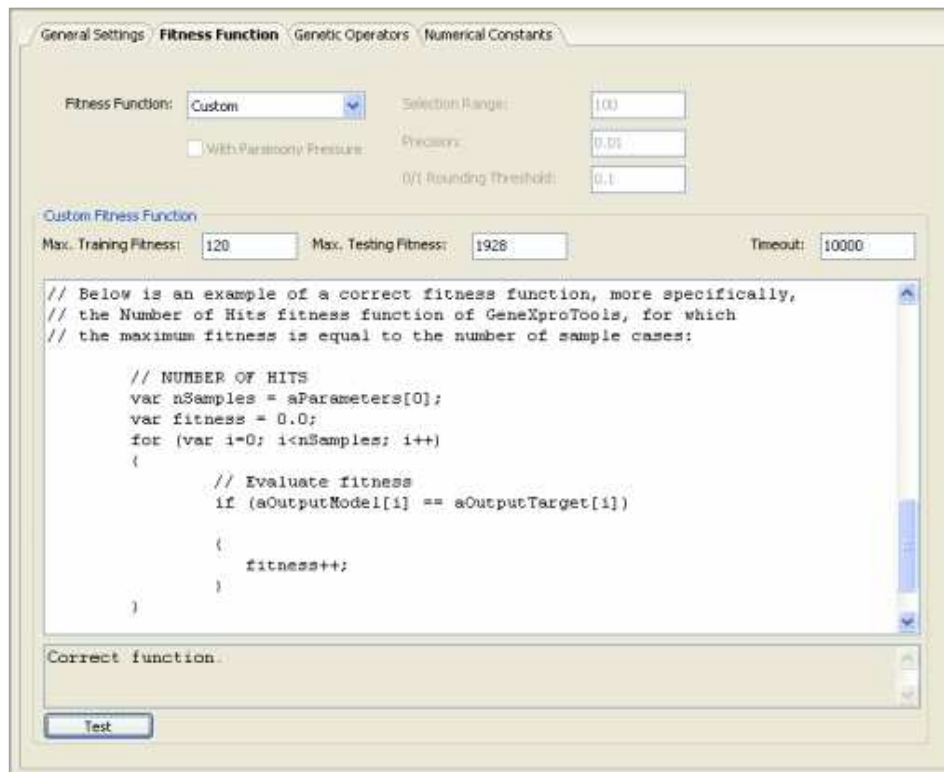
Whenever the number of genes is greater than one, you must also choose a suitable [linking function](#) for linking the logical expressions encoded in each gene. GeneXproTools 4.0 allows you to choose **And**, **Or**, **Nand**, **Nor**, **Xor**, **Nxor**, **Less Than**, **Greater Than**, **Less Or Equal**, or **Greater Or Equal** to link the sub-ETs. As expected, Or (and obviously And) works very well for virtually all problems but sometimes one of the other linkers could be useful for searching different solution spaces and finding a very good, albeit unexpected circuit.

Fitness Functions

For Logic Synthesis problems, in the Fitness Function Tab of the [Settings Panel](#) you have access to 30 [built-in fitness functions](#). Additionally, you can also design your own [custom fitness function](#) and explore the solution space with it. By choosing Custom in the Fitness Function box, the custom fitness editor is activated.



You can design your own [custom fitness function](#) using the Custom Fitness Function window to write the code of your fitness function. The code for the custom fitness function must be in JavaScript and can be tested before evolving a logic circuit with it by pressing the Test button.



The 30 **built-in fitness functions** of GeneXproTools 4.0 for Logic Synthesis are based on the:

- [Number of Hits](#)
- [Hits with Penalty](#)
- [Accuracy](#)
- [Squared Accuracy](#)
- [Sensitivity/Specificity](#)
- [PPV/NPV](#)
- [SSPN](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [R-square](#)
- [Correlation Coefficient](#)
- And all the above with [parsimony pressure](#)

The kind of fitness function you choose will depend most probably on the indicator you are most familiar with. And although there is nothing wrong with this, for all of them can accomplish an efficient evolution, you might want to try different fitness functions for they travel the fitness landscape differently: some of them very straightforwardly in their pursuits while others choose less obvious paths.



Working with Logic Synthesis Tools

Available Algorithms

GeneXproTools 4.0 uses two different [learning algorithms](#) for Logic Synthesis problems. The first – the basic gene expression algorithm or simply [Gene Expression Programming](#) (GEP) – does not support the direct manipulation of random numerical constants (0's and 1's in this case), whereas the second – [GEP with Random Numerical Constants](#) or [GEP-RNC](#) for short – has a facility for handling them directly. So, these two algorithms search the solution landscape differently and therefore you might wish to try them both on your problems.

The [GEP-RNC algorithm](#) is slightly more complex than [GEP](#) as it uses an additional gene domain (Dc) for encoding the random numerical constants. Consequently, this algorithm comes equipped with an additional set of genetic operators ([RNC mutation](#), [Dc mutation](#), [Dc inversion](#), and [Dc IS transposition](#)) especially developed for handling random numerical constants (if you are not familiar with these operators, please use the default values by clicking the Default button for they work very well in all cases).

Taking into consideration not only the slightly higher complexity of the [GEP-RNC algorithm](#) but also the fact that both these algorithms produce equally elegant circuits, [GEP](#) is the default learning algorithm in GeneXproTools 4.0. However, you can activate the [GEP-RNC algorithm](#) in the Settings Panel -> Numerical Constants by checking the Use Random Numerical Constants box.



Working with Logic Synthesis Tools

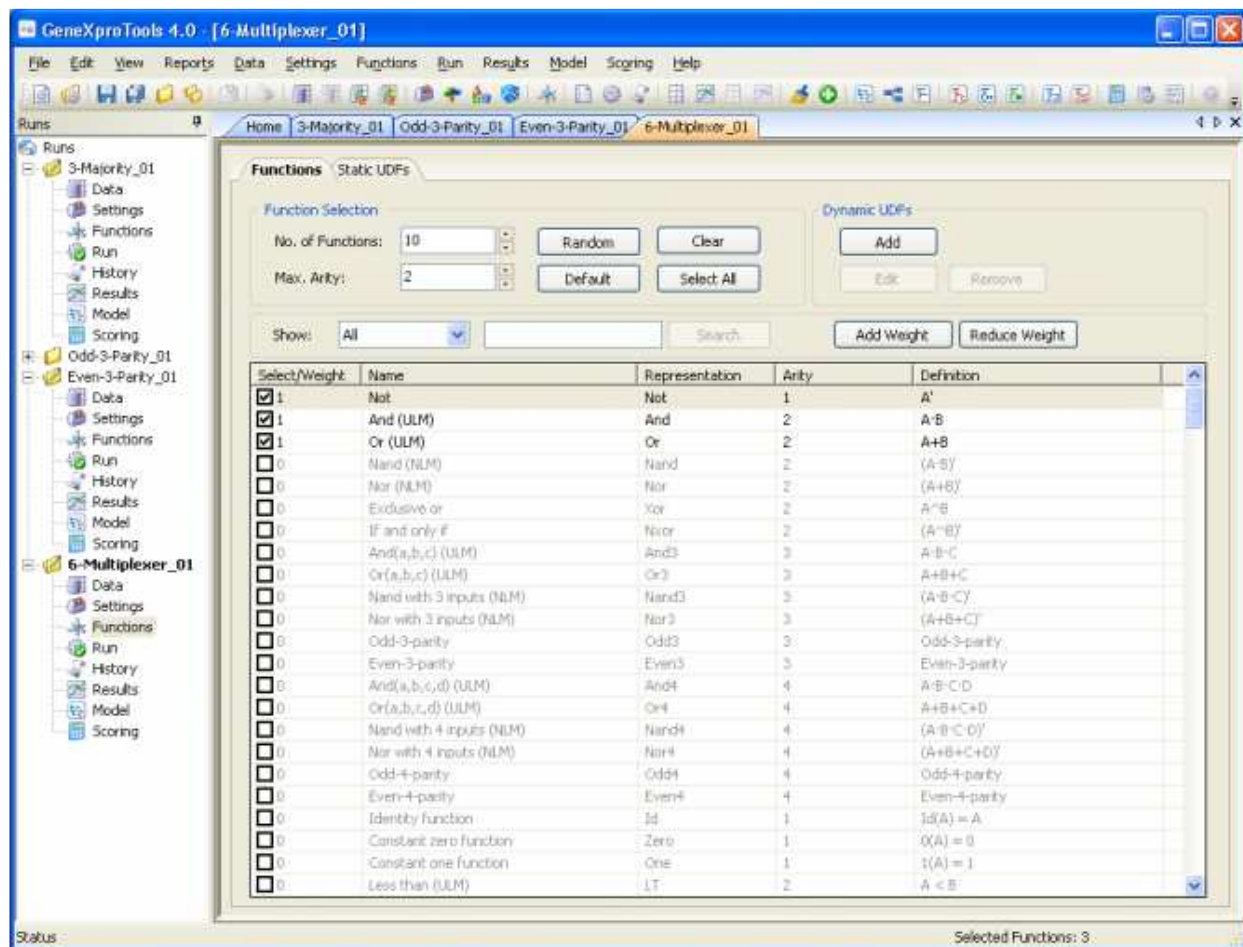
Evolutionary Strategies

Sum-of-products and product-of-sums expressions are alright as solutions to Boolean problems, but most of the times they are far from parsimonious solutions. On the other hand, limiting oneself to the use of just AND-OR-NOT might not be the right approach to finding good solutions to more complex problems. With GeneXproTools you can use an extremely wide spectrum of logical functions (a total of 258 [built-in functions](#) plus all kinds of [user defined functions](#)) to design your logic circuits. And, for instance, if you are just interested in NAND-only or NOR-only circuits, you might try and find a solution to your problem using just this particular function instead of applying De Morgan's theorem to your

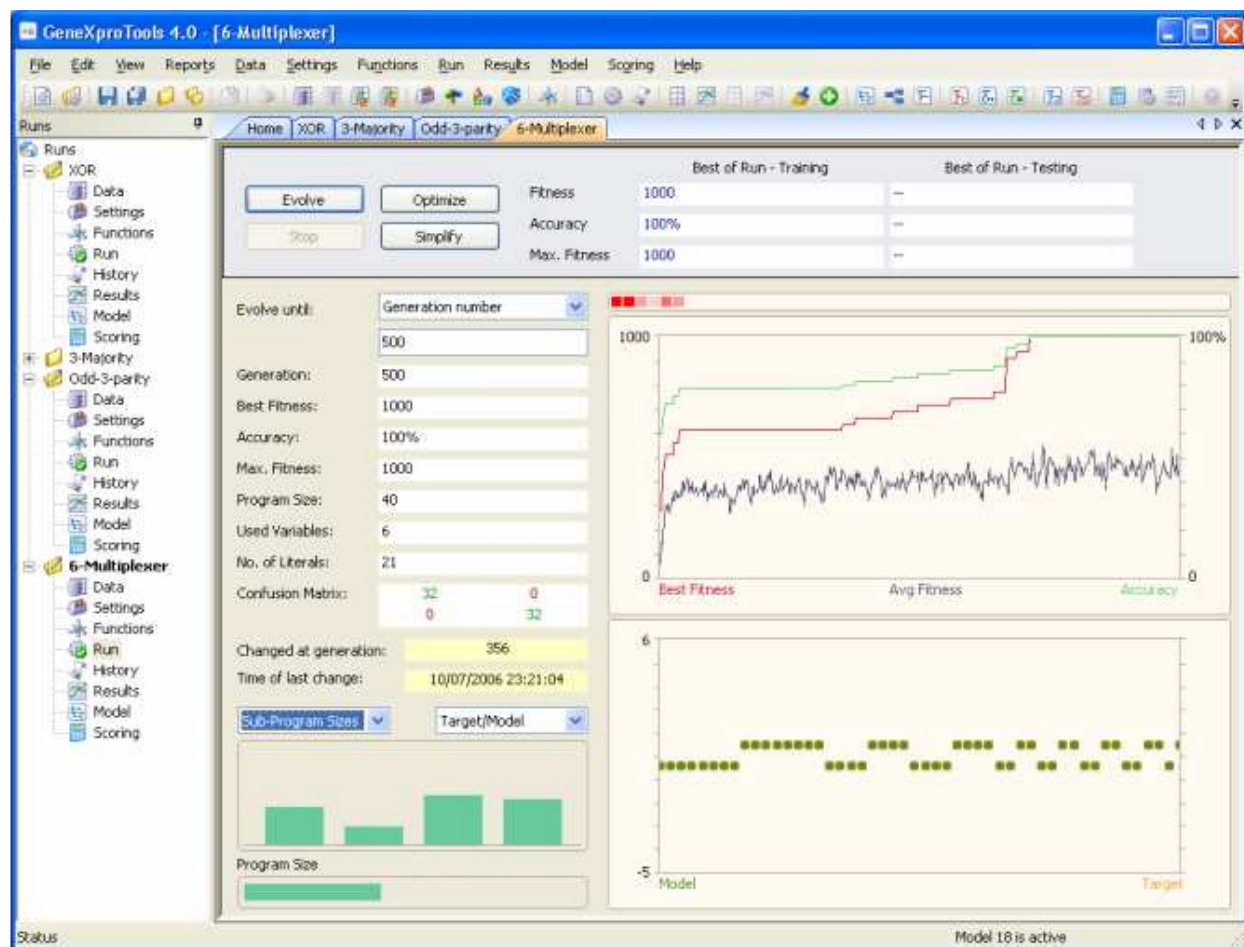
sum-of-products solutions to design your circuits, as most of the times a more parsimonious expression can be found. And if you are interested in less conventional circuits, say a MUX-only circuit or a AND-XOR-NOT circuit, GeneXproTools has no problems working with just these functions to build your circuits.

So, the evolutionary strategies we recommend for Logic Synthesis reflect two main concerns: firstly, evolve your logic circuit with the functions you think are the most appropriate; and secondly, apply parsimony pressure to make it simpler.

The default [run templates](#) GeneXproTools uses for Logic Synthesis use a very simple and universal function set composed of only the basic Boolean operators AND-OR-NOT. If you want to use different gates to design your logic circuits, you'll have to choose them in the Functions Panel.

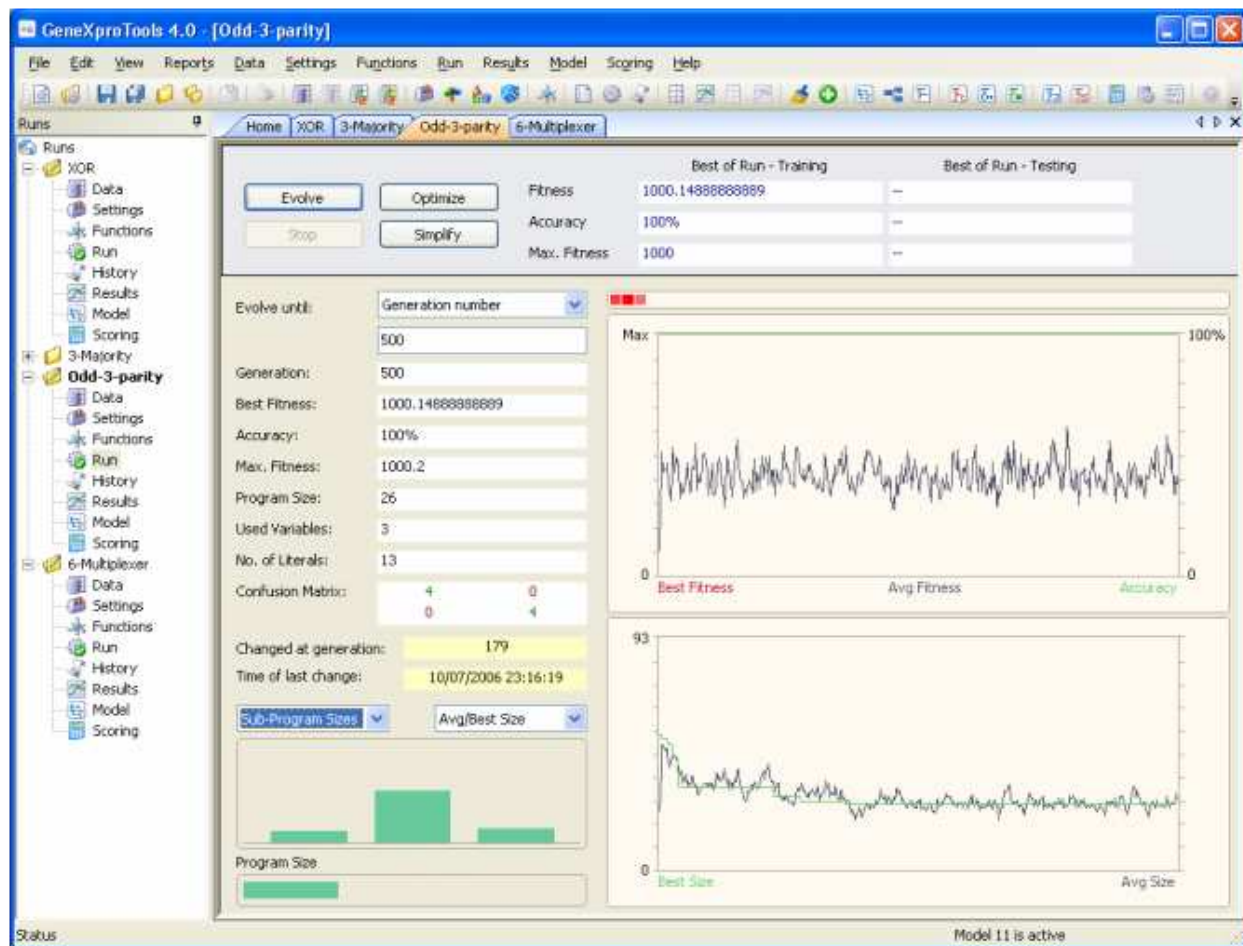


Note also that the GeneXproTools [templates](#) used in Logic Synthesis allow the circuits to evolve without parsimony pressure. This means that you will have to apply the parsimony pressure yourself after the circuit has been designed. For that you just have to press the Simplify button in the Run Panel.



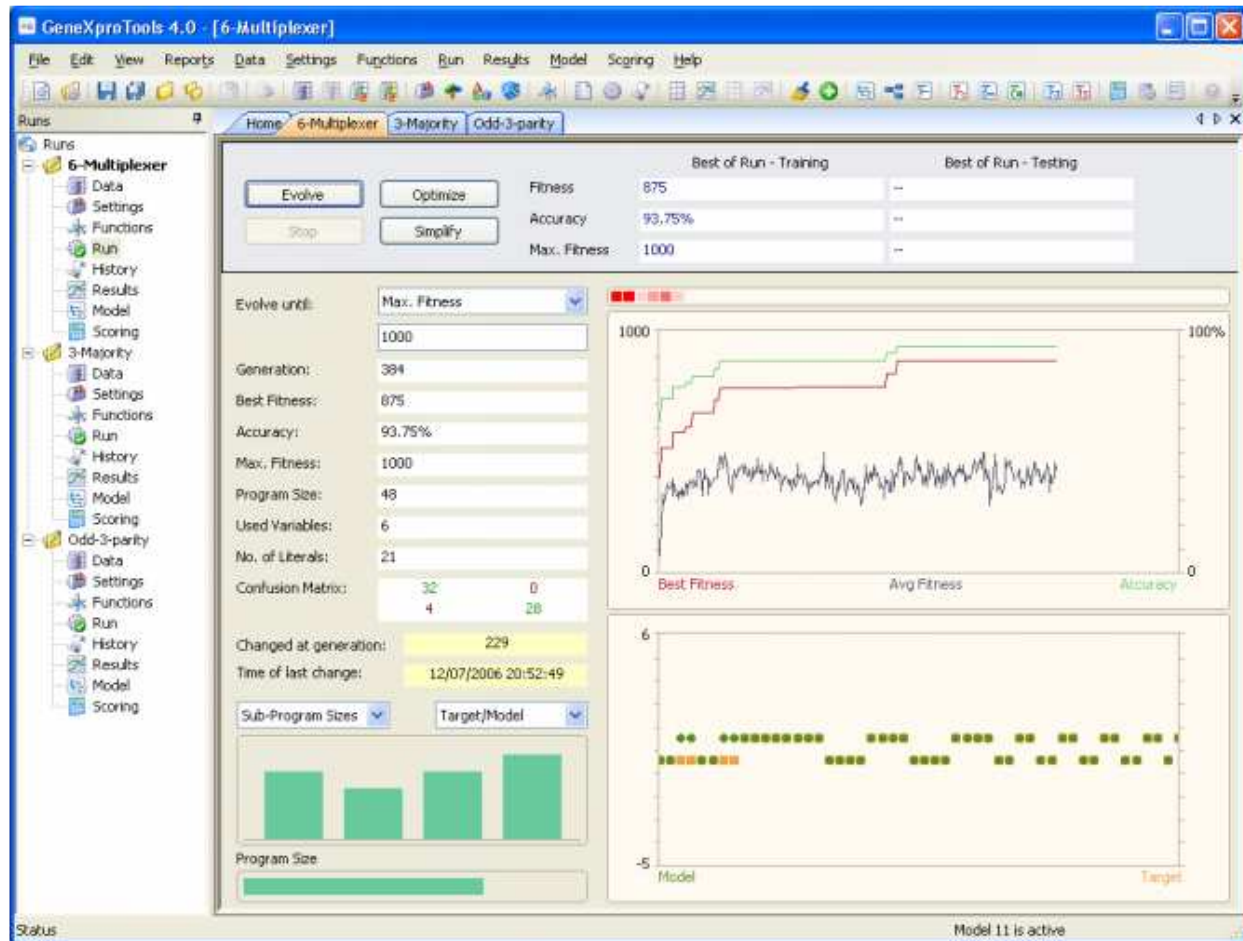
You could obviously start the modeling process by applying parsimony pressure from the onset, but evolutionarily speaking this is not the most efficient approach as better circuits (in terms of fitness) are faster created if more flexible structures are used. So, we recommend you start with a comfortable architecture to allow the discovery of a perfect solution to your problem. Then, by pressing the Simplify button, you use this perfect solution as seed ([evolve with seed](#)) to create simpler versions of it.

GeneXproTools allows you to monitor the simplification process in the Run Panel by choosing Avg/Best Size in the rightmost combo box in the bottom and also by checking the Sub-Program Sizes plot and the Program Size bar at the bottom.



GeneXproTools 4.0 chooses the appropriate template for your problem according to the number of variables in your data. This kind of template is a good starting point that allows you to start the modeling process immediately with just a mouse click. Indeed, even if you are not familiar with evolutionary computation in general and [Gene Expression Programming](#) in particular, you will be able to immediately design sophisticated logic circuits thanks to the templates of GeneXproTools. In these templates, all the adjustable parameters of the learning algorithm are already set and, for instance, you don't have to know how to create genetic diversity, how to set the appropriate population size, the chromosome architecture, the fitness function, how to increase the complexity of your models, and so forth. Then, as you learn more about GeneXproTools, you will be able to explore all its modeling tools and create quickly and efficiently very good logic circuits.

So, after creating a new run you just have to click the Evolve button in the Run Panel in order to design a model. Then you observe carefully the evolutionary process, especially the Target/Model comparison plot. Then, whenever you see fit, you can stop the run without fear of stopping the evolutionary process prematurely as GeneXproTools 4.0 allows you to continue the evolutionary process at a later time by using the best-of-run model as the starting point ([evolve with seed](#)). For that you just have to click on the Optimize button in the Run Panel.



This strategy has enormous advantages as you might choose to stop the run at any time and then take a closer look at the evolved model. For instance, you can analyze its logical structure, translate it into different programming languages, evaluate a wide set of statistical functions for a quick and rigorous assessment of its accuracy, see the complete Model/Target comparison plot in the Results Panel, and so on. Then you might choose to adjust a few parameters, say, choose a different fitness function, expand the function set, add a neutral gene, apply parsimony pressure, and so forth, and then explore this new evolutionary path. You can repeat this process for as long as you want or until a perfect solution has been found.



The Architectures of GeneXproTools Learning Algorithms

This chapter describes the architectures of the learning algorithms used by GeneXproTools 4.0 and teaches you how to explore them so that you can create models that are extremely accurate and robust. In this chapter, you'll learn about:

- [The basic concepts and ideas of Gene Expression Programming.](#)
- [The manipulation of random numerical constants in Gene Expression Programming.](#)

- [The native language of Gene Expression Programming.](#)



The Architectures of GeneXproTools Learning Algorithms

Gene Expression Programming

The understanding of the basic concepts and ideas of Gene Expression Programming (GEP) allows a better understanding of the internal workings of GeneXproTools and, therefore, gives you the right tools to explore its modeling capabilities to the fullest. This section has the following parts:

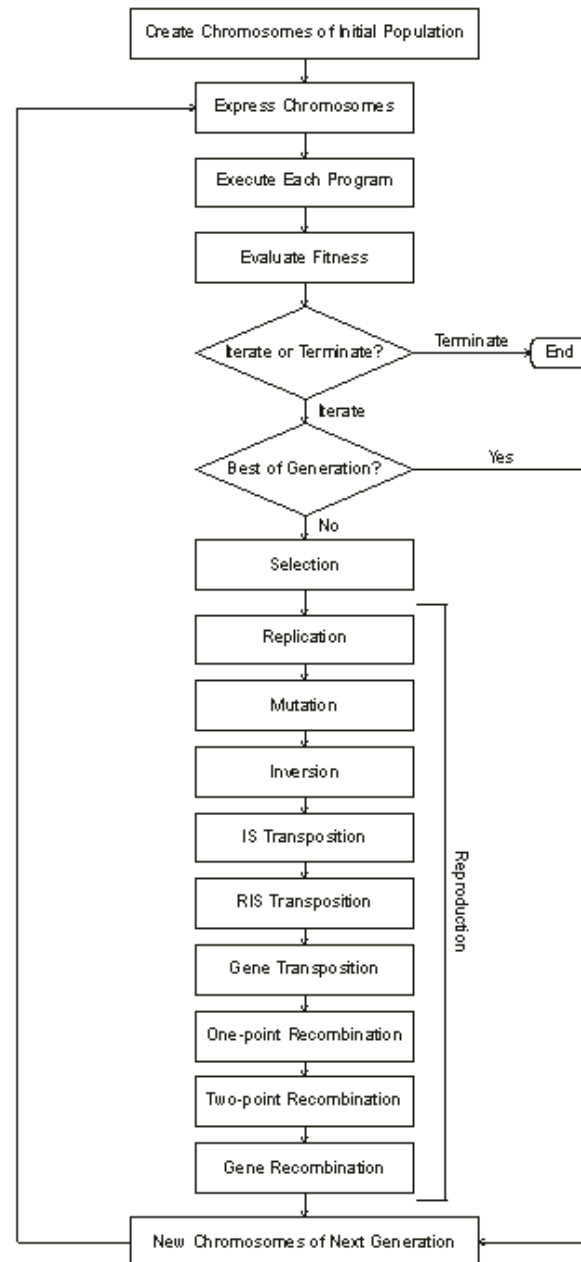
- [An overview of Gene Expression Programming.](#)
- [The architecture of GEP programs.](#)
- [Multigenic chromosomes and linking functions.](#)
- [User defined functions.](#)



The Architectures of GeneXproTools Learning Algorithms

An Overview of Gene Expression Programming

The fundamental steps of Gene Expression Programming are schematically represented in the flowchart below:



The process begins with the random generation of the chromosomes of a certain number of individuals or programs (the **initial population**). Then these chromosomes are expressed and the **fitness** of each program is evaluated against a set of fitness cases (also called selection environment or **training set**). The programs are then selected according to their fitnesses (their performance in that particular environment) to reproduce with modification, leaving progeny with new traits. These new programs are, in their turn, subjected to the same **developmental process**: expression of the chromosomes, confrontation of the selection environment, selection, and reproduction with modification. The process is repeated for a certain number of generations or until a good solution has been found.

The Architectures of GeneXproTools Learning Algorithms

The Architecture of GEP Programs

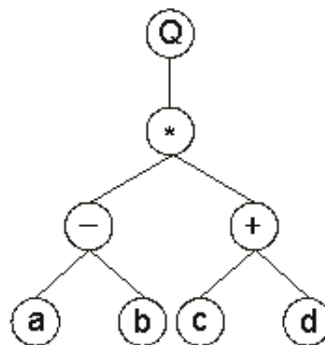
There are two main players in Gene Expression Programming: the **chromosomes** and the **expression trees** (ETs) or programs, being the latter the expression of the genetic information encoded in the former. As in nature, the process of information decoding is called **translation**. And this translation implies obviously a kind of code and a set of rules. The **genetic code** of Gene Expression Programming is very simple: a one-to-one relationship between the symbols of the chromosome and the nodes they represent in the trees. The **rules** are also very simple: they determine the spatial organization of nodes in the expression trees and the type of interaction between sub-ETs. Therefore, there are two languages in GEP: the language of the genes and the language of expression trees and, thanks to the simple rules that determine the structure of ETs and their interactions, it is possible to infer immediately the expression tree given the sequence of a gene, and vice versa. This means that one can choose to have a very complex program represented by its compact genome without losing in meaning. This unequivocal bilingual notation is called **Karva language**. Its details are explained in the remainder of this section.

The **structural organization** of GEP genes is better understood in terms of **open reading frames** (ORFs). In biology, an ORF or coding sequence of a gene begins with the start codon, continues with the amino acid codons, and ends at a termination codon. However, a gene is more than the respective ORF, with sequences upstream of the start codon and sequences downstream of the stop codon. Although in GEP the start site is always the first position of a gene, the termination point does not always coincide with the last position of a gene. Consequently, it is common for GEP genes to have **noncoding regions** downstream of the termination point. These noncoding regions obviously do not interfere with expression but, nonetheless, they play a crucial role in evolution, for they alone allow the creation of valid programs no matter how profoundly their chromosomes are modified. We'll get back to them later.

Consider, for example, the algebraic expression:

$$\sqrt{(a-b)(c+d)}$$

It can also be represented as a diagram or an **expression tree**:



where "Q" represents the square root function.

This kind of diagram representation is what is called the **phenotype** in Gene Expression Programming. And the **genotype** can be easily inferred from the phenotype as follows:

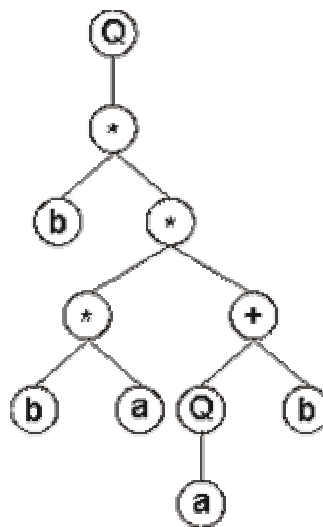
01234567
Q*-+abcd (2)

which is the straightforward reading of the expression tree from left to right and from top to bottom (exactly as we read a page of text). The expression (2) is an ORF, starting at "Q" (position 0) and terminating at "d" (position 7). These ORFs were named **K-expressions** from Karva language.

Consider another ORF, the following K-expression:

01234567890
Q*b**+baQba (3)

Its expression as an ET is also very simple and straightforward. In order to express the ORF correctly, we must follow the rules governing the spatial distribution of functions and terminals. First, the start of a gene corresponds to the **root** of the ET which is placed in the topmost line. Second, in the next line, below each function, are placed as many branch nodes as there are arguments to that function. Third, from left to right, the nodes are filled consecutively with the next elements of the K-expression. Fourth, the process is repeated until a line containing only terminals is formed. In this case, the following expression tree is formed:



Looking at the structure of GEP ORFs only, it is difficult or even impossible to see the advantages of such a representation, except perhaps for its simplicity and elegance. However, when ORFs are analyzed in the context of a gene, the advantages of this representation become obvious. As previously stated, GEP chromosomes have fixed length, and they are composed of one or more genes of equal length. Consequently, the length of a gene is also fixed. Thus, in Gene Expression Programming, what varies is not the length of genes which is constant, but the length of the ORFs. Indeed, the length of an ORF may be equal to or less than the length of the gene. In the first case, the termination point coincides with the end of the gene, and in the latter, the termination point is somewhere upstream of the end of the gene.

What is the function of these noncoding regions of GEP genes? We will see that they are the essence of Gene Expression Programming and evolvability, for they allow the modification of the genome using several [genetic operators](#) without restrictions, always producing syntactically correct programs. Thus, in Gene Expression Programming, the fundamental property of genotype/phenotype systems – **syntactic closure** – is intrinsic, allowing the totally unconstrained restructuring of the genotype and, consequently, an efficient evolution.

Let's analyze then the **structural organization of GEP genes** in order to understand how they invariably code for syntactically correct programs and why they allow an unconstrained application of virtually any genetic operator.

The genes of Gene Expression Programming are composed of a **head** and a **tail**. The head contains

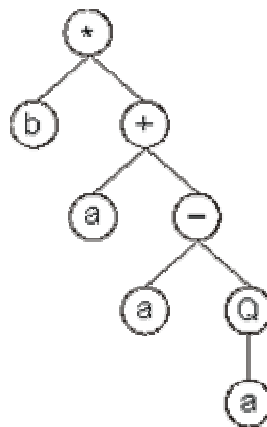
symbols that represent both functions and terminals, whereas the tail contains only terminals. For each problem, the length of the head h is chosen, whereas the length of the tail t is a function of h and the number of arguments n of the function with more arguments (also called **maximum arity**) and is evaluated by the equation:

$$t = h (n-1) + 1$$

Consider a gene for which the **set of functions** $F = \{Q, *, /, -, +\}$ and the **set of terminals** $T = \{a, b\}$. In this case $n = 2$; if we chose an $h = 15$, then $t = 15 (2 - 1) + 1 = 16$; thus, the length of the gene g is $15 + 16 = 31$. One such gene is shown below (the head is shown in blue):

0123456789012345678901234567890
 *b+a-aQab+//+b+babbabbbababbbaaa (4)

It codes for the following ET:

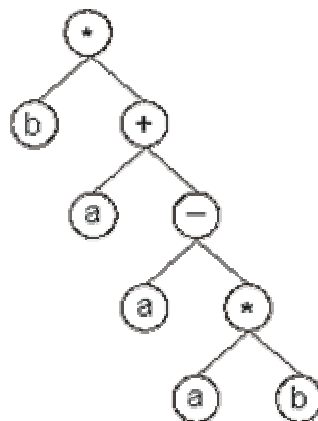


In this case, the K-expression ends at position 7, whereas the gene ends at position 30.

Suppose now a mutation occurred at position 6, changing the "Q" into "*". Then the following gene is obtained:

0123456789012345678901234567890
 *b+a-a*ab+//+b+babbabbbababbbaaa (5)

And its expression gives:



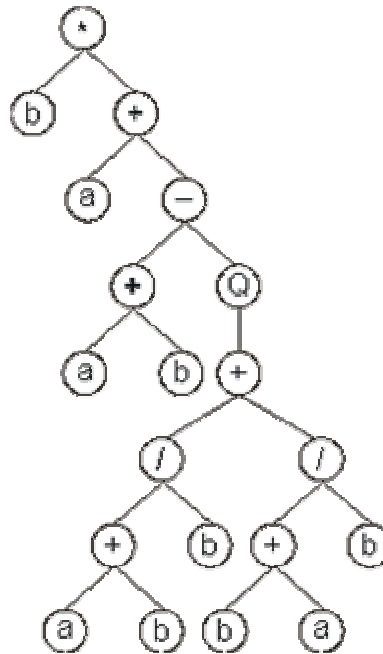
In this case, the termination point shifts one position to the right (position 8), changing slightly the

daughter tree.

Consider another mutation in chromosome (4) above, the substitution of "a" at position 5 by "+", resulting in the following chromosome:

```
0123456789012345678901234567890
*b+a-+Qab+//+b+babbabbbababbbaaa    (6)
```

And its expression gives:

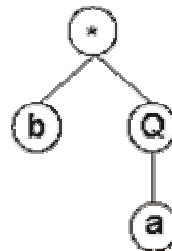


In this case, the termination point shifts twelve positions to the right (position 19), enlarging and changing significantly the daughter tree.

Obviously the opposite also might happen, and the daughter tree might shrink. For example, consider again gene (4) above, and suppose a mutation occurred at position 2, changing the "+" into "Q":

```
0123456789012345678901234567890
*bQa-aQab+//+b+babbabbbababbbaaa    (6)
```

And now its expression results in the following ET:



In this case, the ORF ends at position 3, shortening the original ET in four nodes.

So, despite their fixed length, each gene has the potential to code for expression trees of different sizes and shapes, where the simplest is composed of only one node (when the first element of a gene is a

terminal) and the largest is composed of as many nodes as the length of the gene (when all the elements of the head are functions with maximum arity).

It is evident from the examples above, that any modification made in the genome, no matter how profound, always results in a structurally correct program. Obviously, the structural organization of genes must be preserved, always maintaining the boundaries between head and tail.

If you want to explore further the plasticity of GEP chromosomes, please see the next chapter [Genetic Operators](#) where the mechanisms and effects of all the genetic operators used in GeneXproTools 4.0 are thoroughly analyzed.



The Architectures of GeneXproTools Learning Algorithms

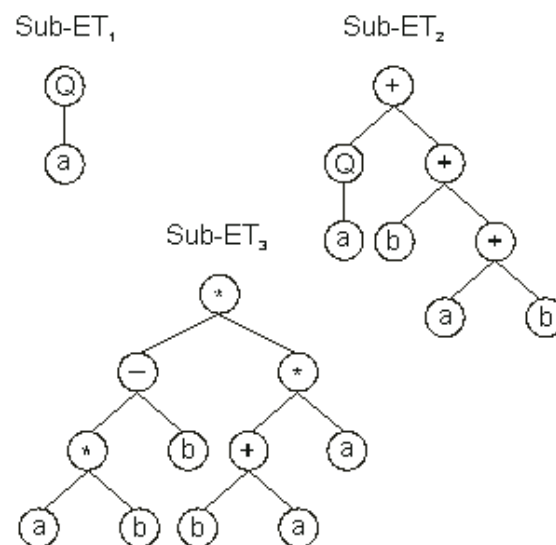
Multigenic Chromosomes and Linking Functions

The chromosomes of Gene Expression Programming are usually composed of more than one [gene](#) of equal length. For each problem or run, the number of genes, as well as the size of the [head](#), are a priori chosen. Each gene codes for a [sub-expression tree](#) (sub-ET) and the sub-ETs interact with one another forming a more complex multi-subunit expression tree.

Consider, for example, the following chromosome with length 39, composed of three genes, each with length 13 (the heads are shown in blue):

```
012345678901201234567890120123456789012
QaQ+-Qbbaaaba+Q+ab+abababa*-**b+aabbaba
```

It has obviously three [open reading frames](#) (ORFs), and each ORF codes for a sub-ET:

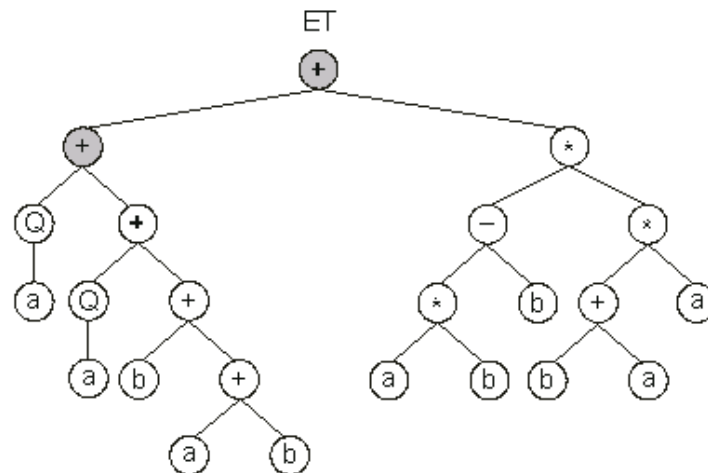


The start of each ORF is always given by position 0; the end of each ORF, though, is only evident upon construction of the corresponding sub-ET. As shown in the Figure above, the first ORF ends at position 1; the second ORF ends at position 7; and the last ORF ends at position 10. Thus, GEP chromosomes contain several ORFs of different sizes, each ORF coding for a structurally and functionally unique sub-

ET.

Obviously, these sub-ETs or sub-programs must interact with one another. And in GeneXproTools 4.0 they interact through special functions, the so called **linking functions**: addition, subtraction, multiplication, and division for mathematical models and And, Or, Nand, Nor, Xor, Nxor, Less Than, Greater Than, Less Or Equal, and Greater Or Equal for Logic Synthesis problems.

The linking by addition of all the three sub-ETs shown in the Figure above is illustrated below (the linking functions are shown in gray):



Note that the final program represented in the Figure above could be linearly encoded as the following [K-expression](#):

```
01234567890123456789012
++*Q+-*aQ+*b+aab+abbaab
```

However, the use of **multigenic chromosomes** is more appropriate to evolve solutions to complex problems, for they permit the modular construction of complex, hierarchical structures, where each gene codes for a smaller and simpler building block. These smaller building blocks are separated from each other, and thus can evolve independently and may, therefore, give rise to new useful building blocks.



The Architectures of GeneXproTools Learning Algorithms

User Defined Functions

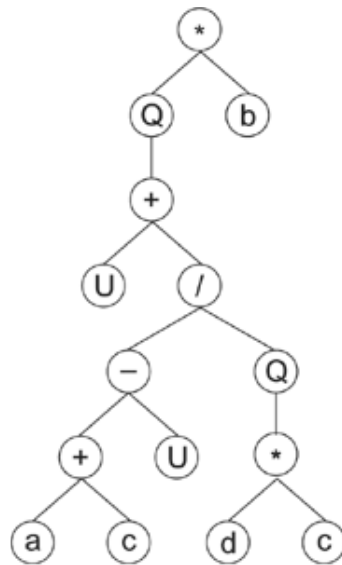
In Gene Expression Programming, [user defined functions](#) (UDFs) may represent functions of several variables although their arity in terms of [expression rules](#) is equal to zero. That is, nodes with UDFs behave as leaf or terminal nodes. So, the implementation of UDFs in Gene Expression Programming can be done using at least two different methods: either they are treated as terminals and are used both in the [heads](#) and [tails](#) or they are treated as functions and are used exclusively in the [heads](#). Either way, the algorithm works very well. Thus, it is a matter of taste which one to choose. In GeneXproTools 4.0 the latter is used as it seems more consistent and less confusing.

Let's see now how genes containing UDFs are expressed. Consider, for instance, the gene below with a

head size of 11 (the head is shown in blue):

```
01234567890123456789012
*Qb+U/-Q+U*acdcbbaaabcc
```

where "U" represents a UDF and "Q" represents the square root function. Its expression results in the following [expression tree](#) (ET):



This UDF "U" could represent any function, for instance, the logistic function $1/(1+e^{-x})$ or the power function x^y or whatever function one wishes to implement. The difference between a UDF and a normal function (any [DDFs](#) or any function from the 279 [built-in mathematical functions](#) or from the 258 [built-in logical functions](#) of GeneXproTools 4.0) is that the arguments to the UDF are fixed during the definition of the function whereas the arguments to the normal function are flexible and depend on the particular configuration of the expression tree. Suppose, for instance, that in the chromosome above,

the UDF "U" represents a particular square root function, more precisely $\sqrt{c}$. It's worth comparing this fixed square root function with the normal, flexible square root function. For instance, in the program above, the square root function at position 7 corresponds to $\sqrt{d \cdot c}$, whereas the "Q" at

position 1 corresponds to $\sqrt{\sqrt{c} + \frac{a+c-\sqrt{c}}{\sqrt{dc}}}$.

However, despite their rigidity, UDFs are extremely useful, especially when they are carefully crafted to the problem at hand for they are not restricted to any particular number of arguments and can be used to express relationships involving hundreds or even thousands of variables. Obviously, a certain amount of inside knowledge about the data is necessary to design well crafted UDFs and with the [data visualization tools](#) of GeneXproTools 4.0 you can easily find important relationships between your variables and explore them to create useful UDFs that can be used to design highly sophisticated models composed of these handcrafted building blocks.



The Architectures of GeneXproTools Learning Algorithms

Gene Expression Programming with Random Numerical Constants

In this section you'll find information about the second learning algorithm used by GeneXproTools 4.0 – Gene Expression Programming with Random Numerical Constants (GEP-RNC), including a brief introduction to:

- [The problem of constants' creation in symbolic regression.](#)
- [The architecture of GEP-RNC programs.](#)



The Architectures of GeneXproTools Learning Algorithms

Function Finding and the Creation of Numerical Constants

It is assumed that the creation of floating-point constants is necessary to do symbolic regression using evolutionary computing methods, perhaps because they seem to be essential for conventional mathematical modeling. For instance, the genetic programming technique solves the problem of constant creation by using a special terminal named *ephemeral random constant*. For each ephemeral random constant used in the trees of the initial population, a random number of a special data type in a specified range is generated. Then these random constants are moved around from tree to tree by the crossover operator.

Gene Expression Programming (GEP) solves the problem of constants' creation more efficiently as it uses an extra **terminal "?"** and an extra **domain Dc** composed of the symbols chosen to represent the random numerical constants (RNCs). For each gene, the random constants are generated during the inception of the initial population and kept in an array. The values of each random constant are only assigned during [gene expression](#). Furthermore, a special mutation operator ([RNC mutation](#)) is used to introduce genetic variation in the available pool of random constants by mutating the random constants directly. In addition, the usual [genetic operators](#) of GEP plus a [Dc specific mutation](#), a [Dc specific inversion](#) and a [Dc specific transposition](#), guarantee the effective circulation of the random constants in the population. Consequently, with this scheme of constants' manipulation, the appropriate diversity of random constants can be generated in the beginning of a run and maintained easily afterwards by the genetic operators.

These two algorithms search the solution landscape differently and therefore you might wish to try them both on your problems. Not surprisingly, the kinds of models these algorithms produce are quite different and, when both of them perform equally well on the problem at hand, you still might prefer one to the other. But there are cases, however, where numerical constants are crucial for an efficient modeling and, therefore, the [GEP-RNC algorithm](#) is the default in GeneXproTools 4.0 for all kinds of mathematical modeling (Function Finding, Classification, and Time Series Prediction). For Logic Synthesis problems, though, the [simpler GEP](#) is used as default as numerical constants here are limited to zero and one and can be easily accessed through other means.



The Architectures of GeneXproTools Learning Algorithms

The Architecture of GEP-RNC Programs

Random numerical constants (RNCs) can be easily implemented in Gene Expression Programming (GEP). For that an additional **domain Dc** is introduced in GEP [genes](#). Structurally, the Dc comes after the [tail](#), has a length t equal to the length of the tail, and is composed of the symbols used to represent the random constants. Therefore, another region (besides the [head](#) and [tail](#)) with defined boundaries and its own alphabet is created in the gene.

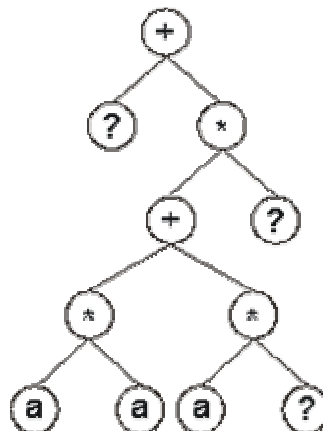
Although this extra domain Dc is essential for handling numerical constants, GeneXproTools 4.0 hides the Dc from the user and takes care of all the decoding. However, if you want to understand the workings of essential features for fine-tuning numerical constants such as special genetic operators and initial diversity of random numerical constants, it's a plus if you know a little about its structure and organization.

For each gene, the numerical constants are randomly generated at the beginning of a run, but their circulation is guaranteed by the usual genetic operators of [mutation](#), [transposition](#), [inversion](#), and [recombination](#). Besides, a special mutation operator ([RNC mutation](#)) allows the permanent introduction of variation in the set of random constants; and three Dc specific operators – [Dc Mutation](#), [Dc IS transposition](#), and [Dc inversion](#) – guarantee the effective shuffling of the constants through the programs of the population.

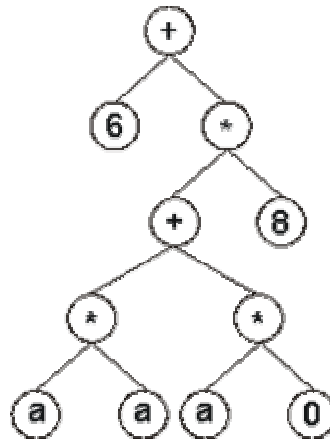
Consider the single-gene chromosome with a head size $h = 7$ (the Dc is shown in blue):

```
01234567890123456789012
+?*+?***aaa??aaa68083295
```

where the terminal "?" represents the random constants. The expression of this kind of chromosome is done exactly as in the [basic gene expression algorithm](#), obtaining:



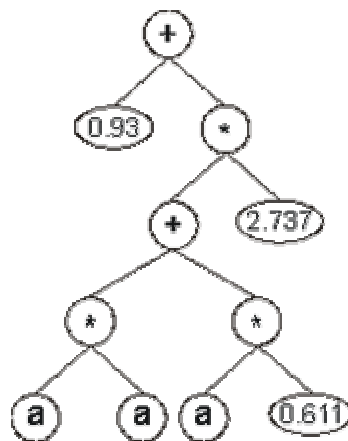
Then the ?'s in the expression tree (ET) are replaced from left to right and from top to bottom by the symbols (numerals, for simplicity) in Dc, obtaining:



The values corresponding to these symbols are kept in an array. For simplicity, the number represented by the numeral indicates the order in the array. For instance, for the 10 elements array:

$$A = \{0.611, 1.184, 2.449, 2.98, 0.496, 2.286, 0.93, 2.305, 2.737, 0.755\}$$

the chromosome above gives:



Obviously, the GEP-RNC algorithm can also be used to evolve highly sophisticated programs composed of multiple sub-programs through the use of [multigenic chromosomes](#). The only difference is that the genes in the GEP-RNC algorithm carry an extra domain for encoding the random numerical constants.



The Architectures of GeneXproTools Learning Algorithms

Karva Notation: The Endogenous Code of GeneXproTools

In this section you'll learn about:

- [Karva notation and how to manipulate K-expressions.](#)
- [The symbols of Karva notation in GeneXproTools 4.0.](#)



The Architectures of GeneXproTools Learning Algorithms

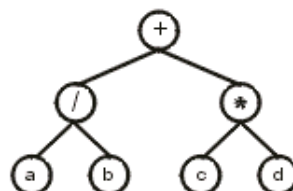
Karva Notation and the Manipulation of K-expressions

Karva notation was developed specifically for [Gene Expression Programming](#) (GEP) and consists of a universal way of representing any mathematical or logical expression that can be represented as a tree. Here we are going to introduce it briefly, but you can find a more detailed explanation on the following sub-sections of this chapter – [The Architecture of GEP Programs](#) and [The Architecture of GEP-RNC Programs](#).

The chromosomes of Gene Expression Programming code for one or more [K-expressions](#), and each K-expression is encoded in a [gene](#). Genes and K-expressions are very easy to interpret. For example, the gene:

0123456
 +/*abcd

can be represented as a diagram or [expression tree](#) (ET):



The **translation** of the gene into the corresponding ET is straightforward: The first element in the gene (position 0) corresponds to the root of the ET; then, below that node, are attached as many nodes as there are arguments to that function (two, in this case); then these nodes are filled consecutively with the elements in the gene (in this case, positions 1 and 2), and so forth. The process is repeated until a line composed of only terminals is formed (in this case, the third line).

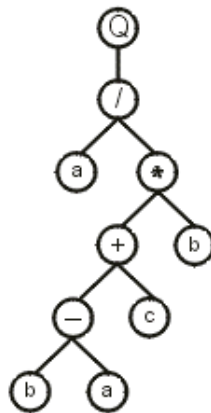
More formally, the gene and ET above are expressed by the expression:

$$\frac{a}{b} + cd$$

Usually the genes evolved by GEP are more complex than the gene presented above, with **noncoding regions** at the end. Consider the following gene (the **head** is shown in blue):

01234567890123456
 Q/a*+b-cbabaccbac

where "Q" represents the square root function. This gene has a **head** of length 8 and a **tail** of length 9. Note that the head contains both functions and terminals, whereas the tail contains exclusively terminals. The translation of such genes is done exactly as in the previous example, obtaining:



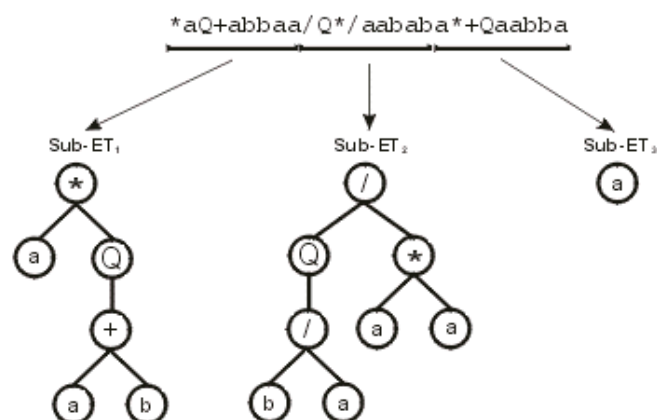
Note that, in this case, not all the elements in the gene were used to construct the ET, as the translation ends whenever a line containing only terminals is formed. In this case, the gene ends at position 16 whereas the K-expression ends at position 9.

Furthermore, GEP chromosomes are usually **multigenic**, and each gene codes for a sub-ET or sub-program. After translation, the sub-ETs are linked by a particular **linking function**: addition, subtraction, multiplication, or division for all kinds of mathematical models in GeneXproTools 4.0 and And, Or, Nand, Nor, Xor, Nxor, Less Than, Greater Than, Less Or Equal, and Greater Or Equal for logical expressions.

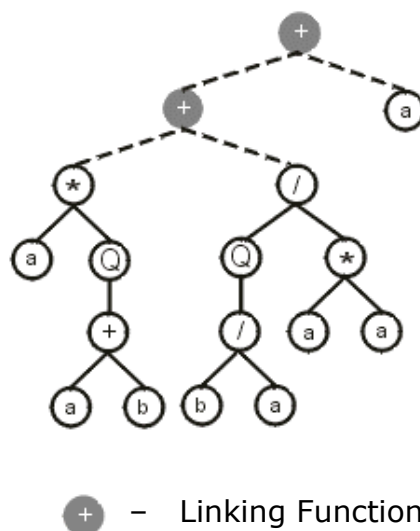
For example, the following chromosome composed of three genes (position 0 indicates the beginning of each gene):

012345678012345678012345678
 aQ+abbba/Q/aababa*+Qaabbba

encodes the following sub-ETs:



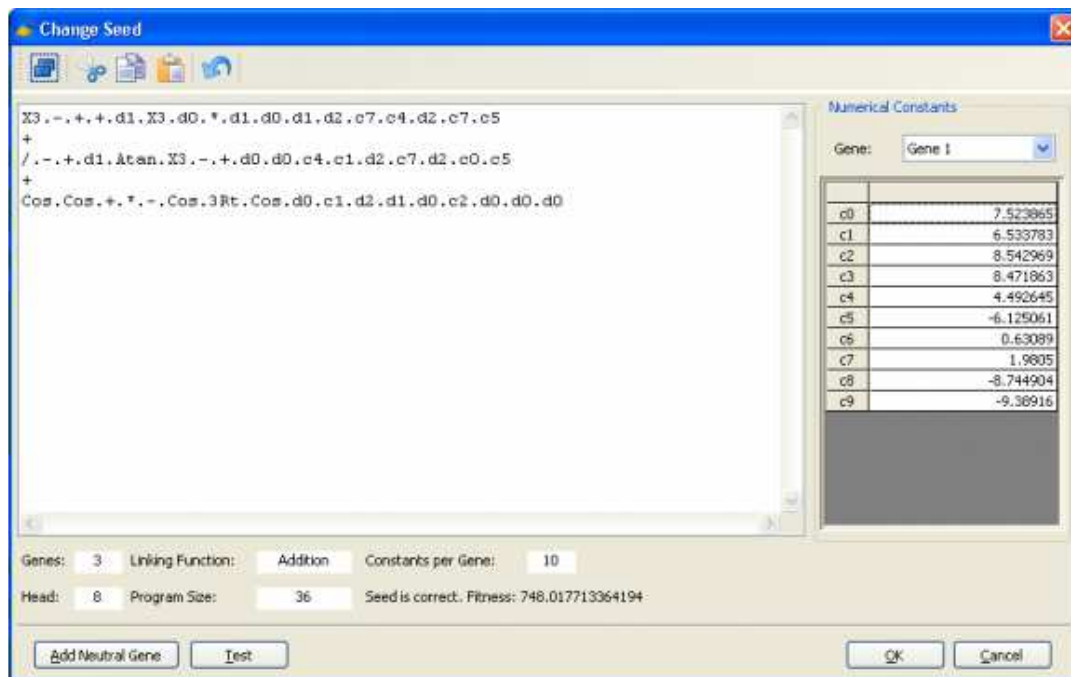
Then the sub-ETs are afterwards linked by one of the available [linking functions](#). For instance, if the linking function were addition, then the following program would be obtained:



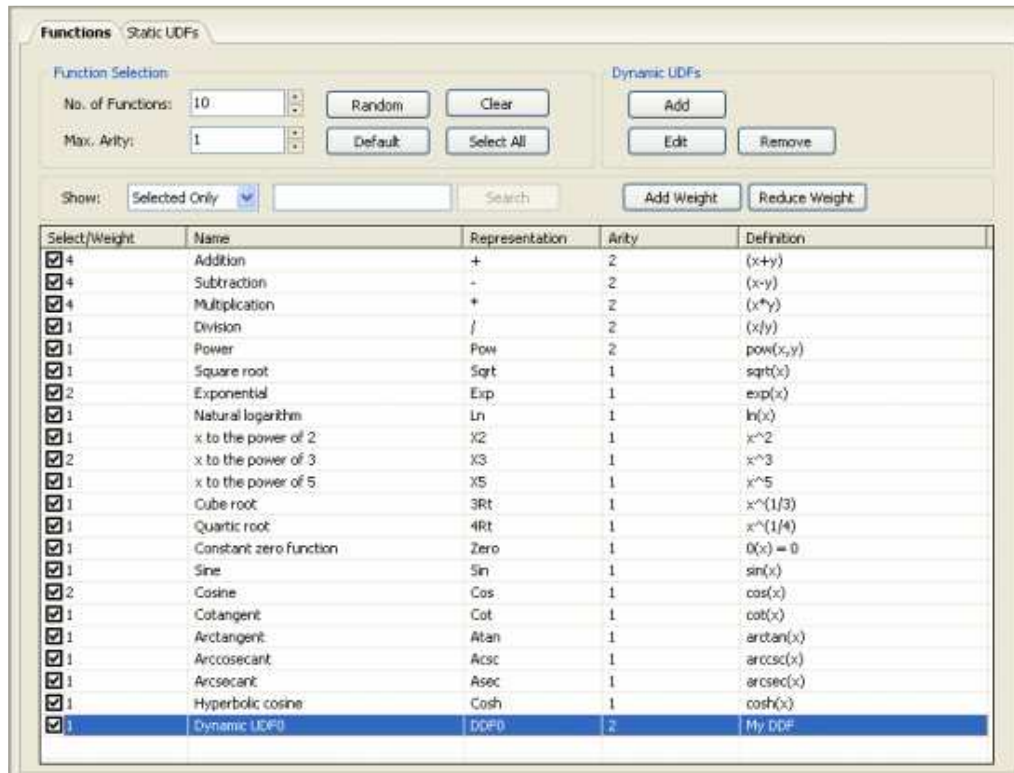
Note that the ET above can be easily linearized into a single [K-expression](#):

```
01234567890123456
++a*/aQQ*/aaabba
```

These manipulations are important in order to fully explore all the features of GeneXproTools, especially the [Change Seed method](#), which allows you to change or create a particular program and then use this program to breed even better programs.

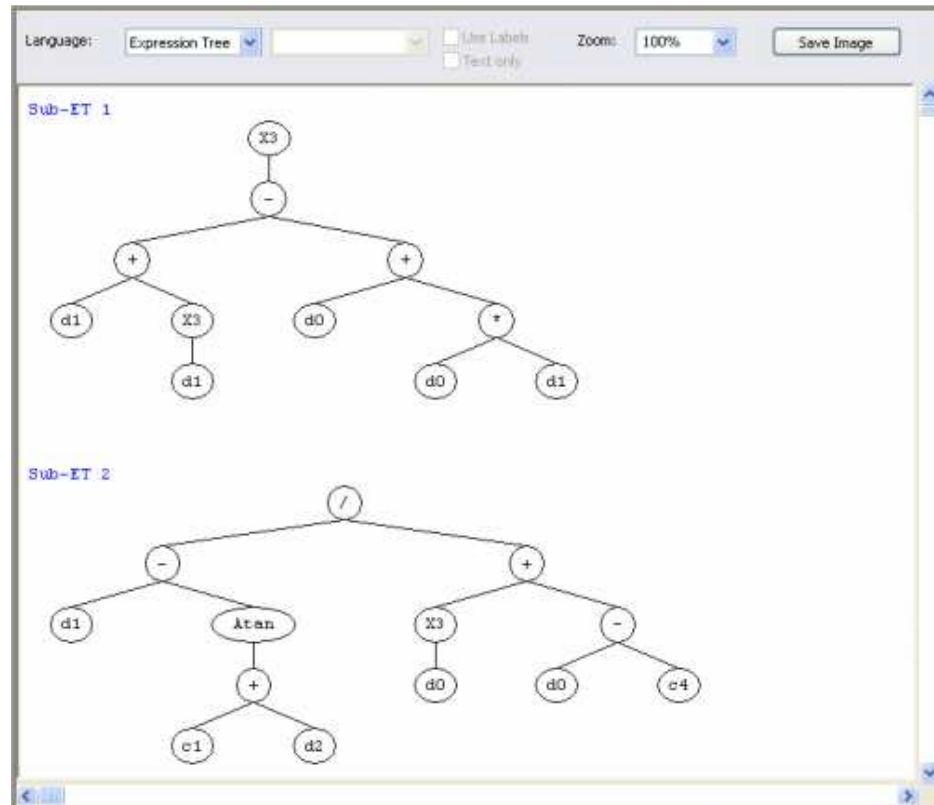


Obviously you'll need to know the symbols used to represent all the 279 [built-in mathematical functions](#) plus the 258 [built-in logical functions](#) of GeneXproTools 4.0 in Karva if you wish to create seed programs with them. But, as will be shown [next](#), they are quite easy to learn as their names obey common rules of naming mathematical/logical functions in programming languages.

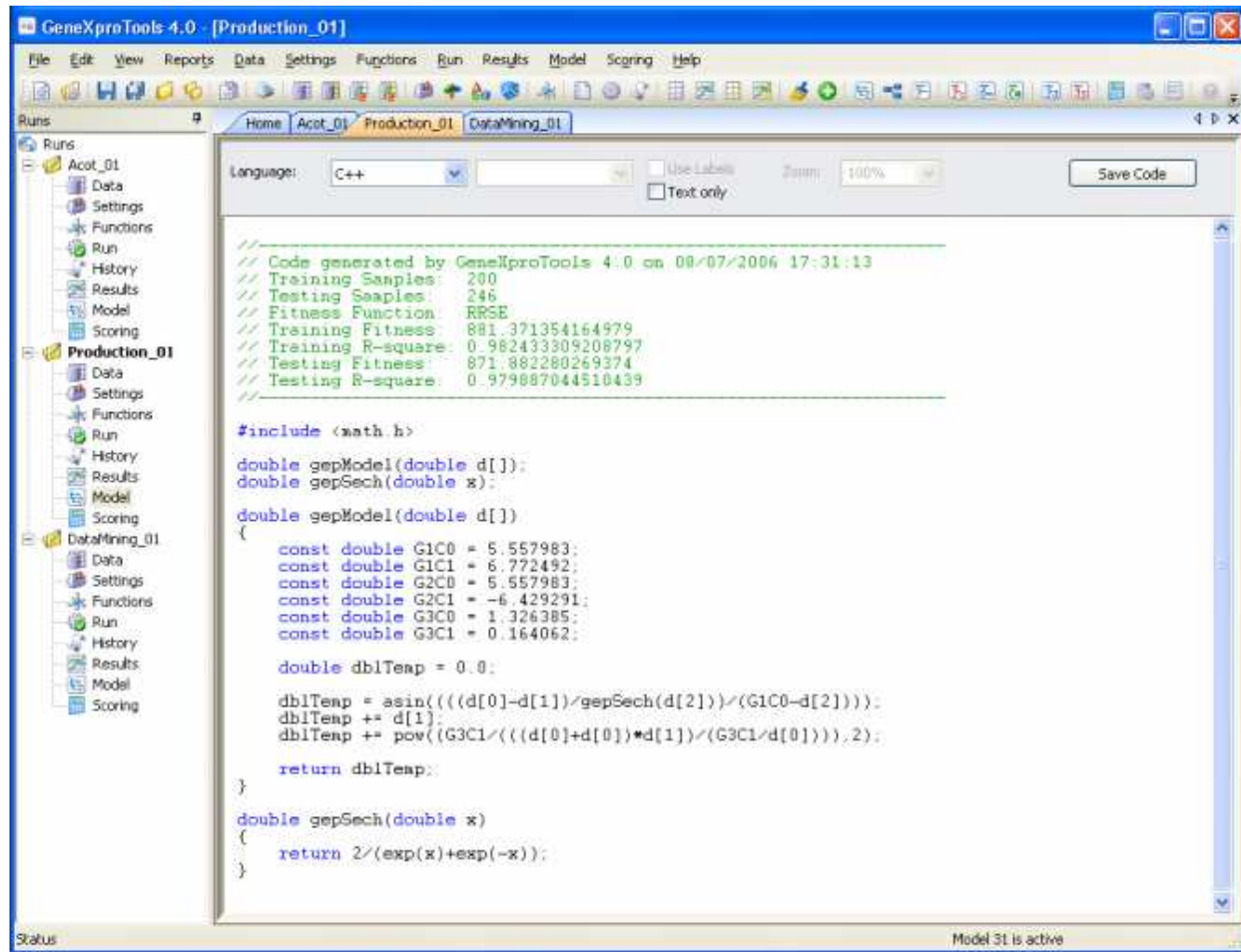


So, GeneXproTools 4.0 evolves models in its native [Karva code](#), but the models evolved by GeneXproTools in its native language can be automatically parsed into visually appealing [expression](#)

trees, allowing a quicker and more complete understanding of their mathematical/logical intricacies.



Additionally, GeneXproTools 4.0 offers a total of 16 [built-in grammars](#) so that the models evolved by GeneXproTools in its native language can be automatically translated into the most commonly used programming languages (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL). This code can then be used in other applications.



And finally, GeneXproTools 4.0 allows also the design of [User Defined Grammars](#) so that the models evolved by GeneXproTools in its native language can be automatically translated into the programming language of your choice if you happen to prefer a programming language not already covered by the 16 [built-in grammars](#) of GeneXproTools 4.0 (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL).

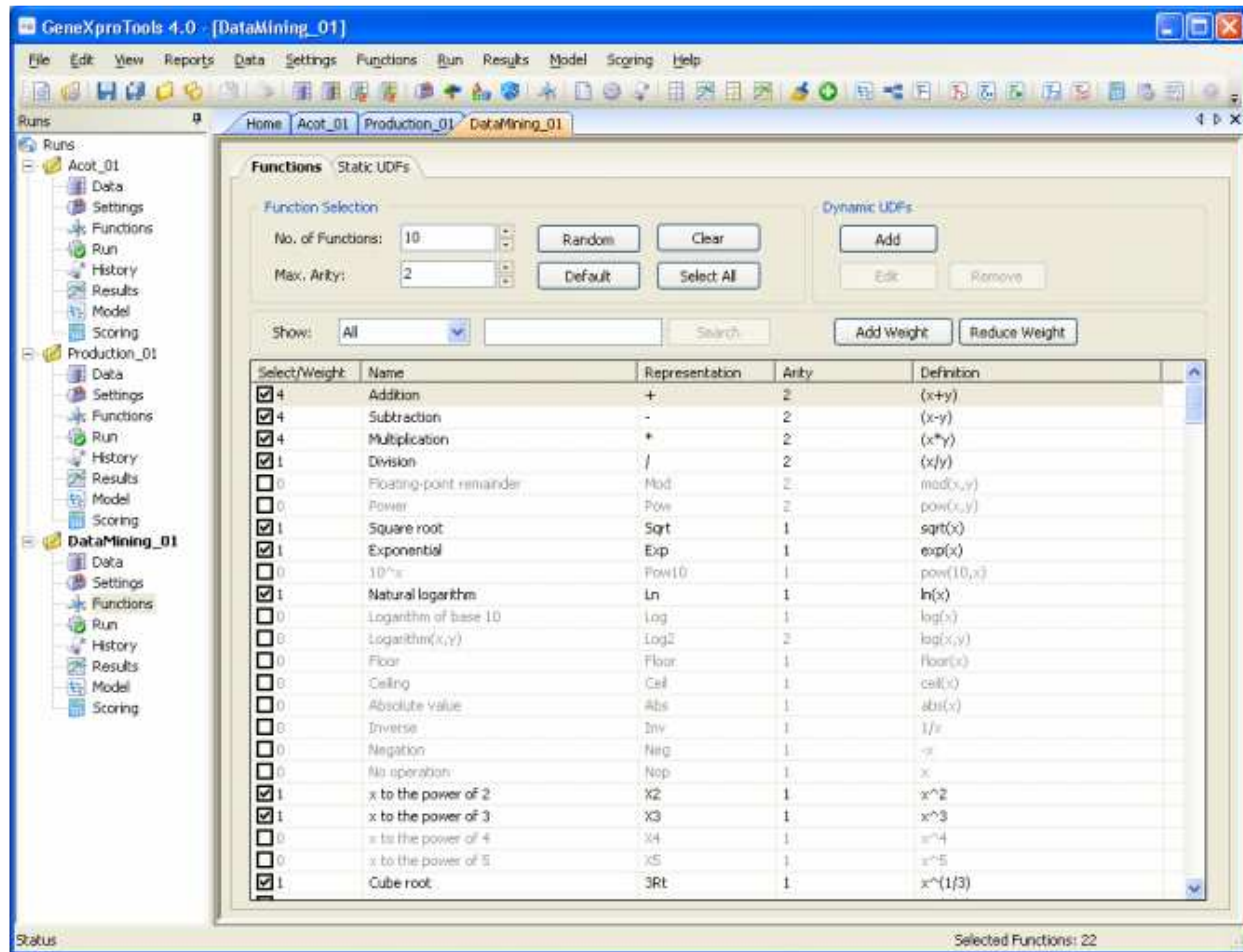


The Architectures of GeneXproTools Learning Algorithms

The Symbols of Karva Notation

In this section are listed:

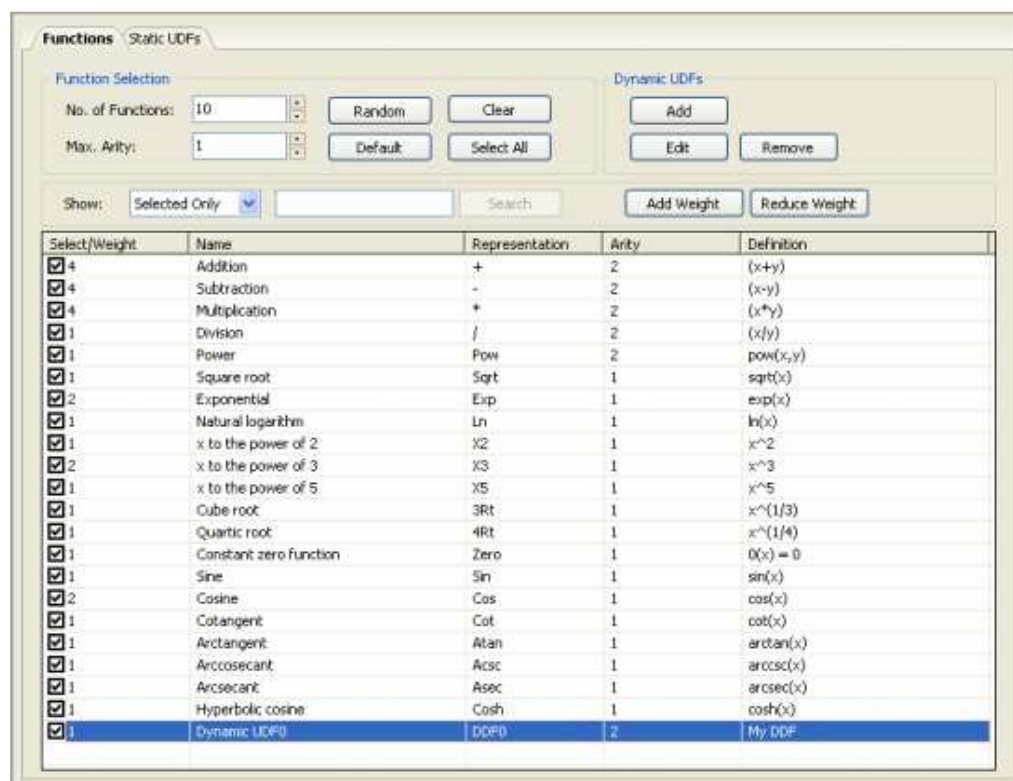
- All the 279 built-in mathematical functions of GeneXproTools 4.0.
- All the 258 built-in logical functions of GeneXproTools 4.0.



The Architectures of GeneXproTools Learning Algorithms

Built-in Mathematical Functions

Below are listed all the 279 [built-in mathematical functions](#) available in GeneXproTools 4.0, starting with their representation in [Karva Notation](#) and their mathematical definition (for more details about their implementations, see the [built-in grammars](#) of GeneXproTools):



Basic arithmetic functions:

- **+**: addition
- **-**: subtraction
- *****: multiplication
- **/**: division
- **Add3**: addition with 3 inputs
- **Sub3**: subtraction with 3 inputs
- **Mul3**: multiplication with 3 inputs
- **Div3**: division with 3 inputs
- **Add4**: addition with 4 inputs
- **Sub4**: subtraction with 4 inputs
- **Mul4**: multiplication with 4 inputs
- **Div4**: division with 4 inputs

Logarithmic functions:

- **Ln**: natural logarithm
- **Log**: logarithm of base 10
- **Log2**: logarithm(x,y)

Exponential and power functions:

- **Sqrt**: square root
- **Exp**: exponential
- **Pow**: power
- **Pow10**: 10^x
- **X2**: x^2
- **X3**: x^3
- **X4**: x^4

- **X5**: x^5
- **3Rt**: cube root
- **4Rt**: quartic root
- **5Rt**: quintic root
- **Logi**: $\text{logistic}(x)$
- **Logi2**: $\text{logistic}(x,y)$
- **Logi3**: $\text{logistic}(x,y,z)$
- **Logi4**: $\text{logistic}(a,b,c,d)$
- **Gau**: $\text{Gaussian}(x)$
- **Gau2**: $\text{Gaussian}(x,y)$
- **Gau3**: $\text{Gaussian}(x,y,z)$
- **Gau4**: $\text{Gaussian}(a,b,c,d)$

Rounding and other simple functions:

- **Floor**: floor
- **Ceil**: ceiling
- **Mod**: floating-point remainder
- **Abs**: absolute value
- **Inv**: inverse
- **Neg**: negation
- **NOT**: complement
- **Nop**: no operation

Minimum, maximum, and average functions:

- **Min2**: minimum of 2 inputs
- **Min3**: minimum of 3 inputs
- **Min4**: minimum of 4 inputs
- **Max2**: maximum of 2 inputs
- **Max3**: maximum of 3 inputs
- **Max4**: maximum of 4 inputs
- **Avg2**: average of 2 inputs
- **Avg3**: average of 3 inputs
- **Avg4**: average of 4 inputs

Constant functions:

- **Zero**: constant zero function
- **One**: constant one function
- **Zero2**: $0(x,y)$
- **One2**: $1(x,y)$
- **Pi**: number π
- **E**: Euler's number

Trigonometric functions (x is in radians):

- **Sin**: sine
- **Cos**: cosine
- **Tan**: tangent
- **Csc**: cosecant
- **Sec**: secant
- **Cot**: cotangent

Inverse trigonometric functions:

- **Asin**: arcsine

- [Acos](#): arccosine
- [Atan](#): arctangent
- [Acsc](#): arccosecant
- [Asec](#): arcsecant
- [Acot](#): arccotangent

Hyperbolic functions:

- [Sinh](#): hyperbolic sine
- [Cosh](#): hyperbolic cosine
- [Tanh](#): hyperbolic tangent
- [Csch](#): hyperbolic cosecant
- [Sech](#): hyperbolic secant
- [Coth](#): hyperbolic cotangent

Inverse hyperbolic functions:

- [Asinh](#): inverse hyperbolic sine
- [Acosh](#): inverse hyperbolic cosine
- [Atanh](#): inverse hyperbolic tangent
- [Acsch](#): inverse hyperbolic cosecant
- [Asech](#): inverse hyperbolic secant
- [Acoth](#): inverse hyperbolic cotangent

Comparison 0/1 functions of two arguments (OR series):

- [OR1](#): if $x < 0$ OR $y < 0$, then 1, else 0
- [OR2](#): if $x \geq 0$ OR $y \geq 0$, then 1, else 0
- [OR3](#): if $x \leq 0$ OR $y \leq 0$, then 1, else 0
- [OR4](#): if $x < 1$ OR $y < 1$, then 1, else 0
- [OR5](#): if $x \geq 1$ OR $y \geq 1$, then 1, else 0
- [OR6](#): if $x \leq 1$ OR $y \leq 1$, then 1, else 0

Comparison 0/1 functions of two arguments (AND series):

- [AND1](#): if $x < 0$ AND $y < 0$, then 1, else 0
- [AND2](#): if $x \geq 0$ AND $y \geq 0$, then 1, else 0
- [AND3](#): if $x \leq 0$ AND $y \leq 0$, then 1, else 0
- [AND4](#): if $x < 1$ AND $y < 1$, then 1, else 0
- [AND5](#): if $x \geq 1$ AND $y \geq 1$, then 1, else 0
- [AND6](#): if $x \leq 1$ AND $y \leq 1$, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series A):

- [LT2A](#): if $x < y$, then x , else y
- [GT2A](#): if $x > y$, then x , else y
- [LOE2A](#): if $x \leq y$, then x , else y
- [GOE2A](#): if $x \geq y$, then x , else y
- [ET2A](#): if $x = y$, then x , else y
- [NET2A](#): if $x \neq y$, then x , else y

Comparison 0/1 IF THEN ELSE functions of two arguments (series B):

- [LT2B](#): if $x < y$, then 1, else 0
- [GT2B](#): if $x > y$, then 1, else 0
- [LOE2B](#): if $x \leq y$, then 1, else 0
- [GOE2B](#): if $x \geq y$, then 1, else 0

- **ET2B**: if $x = y$, then 1, else 0
- **NET2B**: if $x \neq y$, then 1, else 0

Comparison IF THEN ELSE functions of two arguments (series C):

- **LT2C**: if $x < y$, then $(x+y)$, else $(x-y)$
- **GT2C**: if $x > y$, then $(x+y)$, else $(x-y)$
- **LOE2C**: if $x \leq y$, then $(x+y)$, else $(x-y)$
- **GOE2C**: if $x \geq y$, then $(x+y)$, else $(x-y)$
- **ET2C**: if $x = y$, then $(x+y)$, else $(x-y)$
- **NET2C**: if $x \neq y$, then $(x+y)$, else $(x-y)$

Comparison IF THEN ELSE functions of two arguments (series D):

- **LT2D**: if $x < y$, then $(x*y)$, else (x/y)
- **GT2D**: if $x > y$, then $(x*y)$, else (x/y)
- **LOE2D**: if $x \leq y$, then $(x*y)$, else (x/y)
- **GOE2D**: if $x \geq y$, then $(x*y)$, else (x/y)
- **ET2D**: if $x = y$, then $(x*y)$, else (x/y)
- **NET2D**: if $x \neq y$, then $(x*y)$, else (x/y)

Comparison IF THEN ELSE functions of two arguments (series E):

- **LT2E**: if $x < y$, then $(x+y)$, else $(x*y)$
- **GT2E**: if $x > y$, then $(x+y)$, else $(x*y)$
- **LOE2E**: if $x \leq y$, then $(x+y)$, else $(x*y)$
- **GOE2E**: if $x \geq y$, then $(x+y)$, else $(x*y)$
- **ET2E**: if $x = y$, then $(x+y)$, else $(x*y)$
- **NET2E**: if $x \neq y$, then $(x+y)$, else $(x*y)$

Comparison IF THEN ELSE functions of two arguments (series F):

- **LT2F**: if $x < y$, then $(x+y)$, else $\sin(x*y)$
- **GT2F**: if $x > y$, then $(x+y)$, else $\sin(x*y)$
- **LOE2F**: if $x \leq y$, then $(x+y)$, else $\sin(x*y)$
- **GOE2F**: if $x \geq y$, then $(x+y)$, else $\sin(x*y)$
- **ET2F**: if $x = y$, then $(x+y)$, else $\sin(x*y)$
- **NET2F**: if $x \neq y$, then $(x+y)$, else $\sin(x*y)$

Comparison IF THEN ELSE functions of two arguments (series G):

- **LT2G**: if $x < y$, then $(x+y)$, else $\text{atan}(x*y)$
- **GT2G**: if $x > y$, then $(x+y)$, else $\text{atan}(x*y)$
- **LOE2G**: if $x \leq y$, then $(x+y)$, else $\text{atan}(x*y)$
- **GOE2G**: if $x \geq y$, then $(x+y)$, else $\text{atan}(x*y)$
- **ET2G**: if $x = y$, then $(x+y)$, else $\text{atan}(x*y)$
- **NET2G**: if $x \neq y$, then $(x+y)$, else $\text{atan}(x*y)$

Comparison IF THEN ELSE functions of three arguments (series A):

- **LT3A**: if $x < 0$, then y , else z
- **GT3A**: if $x > 0$, then y , else z
- **LOE3A**: if $x \leq 0$, then y , else z
- **GOE3A**: if $x \geq 0$, then y , else z
- **ET3A**: if $x = 0$, then y , else z
- **NET3A**: if $x \neq 0$, then y , else z

Comparison IF THEN ELSE functions of three arguments (series B):

- **LT3B:** if $(x+y) < z$, then $(x+y)$, else z
- **GT3B:** if $(x+y) > z$, then $(x+y)$, else z
- **LOE3B:** if $(x+y) \leq z$, then $(x+y)$, else z
- **GOE3B:** if $(x+y) \geq z$, then $(x+y)$, else z
- **ET3B:** if $(x+y) = z$, then $(x+y)$, else z
- **NET3B:** if $(x+y) \neq z$, then $(x+y)$, else z

Comparison IF THEN ELSE functions of three arguments (series C):

- **LT3C:** if $(x+y) < z$, then $(x+y)$, else $(x+z)$
- **GT3C:** if $(x+y) > z$, then $(x+y)$, else $(x+z)$
- **LOE3C:** if $(x+y) \leq z$, then $(x+y)$, else $(x+z)$
- **GOE3C:** if $(x+y) \geq z$, then $(x+y)$, else $(x+z)$
- **ET3C:** if $(x+y) = z$, then $(x+y)$, else $(x+z)$
- **NET3C:** if $(x+y) \neq z$, then $(x+y)$, else $(x+z)$

Comparison IF THEN ELSE functions of three arguments (series D):

- **LT3D:** if $(x+y) < z$, then $(x+y)$, else $(x-z)$
- **GT3D:** if $(x+y) > z$, then $(x+y)$, else $(x-z)$
- **LOE3D:** if $(x+y) \leq z$, then $(x+y)$, else $(x-z)$
- **GOE3D:** if $(x+y) \geq z$, then $(x+y)$, else $(x-z)$
- **ET3D:** if $(x+y) = z$, then $(x+y)$, else $(x-z)$
- **NET3D:** if $(x+y) \neq z$, then $(x+y)$, else $(x-z)$

Comparison IF THEN ELSE functions of three arguments (series E):

- **LT3E:** if $(x+y) < z$, then $(x+y)$, else $(x*z)$
- **GT3E:** if $(x+y) > z$, then $(x+y)$, else $(x*z)$
- **LOE3E:** if $(x+y) \leq z$, then $(x+y)$, else $(x*z)$
- **GOE3E:** if $(x+y) \geq z$, then $(x+y)$, else $(x*z)$
- **ET3E:** if $(x+y) = z$, then $(x+y)$, else $(x*z)$
- **NET3E:** if $(x+y) \neq z$, then $(x+y)$, else $(x*z)$

Comparison IF THEN ELSE functions of three arguments (series F):

- **LT3F:** if $(x+y) < z$, then $(x+y)$, else (x/z)
- **GT3F:** if $(x+y) > z$, then $(x+y)$, else (x/z)
- **LOE3F:** if $(x+y) \leq z$, then $(x+y)$, else (x/z)
- **GOE3F:** if $(x+y) \geq z$, then $(x+y)$, else (x/z)
- **ET3F:** if $(x+y) = z$, then $(x+y)$, else (x/z)
- **NET3F:** if $(x+y) \neq z$, then $(x+y)$, else (x/z)

Comparison IF THEN ELSE functions of three arguments (series G):

- **LT3G:** if $(x+y) < z$, then $(x*y)$, else $(x+z)$
- **GT3G:** if $(x+y) > z$, then $(x*y)$, else $(x+z)$
- **LOE3G:** if $(x+y) \leq z$, then $(x*y)$, else $(x+z)$
- **GOE3G:** if $(x+y) \geq z$, then $(x*y)$, else $(x+z)$
- **ET3G:** if $(x+y) = z$, then $(x*y)$, else $(x+z)$
- **NET3G:** if $(x+y) \neq z$, then $(x*y)$, else $(x+z)$

Comparison IF THEN ELSE functions of three arguments (series H):

- **LT3H:** if $(x+y) < z$, then $(x*y)$, else $(x-z)$

- **GT3H**: if $(x+y) > z$, then $(x*y)$, else $(x-z)$
- **LOE3H**: if $(x+y) \leq z$, then $(x*y)$, else $(x-z)$
- **GOE3H**: if $(x+y) \geq z$, then $(x*y)$, else $(x-z)$
- **ET3H**: if $(x+y) = z$, then $(x*y)$, else $(x-z)$
- **NET3H**: if $(x+y) \neq z$, then $(x*y)$, else $(x-z)$

Comparison IF THEN ELSE functions of three arguments (series I):

- **LT3I**: if $(x+y) < z$, then $(x*y)$, else $(x*z)$
- **GT3I**: if $(x+y) > z$, then $(x*y)$, else $(x*z)$
- **LOE3I**: if $(x+y) \leq z$, then $(x*y)$, else $(x*z)$
- **GOE3I**: if $(x+y) \geq z$, then $(x*y)$, else $(x*z)$
- **ET3I**: if $(x+y) = z$, then $(x*y)$, else $(x*z)$
- **NET3I**: if $(x+y) \neq z$, then $(x*y)$, else $(x*z)$

Comparison IF THEN ELSE functions of three arguments (series J):

- **LT3J**: if $(x+y) < z$, then $(x*y)$, else (x/z)
- **GT3J**: if $(x+y) > z$, then $(x*y)$, else (x/z)
- **LOE3J**: if $(x+y) \leq z$, then $(x*y)$, else (x/z)
- **GOE3J**: if $(x+y) \geq z$, then $(x*y)$, else (x/z)
- **ET3J**: if $(x+y) = z$, then $(x*y)$, else (x/z)
- **NET3J**: if $(x+y) \neq z$, then $(x*y)$, else (x/z)

Comparison IF THEN ELSE functions of three arguments (series K):

- **LT3K**: if $(x+y) < z$, then $(x+y+z)$, else $\sin(x*y*z)$
- **GT3K**: if $(x+y) > z$, then $(x+y+z)$, else $\sin(x*y*z)$
- **LOE3K**: if $(x+y) \leq z$, then $(x+y+z)$, else $\sin(x*y*z)$
- **GOE3K**: if $(x+y) \geq z$, then $(x+y+z)$, else $\sin(x*y*z)$
- **ET3K**: if $(x+y) = z$, then $(x+y+z)$, else $\sin(x*y*z)$
- **NET3K**: if $(x+y) \neq z$, then $(x+y+z)$, else $\sin(x*y*z)$

Comparison IF THEN ELSE functions of three arguments (series L):

- **LT3L**: if $(x+y) < z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- **GT3L**: if $(x+y) > z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- **LOE3L**: if $(x+y) \leq z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- **GOE3L**: if $(x+y) \geq z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- **ET3L**: if $(x+y) = z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$
- **NET3L**: if $(x+y) \neq z$, then $(x+y+z)$, else $\text{atan}(x*y*z)$

Comparison IF THEN ELSE functions of four arguments (series A):

- **LT4A**: if $a < b$, then c , else d
- **GT4A**: if $a > b$, then c , else d
- **LOE4A**: if $a \leq b$, then c , else d
- **GOE4A**: if $a \geq b$, then c , else d
- **ET4A**: if $a = b$, then c , else d
- **NET4A**: if $a \neq b$, then c , else d

Comparison IF THEN ELSE functions of four arguments (series B):

- **LT4B**: if $(a+b) < (c+d)$, then c , else d
- **GT4B**: if $(a+b) > (c+d)$, then c , else d
- **LOE4B**: if $(a+b) \leq (c+d)$, then c , else d
- **GOE4B**: if $(a+b) \geq (c+d)$, then c , else d

- **ET4B**: if $(a+b) = (c+d)$, then c , else d
- **NET4B**: if $(a+b) \neq (c+d)$, then c , else d

Comparison IF THEN ELSE functions of four arguments (series C):

- **LT4C**: if $(a+b) < (c+d)$, then $(a+b)$, else $(c+d)$
- **GT4C**: if $(a+b) > (c+d)$, then $(a+b)$, else $(c+d)$
- **LOE4C**: if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c+d)$
- **GOE4C**: if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c+d)$
- **ET4C**: if $(a+b) = (c+d)$, then $(a+b)$, else $(c+d)$
- **NET4C**: if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c+d)$

Comparison IF THEN ELSE functions of four arguments (series D):

- **LT4D**: if $(a+b) < (c+d)$, then $(a+b)$, else $(c-d)$
- **GT4D**: if $(a+b) > (c+d)$, then $(a+b)$, else $(c-d)$
- **LOE4D**: if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c-d)$
- **GOE4D**: if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c-d)$
- **ET4D**: if $(a+b) = (c+d)$, then $(a+b)$, else $(c-d)$
- **NET4D**: if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c-d)$

Comparison IF THEN ELSE functions of four arguments (series E):

- **LT4E**: if $(a+b) < (c+d)$, then $(a+b)$, else $(c*d)$
- **GT4E**: if $(a+b) > (c+d)$, then $(a+b)$, else $(c*d)$
- **LOE4E**: if $(a+b) \leq (c+d)$, then $(a+b)$, else $(c*d)$
- **GOE4E**: if $(a+b) \geq (c+d)$, then $(a+b)$, else $(c*d)$
- **ET4E**: if $(a+b) = (c+d)$, then $(a+b)$, else $(c*d)$
- **NET4E**: if $(a+b) \neq (c+d)$, then $(a+b)$, else $(c*d)$

Comparison IF THEN ELSE functions of four arguments (series F):

- **LT4F**: if $(a+b) < (c+d)$, then $(a+b)$, else (c/d)
- **GT4F**: if $(a+b) > (c+d)$, then $(a+b)$, else (c/d)
- **LOE4F**: if $(a+b) \leq (c+d)$, then $(a+b)$, else (c/d)
- **GOE4F**: if $(a+b) \geq (c+d)$, then $(a+b)$, else (c/d)
- **ET4F**: if $(a+b) = (c+d)$, then $(a+b)$, else (c/d)
- **NET4F**: if $(a+b) \neq (c+d)$, then $(a+b)$, else (c/d)

Comparison IF THEN ELSE functions of four arguments (series G):

- **LT4G**: if $(a+b) < (c+d)$, then $(a*b)$, else $(c+d)$
- **GT4G**: if $(a+b) > (c+d)$, then $(a*b)$, else $(c+d)$
- **LOE4G**: if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c+d)$
- **GOE4G**: if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c+d)$
- **ET4G**: if $(a+b) = (c+d)$, then $(a*b)$, else $(c+d)$
- **NET4G**: if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c+d)$

Comparison IF THEN ELSE functions of four arguments (series H):

- **LT4H**: if $(a+b) < (c+d)$, then $(a*b)$, else $(c-d)$
- **GT4H**: if $(a+b) > (c+d)$, then $(a*b)$, else $(c-d)$
- **LOE4H**: if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c-d)$
- **GOE4H**: if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c-d)$
- **ET4H**: if $(a+b) = (c+d)$, then $(a*b)$, else $(c-d)$
- **NET4H**: if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c-d)$

Comparison IF THEN ELSE functions of four arguments (series I):

- **LT4I**: if $(a+b) < (c+d)$, then $(a*b)$, else $(c*d)$
- **GT4I**: if $(a+b) > (c+d)$, then $(a*b)$, else $(c*d)$
- **LOE4I**: if $(a+b) \leq (c+d)$, then $(a*b)$, else $(c*d)$
- **GOE4I**: if $(a+b) \geq (c+d)$, then $(a*b)$, else $(c*d)$
- **ET4I**: if $(a+b) = (c+d)$, then $(a*b)$, else $(c*d)$
- **NET4I**: if $(a+b) \neq (c+d)$, then $(a*b)$, else $(c*d)$

Comparison IF THEN ELSE functions of four arguments (series J):

- **LT4J**: if $(a+b) < (c+d)$, then $(a*b)$, else (c/d)
- **GT4J**: if $(a+b) > (c+d)$, then $(a*b)$, else (c/d)
- **LOE4J**: if $(a+b) \leq (c+d)$, then $(a*b)$, else (c/d)
- **GOE4J**: if $(a+b) \geq (c+d)$, then $(a*b)$, else (c/d)
- **ET4J**: if $(a+b) = (c+d)$, then $(a*b)$, else (c/d)
- **NET4J**: if $(a+b) \neq (c+d)$, then $(a*b)$, else (c/d)

Comparison IF THEN ELSE functions of four arguments (series K):

- **LT4K**: if $(a+b) < (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- **GT4K**: if $(a+b) > (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- **LOE4K**: if $(a+b) \leq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- **GOE4K**: if $(a+b) \geq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- **ET4K**: if $(a+b) = (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$
- **NET4K**: if $(a+b) \neq (c+d)$, then $\sin(a*b)$, else $\sin(c*d)$

Comparison IF THEN ELSE functions of four arguments (series L):

- **LT4L**: if $(a+b) < (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- **GT4L**: if $(a+b) > (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- **LOE4L**: if $(a+b) \leq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- **GOE4L**: if $(a+b) \geq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- **ET4L**: if $(a+b) = (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$
- **NET4L**: if $(a+b) \neq (c+d)$, then $\text{atan}(a*b)$, else $\text{atan}(c*d)$

Dynamic UDFs are indexed and are represented by:

- **DDF0**
- **DDF1**
- etc.

Static UDFs are also indexed and are represented by:

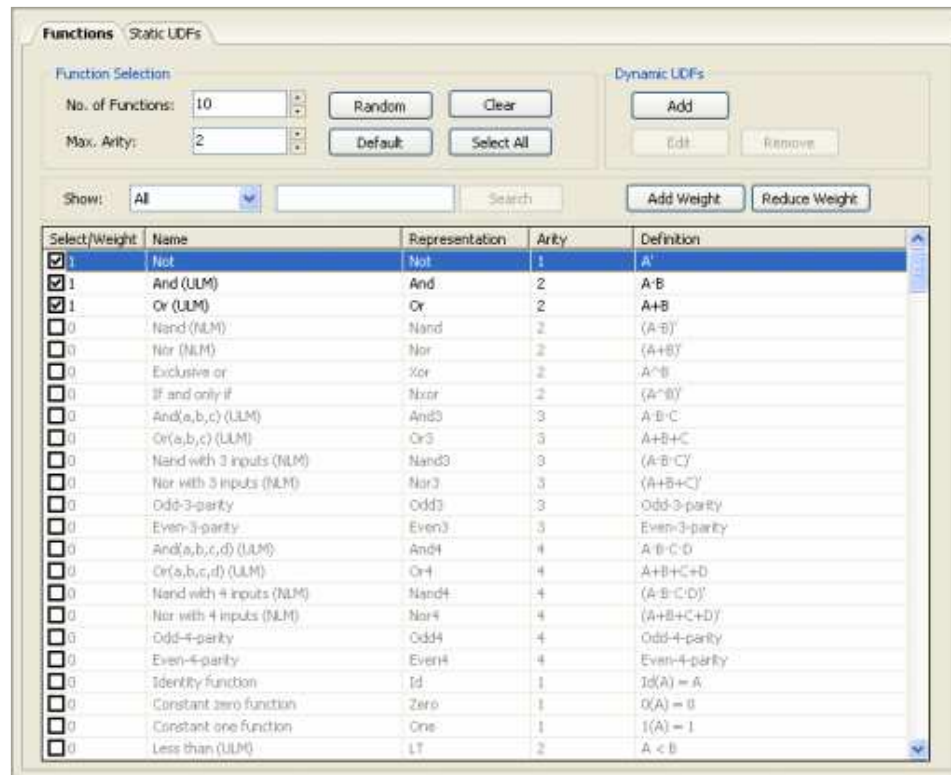
- **UDF0**
- **UDF1**
- etc.



The Architectures of GeneXproTools Learning Algorithms

Built-in Logical Functions

Below are listed all the 258 [built-in logical functions](#) available in GeneXproTools 4.0, starting with their representation in [Karva Notation](#) and their logical definition (for more details about their implementations, see the [built-in grammars](#) of GeneXproTools):



Basic logical functions with 1 and 2 inputs:

- [Not](#): A'
- [And](#): $A \cdot B$
- [Or](#): $A + B$
- [Nand](#): $(A \cdot B)'$
- [Nor](#): $(A + B)'$
- [Xor](#): $A \wedge B$
- [Nxor](#): $(A \wedge B)'$

Basic logical functions with 3 inputs:

- [And3](#): $A \cdot B \cdot C$
- [Or3](#): $A + B + C$
- [Nand3](#): $(A \cdot B \cdot C)'$
- [Nor3](#): $(A + B + C)'$
- [Odd3](#): Odd-3-parity
- [Even3](#): Even-3-parity

Basic logical functions with 4 inputs:

- [And4](#): $A \cdot B \cdot C \cdot D$
- [Or4](#): $A + B + C + D$
- [Nand4](#): $(A \cdot B \cdot C \cdot D)'$
- [Nor4](#): $(A + B + C + D)'$
- [Odd4](#): Odd-4-parity

- [Even4](#): Even-4-parity

Additional logical functions with 1 input:

- [Id](#): $\text{Id}(A) = A$
- [Zero](#): $0(A) = 0$
- [One](#): $1(A) = 1$

Additional logical functions with 2 inputs:

- [LT](#): $A < B$
- [GT](#): $A > B$
- [LOE](#): $A \leq B$
- [GOE](#): $A \geq B$
- [NotA](#): $\text{NOT}(A,B) = \text{NOT } A$
- [NotB](#): $\text{NOT}(A,B) = \text{NOT } B$
- [IdA](#): $\text{IdA}(A,B) = A$
- [IdB](#): $\text{IdB}(A,B) = B$
- [Zero2](#): $0(A,B) = 0$
- [One2](#): $1(A,B) = 1$

Derived logical functions with 3 inputs:

- [LT3](#): $A < B < C$
- [GT3](#): $A > B > C$
- [LOE3](#): $A \leq B \leq C$
- [GOE3](#): $A \geq B \geq C$

Common logical functions with 3 inputs:

- [Mux](#): 3-Multiplexer
- [If](#): If $A = 1$, then B , else C
- [Maj](#): Majority(A,B,C)
- [Min](#): Minority(A,B,C)
- [2Off](#): Exactly two off
- [2On](#): Exactly two on

Universal logical modules with 3 inputs (series A):

- [LM3A1](#): $AC' + BC$
- [LM3A2](#): $AC' + B'C$
- [LM3A3](#): $A'C' + BC$
- [LM3A4](#): $A'C' + B'C$

Universal logical modules with 3 inputs (series B):

- [LM3B1](#): $(A+C') \cdot (B+C)$
- [LM3B2](#): $(A+C') \cdot (B'+C)$
- [LM3B3](#): $(A'+C') \cdot (B+C)$
- [LM3B4](#): $(A'+C') \cdot (B'+C)$

Universal logical modules with 3 inputs (series C):

- [LM3C1](#): $AB' + BC$
- [LM3C2](#): $AB' + BC'$
- [LM3C3](#): $A'B' + BC$

- **LM3C4**: $A'B' + BC'$

Universal logical modules with 3 inputs (series D):

- **LM3D1**: $(A+B') \cdot (B+C)$
- **LM3D2**: $(A+B') \cdot (B+C')$
- **LM3D3**: $(A'+B') \cdot (B+C)$
- **LM3D4**: $(A'+B') \cdot (B+C')$

Universal logical modules with 3 inputs (series E):

- **LM3E1**: $A'C + AB'$
- **LM3E2**: $A'C' + AB$
- **LM3E3**: $A'C' + AB'$

Universal logical modules with 3 inputs (series F):

- **LM3F1**: $(A'+C) \cdot (A+B')$
- **LM3F2**: $(A'+C') \cdot (A+B)$
- **LM3F3**: $(A'+C') \cdot (A+B')$

Universal logical modules with 3 inputs (series G):

- **LM3G1**: $(A \wedge C') \cdot (B \wedge C)$
- **LM3G2**: $(A \wedge C') \cdot (B' \wedge C)$
- **LM3G3**: $(A' \wedge C') \cdot (B \wedge C)$
- **LM3G4**: $(A' \wedge C') \cdot (B' \wedge C)$

Universal logical modules with 3 inputs (series H):

- **LM3H1**: $((A \cdot B)' \cdot C)'$
- **LM3H2**: $(A \cdot (B \cdot C)')'$
- **LM3H3**: $((A+B)' + C)'$
- **LM3H4**: $(A + (B+C)')'$

Comparison IF THEN ELSE functions with 3 inputs (series A):

- **LT3A**: If $A < B$, then $(A \cdot C)$, else $(B \cdot C)'$
- **GT3A**: If $A > B$, then $(A \cdot C)$, else $(B \cdot C)'$
- **LOE3A**: If $A \leq B$, then $(A \cdot C)$, else $(B \cdot C)'$
- **GOE3A**: If $A \geq B$, then $(A \cdot C)$, else $(B \cdot C)'$
- **ET3A**: If $A = B$, then $(A \cdot C)$, else $(B \cdot C)'$
- **NET3A**: If $A \neq B$, then $(A \cdot C)$, else $(B \cdot C)'$

Comparison IF THEN ELSE functions with 3 inputs (series B):

- **LT3B**: If $A < B$, then $(A \cdot C)'$, else $(A \cdot C)$
- **GT3B**: If $A > B$, then $(A \cdot C)'$, else $(A \cdot C)$
- **LOE3B**: If $A \leq B$, then $(A \cdot C)'$, else $(A \cdot C)$
- **GOE3B**: If $A \geq B$, then $(A \cdot C)'$, else $(A \cdot C)$
- **ET3B**: If $A = B$, then $(A \cdot C)'$, else $(A \cdot C)$
- **NET3B**: If $A \neq B$, then $(A \cdot C)'$, else $(A \cdot C)$

Comparison IF THEN ELSE functions with 3 inputs (series C):

- **LT3C**: If $A < B$, then $(A+C)'$, else $(B+C)$

- [GT3C](#): If $A > B$, then $(A+C)'$, else $(B+C)$
- [LOE3C](#): If $A \leq B$, then $(A+C)'$, else $(B+C)$
- [GOE3C](#): If $A \geq B$, then $(A+C)'$, else $(B+C)$
- [ET3C](#): If $A = B$, then $(A+C)'$, else $(B+C)$
- [NET3C](#): If $A \neq B$, then $(A+C)'$, else $(B+C)$

Additional universal logical modules with 3 inputs:

- [T004](#): 00000100
- [T008](#): 00001000
- [T009](#): 00001001
- [T032](#): 00100000
- [T033](#): 00100001
- [T041](#): 00101001
- [T055](#): 00110111
- [T057](#): 00111001
- [T064](#): 01000000
- [T065](#): 01000001
- [T069](#): 01000101
- [T073](#): 01001001
- [T081](#): 01010001
- [T089](#): 01011001
- [T093](#): 01011101
- [T096](#): 01100000
- [T101](#): 01100101
- [T109](#): 01101101
- [T111](#): 01101111
- [T121](#): 01111001
- [T123](#): 01111011
- [T125](#): 01111101
- [T154](#): 10011010
- [T223](#): 11011111
- [T239](#): 11101111
- [T249](#): 11111001
- [T251](#): 11111011
- [T253](#): 11111101

Derived logical functions with 4 inputs:

- [LT4](#): $A < B < C < D$
- [GT4](#): $A > B > C > D$
- [LOE4](#): $A \leq B \leq C \leq D$
- [GOE4](#): $A \geq B \geq C \geq D$

Common logical functions with 4 inputs:

- [Tie](#): Tie
- [Ntie](#): Not tie
- [3Off](#): Exactly three off
- [3On](#): Exactly three on

Universal logical modules with 4 inputs (series A):

- [LM4A1](#): $AD' + BD + CD$
- [LM4A2](#): $AD' + B'D + CD$
- [LM4A3](#): $AD' + BD + C'D$
- [LM4A4](#): $AD' + B'D + C'D$

- [LM4A5](#): $A'D' + BD + CD$
- [LM4A6](#): $A'D' + B'D + CD$
- [LM4A7](#): $A'D' + BD + C'D$
- [LM4A8](#): $A'D' + B'D + C'D$

Universal logical modules with 4 inputs (series B):

- [LM4B1](#): $(A+D') \cdot (B+D) \cdot (C+D)$
- [LM4B2](#): $(A+D') \cdot (B'+D) \cdot (C+D)$
- [LM4B3](#): $(A+D') \cdot (B+D) \cdot (C'+D)$
- [LM4B4](#): $(A+D') \cdot (B'+D) \cdot (C'+D)$
- [LM4B5](#): $(A'+D') \cdot (B+D) \cdot (C+D)$
- [LM4B6](#): $(A'+D') \cdot (B'+D) \cdot (C+D)$
- [LM4B7](#): $(A'+D') \cdot (B+D) \cdot (C'+D)$
- [LM4B8](#): $(A'+D') \cdot (B'+D) \cdot (C'+D)$

Universal logical modules with 4 inputs (series C):

- [LM4C1](#): $AB' + BC + BD$
- [LM4C2](#): $AB' + B'C + BD$
- [LM4C3](#): $AB' + BC + B'D$
- [LM4C4](#): $AB' + B'C + B'D$
- [LM4C5](#): $A'B' + BC + BD$
- [LM4C6](#): $A'B' + B'C + BD$
- [LM4C7](#): $A'B' + BC + B'D$
- [LM4C8](#): $A'B' + B'C + B'D$

Universal logical modules with 4 inputs (series D):

- [LM4D1](#): $(A+B') \cdot (B+C) \cdot (B+D)$
- [LM4D2](#): $(A+B') \cdot (B'+C) \cdot (B+D)$
- [LM4D3](#): $(A+B') \cdot (B+C) \cdot (B'+D)$
- [LM4D4](#): $(A+B') \cdot (B'+C) \cdot (B'+D)$
- [LM4D5](#): $(A'+B') \cdot (B+C) \cdot (B+D)$
- [LM4D6](#): $(A'+B') \cdot (B'+C) \cdot (B+D)$
- [LM4D7](#): $(A'+B') \cdot (B+C) \cdot (B'+D)$
- [LM4D8](#): $(A'+B') \cdot (B'+C) \cdot (B'+D)$

Universal logical modules with 4 inputs (series E):

- [LM4E1](#): $AC' + BC + CD$
- [LM4E2](#): $AC' + B'C + CD$
- [LM4E3](#): $AC' + BC + C'D$
- [LM4E4](#): $AC' + B'C + C'D$
- [LM4E5](#): $A'C' + BC + CD$
- [LM4E6](#): $A'C' + B'C + CD$
- [LM4E7](#): $A'C' + BC + C'D$
- [LM4E8](#): $A'C' + B'C + C'D$

Universal logical modules with 4 inputs (series F):

- [LM4F1](#): $(A+C') \cdot (B+C) \cdot (C+D)$
- [LM4F2](#): $(A+C') \cdot (B'+C) \cdot (C+D)$
- [LM4F3](#): $(A+C') \cdot (B+C) \cdot (C'+D)$
- [LM4F4](#): $(A+C') \cdot (B'+C) \cdot (C'+D)$
- [LM4F5](#): $(A'+C') \cdot (B+C) \cdot (C+D)$
- [LM4F6](#): $(A'+C') \cdot (B'+C) \cdot (C+D)$

- **LM4F7:** $(A'+C') \cdot (B+C) \cdot (C'+D)$
- **LM4F8:** $(A'+C') \cdot (B'+C) \cdot (C'+D)$

Universal logical modules with 4 inputs (series G):

- **LM4G1:** $A'D+AB+AC$
- **LM4G2:** $A'D+AB'+AC$
- **LM4G3:** $A'D+AB+ AC'$
- **LM4G4:** $A'D+AB'+ AC'$
- **LM4G5:** $A'D'+AB+AC$
- **LM4G6:** $A'D'+AB'+AC$
- **LM4G7:** $A'D'+AB+ AC'$
- **LM4G8:** $A'D'+AB'+ AC'$

Universal logical modules with 4 inputs (series H):

- **LM4H1:** $(A'+D) \cdot (A+B) \cdot (A+C)$
- **LM4H2:** $(A'+D) \cdot (A+B') \cdot (A+C)$
- **LM4H3:** $(A'+D) \cdot (A+B) \cdot (A+C')$
- **LM4H4:** $(A'+D) \cdot (A+B') \cdot (A+C')$
- **LM4H5:** $(A'+D') \cdot (A+B) \cdot (A+C)$
- **LM4H6:** $(A'+D') \cdot (A+B') \cdot (A+C)$
- **LM4H7:** $(A'+D') \cdot (A+B) \cdot (A+C')$
- **LM4H8:** $(A'+D') \cdot (A+B') \cdot (A+C')$

Universal logical modules with 4 inputs (series I):

- **LM4I1:** $((A \cdot B)' \cdot C)' \cdot D'$
- **LM4I2:** $(A \cdot (B \cdot (C \cdot D)'))' \cdot D'$
- **LM4I3:** $((A \cdot (B \cdot C'))' \cdot D)'$
- **LM4I4:** $(A \cdot ((B \cdot C)' \cdot D))'$
- **LM4I5:** $((A+B) \cdot C)' \cdot D'$
- **LM4I6:** $(A \cdot (B \cdot (C \cdot D)'))' \cdot D'$
- **LM4I7:** $((A \cdot (B \cdot C))' \cdot D)'$
- **LM4I8:** $(A \cdot ((B \cdot C)' \cdot D))'$

Comparison IF THEN ELSE functions with 4 inputs (series A):

- **LT4A:** If $A < B$, then C , else D
- **GT4A:** If $A > B$, then C , else D
- **LOE4A:** If $A \leq B$, then C , else D
- **GOE4A:** If $A \geq B$, then C , else D
- **ET4A:** If $A = B$, then C , else D
- **NET4A:** If $A \neq B$, then C , else D

Comparison IF THEN ELSE functions with 4 inputs (series B):

- **LT4B:** If $A < B$, then $(C \cdot D)$, else D'
- **GT4B:** If $A > B$, then $(C \cdot D)$, else D'
- **LOE4B:** If $A \leq B$, then $(C \cdot D)$, else D'
- **GOE4B:** If $A \geq B$, then $(C \cdot D)$, else D'
- **ET4B:** If $A = B$, then $(C \cdot D)$, else D'
- **NET4B:** If $A \neq B$, then $(C \cdot D)$, else D'

Comparison IF THEN ELSE functions with 4 inputs (series C):

- **LT4C:** If $A < B$, then $(C+D)$, else D'

- [GT4C](#): If $A > B$, then $(C+D)$, else D'
- [LOE4C](#): If $A \leq B$, then $(C+D)$, else D'
- [GOE4C](#): If $A \geq B$, then $(C+D)$, else D'
- [ET4C](#): If $A = B$, then $(C+D)$, else D'
- [NET4C](#): If $A \neq B$, then $(C+D)$, else D'

Comparison IF THEN ELSE functions with 4 inputs (series D):

- [LT4D](#): If $A < B$, then $(A \cdot D)$, else $(C \cdot D)'$
- [GT4D](#): If $A > B$, then $(A \cdot D)$, else $(C \cdot D)'$
- [LOE4D](#): If $A \leq B$, then $(A \cdot D)$, else $(C \cdot D)'$
- [GOE4D](#): If $A \geq B$, then $(A \cdot D)$, else $(C \cdot D)'$
- [ET4D](#): If $A = B$, then $(A \cdot D)$, else $(C \cdot D)'$
- [NET4D](#): If $A \neq B$, then $(A \cdot D)$, else $(C \cdot D)'$

Comparison IF THEN ELSE functions with 4 inputs (series E):

- [LT4E](#): If $A < B$, then $(A \cdot D)'$, else $(A \cdot C)$
- [GT4E](#): If $A > B$, then $(A \cdot D)'$, else $(A \cdot C)$
- [LOE4E](#): If $A \leq B$, then $(A \cdot D)'$, else $(A \cdot C)$
- [GOE4E](#): If $A \geq B$, then $(A \cdot D)'$, else $(A \cdot C)$
- [ET4E](#): If $A = B$, then $(A \cdot D)'$, else $(A \cdot C)$
- [NET4E](#): If $A \neq B$, then $(A \cdot D)'$, else $(A \cdot C)$

Additional universal logical modules with 4 inputs:

- [Q0002](#): 00000000000000010
- [Q001C](#): 00000000000011100
- [Q0048](#): 0000000001001000
- [Q0800](#): 0000100000000000
- [Q3378](#): 0011001101111000
- [Q3475](#): 0011010001110101
- [Q3CB0](#): 0011110010110000
- [Q3DEF](#): 0011110111101111
- [Q3DFF](#): 0011110111111111
- [Q4200](#): 0100001000000000
- [Q4C11](#): 0100110000010001
- [Q5100](#): 0101000100000000
- [Q5EEF](#): 0101111011101111
- [Q5EFF](#): 0101111011111111
- [Q6A6D](#): 0110101001101101
- [Q6F75](#): 0110111101110101
- [Q74C4](#): 0111010011000100
- [Q7DA3](#): 0111110110100011
- [Q8304](#): 1000001100000100
- [Q8430](#): 1000010000110000
- [Q8543](#): 1000010101000011
- [Q9D80](#): 1001110110000000
- [QA092](#): 1010000010010010
- [QB36A](#): 1011001101101010
- [QCBCF](#): 1100101111001111
- [QEEB1](#): 1110111010110001
- [QEFFF](#): 1110111111111111
- [QFF7B](#): 1111111101111011
- [QFFF6](#): 1111111111110110
- [QFFFB](#): 1111111111111011

Dynamic UDFs are indexed and are represented by:

- [DDF0](#)
- [DDF1](#)
- etc.

Static UDFs are also indexed and are represented by:

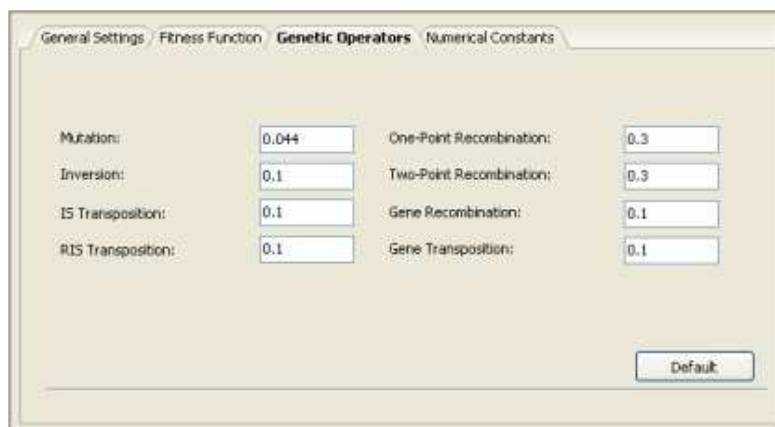
- [UDF0](#)
- [UDF1](#)
- etc.



Genetic Operators

This chapter describes the mechanisms of all the genetic operators of GeneXproTools 4.0 and teaches you how to use them to allow an efficient evolution. In this chapter, you'll learn about:

- [Mutation and its power.](#)
- [Inversion and its power.](#)
- [Transposition and its power.](#)
- [Recombination and its power.](#)
- [Special genetic operators for fine-tuning numerical constants.](#)



Genetic Operators

Mutation

For the sake of simplicity, we are going to illustrate the mechanisms of mutation using the compact,

linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function or terminal) is represented by a single character so that each element can be easily identified by its position in the chromosome.

Mutation is by far the most efficient operator in [Gene Expression Programming](#) (GEP). With mutation, populations of individuals adapt very efficiently, allowing the evolution of good solutions to virtually all problems. The default value for the **mutation rate** in GeneXproTools 4.0 is 0.044, but a good rule of thumb consists of using a mutation rate equivalent to two one-point mutations per chromosome. Considering the length of GEP genomes, this mutation rate is much higher than the mutation rates found in nature, but thanks to the cloning of the best individual of each generation, we can have GEP populations undergoing high mutation rates and, nevertheless, evolving very efficiently.

In Gene Expression Programming, mutations can occur anywhere in the chromosome. But obviously the structural organization of chromosomes must remain intact, that is, in the [heads](#) any symbol can change into another (function or terminal), whereas in the [tails](#) terminals can only change into other terminals. This way, the [structural organization of chromosomes](#) is preserved, and all the new individuals produced by mutation are structurally correct programs.

Consider the following chromosome composed of three genes, each with a head length of 7:

```
012345678901234012345678901234012345678901234
-cQd/-/aadddcca+*b**cQbcaacbda+c*/cdaabdbcbba
```

Suppose a mutation changed the "-" at position 5 in gene 1 to "b"; the "d" at position 10 in gene 1 to "c"; the "*" at position 1 in gene 2 to "Q"; and the "c" at position 1 in gene 3 to "/". In this case the following chromosome is obtained:

```
012345678901234012345678901234012345678901234
-cQd/b/aadcdcca+Qb**cQbcaacbda+/*/cdaabdbcbba
```

Note that if a function is mutated into a terminal or vice versa, or a function of one argument is mutated into a function of two arguments or vice versa, the [expression tree](#) is usually modified drastically, growing or shrinking between the boundaries determined by the size of the gene. In summary, in [Gene Expression Programming](#) there are no constraints both in the kind of mutation and the number of mutations in a chromosome as, in all cases, the newly created individuals are syntactically correct programs.



Genetic Operators

Inversion

For the sake of simplicity, we are going to illustrate the mechanisms of inversion using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function or terminal) is represented by a single character so that each element can be easily identified by its position in the chromosome.

In [Gene Expression Programming](#), the **inversion** operator is restricted to the [heads of genes](#). Here any sequence might be randomly selected and inverted. The default value for the **inversion rate** in GeneXproTools 4.0 is 0.1, as this operator is usually used together with other, more powerful operators such as [mutation](#). But if you want to introduce genetic modification by using this operator alone, you will obtain better results with inversion rates of 1.0.

In Gene Expression Programming, the inversion operator randomly chooses the chromosome, the gene to be modified, and the start and termination points of the sequence to be inverted.

Consider the following chromosome composed of three genes, each with a head length of 7:

```
012345678901234012345678901234012345678901234
- /Qbaadcbadbdc-cbd+Qdbaabdacd+QbddQ-ddabdbbb
```

Suppose that the sequence "/Qb" in gene 1 (positions 1-3) was picked up to be inverted. Then the following chromosome is formed:

```
012345678901234012345678901234012345678901234
-bQ/aadcbadbdc-cbd+Qdbaabdacd+QbddQ-ddabdbbb
```

Note that, since the inversion operator was restricted to the [heads of genes](#), there is no danger of a function ending up in the [tails](#) and, consequently, all the new individuals created by inversion are syntactically correct programs.



Genetic Operators

Transposition

The transposable elements (also called **transposons**) of [Gene Expression Programming](#) (GEP) are fragments of the genome that can be activated and then jump to another place in the chromosome. In GEP there are three kinds of transposable elements:

- Short fragments with a function or terminal in the first position that transpose to the head of genes except the root (insertion sequence elements or **IS elements**).
- Short fragments with a function in the first position that transpose to the start position of genes (root IS elements or **RIS elements**).
- Entire genes that transpose to the beginning of chromosomes.

Consequently, GeneXproTools 4.0 has three different kinds of transposition:

- [IS transposition](#).
- [RIS transposition](#).
- [Gene transposition](#).



Genetic Operators

IS Transposition

For the sake of simplicity, we are going to illustrate the mechanisms of IS transposition using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function or terminal) is represented by a single character so that each element can be easily identified by its position in the

chromosome.

Insertion sequence elements or **IS elements** are short fragments of the genome with a function or terminal in the first position that transpose to the [head of genes](#) except the root. The default value for the **IS transposition rate** in GeneXproTools 4.0 is 0.1, as this operator is usually used together with other, more powerful operators such as [mutation](#). But if you want to introduce genetic modification by using this operator alone, you will obtain better results with IS transposition rates of 1.0.

Any sequence in the genome might become an IS element and, therefore, these elements are randomly selected throughout the chromosome. A copy of the IS element is made and inserted at any position in the [head of a gene](#), except the first position.

The **IS transposition** operator randomly chooses the chromosome, the start and termination points of the IS element, and the target site.

Consider the following chromosome composed of three genes, each with a head length of 7:

```
0123456789012340123456789012345678901234
/cQQ*b+bccabdaaQb/Qdd-dcbdbadd-*dQc+bccaabaad
```

Suppose that the sequence "Qc+" in gene 3 (positions 3-5) was picked up as an IS element to be then inserted between positions 1-2 in gene 2, obtaining:

```
0123456789012340123456789012345678901234
/cQQ*b+bccabdaaQbQc+/Qdcdbdadd-*dQc+bccaabaad
```

Note that, in this case, a perfect copy of the transposon appears at the site of insertion. Note also that a sequence with as many symbols as the IS element is deleted at the end of the head of gene 2 (in this case, the sequence "dd-" was deleted). Thus, despite this insertion, the [structural organization of chromosomes](#) is maintained and, therefore, all the new individuals created by IS transposition are syntactically correct programs.



Genetic Operators

RIS Transposition

For the sake of simplicity, we are going to illustrate the mechanisms of RIS transposition using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function or terminal) is represented by a single character so that each element can be easily identified by its position in the chromosome.

Root insertion sequence elements or **RIS elements** are short fragments with a function in the first position that transpose to the start position of [genes](#). The default value for the **RIS transposition rate** in GeneXproTools 4.0 is 0.1, as this operator is usually used together with other, more powerful operators such as [mutation](#). But if you want to introduce genetic modification by using this operator alone, you will obtain better results with RIS transposition rates of 1.0.

All RIS elements start with a function, and therefore must be chosen among the sequences of the [heads](#). For that, a point is randomly chosen in the head and the gene is scanned downstream until a function is found. This function becomes the start position of the RIS element. If no functions are found, the operator does nothing.

The **RIS transposition** operator randomly chooses the chromosome, the gene to be modified, and the start and termination points of the RIS element.

Consider the following chromosome composed of three genes, each with a head length of 7:

```
012345678901234012345678901234012345678901234
*a/cadQcabcdaca*aQb/cccbbaaca/*c-Qd+abccdabb
```

Suppose that the sequence "-Q" in gene 3 was randomly chosen to become an RIS element. The transposon copies itself and then transposes to the root of the gene, giving:

```
012345678901234012345678901234012345678901234
*a/cadQcabcdaca*aQb/cccbbaaca-Q/*c-Qabccdabb
```

Note that during transposition, the whole **head** shifts to accommodate the RIS element, losing, at the same time, the last symbols of the head (as many as there exist in the transposon). In this case, the sequence "d+" was deleted and an extra copy of the transposon was created. As with **IS transposition**, the **tail** of the gene subjected to RIS transposition and all nearby genes remain unchanged. Note, again, that all the programs newly created by this operator are syntactically correct as it also preserves the **structural organization of the chromosome**.



Genetic Operators

Gene Transposition

For the sake of simplicity, we are going to illustrate the mechanisms of gene transposition using the compact, linear representation of chromosomes used to describe the **structural organization of chromosomes** in the previous chapter. In this representation, each element (function or terminal) is represented by a single character so that each element can be easily identified by its position in the chromosome.

In **gene transposition** an entire gene works as a **transposon** and transposes itself to the beginning of the chromosome. In contrast to the other forms of transposition, in gene transposition, the transposon (the gene) is deleted at the place of origin.

The default value for the **gene transposition rate** in GeneXproTools 4.0 is 0.1, as this operator is usually used together with other, more powerful operators such as **mutation**. But if you want to introduce genetic modification by using this operator alone, you will obtain better results with gene transposition rates of 1.0.

The gene transposition operator randomly chooses the chromosome to be modified and then randomly chooses one of its genes (except the first, obviously) to transpose.

Consider the following chromosome composed of three genes, each with a **head** length of 7:

```
012345678901234012345678901234012345678901234
/cQQ*bQbccadbdc-cbd+Qdbaabdacd+*dbbcbccaabcad
```

Suppose gene 2 was chosen to undergo gene transposition. In this case the following chromosome is obtained:


```
012345678901234012345678901234012345678901234
-cbd+Qdbaabdacd/cQQ*bQbccadbd+*dbbcbccaabcad
```

Apparently, gene transposition is only capable of shuffling genes and, for [sub-expression trees](#) (sub-ETs) linked by commutative functions such as addition or multiplication or AND or OR, this contributes nothing to adaptation in the short run. Note, however, that when the sub-ETs are linked by a non-commutative function such as subtraction or division, the order of the genes matters and, in this case, gene transposition becomes a macromutator. However, gene transposition becomes particularly interesting when it is used in conjunction with [recombination](#), for it allows not only the duplication of genes but also a more generalized shuffling of genes or smaller building blocks.



Genetic Operators

Recombination

GeneXproTools 4.0 has three different kinds of recombination or crossover:

- [One-point recombination](#).
- [Two-point recombination](#).
- [Gene recombination](#).



Genetic Operators

One-point Recombination

For the sake of simplicity, we are going to illustrate the mechanisms of one-point recombination using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function or terminal) is represented by a single character so that each element can be easily identified by its position in the chromosome.

In **one-point recombination** the parent chromosomes are paired and split up at exactly the same point. The material downstream of the recombination point is afterwards exchanged between the two chromosomes.

The default value for the **one-point recombination rate** in GeneXproTools 4.0 is 0.3, as this operator is usually used together with other, more powerful operators such as [mutation](#). But if you want to introduce genetic modification by using this operator alone, you will obtain better results with one-point recombination rates of 1.0.

Consider the following parent chromosomes, each composed of three genes:

```
012345678901234012345678901234012345678901234
+++*QQbbdddbbb*-d*cbbbcdadddbd+-baaaaacdccbba
Q+-Q///bbaacccb-cda+-dbcacdad+d/c**abdbcabdb
```

Suppose bond 4 in gene 2 (between positions 3 and 4) was randomly chosen as the crossover point.

Then, the paired chromosomes are both cut at this bond, and exchange between them the material downstream of the crossover point, forming the offspring below:

```
012345678901234012345678901234012345678901234
+++ -QQbbdddbbb* -d*+ -dbcacacad+d/c**abdbcabdb
Q+-Q///bbaaccb-cdacbbbcdaddbd+-baaaaacdcbbba
```

It is worth emphasizing that the chromosomes of [Gene Expression Programming](#) (GEP) can cross over any point in the genome, continually disrupting old building blocks and continually forming new ones. Furthermore, due to both the [multigenic nature](#) of GEP chromosomes and the existence of [noncoding regions](#) in most genes, entire genes and intact [K-expressions](#) can be swapped between parent chromosomes. Thus, the disruptive tendencies of one-point recombination (splitting of building blocks) coexist side by side with its more conservative tendencies (swapping of genes and K-expressions), making one-point recombination (and of course [two-point recombination](#) too) a very well balanced genetic operator. Furthermore, like all the other [recombinational operators](#), when one-point recombination is used together with [gene transposition](#), it is also capable of duplicating genes.

Notwithstanding, if the goal is to evolve good solutions, one-point or two-point recombination should never be used as the only source of genetic variation as, with time, they tend to homogenize populations. However, together with [mutation](#), [inversion](#) and [transposition](#), these operators are an excellent source of genetic variation and are more than sufficient to evolve good solutions to virtually all problems.



Genetic Operators

Two-point Recombination

For the sake of simplicity, we are going to illustrate the mechanisms of two-point recombination using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function or terminal) is represented by a single character so that each element can be easily identified by its position in the chromosome.

In **two-point recombination** two parent chromosomes are paired and two points are randomly chosen as crossover points. The material between the recombination points is afterwards exchanged between the parent chromosomes, forming two new daughter chromosomes.

The default value for the **two-point recombination rate** in GeneXproTools 4.0 is 0.3, as this operator is usually used together with other, more powerful operators such as [mutation](#). But if you want to introduce genetic modification by using this operator alone, you will obtain better results with two-point recombination rates of 1.0.

Consider the following parent chromosomes, each composed of three genes:

```
012345678901234012345678901234012345678901234
+ca* -+cdacabbca/Qdd -c+cddacdac+/ccbd/bcbadabb
Qcb+b+acddaadad*/*b/adadaddcba/bc-a+Qacdbbbbaa
```

Suppose bond 7 in gene 1 (between positions 6 and 7) and bond 4 in gene 3 (between positions 3 and 4) were chosen as crossover points. Then, the following daughter chromosomes are created:

```
012345678901234012345678901234012345678901234
```

```
+ca* -+cddaadad*/*b/adadaddcba/bc -bd/bcbadabb
Qcb+b+adacabbca/Qdd-c+cddacdac+/cca+Qacdbbbaa
```

It is worth emphasizing that two-point recombination is more disruptive than [one-point recombination](#) in the sense that it recombines the genetic material more thoroughly, constantly destroying old building blocks and creating new ones. But like one-point recombination, two-point recombination has also a conservative side and it is good at swapping entire genes and [K-expressions](#). And, as observed for one-point recombination, two-point recombination can also give rise to duplicated genes if it were used together with [gene transposition](#).

Notwithstanding, if the goal is to evolve good solutions, one-point or two-point recombination should never be used as the only source of genetic variation as, with time, they tend to homogenize populations. However, together with [mutation](#), [inversion](#) and [transposition](#), these operators are an excellent source of genetic variation and are more than sufficient to evolve good solutions to virtually all problems.



Genetic Operators

Gene Recombination

For the sake of simplicity, we are going to illustrate the mechanisms of gene recombination using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function or terminal) is represented by a single character so that each element can be easily identified by its position in the chromosome.

In **gene recombination**, entire genes are exchanged between two parent chromosomes, forming two daughter chromosomes containing genes from both parents. The exchanged genes are randomly chosen and occupy exactly the same position in the parent chromosomes.

The default value for the **gene recombination rate** in GeneXproTools 4.0 is 0.1, as this operator is usually used together with other, more powerful operators such as [mutation](#). But if you want to introduce genetic modification by using this operator alone, you will obtain better results with gene recombination rates of 1.0.

Consider the following parent chromosomes, each composed of three genes:

```
012345678901234012345678901234012345678901234
+Qaa-dcabdaddac-a*b- /aabdbbdba+caQ*bQcdcbdcac
//Q//bacdacabba/b/d+/acddbbaac*-c*- /acdbacddd
```

Suppose gene 2 was chosen to be exchanged. In this case the following offspring is formed:

```
012345678901234012345678901234012345678901234
+Qaa-dcabdaddac/b/d+/acddbbaac+caQ*bQcdcbdcac
//Q//bacdacabba-a*b- /aabdbbdba*-c*- /acdbacddd
```

Note that, with this kind of recombination, similar genes can be exchanged but, most of the times, the exchanged genes are very different and new material is introduced in the population.

It is worth emphasizing that gene recombination is unable to create new genes: the individuals created by this operator are different arrangements of existing genes. Obviously, if gene recombination were

used as the unique source of genetic variation, more complex problems could only be solved using very large initial populations in order to provide for the necessary diversity of genes. However, the evolvability of [Gene Expression Programming](#) is based not only in the shuffling of genes (achieved by gene recombination and [gene transposition](#)), but also in the constant creation of new genetic material which is carried out essentially by [mutation](#), [inversion](#) and [transposition](#) (both [IS](#) and [RIS transposition](#)) and, to a lesser extent, by [recombination](#) (both [one-point](#) and [two-point recombination](#)).

So, if the goal is to evolve good solutions, gene recombination should never be used as the only source of genetic variation as, with time, it tends to homogenize populations. However, as observed for [one-point](#) and [two-point recombination](#), gene recombination can also give rise to duplicated genes if it were used together with [gene transposition](#).



Genetic Operators

Genetic Operators for Random Numerical Constants

Whenever the [GEP-RNC algorithm](#) of GeneXproTools 4.0 is active, an additional set of genetic operators, especially developed for handling random numerical constants, are available for fine-tuning the evolution of constants:

- [Dc mutation](#).
- [Dc inversion](#).
- [Dc IS transposition](#).
- [RNC mutation](#).

General Settings Fitness Function Genetic Operators **Numerical Constants**

☒ Use Random Numerical Constants

Constants per Gene: 2 Lower Bound: -10

Data Type: Floating-Point Upper Bound: 10

RNC Mutation: 0.01

Dc Mutation: 0.044

Dc Inversion: 0.1

Dc IS Transposition: 0.1

Default



Genetic Operators

Dc Mutation

The [GEP-RNC algorithm](#) of GeneXproTools 4.0 uses an additional [gene domain](#) (Dc) for encoding random numerical constants (RNCs). In the Dc, the symbols used to represent the random numerical constants can obviously be replaced by other symbols of the same kind.

For the sake of simplicity, we are going to illustrate the mechanisms of Dc mutation using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function, terminal, or random constant) is represented by a single character so that each element can be easily identified by its position in the chromosome.

Dc mutation is similar to the [mutation](#) that occurs in the [heads](#) and [tails](#) of genes, extending this powerful operator also to Dc. Thus, the default value for the **Dc mutation rate** in GeneXproTools 4.0 is also 0.044.

Consider the following chromosome composed of two genes, each with a Dc length of 8:

```
0123456789012345678901201234567890123456789012
*/-?c++?c?acadc60569331+-?Q*?/cd?ddc?a86358705
```

Suppose a mutation changed the numeral "5" at position 17 in gene 1 to "4"; and the "8" at position 15 in gene 2 to "1". In this case the following chromosome is obtained:

```
0123456789012345678901201234567890123456789012
*/-?c++?c?acadc60469331+-?Q*?/cd?ddc?a16358705
```

Now suppose that the arrays below represent the random numerical constants of the respective genes:

$$C_1 = \{-1.64, -1.834, -0.295, 1.205, -0.807, 0.856, 1.702, -1.026, -0.417, -1.061\}$$

$$C_2 = \{-1.14, 1.177, -1.179, -0.74, 0.393, 1.135, -0.625, 1.643, -0.029, -1.639\}$$

As you can see (you'll need to draw the [expression trees](#), of course), these chromosomes encode different solutions because the random constants expressed in the original chromosome are slightly different from the ones expressed in the daughter chromosome.



Genetic Operators

Dc Inversion

The [GEP-RNC algorithm](#) of GeneXproTools 4.0 uses an additional [gene domain](#) (Dc) for encoding random numerical constants (RNCs). In the Dc, the symbols used to represent the random numerical constants can obviously be replaced by other symbols of the same kind.

For the sake of simplicity, we are going to illustrate the mechanisms of Dc inversion using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function, terminal, or random constant) is represented by a single character so that each element can be easily identified by its position in the chromosome.

Dc inversion is similar to the [inversion](#) that occurs in the [heads of genes](#), extending this operator also to Dc. Thus, the default value for the **Dc inversion rate** in GeneXproTools 4.0 is also 0.1.

The Dc inversion operator randomly chooses the chromosome, the gene to be modified, and the start and termination points of the sequence to be inverted.

Consider the following chromosome composed of two genes, each with a Dc length of 8:

```
0123456789012345678901201234567890123456789012
-Q/*? -?da?bac?b21650337Q*+b?/*?addb??b10715948
```

Suppose that the sequence "1650" in gene 1 (positions 16-19) was picked up to be inverted. Then the following chromosome is formed:

```
0123456789012345678901201234567890123456789012
-Q/*? -?da?bac?b20561337Q*+b?/*?addb??b10715948
```

Now suppose that the arrays below represent the random numerical constants of the respective genes:

$$C_1 = \{-0.78, -0.521, -1.224, 1.891, 0.554, 1.237, -0.444, 0.472, 1.012, 0.679\}$$

$$C_2 = \{-1.553, 1.425, -1.606, -0.487, 1.255, -0.253, -1.91, 1.427, -0.103, -1.625\}$$

As you can see (you'll need to draw the [expression trees](#), of course), these chromosomes encode different solutions because the random constants expressed in the original chromosome are slightly different from the ones expressed in the daughter chromosome.



Genetic Operators

Dc IS Transposition

The [GEP-RNC algorithm](#) of GeneXproTools 4.0 uses an additional [gene domain](#) (Dc) for encoding random numerical constants (RNCs). In the Dc, the symbols used to represent the random numerical constants can obviously be replaced by other symbols of the same kind.

For the sake of simplicity, we are going to illustrate the mechanisms of Dc IS transposition using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function, terminal, or random constant) is represented by a single character so that each element can be easily identified by its position in the chromosome.

Dc IS transposition similar to the [IS transposition](#) that occurs in the [heads of genes](#), extending this operator also to Dc. Thus, the default value for the **Dc IS transposition rate** in GeneXproTools 4.0 is also 0.1.

The Dc IS transposition operator randomly chooses the chromosome, the start and termination points of the IS element, and the target site.

Consider the following chromosome composed of two genes, each with a Dc length of 8:

```
0123456789012345678901201234567890123456789012
/- - -Q+c??a?cddb05829607 - -?/* -ab?dd?cb25391083
```


Suppose that the sequence "108" in gene 2 (positions 19-21) was picked up as an IS element to be then inserted between positions 15 and 16 in gene 2, obtaining:

```
0123456789012345678901201234567890123456789012
/---Q+c??a?cddb05829607--?//*-ab?dd?cb21085391
```

Note that a sequence with as many symbols as the IS element is deleted at the end of the Dc of gene 2 (in this case, the sequence "083" was deleted).

Now suppose that the arrays below represent the random numerical constants of the respective genes:

$$C_1 = \{-0.148, 1.83, -0.503, -1.786, 0.313, -0.302, 0.768, -0.947, 1.487, 0.075\}$$

$$C_2 = \{-0.256, -0.026, 1.874, 1.488, -0.8, -0.804, 0.039, -0.957, 0.462, 1.677\}$$

As you can see (you'll need to draw the [expression trees](#), of course), these chromosomes encode different solutions because the random constants expressed in the original chromosome are slightly different from the ones expressed in the daughter chromosome.



Genetic Operators

Direct Mutation of Random Numerical Constants

The [GEP-RNC algorithm](#) of GeneXproTools 4.0 uses an additional [gene domain](#) (Dc) for encoding random numerical constants (RNCs). In the Dc, the symbols used to represent the random numerical constants can obviously be replaced by other symbols of the same kind.

For the sake of simplicity, we are going to illustrate the mechanisms of RNC mutation using the compact, linear representation of chromosomes used to describe the [structural organization of chromosomes](#) in the previous chapter. In this representation, each element (function, terminal, or random constant) is represented by a single character so that each element can be easily identified by its position in the chromosome.

You know already that all the [genetic operators](#) contribute, directly or indirectly, to move the random numerical constants around. And, in fact, this constant shuffling is more than sufficient to allow an effective evolution of good models as the appropriate number of numerical constants is randomly created at the beginning of each run. However, GeneXproTools 4.0 has a special mutation operator that can be used to replace the value of a particular numerical constant by another one also randomly generated.

This operator, called **direct mutation of random numerical constants** or **RNC mutation** for short, randomly selects particular targets in the arrays where the random numerical constants are kept, and randomly generates a new constant.

Consider for instance the array:

$$C = \{-0.433, -1.823, 1.255, 0.028, -1.755, -0.036, -0.128, \textbf{-1.163}, 1.806, 0.083\}$$

encoding the random numerical constants of a particular gene. Suppose a mutation occurred at position 7, changing the constant -1.163 occupying that position by -0.494, giving:

$$C = \{-0.433, -1.823, 1.255, 0.028, -1.755, -0.036, -0.128, \textbf{-0.494}, 1.806, 0.083\}$$

The consequences of this kind of mutation are very diverse: they might be neutral in effect (for instance, when the gene itself is neutral; or when the random numerical constant has no expression on the [sub-ET](#)) or they might have manifold effects. The latter occurs whenever the mutated random numerical constant happens to be used more than once in the [expression](#) of the affected gene.

Let's illustrate this with an example. Consider, for instance, the following chromosome composed of two genes, each with a Dc length of 8:

```
0123456789012345678901201234567890123456789012
+-Q?+-c?c?aabcb13168489Q++/d?+cca??dab06383015
```

For which the random numerical constants of each gene are kept in the following arrays:

$$C_1 = \{-0.148, \mathbf{1.83}, -0.503, -1.786, 0.313, -0.302, 0.768, -0.947, 1.487, 0.075\}$$

$$C_2 = \{-0.256, -0.026, 1.874, 1.488, -0.8, -0.804, \mathbf{0.039}, -0.957, 0.462, 1.677\}$$

Now suppose a mutation occurred at position 1 in C_1 , changing the constant 1.83 into -0.036; and the constant 0.039 at position 6 in C_2 into 1.255. In this case, the following arrays are obtained:

$$C_1 = \{-0.148, \mathbf{-0.036}, -0.503, -1.786, 0.313, -0.302, 0.768, -0.947, 1.487, 0.075\}$$

$$C_2 = \{-0.256, -0.026, 1.874, 1.488, -0.8, -0.804, \mathbf{1.255}, -0.957, 0.462, 1.677\}$$

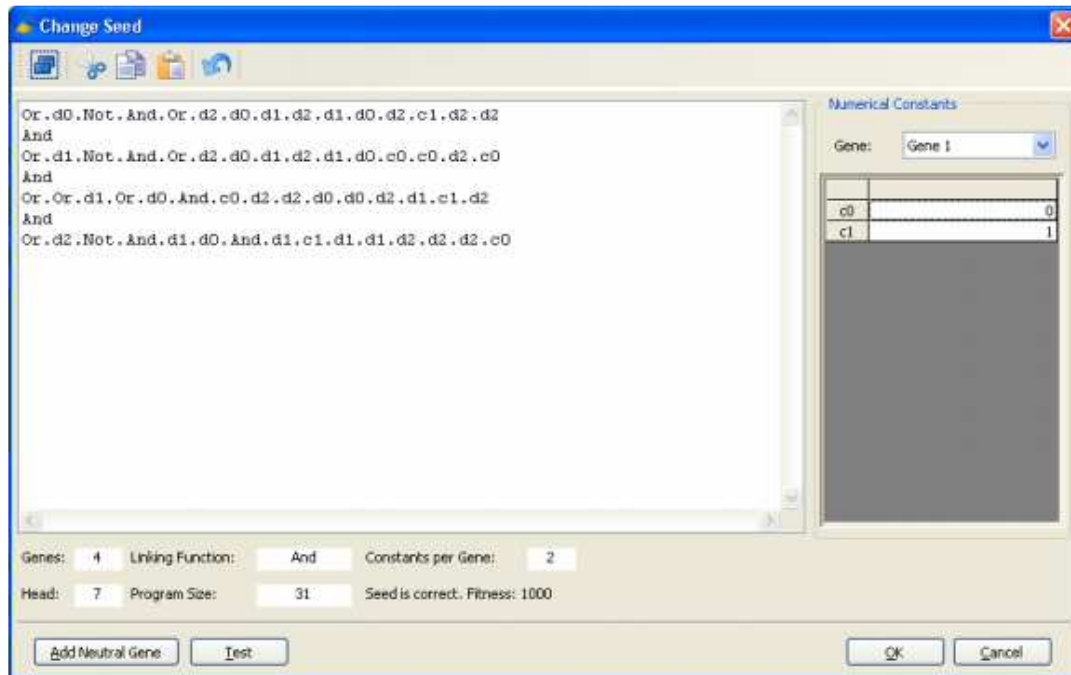
As you can see (you'll need to draw the [expression trees](#), of course), these chromosomes (before and after mutation) encode different solutions because the random constants expressed in the original individual are slightly different from the ones expressed in the daughter chromosome.



Designing New Models from Old Models

This chapter teaches you how to:

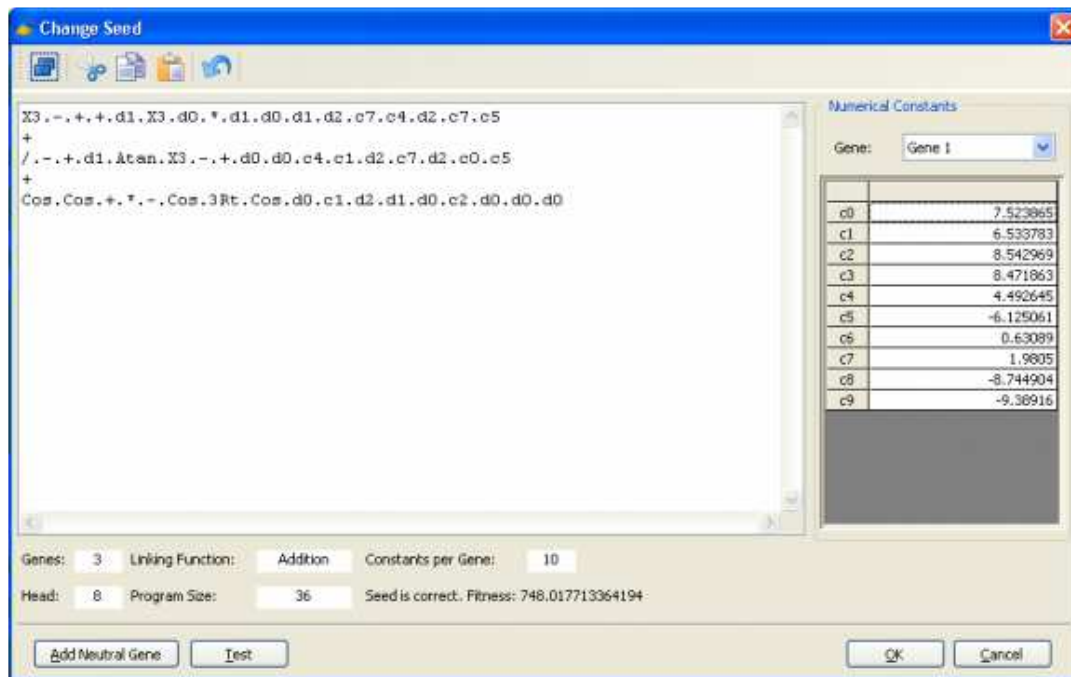
- [Optimize models previously evolved by GeneXproTools.](#)
- [Modify models previously evolved by GeneXproTools.](#)
- [Optimize a model created elsewhere.](#)
- [Simplify a model.](#)
- [Refresh a model.](#)
- [Add a neutral gene to allow a better optimization.](#)



Designing New Models from Old Models

Optimizing Models Evolved by GeneXproTools

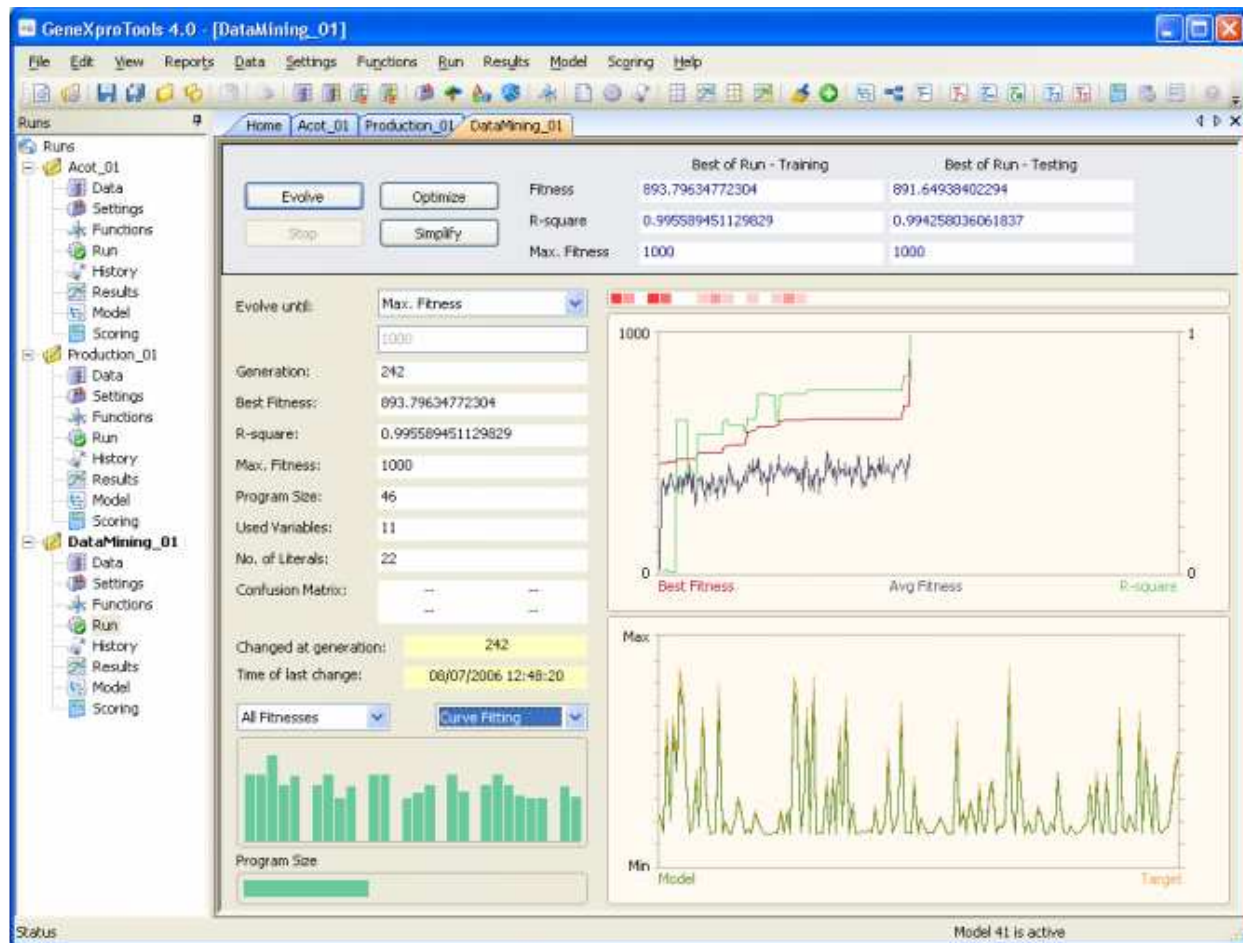
Sometimes, even if you are using a model previously evolved by GeneXproTools as seed, you might no longer have access to the model in [Karva code](#). In that case, you will need to know how to convert the model into Karva, as the Change Seed window only accepts Karva code.



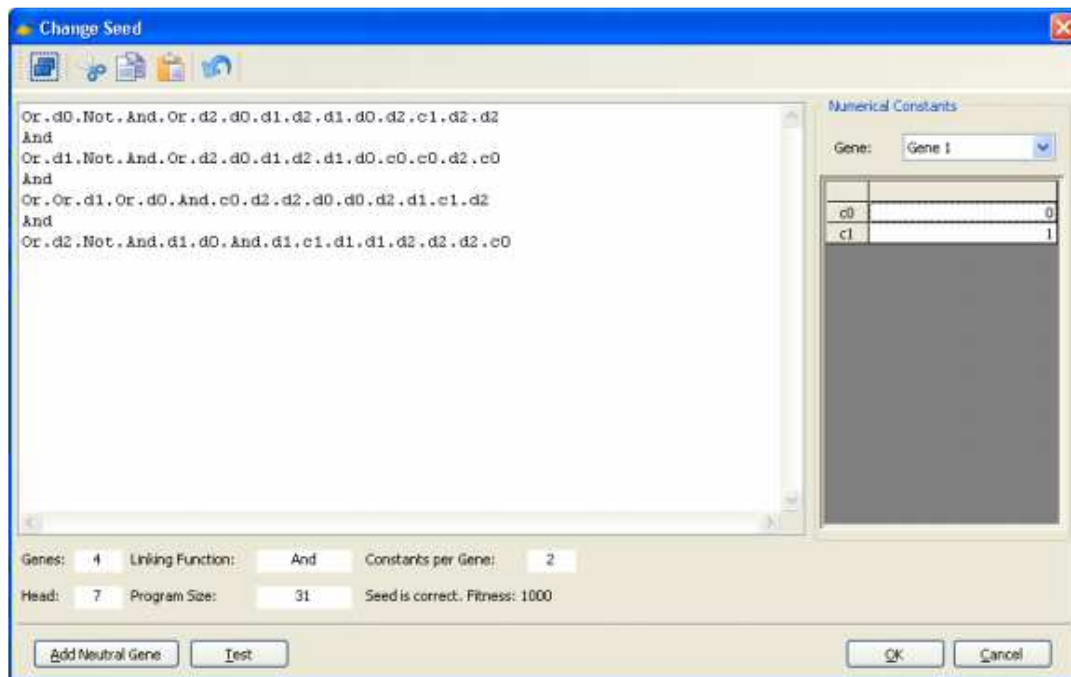
But most of the times, your seed model is already in [Karva code](#) and, in this case, creating better models from it should be a breeze.



The **modeling from seed facility** of GeneXproTools is a very powerful modeling tool as it enables you to refine your models by using an already good model as starter. When this starting model had been created by GeneXproTools, this process is very easy and most of the times you just have to press the Optimize button in the [Run Panel](#). In this case, the active model of that run (usually the best-of-run but any model kept in the [History Panel](#) can become active) can be used immediately as seed.



Other times, though, a model could have been evolved by GeneXproTools but you no longer have access to the original .gep file. In this case, you'll need to create a new run and choose a **chromosomal architecture** that could accommodate your starting model. This includes the **number of genes**, the **head size**, the **linking function**, the **function set**, and if your model has **numerical constants**, the number of constants per gene. Also important is the kind of **fitness function** you'll choose, for you must be careful that your seed model is viable (that is, has a positive fitness). To make sure of that, GeneXproTools 4.0 allows you to **pre-test your model** in the Change Seed window.



This pre-test not only evaluates the fitness of your starting model in the particular training set you've chosen to fine-tune your model, but also makes sure that the [structural organization](#) of your model is correct. For instance, if your model uses a function that was not selected in the [Functions Panel](#) or if you are not respecting the [chromosome structure](#) (head and tail sizes plus the linking function), you are prompted to fix the errors before continuing. And only when the seed is structurally correct are you allowed to go to the [Run Panel](#) and press the Optimize button so that you can breed new programs with it. Be advised, however, that if your seed, although viable, has a very low fitness, you will most probably lose it very soon in the evolutionary process and its contribution to the creation of better models could be negligible.



Designing New Models from Old Models

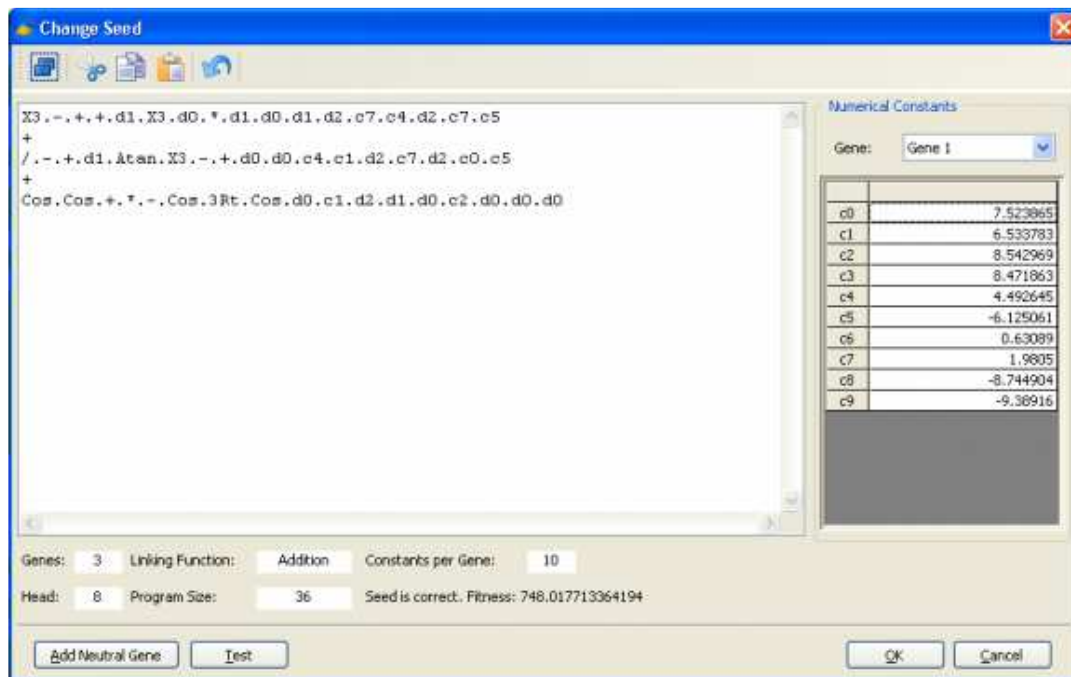
Modifying Models Evolved by GeneXproTools

Sometimes it is important to introduce small (or not so small) changes in an existing model, say, adjust a coefficient or remove a term. For that you must know how to change your seed correctly.

When your seed is already in [Karva notation](#) and the change is a small one, most of the times you will be able to change your seed directly in the Change Seed window and, by checking its representation as an [expression tree](#) in the Model Panel, you will be able to twinkle with it until it meets your standards.

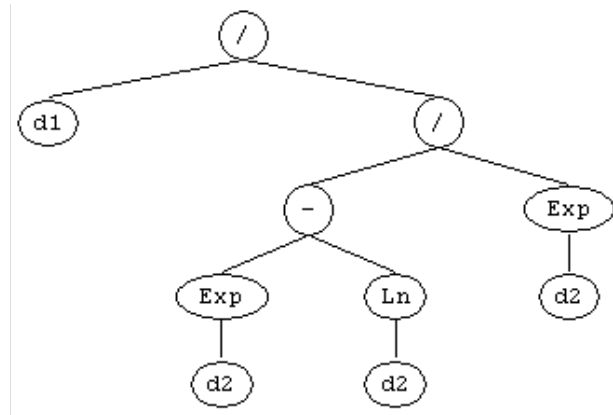


But if the change is a big one or if your seed is not in [Karva notation](#), then you will need to translate your model into Karva as the Change Seed window can only accept models in Karva code.



The easiest way of translating your model into [Karva code](#), consists of representing your model in a [tree structure](#) and then create a linear representation of the parse tree by writing down all the nodes in the tree from top to bottom and from left to right. You will end up with a [K-expression](#) which you can, with only a small modification, use as seed.

For instance, the expression tree below:



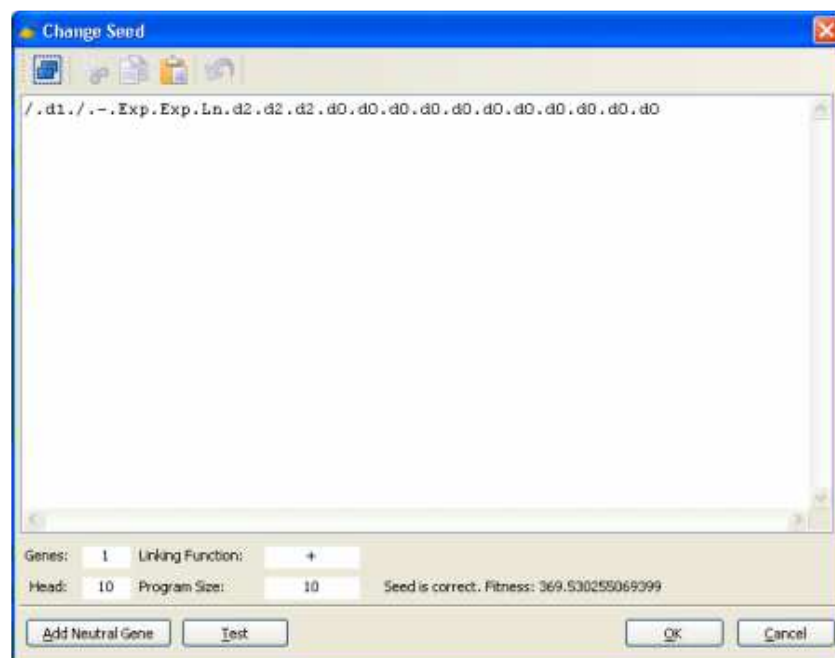
can be represented linearly by the following K-expression:

`/.d1./.-.Exp.Exp.Ln.d2.d2.d2`

Now you only need to incorporate this K-expression into a correct gene structure, with a [head](#) and a [tail](#) correctly defined. A simple way of doing this consists of considering the entire K-expression the head of your gene and then add a tail (with the correct length, of course) containing only terminals (which terminals you add is irrelevant as these terminals will not interfere with the expression of your K-expression). For instance, the K-expression above could be correctly accommodated into the following gene (the head is shown in blue):

`/.d1./.-.Exp.Exp.Ln.d2.d2.d2.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0`

Then you just have to create a new run in GeneXproTools and evolve quickly (for only one generation is enough) a proto-model with exactly the same structure you'll need for integrating your seed model (that is, number of genes, head size, linking function, function set, etc.), and then replace this proto-model in the Change Seed window by your seed.



These are only some examples of the things you can accomplish with the Change Seed method of

GeneXproTools 4.0. If you want to make other kinds of modification, for instance, use different [K-expressions](#) to represent different terms, then you must be familiar with the architectures of GeneXproTools 4.0 [learning algorithms](#) so that you can map correctly all your modifications into [Karva notation](#).



Designing New Models from Old Models

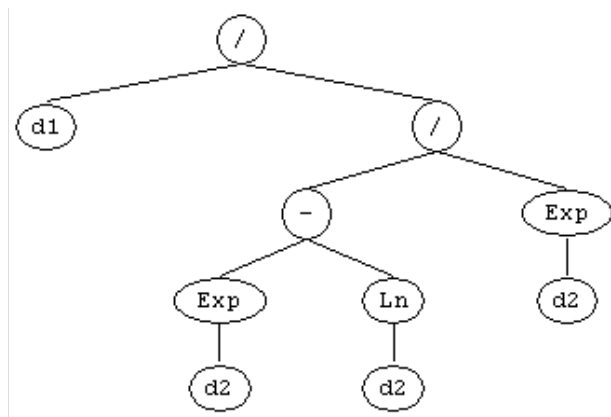
Optimizing an External Model

Optimizing a model not created by GeneXproTools, requires pretty much the same expertise necessary to make big modifications in models evolved by GeneXproTools as, in both cases, you will need to know how to create [K-expressions](#) from [expression trees](#).

So, assuming that the model you want to use as seed is not in [Karva notation](#), you will need to translate your model into Karva as the Change Seed window can only accept models in Karva code.

The easiest way of translating your model into [Karva code](#), consists of representing your model in a [tree structure](#) and then create a linear representation of the parse tree by writing down all the nodes in the tree from top to bottom and from left to right. You will end up with a [K-expression](#) that you can, with only a small modification, use as seed.

For instance, the expression tree below:



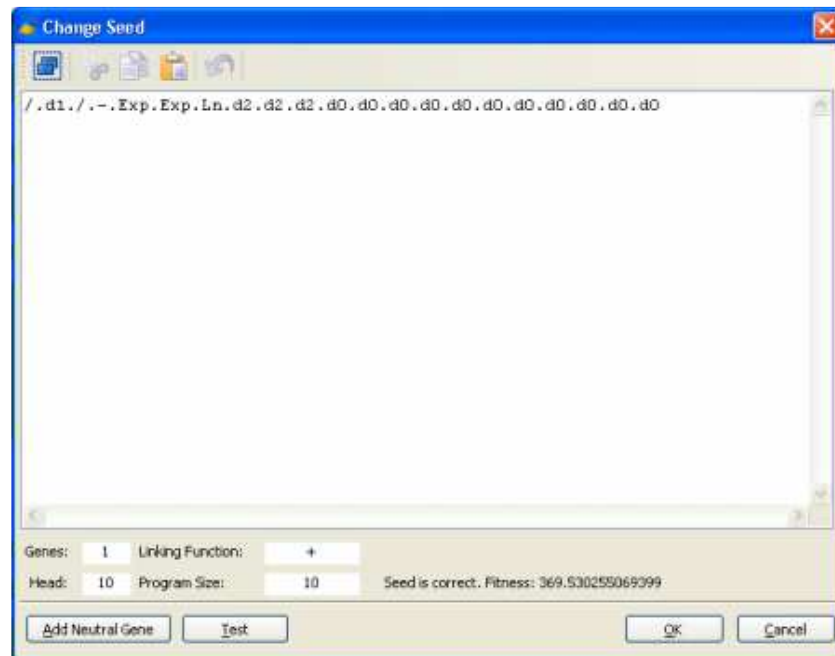
can be represented linearly by the following K-expression:

[/.d1./.-.Exp.Exp.Ln.d2.d2.d2](#)

Now you only need to incorporate this K-expression into a correct gene structure, with a [head](#) and a [tail](#) correctly defined. A simple way of doing this, consists of considering the entire K-expression the head of your gene and then add a tail (with the correct length, of course) containing only terminals (which terminals you add is irrelevant as these terminals will not interfere with the expression of your K-expression). For instance, the K-expression above could be correctly accommodated into the following gene:

[/.d1./.-.Exp.Exp.Ln.d2.d2.d2.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0](#)

Then you just have to create a new run in GeneXproTools and evolve quickly (for one generation is enough) a proto-model with exactly the same structure you'll need for integrating your seed model (that is, number of genes, head size, linking function, function set, etc.), and then replace this proto-model in the Change Seed window by your seed.



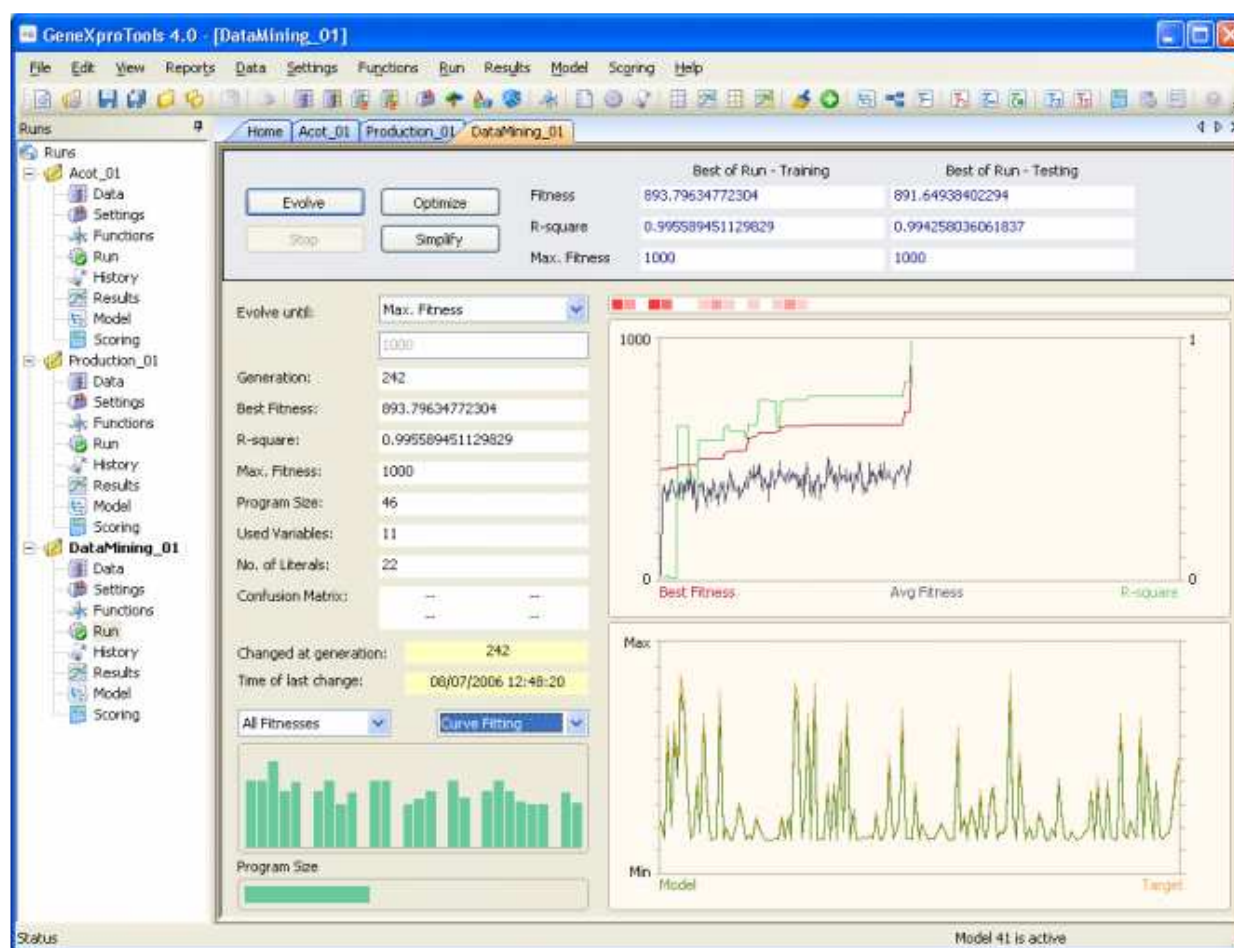
These are only some examples of the things you can accomplish with the Change Seed method of GeneXproTools 4.0. If you want to make other kinds of modification, for instance, use different [K-expressions](#) to represent different terms, then you must be familiar with the architectures of GeneXproTools 4.0 [learning algorithms](#) so that you can map correctly all your modifications into [Karva notation](#).



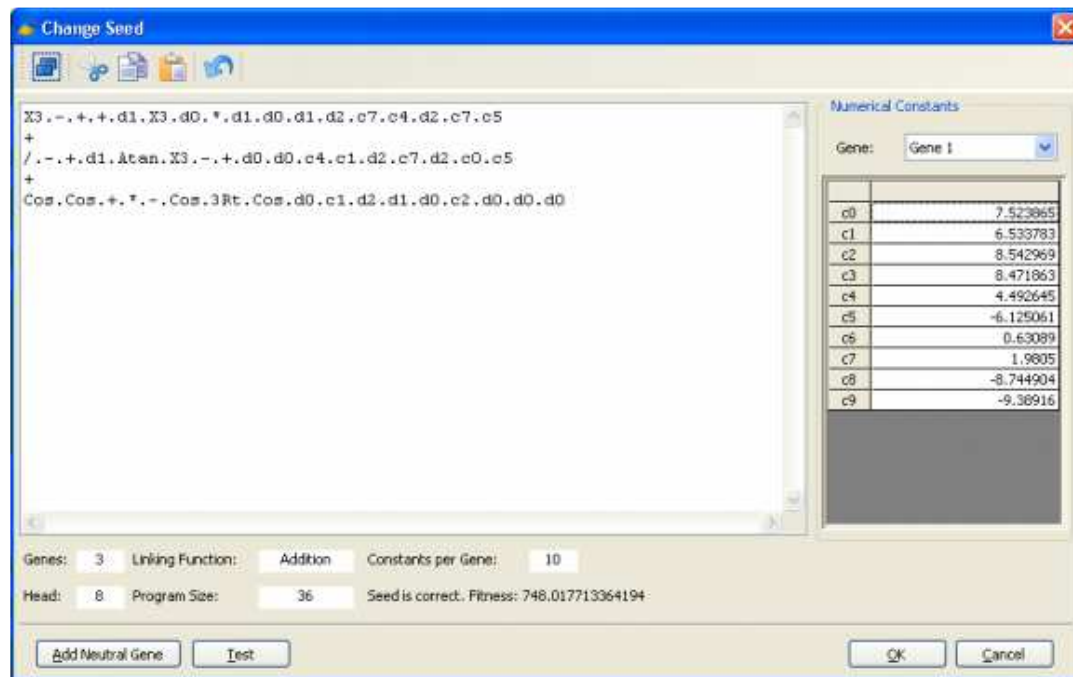
Designing New Models from Old Models

Simplifying a Model

GeneXproTools 4.0 allows you to simplify an existing model (which could have been either generated by GeneXproTools or by another modeling tool) either by pressing the Simplify button on the Run Panel or by choosing a fitness function with parsimony pressure and then press Optimize or Simplify on the Run Panel.

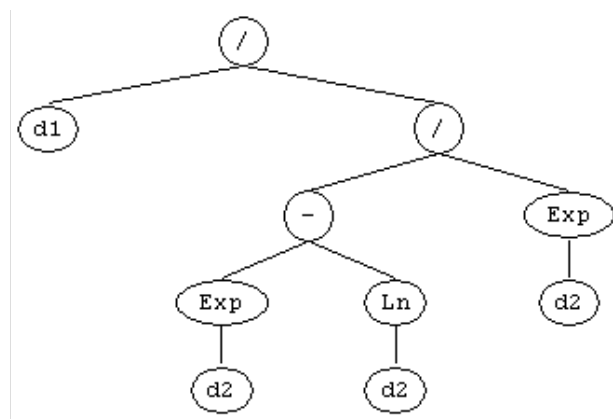


For models created outside of GeneXproTools or for GeneXproTools models modified by the user, the starting model is fed to the algorithm through the Change Seed window where both the fitness and structural soundness of the model are tested. In this case, you will need to know how to convert the model into [Karva code](#), as the Change Seed window only accepts Karva code.



The easiest way of translating your model into [Karva code](#), consists of representing your model in a [tree structure](#) and then create a linear representation of the parse tree by writing down all the nodes in the tree from top to bottom and from left to right. You will end up with a [K-expression](#) that you can, with only a small modification, use as seed.

For instance, the expression tree below:



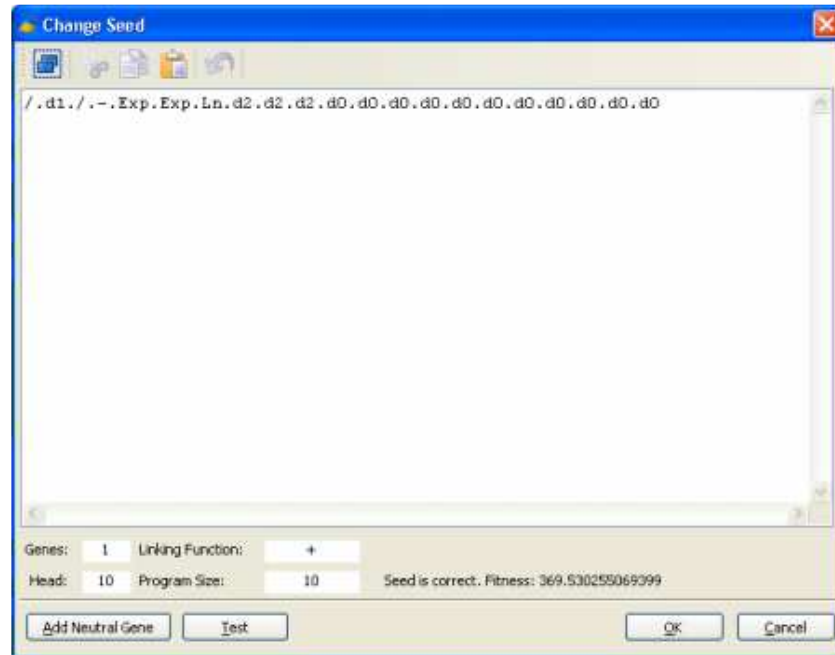
can be represented linearly by the following K-expression:

[/.d1./.-.Exp.Exp.Ln.d2.d2.d2](#)

Now you only need to incorporate this K-expression into a correct gene structure, with a [head](#) and a [tail](#) correctly defined. A simple way of doing this, consists of considering the entire K-expression the head of your gene and then add a tail (with the correct length, of course) containing only terminals (which terminals you add is irrelevant as these terminals will not interfere with the expression of your K-expression). For instance, the K-expression above could be correctly accommodated into the following gene:

[/.d1./.-.Exp.Exp.Ln.d2.d2.d2.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0](#)

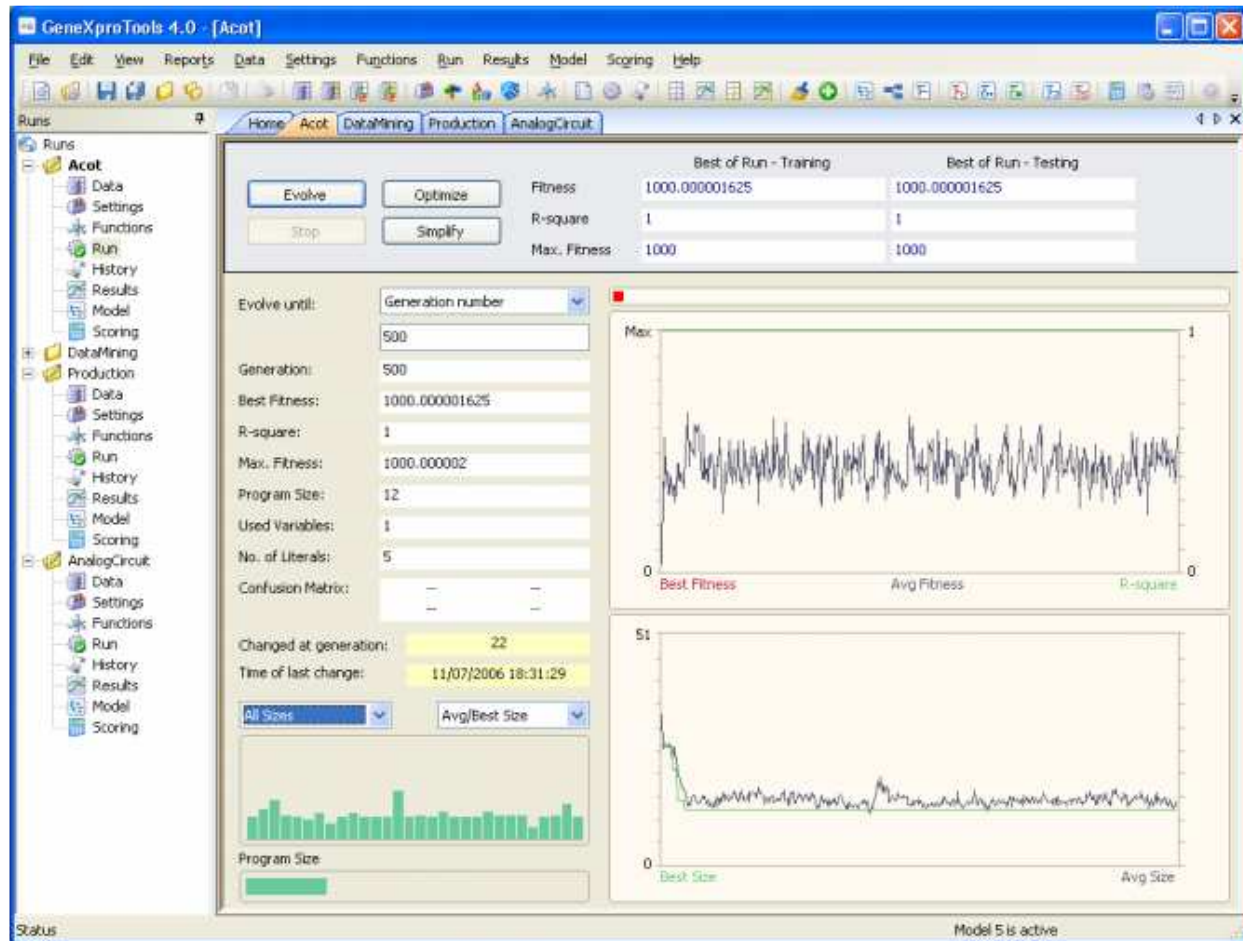
Then you just have to create a new run in GeneXproTools and evolve quickly (for one generation is enough) a proto-model with exactly the same structure you'll need for integrating your seed model (that is, number of genes, head size, linking function, function set, etc.), and then replace this proto-model in the Change Seed window by your seed.



Then, in the [Run Panel](#), by clicking the Simplify button, an evolutionary process starts in which all the subsequent models will be descendants of the model you want to simplify. Keep in mind, however, that the simplification algorithm used by GeneXproTools is an evolutionary one and, despite parsimony pressure being applied, models continue to be selected preferentially by fitness and sometimes their complexity might even increase temporarily if the gain in fitness outweighs the loss in simplicity.

For models created in the GeneXproTools 4.0 environment, you just have to select the model you want to simplify (either the best-of-run or an intermediate model picked up in the [History Panel](#)) and then click the Simplify button and let the algorithm create better descendants not only in terms of fitness but also in terms of size.

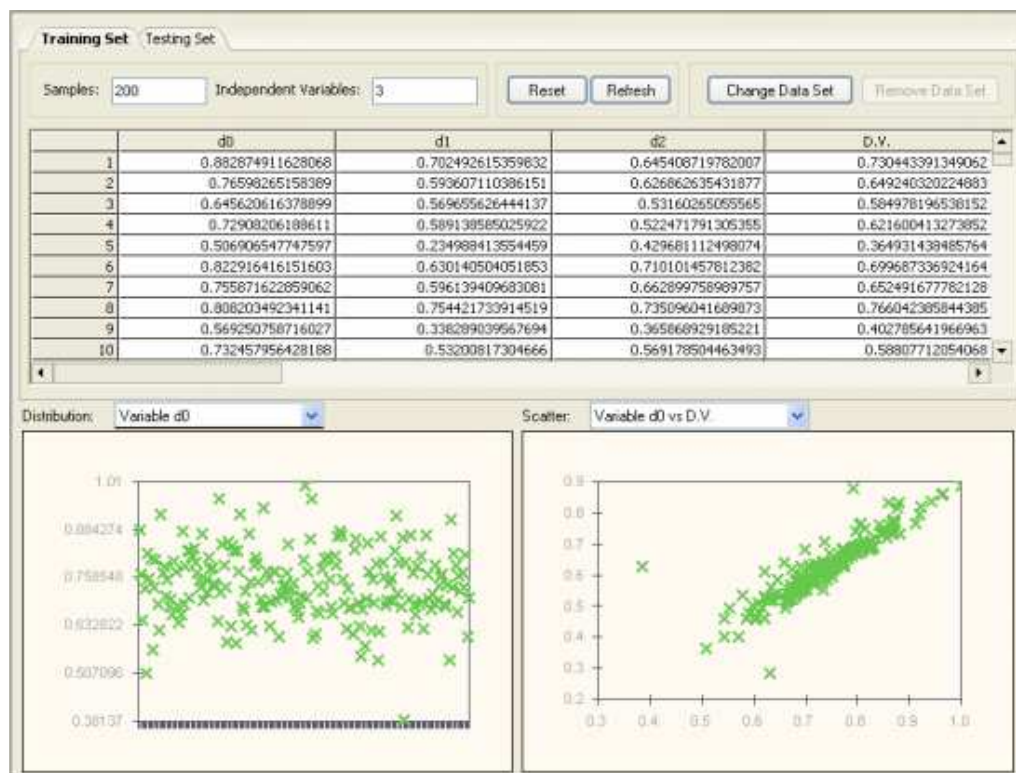
This can be done for as long as you wish or until you feel the program is as parsimonious as it will get. By monitoring the plots Avg/Best Size and Sub-Program Sizes as well as the Heat Map in the [Run Panel](#), you will know exactly how the simplification process is progressing.



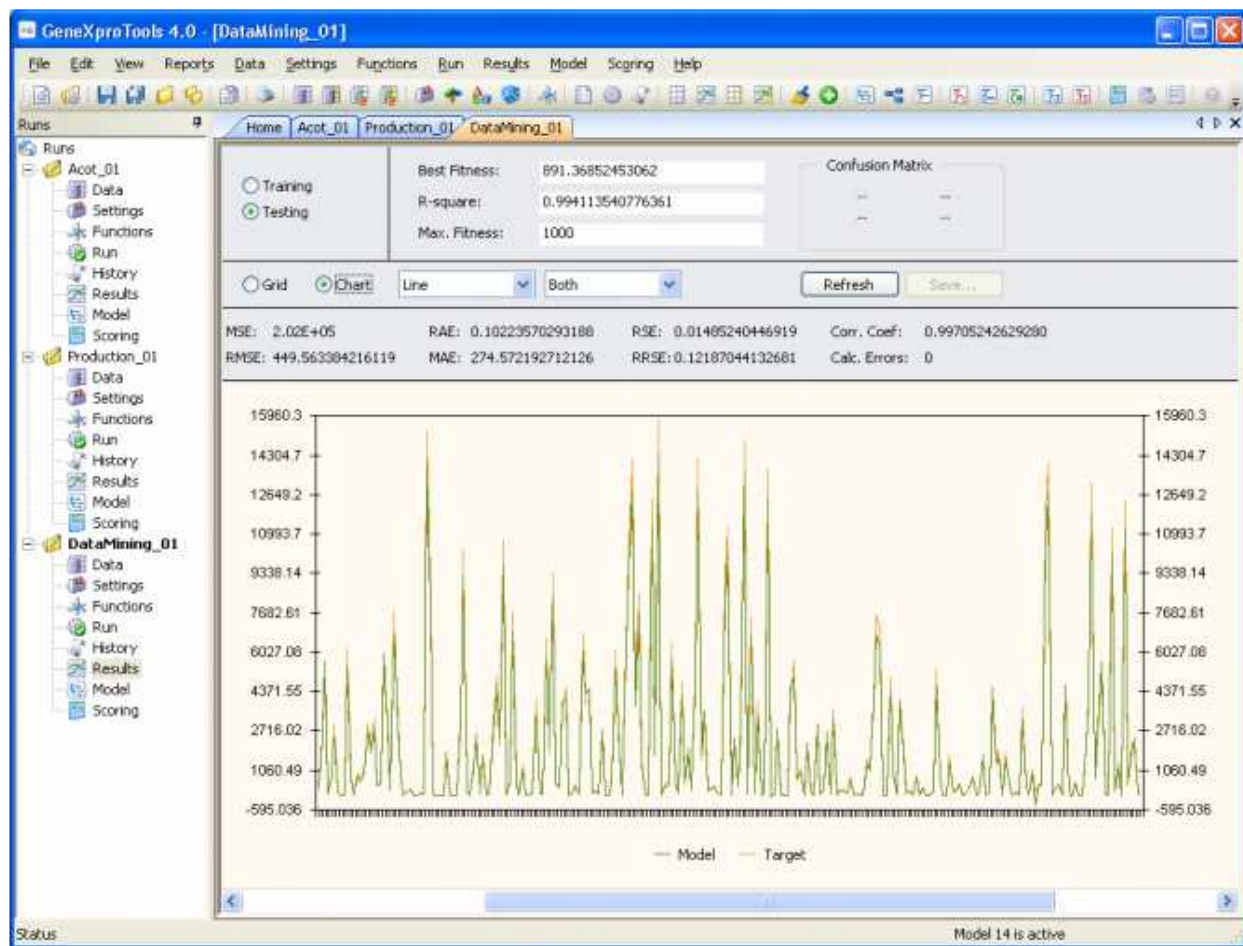
Designing New Models from Old Models

Refreshing a Model

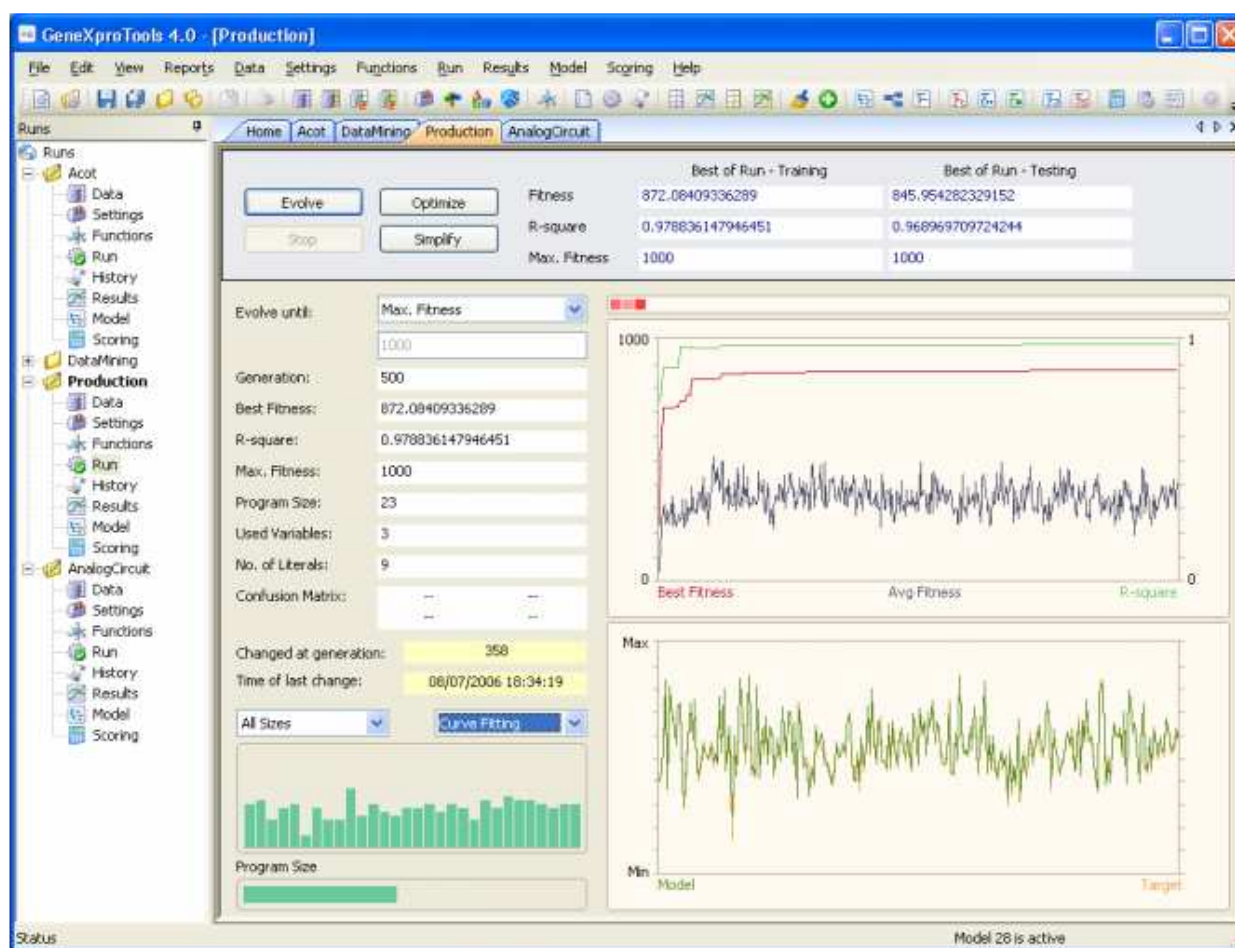
With time models get out of date and a little refreshing is required. With GeneXproTools 4.0 refreshing models is easy. For models created by GeneXproTools, you just have to replace the old training set (that is, the data set that was used to create the model you want to refresh) with the new one. You can do that in the Data Panel -> Training Set Tab or by choosing Change Training Data in the Data Menu.



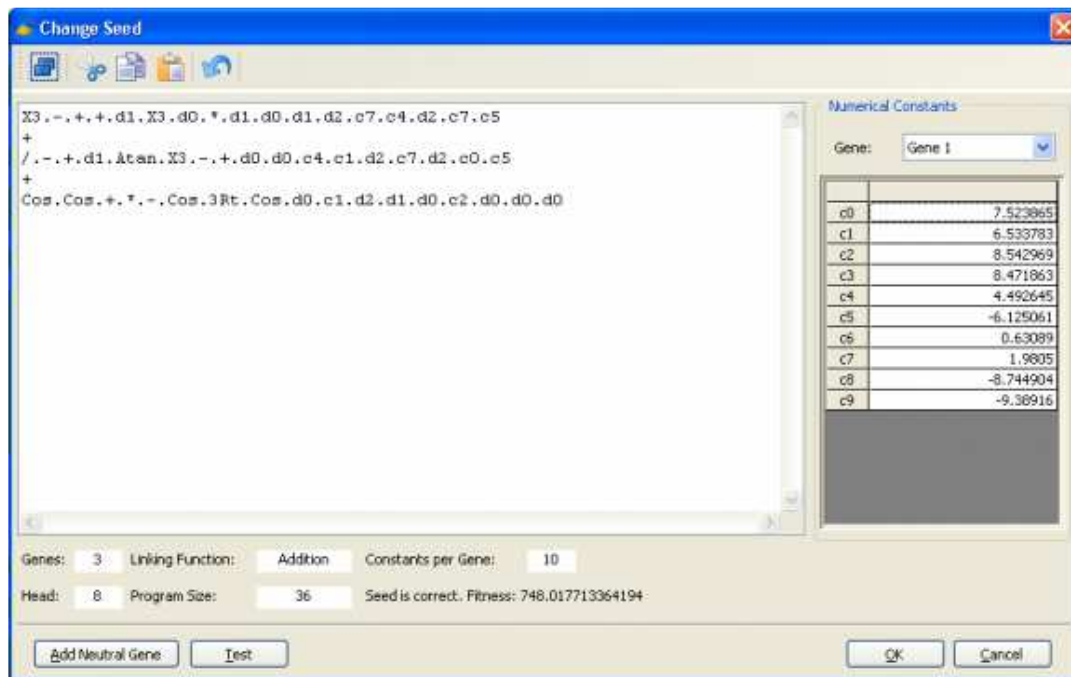
Then we recommend you go to the Results Panel and check how the old model fares in this training set, as sometimes calculation errors might occur with new data, which obviously will render the model unviable in this new environment. If that is the case, you'll have to check carefully your new data and get rid of the problematic samples that are causing the errors in order to make it viable, otherwise you won't be able to use it as seed for starting the refreshing process.



Thus, after making sure that your old model is viable in this new data set, you just go to the [Run Panel](#) and press the Optimize button. By doing that you are starting a new evolutionary process where all the best-of-generation models are refreshed models, much better adapted to the new circumstances than the old model.

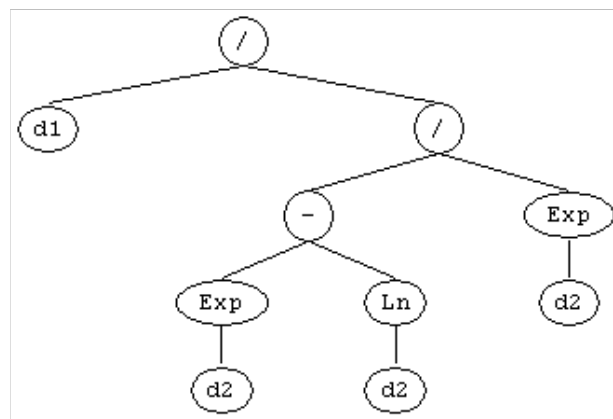


For models created outside of GeneXproTools, a new .gep run must be created using the new data set for training. Then the old model is fed into GeneXproTools through the Change Seed window where both the fitness and structural soundness of the model can be tested. In this case, you will need to know how to convert the model into [Karva code](#), as the Change Seed window only accepts Karva code.



The easiest way of translating your model into [Karva code](#), consists of representing your model in a [tree structure](#) and then create a linear representation of the parse tree by writing down all the nodes in the tree from top to bottom and from left to right. You will end up with a [K-expression](#) that you can, with only a small modification, use as seed.

For instance, the expression tree below:



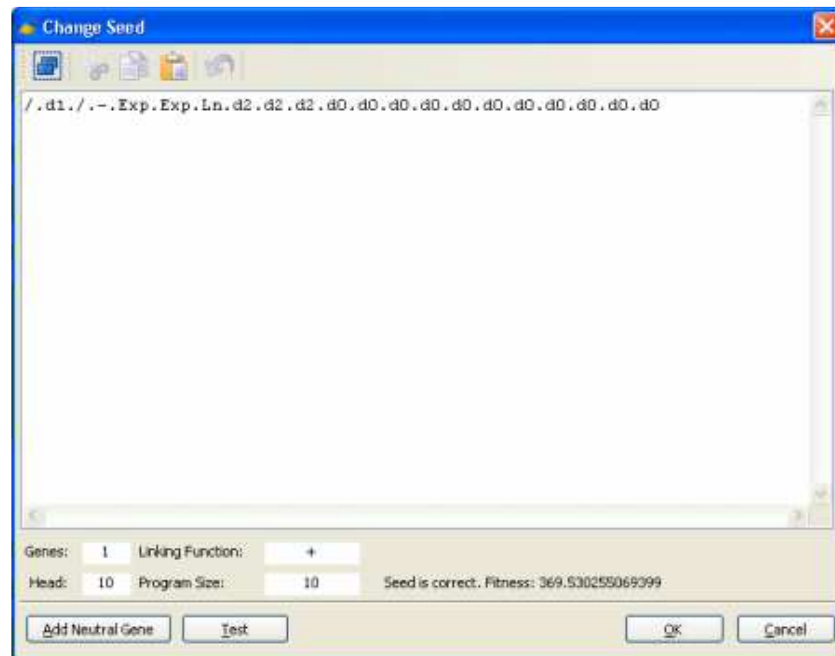
can be represented linearly by the following K-expression:

[/.d1./.-.Exp.Exp.Ln.d2.d2.d2](#)

Now you only need to incorporate this K-expression into a correct gene structure, with a [head](#) and a [tail](#) correctly defined. A simple way of doing this, consists of considering the entire K-expression the head of your gene and then add a tail (with the correct length, of course) containing only terminals (which terminals you add is irrelevant as these terminals will not interfere with the expression of your K-expression). For instance, the K-expression above could be correctly accommodated into the following gene:

[/.d1./.-.Exp.Exp.Ln.d2.d2.d2.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0.d0](#)

Then you just have to create a new run in GeneXproTools and evolve quickly (for one generation is enough) a proto-model with exactly the same structure you'll need for integrating your seed model (that is, number of genes, head size, linking function, function set, etc.), and then replace this proto-model in the Change Seed window by your seed.



Then, in the [Run Panel](#), by clicking the Optimize button, an evolutionary process starts in which all the subsequent models will be descendants of the model you want to refresh, being all the best-of-generation models refreshed models, much better adapted to the new circumstances than the old model.

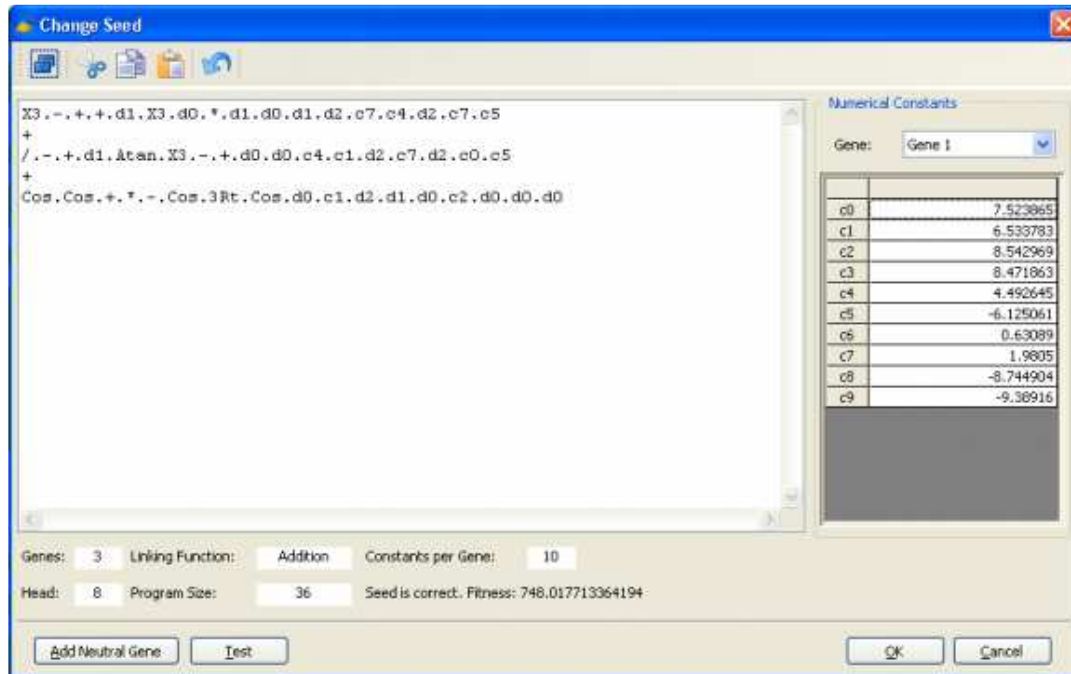


Designing New Models from Old Models

Adding a Neutral Gene

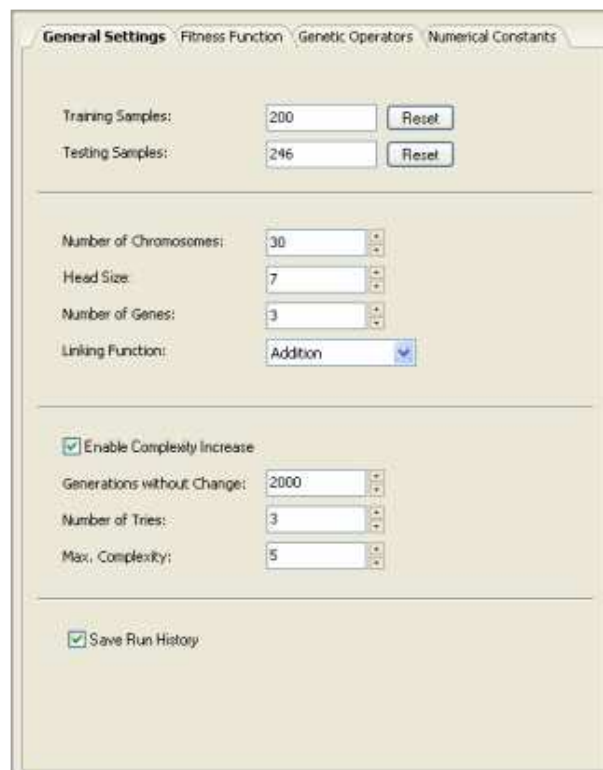
The addition of a neutral gene or term to a model might seem, at first sight, the wrong thing to do as most of the times we are interested in creating more parsimonious models, not the opposite. But one should look at this as modeling in progress as, for really complex phenomena, it is not uncommon to approximate a complex function progressively.

Thus, being able of introducing extra terms to your seed is a powerful modeling tool and GeneXproTools 4.0 allows you to do that either by selecting Add Neutral Gene in the Edit Menu or by pressing the Add Neutral Gene button in the Change Seed window.



Here, by pressing the Add Neutral Gene button, you will see a neutral gene being added to your model. By doing this, you are giving the learning algorithm more room to play and, hopefully, a better, more complex program will evolve.

GeneXproTools 4.0 also allows you to introduce neutral genes automatically by activating the Complexity Increase Engine in the Settings Panel -> General Settings Tab.



Whenever you are using the **Complexity Increase Engine** of GeneXproTools 4.0, you must fill the Generations Without Change box to set the period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction is created or a neutral gene (an extra term) is automatically added to the evolving models; the Number of Tries box corresponds to the number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models; in the Max. Complexity box you write the maximum number of terms (genes) you'll allow in your model and no other terms will be introduced beyond this threshold.

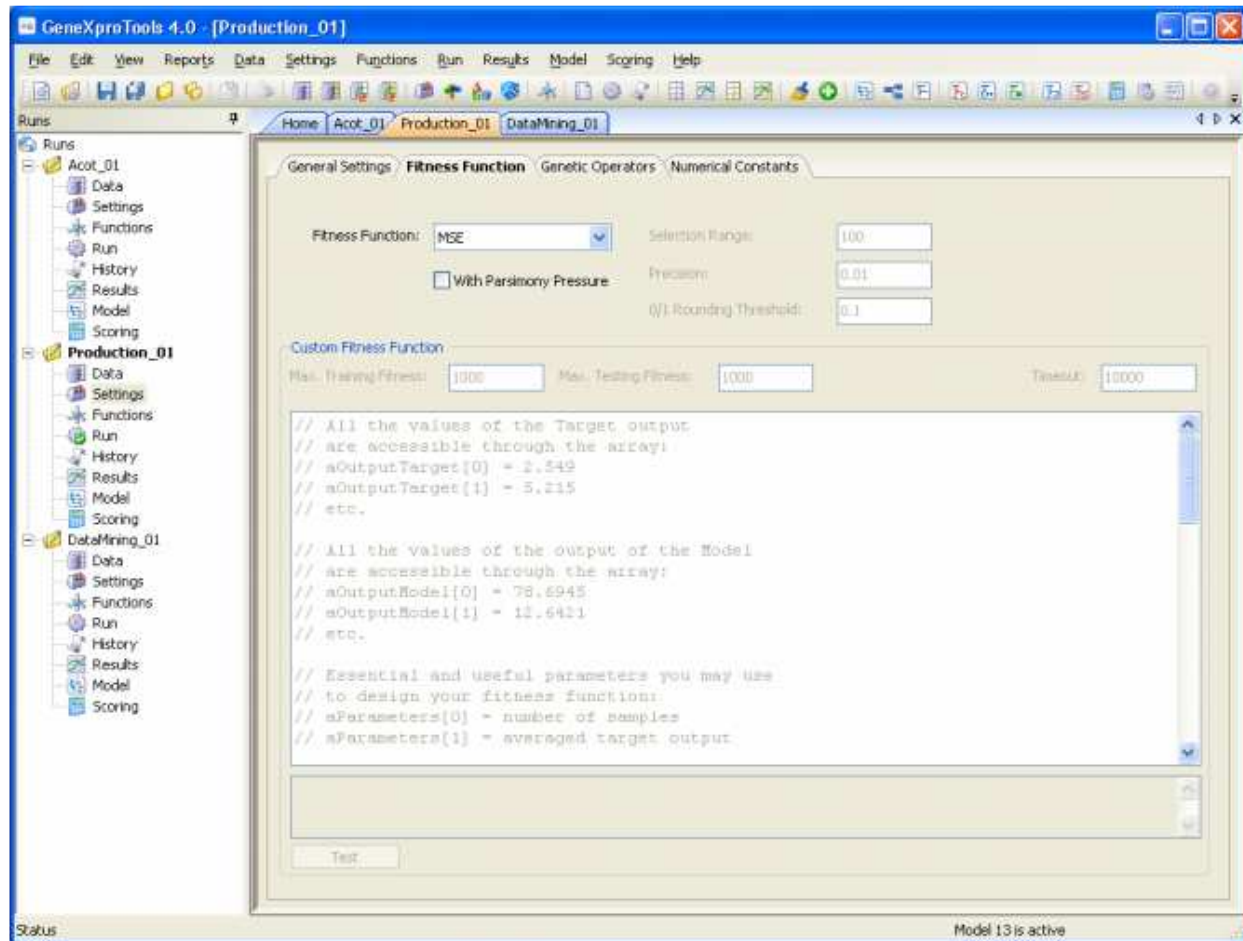
As stated previously, the Complexity Increase Engine of GeneXproTools 4.0 might be a very powerful modeling tool, especially for [Time Series Prediction](#) where good models might require some serious modeling, but you must be careful not to create excessively complex models for, most of the times, a greater complexity does not necessarily imply a greater efficiency.



Choosing the Fitness Function

This chapter describes all the fitness functions used for:

- [Function finding problems.](#)
- [Classification problems.](#)
- [Time series prediction problems.](#)
- [Logic synthesis problems.](#)



Choosing the Fitness Function

Fitness Functions for Function Finding

For Function Finding problems, the fitness functions of GeneXproTools 4.0 are organized as follows:

Fitness functions based on the absolute error:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)

Fitness functions based on the relative error:

- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)
- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)

Fitness functions based on correlation coefficients:

- [R-square](#)
- [Correlation Coefficient](#)

Fitness functions with parsimony pressure:

- And all the above with [parsimony pressure](#)

And user defined fitness functions:

- [Custom Fitness Function](#)

General Settings **Fitness Function** Genetic Operators Numerical Constants

Fitness Function: Custom Selection Ranges:

☐ With Parsimony Pressure: Precision:

Rounding Threshold:

Custom Fitness Function

Max. Training Fitness: Max. Testing Fitness: Timeout:

```
// Below is an example of a correct fitness function:
// the Mean Squared Error (MSE) fitness function of GeneXproTools,
// for which the maximum fitness is equal to 1000:

// MEAN SQUARED ERROR
var nSamples = aParameters[0];
var fitness = 0.0;
var modelMinusTargetSquared = 0.0;
var MSE = 0.0;
for (var i=0; i<nSamples; i++)
{
    var temp1 = 0.0;
    temp1 = aOutputModel[i] - aOutputTarget[i];
    temp1 *= temp1;
    modelMinusTargetSquared += temp1;
}

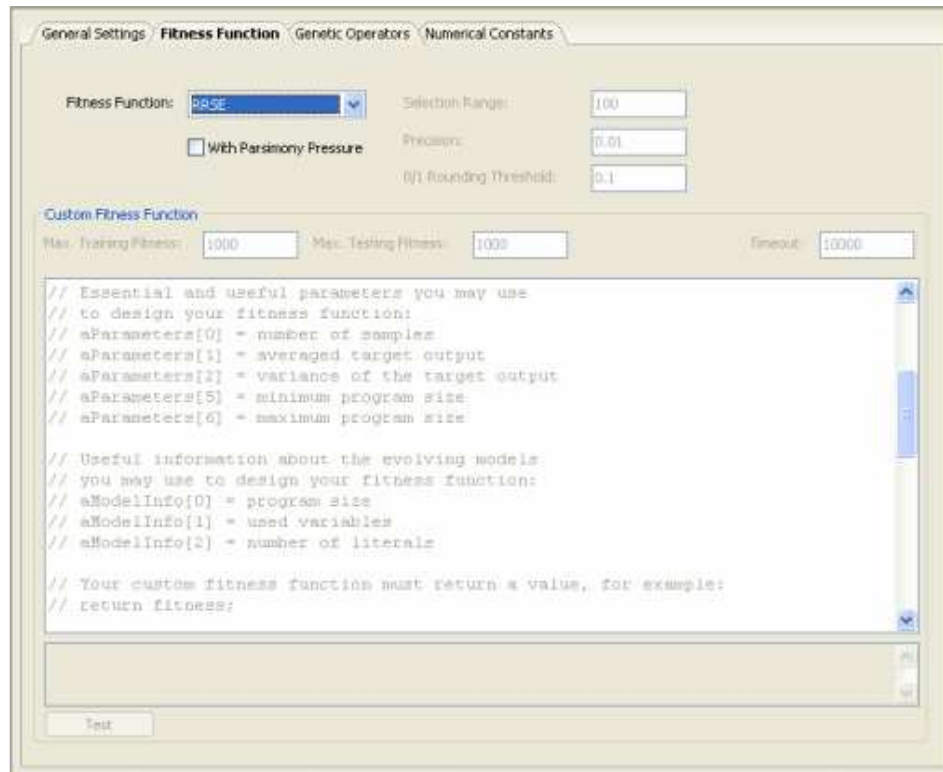
Correct function.
```

◀ | ▶

Choosing the Fitness Function

Fitness Functions and Parsimony Pressure

GeneXproTools allows you to select models based not only on their raw fitness but also on their size by choosing a fitness function with parsimony pressure. For that you just have to check the box With Parsimony Pressure on the Fitness Function Tab.



Designing parsimonious solutions makes sense not only in terms of efficiency but also clarity, and modelers always try to balance both these factors. The fitness functions with parsimony pressure of GeneXproTools allow you to achieve the same balance between efficiency and simplicity by favoring good solutions with more compact structures.

GeneXproTools has a total of 15 **fitness functions with parsimony pressure**, finely tuned to allow an efficient evolution:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)
- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)
- [R-square](#)

- [Correlation Coefficient](#)

In addition, GeneXproTools 4.0 allows you to play with program size and weight it as you see fit in your own [custom fitness functions](#).



Choosing the Fitness Function

Absolute Error Based Fitness Functions

GeneXproTools 4.0 has a total of 16 fitness functions based on the **absolute error**:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- All the above with [parsimony pressure](#)



Choosing the Fitness Function

Absolute Error with Selection Range

GeneXproTools 4.0 implements the **Absolute Error With Selection Range** (Absolute with SR) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **Absolute with SR** fitness function explores the idea of a selection range and a precision. The **selection range** is used as a limit for selection to operate, above which the performance of a program on a particular fitness case contributes nothing to its fitness. And the **precision** is the limit for improvement, as it allows the fine-tuning of the evolved solutions as accurately as possible.

Mathematically, the **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = \sum_{j=1}^n (R - |P_{(ij)} - T_j|)$$

where R is the selection range, $P_{(ij)}$ the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases) and T_j is the target value for fitness case j . Note that the absolute value term corresponds to the **absolute error**. This term is what is called the **precision** and if the error is smaller than or equal to the precision then the error becomes zero. Thus, for a good match, the absolute value term is zero and $f_i = f_{\max} = nR$.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = nR$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Absolute/Hits

GeneXproTools 4.0 implements the **Absolute/Hits** fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For some Function Finding problems it is important to evolve a model that performs well for all fitness cases within a certain absolute error (the **precision**) of the correct value.

For those cases, the **fitness** $f_{(ij)}$ of an individual program i for fitness case j is evaluated by the formula:

$$\text{If } E_{(ij)} \leq p, \text{ then } f_{(ij)} = 1; \text{ else } f_{(ij)} = 0$$

where p is the precision and $E_{(ij)}$ is the **absolute error** of an individual program i for fitness case j evaluated by the equation:

$$E_{(ij)} = |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j and T_j is the target value for fitness case j .

And overall, the **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = \sum_{j=1}^n f_{(ij)}$$

where n is the total number of fitness cases.

So, for this fitness function, maximum fitness $f_{\max}$ is given by:

$$f_{\max} = n$$

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = n$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Squared Error

GeneXproTools 4.0 implements the **Mean Squared Error** (MSE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **MSE** fitness function of GeneXproTools is, as expected, based on the standard **mean squared error**, which, on its turn, is based on the **absolute error**.

The mean squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the MSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$

Choosing the Fitness Function

Root Mean Squared Error

GeneXproTools 4.0 implements the **Root Mean Squared Error** (RMSE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **RMSE** fitness function of GeneXproTools is, as expected, based on the standard [root mean squared error](#). By taking the square root of the [mean squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

The root mean squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the RMSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Absolute Error

GeneXproTools 4.0 implements the **Mean Absolute Error** (MAE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **MAE** fitness function of GeneXproTools is, as expected, based on the standard [mean absolute error](#), which, on its turn, is based on the **absolute error**.

The mean absolute error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the MAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal (the coefficient 1000 allows a fairer distribution of fitnesses for selection).

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Squared Error

GeneXproTools 4.0 implements the **Relative Squared Error** (RSE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **RSE** fitness function of GeneXproTools is, as expected, based on the standard [relative squared error](#), which, on its turn, is based on the **absolute error**.

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the relative squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Relative Squared Error

GeneXproTools 4.0 implements the **Root Relative Squared Error** (RRSE) fitness function both with

and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **RRSE** fitness function of GeneXproTools is, as expected, based on the standard [root relative squared error](#), which, on its turn, is based on the **absolute error**.

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the root relative squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RRSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Absolute Error

GeneXproTools 4.0 implements the **Relative Absolute Error** (RAE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **RAE** fitness function of GeneXproTools is, as expected, based on the standard [relative absolute error](#), which, on its turn, is based on the **absolute error**.

The relative absolute error is very similar to the [relative squared error](#) in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the relative absolute error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n |P_{ij} - T_j|}{\sum_{j=1}^n |T_j - \bar{T}|}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Error Based Fitness Functions

GeneXproTools 4.0 has a total of 16 fitness functions based on the **relative error**:

- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)

- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)
- All the above with [parsimony pressure](#)



Choosing the Fitness Function

Relative Error with Selection Range

GeneXproTools 4.0 implements the **Relative Error With Selection Range** (Relative with SR) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **Relative with SR** fitness function explores the idea of a selection range and a precision. The **selection range** is used as a limit for selection to operate, above which the performance of a program on a particular fitness case contributes nothing to its fitness. And the **precision** is the limit for improvement, as it allows the fine-tuning of the evolved solutions as accurately as possible.

Mathematically, the **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = \sum_{j=1}^n \left(R - \left| \frac{P_{(ij)} - T_j}{T_j} \cdot 100 \right| \right)$$

where R is the selection range, $P_{(ij)}$ the value predicted by the individual program i for fitness case j (out of n fitness cases) and T_j is the target value for fitness case j . Note that the absolute value term corresponds to the **relative error**. This term is what is called the **precision** and if the error is smaller than or equal to the precision then the error becomes zero. Thus, for a good match the absolute value term is zero and $f_i = f_{\max} = nR$.

Note that when the target value for a particular fitness case is zero, the relative error is undefined but, for fitness evaluation purposes during training, GeneXproTools 4.0 salvages these cases so that they can also be used for fine-tuning solutions.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = nR$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative/Hits

GeneXproTools 4.0 implements the **Relative/Hits** fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For some Function Finding problems it is important to evolve a model that performs well for all fitness cases within a certain relative error (the **precision**) of the correct value.

For those cases, the **fitness** $f_{(ij)}$ of an individual program i for fitness case j is evaluated by the formula:

$$\text{If } E_{(ij)} \leq p, \text{ then } f_{(ij)} = 1; \text{ else } f_{(ij)} = 0$$

where p is the precision and $E_{(ij)}$ is the relative error of an individual program i for fitness case j evaluated by the equation:

$$E_{(ij)} = \left| \frac{P_{(ij)} - T_j}{T_j} \cdot 100 \right|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j and T_j is the target value for fitness case j .

And overall, the **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = \sum_{j=1}^n f_{(ij)}$$

where n is the total number of fitness cases.

So, for this fitness function, maximum fitness $f_{\max}$ is given by:

$$f_{\max} = n$$

Note that when the target value for a particular fitness case is zero, the relative error is undefined but, for fitness evaluation purposes during training, GeneXproTools 4.0 salvages these cases so that they can also be used for fine-tuning solutions.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = n$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Squared Error

GeneXproTools 4.0 implements the **Mean Squared Error** (rMSE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rMSE** fitness function of GeneXproTools is, as expected, based on the standard [mean squared error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The rMSE E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n \left(\frac{P_{(ij)} - T_j}{T_j} \right)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the rMSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Mean Squared Error

GeneXproTools 4.0 implements the **Root Mean Squared Error** (rRMSE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rRMSE** fitness function of GeneXproTools is, as expected, based on the standard [root mean squared error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure. By taking the square root of the [mean squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

The rRMSE E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n \left(\frac{P_{(ij)} - T_j}{T_j} \right)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the rRMSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the

linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Absolute Error

GeneXproTools 4.0 implements the **Mean Absolute Error** (rMAE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rMAE** fitness function of GeneXproTools is, as expected, based on the standard [mean absolute error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The rMAE E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n \left| \frac{P_{(ij)} - T_j}{T_j} \right|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the rMAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal (the coefficient 1000 allows a fairer distribution of fitnesses for selection).

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with

parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Squared Error

GeneXproTools 4.0 implements the **Relative Squared Error** (rRSE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rRSE** fitness function of GeneXproTools is, as expected, based on the standard [relative squared error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the rRSE E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n \left(\frac{P_{(j)} - T_j}{T_j} \right)^2}{\sum_{j=1}^n \left(\frac{T_j - \bar{T}}{\bar{T}} \right)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the rRSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Relative Squared Error

GeneXproTools 4.0 implements the **Root Relative Squared Error** (rRRSE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rRRSE** fitness function of GeneXproTools is, as expected, based on the standard [root relative squared error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the rRRSE E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^n \left(\frac{P_{ij} - T_j}{T_j} \right)^2}{\sum_{j=1}^n \left(\frac{T_j - \bar{T}}{\bar{T}} \right)^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the rRRSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Absolute Error

GeneXproTools 4.0 implements the **Relative Absolute Error** (rRAE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rRAE** fitness function of GeneXproTools is, as expected, based on the standard [relative absolute error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The relative absolute error is very similar to the [relative squared error](#) in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the rRAE E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n \left| \frac{P_{(ij)} - T_j}{T_j} \right|}{\sum_{j=1}^n \left| \frac{T_j - \bar{T}}{\bar{T}} \right|}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the rRAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

R-square

GeneXproTools 4.0 implements the **R-square** fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **R-square** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [R-square](#), which returns the square of the Pearson product moment correlation coefficient.

The Pearson product moment correlation coefficient is a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between the predicted values and the target values.

The Pearson product moment correlation coefficient R_i of an individual program i is evaluated by the equation:

$$R_i = \frac{n \sum_{j=1}^n (T_j P_{(ij)}) - \left(\sum_{j=1}^n T_j \right) \left(\sum_{j=1}^n P_{(ij)} \right)}{\sqrt{\left[n \sum_{j=1}^n T_j^2 - \left(\sum_{j=1}^n T_j \right)^2 \right] \left[n \sum_{j=1}^n P_{(ij)}^2 - \left(\sum_{j=1}^n P_{(ij)} \right)^2 \right]}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * R_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Correlation Coefficient

GeneXproTools 4.0 implements the **Correlation Coefficient** fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **Correlation Coefficient** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [correlation coefficient](#), which is a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between the predicted values and the target values.

The correlation coefficient C_i of an individual program i is evaluated by the equation:

$$C_i = \frac{Cov(T, P)}{\sigma_t \cdot \sigma_p}$$

where $Cov(T, P)$ is the covariance of the target and model outputs; and σ_t and σ_p are the corresponding standard deviations, which are given by:

$$Cov(T, P) = \frac{1}{n} \sum_{j=1}^n (T_j - \bar{T})(P_{(ij)} - \bar{P})$$

$$\sigma_t = \sqrt{\frac{\sum_{j=1}^n (T_j - \bar{T})^2}{n}}$$

$$\sigma_p = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - \bar{P})^2}{n}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n fitness cases or sample cases); T_j is the target value for fitness case j ; and $\bar{T}$ and $\bar{P}$ are given by the formulas:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

$$\bar{P} = \frac{1}{n} \sum_{j=1}^n P_{(j)}$$

The correlation coefficient is confined to the range $[-1, 1]$. When $C_i = 1$, there is a perfect positive linear correlation between T and P , that is, they vary by the same amount. When $C_i = -1$, there is a perfect negative linear correlation between T and P , that is, they vary in opposite ways (when T increases, P decreases by the same amount). When $C_i = 0$, there is no correlation between T and P . Intermediate values describe partial correlations and the closer to 1 or -1 the better the model.

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * C_i * C_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



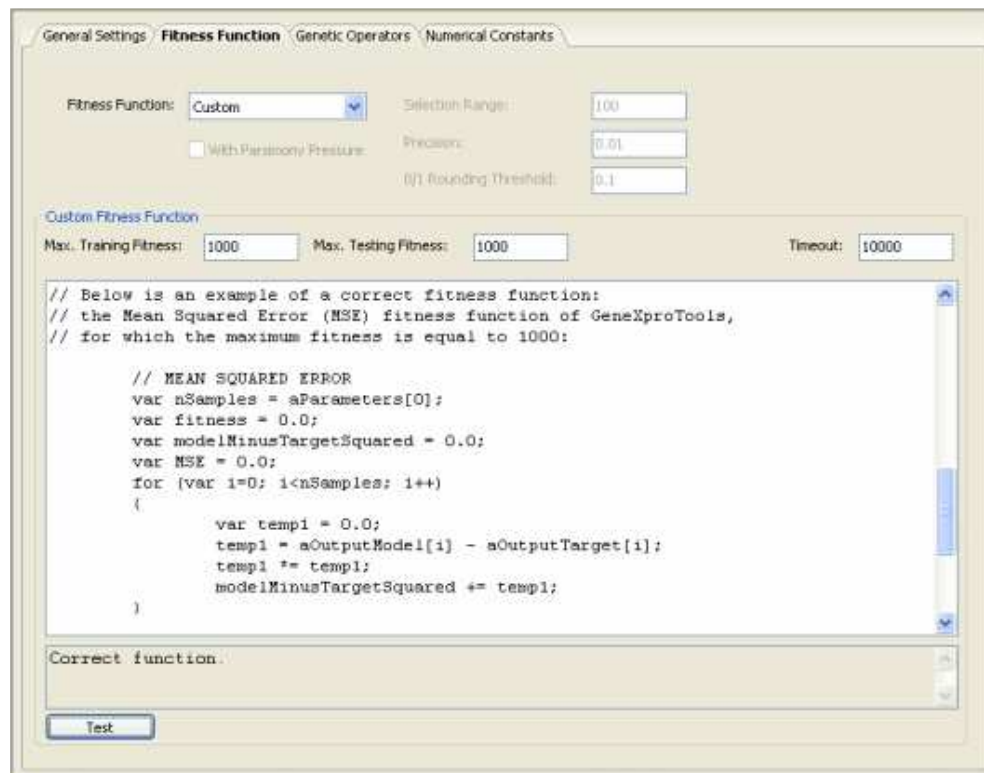
Choosing the Fitness Function

User Defined Fitness Functions

GeneXproTools 4.0 comes equipped with 36 [built-in fitness functions](#) for Function Finding problems and, most probably, you will find among them a suitable fitness function for your problem. But if you

want to try other fitness functions to model your data, GeneXproTools 4.0 allows you to design fitness functions of your own and use them to search the fitness landscape.

You can design your own fitness function using the Custom Fitness Function window.



GeneXproTools 4.0 gives you access to a wide set of essential and useful parameters you may use to design your fitness function:

- `aParameters[0]` = number of samples
- `aParameters[1]` = averaged target output
- `aParameters[2]` = variance of the target output
- `aParameters[5]` = minimum program size
- `aParameters[6]` = maximum program size

In addition, GeneXproTools also gives you access to useful information about the evolving models that are essential for designing custom fitness functions with [parsimony pressure](#):

- `aModelInfo[0]` = program size
- `aModelInfo[1]` = used variables
- `aModelInfo[2]` = number of literals

The code for the custom fitness function must be in **JavaScript** and can be tested before evolving a model with it. Please remember that GeneXproTools 4.0 uses fitness proportionate selection and, therefore, fitness must increase with performance and only non-negative values are acceptable. Below is the sample code of a correct fitness function, which is identical to the [MSE fitness function](#) of GeneXproTools 4.0.

```

////////////////////////////////////////
// All the values of the Target output
// are accessible through the array:

```

```

// aOutputTarget[0] = 2.549
// aOutputTarget[1] = 5.215
// etc.

// All the values of the output of the Model
// are accessible through the array:
// aOutputModel[0] = 78.6945
// aOutputModel[1] = 12.6421
// etc.

// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals

// Your custom fitness function must return a value, for example:
// return fitness;

// Below is an example of a correct fitness function:
// the Mean Squared Error (MSE) fitness function of GeneXproTools,
// for which the maximum fitness is equal to 1000:

// MEAN SQUARED ERROR
var nSamples = aParameters[0];
var fitness = 0.0;
var modelMinusTargetSquared = 0.0;
var MSE = 0.0;
for (var i=0; i<nSamples; i++)
{
    var temp1 = 0.0;
    temp1 = aOutputModel[i] - aOutputTarget[i];
    temp1 *= temp1;
    modelMinusTargetSquared += temp1;
}

MSE = modelMinusTargetSquared / nSamples;

if (MSE <= 0.000000001)
    MSE = 0.0;

fitness = (1/(1+MSE))*1000;

return fitness;

////////////////////////////////////

```

Obviously, in this case, maximum fitness equals 1000 both for the training and testing sets, and you must also feed this information into GeneXproTools through the Custom Fitness Function window in the boxes Max. Training Fitness and Max. Testing Fitness.

Choosing the Fitness Function

Fitness Functions for Classification

For Classification problems you can select one of the 30 built-in fitness functions of GeneXproTools 4.0:

- [Number of Hits](#)
- [Hits with Penalty](#)
- [Accuracy](#)
- [Squared Accuracy](#)
- [Sensitivity/Specificity](#)
- [PPV/NPV](#)
- [SSPN](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [R-square](#)
- [Correlation Coefficient](#)
- And all the above with [parsimony pressure](#)

Or create your own fitness function by choosing:

- [Custom Fitness Function](#)

The screenshot shows the 'Fitness Function' tab in the GeneXproTools 4.0 configuration window. The 'Fitness Functions' dropdown is set to 'Custom'. Below it, there is a checkbox for 'With Parsimony Pressure' which is unchecked. To the right, there are input fields for 'Selection Range' (100), 'Precision' (0.01), and '0/1 Rounding Threshold' (0.1). The 'Custom Fitness Function' section has input fields for 'Max. Training Fitness' (350), 'Max. Testing Fitness' (174), and 'Timeout' (10000). A large text area contains a JavaScript code example for the 'Number of Hits' fitness function. Below the code area is a 'Correct function.' checkbox and a 'Test' button.

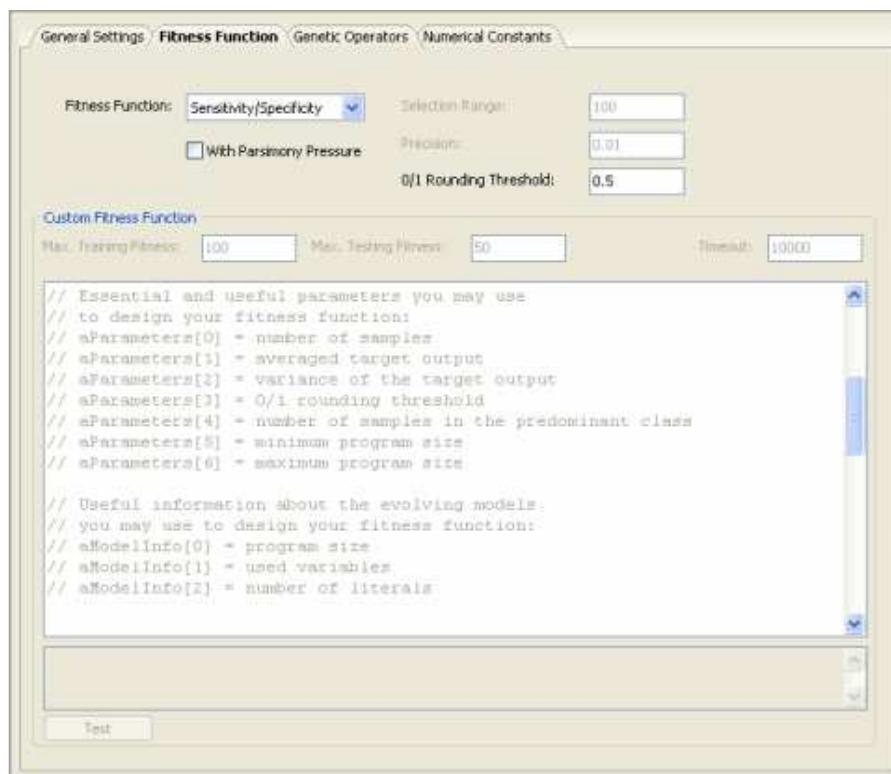
```
// Below is an example of a correct fitness function, more specifically,  
// the Number of Hits fitness function of GeneXproTools, for which  
// the maximum fitness is equal to the number of sample cases:  
  
// NUMBER OF HITS  
var nSamples = aParameters[0];  
var roundingThreshold = aParameters[3];  
var fitness = 0.0;  
for (var i=0; i<nSamples; i++)  
{  
    // Convert the value returned by the model into 1 or 0  
    if (aOutputModel[i] >= roundingThreshold)  
    {  
        aOutputModel[i] = 1;  
    }  
    else  
    {  
        aOutputModel[i] = 0;  
    }  
    fitness += aOutputModel[i];  
}
```



Choosing the Fitness Function

Fitness Functions and Parsimony Pressure

GeneXproTools allows you to select models based not only on their raw fitness but also on their size by choosing a fitness function with parsimony pressure. For that you just have to check the box With Parsimony Pressure on the Fitness Function Tab.



Designing parsimonious solutions makes sense not only in terms of efficiency but also clarity, and modelers always try to balance both these factors. The fitness functions with parsimony pressure of GeneXproTools allow you to achieve the same balance between efficiency and simplicity by favoring good solutions with more compact structures.

GeneXproTools has a total of 15 **fitness functions with parsimony pressure**, finely tuned to allow an efficient evolution:

- [Number of Hits](#)
- [Hits with Penalty](#)
- [Accuracy](#)
- [Squared Accuracy](#)
- [Sensitivity/Specificity](#)
- [PPV/NPV](#)
- [SSPN](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)

- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [R-square](#)
- [Correlation Coefficient](#)

In addition, GeneXproTools 4.0 allows you to play with program size and weight it as you see fit in your own [custom fitness functions](#).



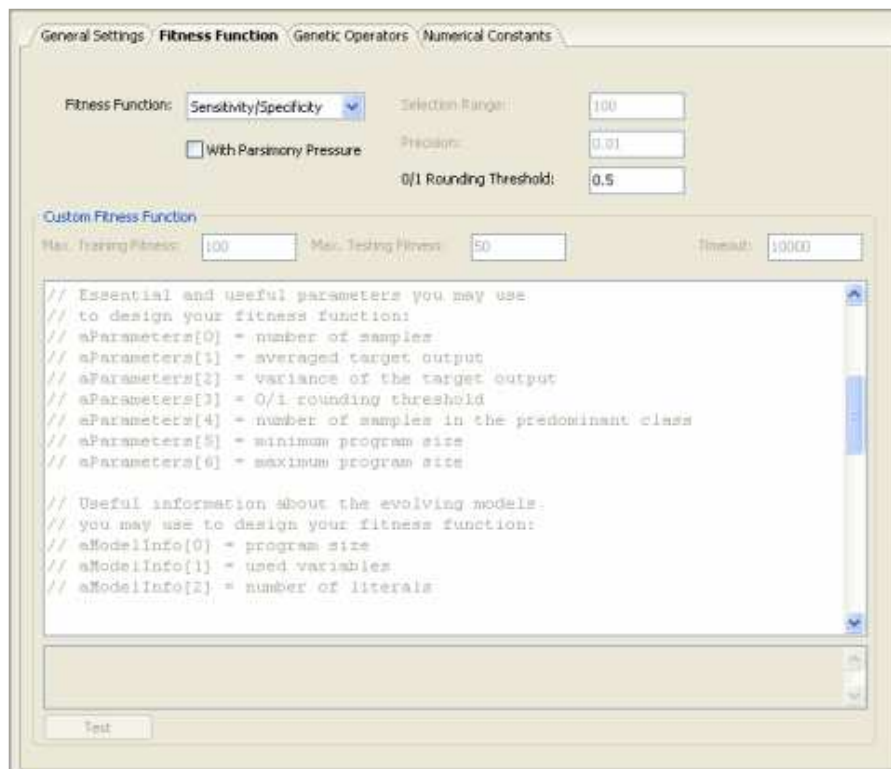
Choosing the Fitness Function

Number of Hits

GeneXproTools 4.0 implements the **Number of Hits** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **Number of Hits** fitness function is very simple and highly efficient, and is based on the number of samples correctly classified.

More formally, the fitness f_i of an individual program corresponds to the number of hits and is evaluated by the formula:

$$f_i = h$$

where h is the number of fitness cases correctly evaluated (number of hits).

So, for this fitness function, maximum fitness $f_{\max}$ is given by:

$$f_{\max} = n$$

where n is the number of fitness cases.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = n$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

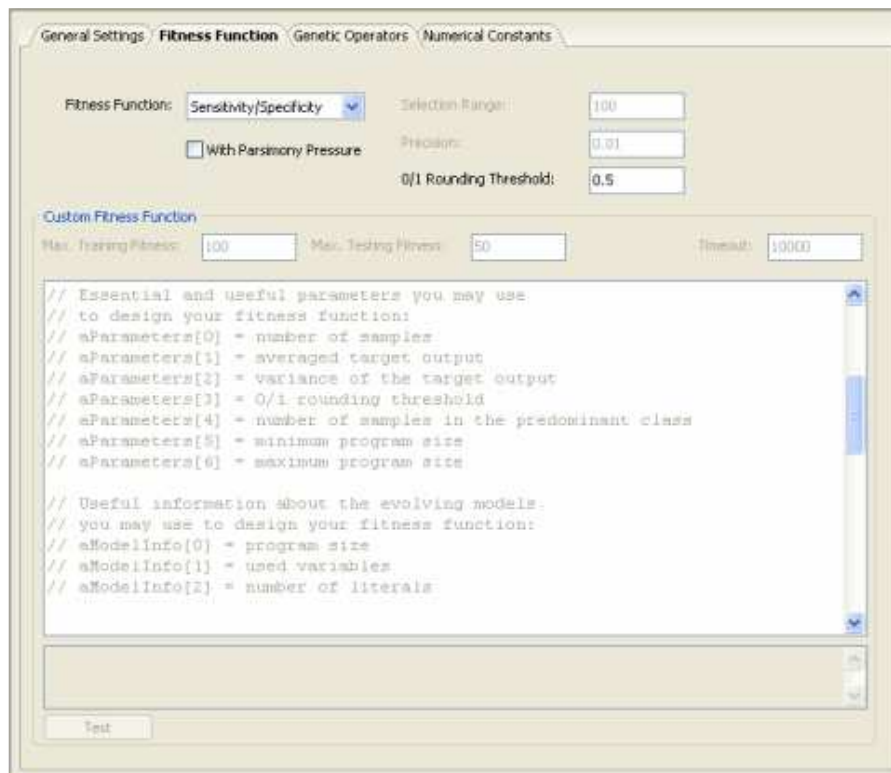
Hits with Penalty

GeneXproTools 4.0 implements the **Hits with Penalty** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the

evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **Hits with Penalty** fitness function is very simple and highly efficient, and is based on the number of samples correctly classified.

More formally, the fitness f_i of an individual program corresponds to the number of hits and is evaluated by the formula:

$$\text{If } (TP_i = 0 \text{ OR } TN_i = 0), \text{ then } f_i = TP_i + TN_i; \text{ else } f_i = 0$$

where TP_i is the number of **true positives** and TN_i is the number of **true negatives**.

So, for this fitness function, maximum fitness $f_{\max}$ is given by:

$$f_{\max} = n$$

where n is the number of fitness cases.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = n$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



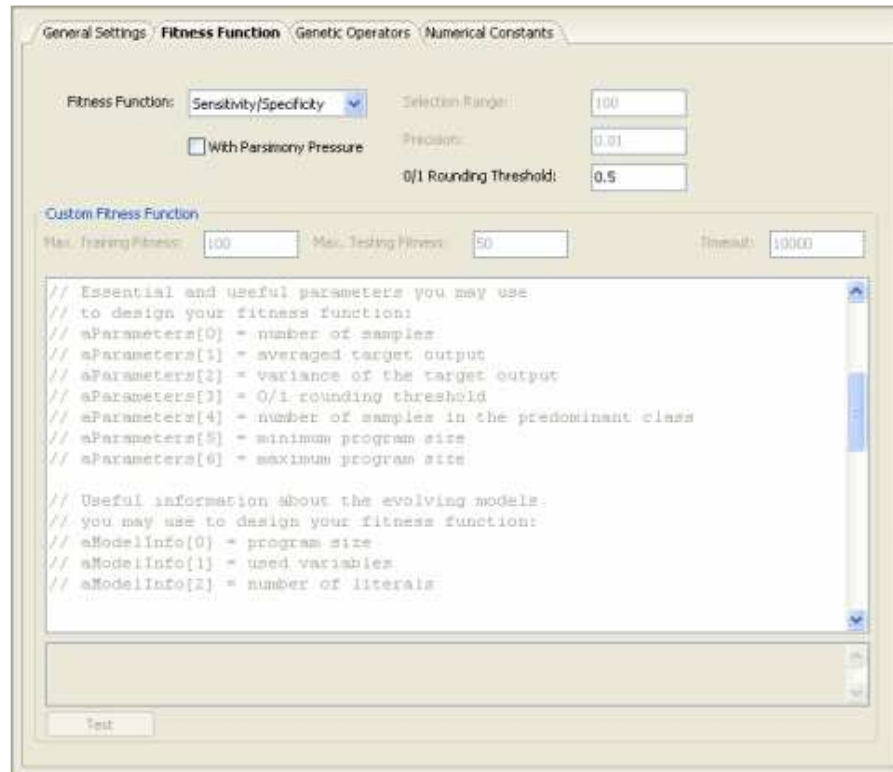
Choosing the Fitness Function

Accuracy

GeneXproTools 4.0 implements the **Accuracy** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **Accuracy** fitness function of GeneXproTools is, as expected, based on the [classification accuracy](#).

The classification accuracy A_i of an individual program i depends on the number of fitness cases correctly classified (**true positives** plus **true negatives**) and is evaluated by the formula:

$$A_i = \frac{t}{n}$$

where t is the number of sample cases correctly classified, and n is the total number of sample cases.

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * A_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



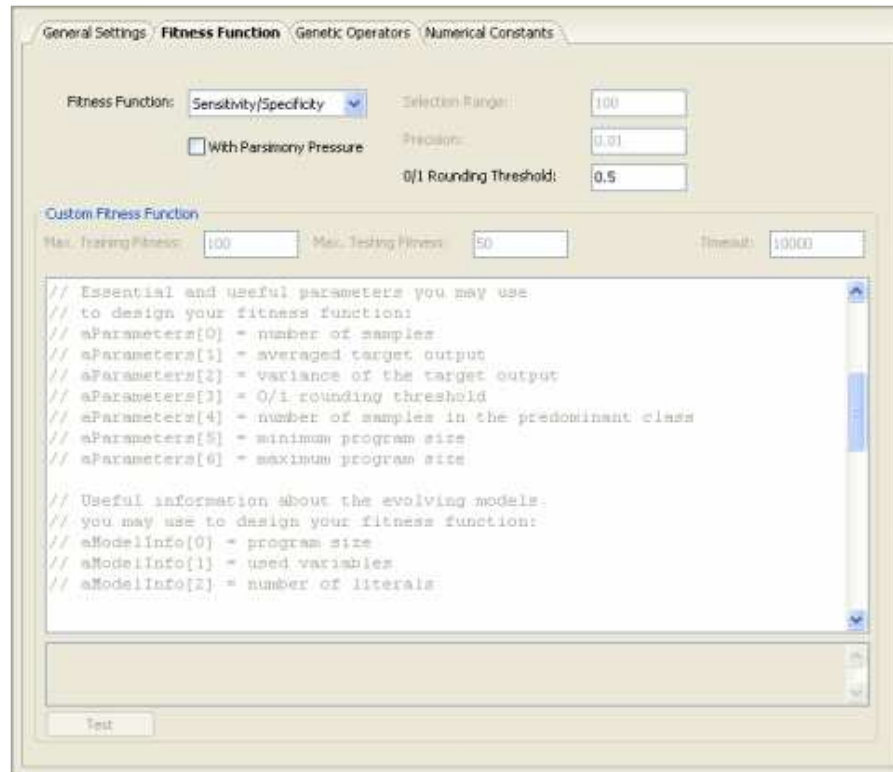
Choosing the Fitness Function

Squared Accuracy

GeneXproTools 4.0 implements the **Squared Accuracy** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **Squared Accuracy** fitness function of GeneXproTools is, as expected, based on the [classification accuracy](#).

The classification accuracy A_i of an individual program i depends on the number of fitness cases correctly classified (**true positives** plus **true negatives**) and is evaluated by the formula:

$$A_i = \frac{t}{n}$$

where t is the number of sample cases correctly classified, and n is the total number of sample cases.

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * A_i * A_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum

program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



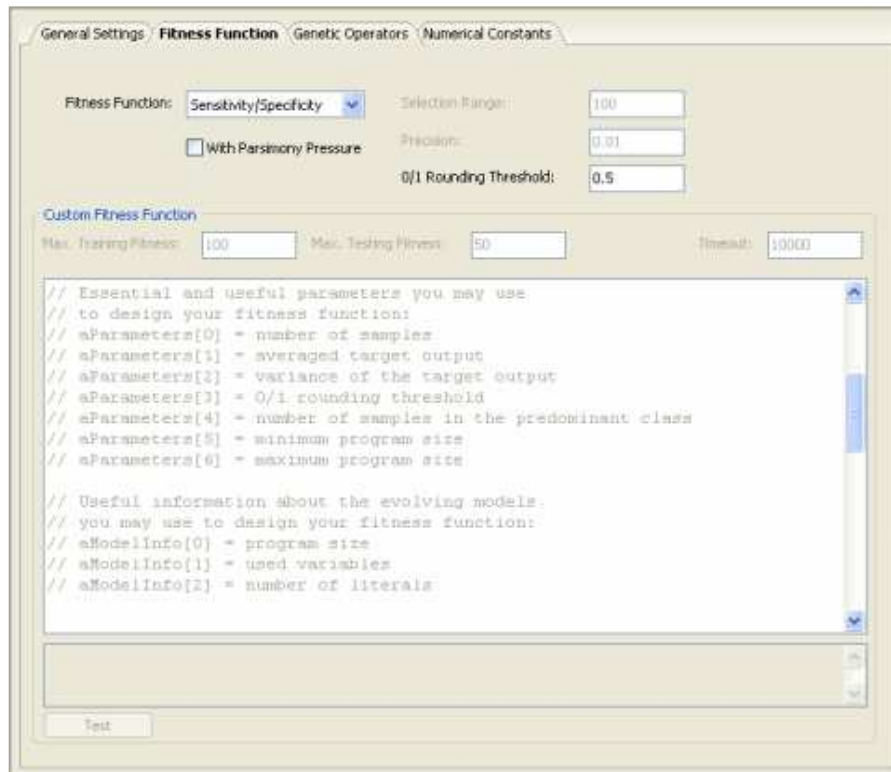
Choosing the Fitness Function

Sensitivity/Specificity

GeneXproTools 4.0 implements the **Sensitivity/Specificity** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **Sensitivity/Specificity** fitness function of GeneXproTools 4.0 is, as expected, based both on the [sensitivity](#) and [specificity](#).

The sensitivity/specificity SS_i of an individual program i is evaluated by the equation:

$$SS_i = SE_i \cdot SP_i$$

where SE_i is the sensitivity and SP_i is the specificity of the individual program i , and are given by the formulas:

$$SE_i = \frac{TP_i}{TP_i + FN_i}$$

$$SP_i = \frac{TN_i}{TN_i + FP_i}$$

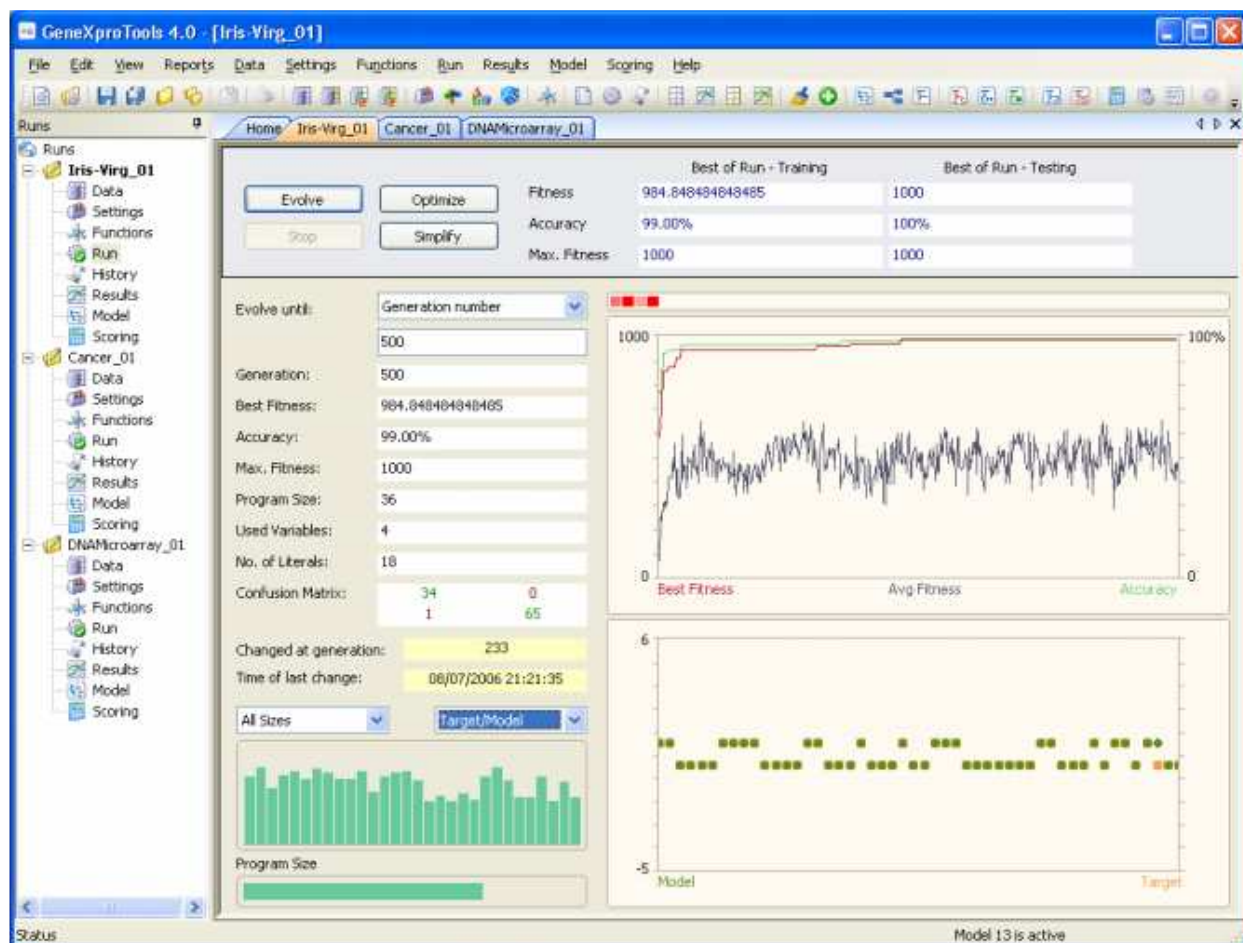
where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually

shown in what is called a **confusion matrix**. And for all classification problems, regardless of the fitness function, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



Thus, for evaluating the fitness f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot SS_i$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and

complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



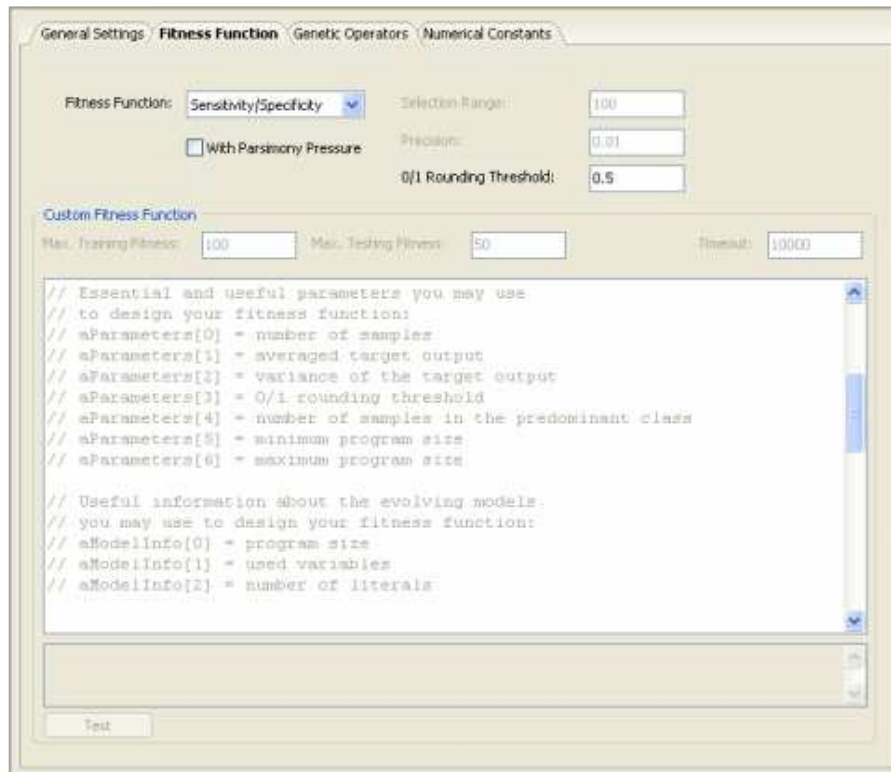
Choosing the Fitness Function

PPV/NPV

GeneXproTools 4.0 implements the **Positive Predictive Value / Negative Predictive Value** (PPV/NPV) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **PPV/NPV** fitness function of GeneXproTools 4.0 is, as expected, based both on the **positive predictive value** (PPV) and **negative predictive value** (NPV).

The PPV/NPV PN_i of an individual program i is evaluated by the equation:

$$PN_i = PPV_i \cdot NPV_i$$

where PPV_i is the positive predictive value and NPV_i is the negative predictive value of the individual program i , and are given by the formulas:

$$PPV_i = \frac{TP_i}{TP_i + FP_i} \quad \text{where } TP_i + FP_i \neq 0$$

$$NPV_i = \frac{TN_i}{TN_i + FN_i} \quad \text{where } TN_i + FN_i \neq 0$$

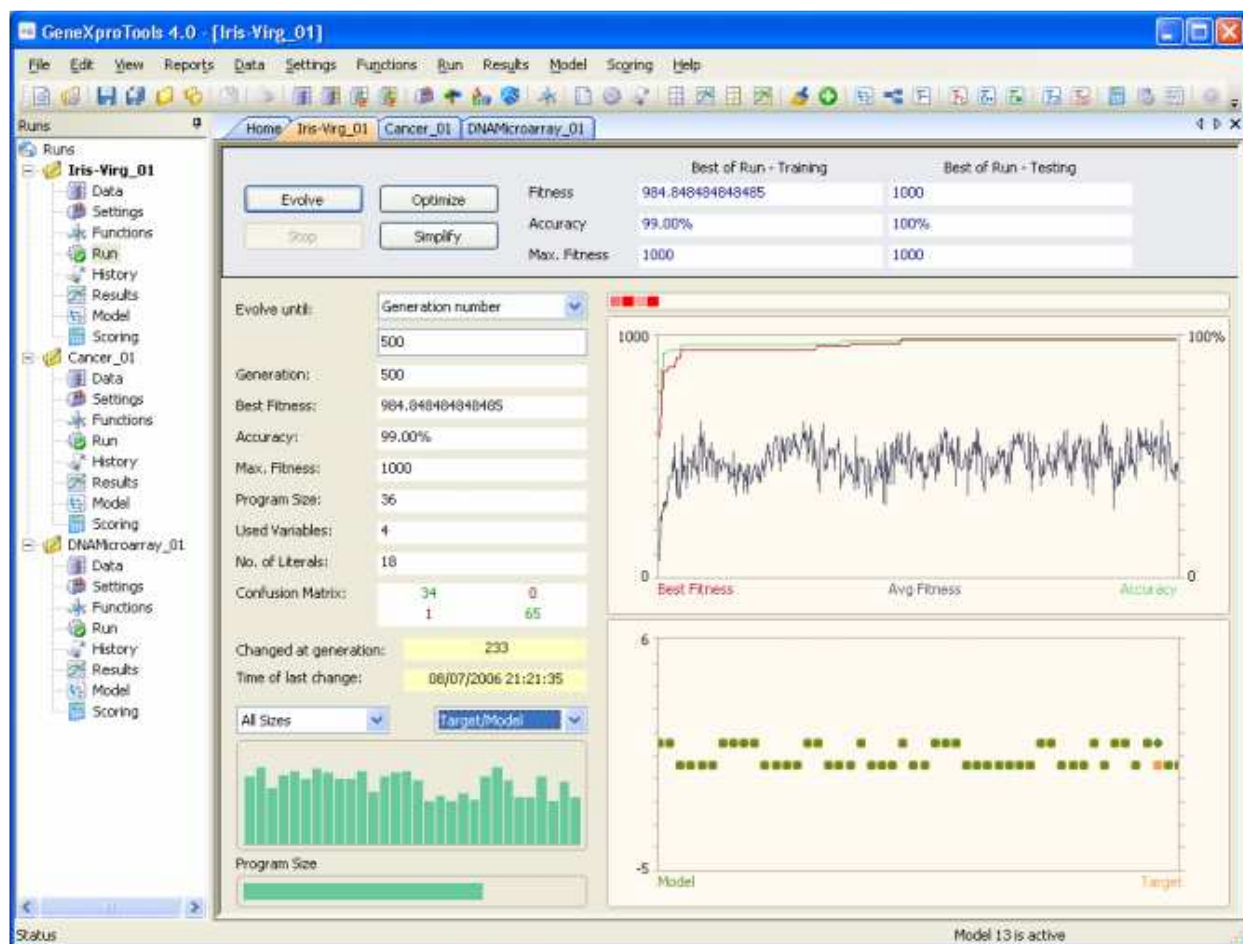
where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually

shown in what is called a **confusion matrix**. And for all classification problems, regardless of the fitness function, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



Thus, for evaluating the fitness f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot PN_i$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and

complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



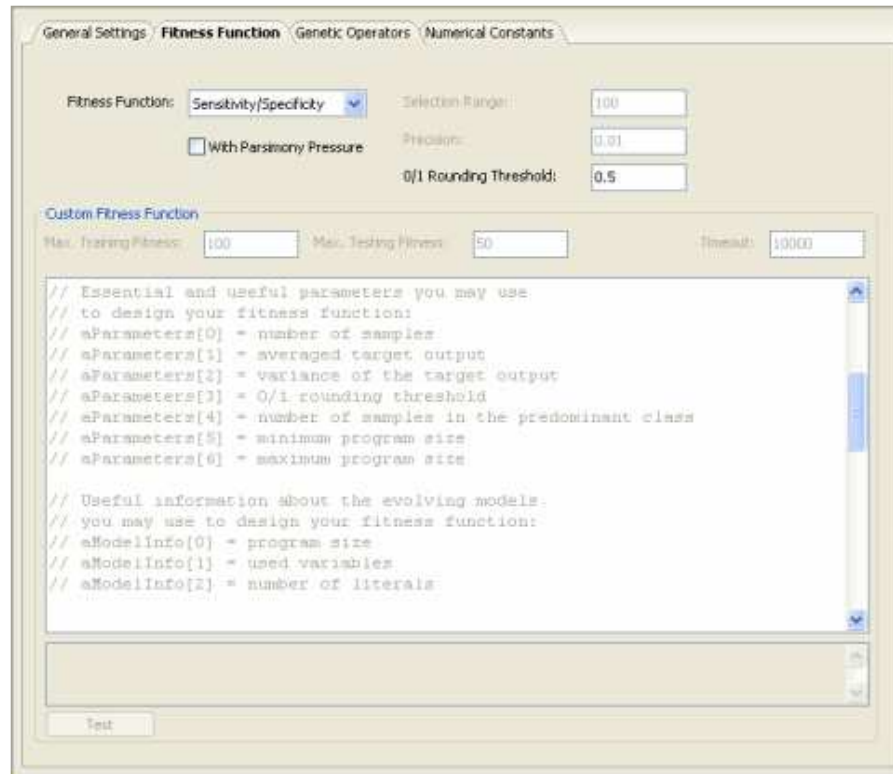
Choosing the Fitness Function

Sensitivity/Specificity/PPV/NPV

GeneXproTools 4.0 implements the **Sensitivity/Specificity/PPV/NPV** (SSPN) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **SSPN** fitness function of GeneXproTools 4.0 is, as expected, based on all the four statistical indexes: **sensitivity**, **specificity**, **positive predictive value (PPV)**, and **negative predictive value (NPV)**.

The sensitivity/specificity/PPV/NPV $SSPN_i$ of an individual program i is evaluated by the equation:

$$SSPN_i = SE_i \cdot SP_i \cdot PPV_i \cdot NPV_i$$

where SE_i is the sensitivity, SP_i the specificity, PPV_i the positive predictive value, and NPV_i the negative predictive value of the individual program i , and are given by the formulas:

$$SE_i = \frac{TP_i}{TP_i + FN_i}$$

$$SP_i = \frac{TN_i}{TN_i + FP_i}$$

$$PPV_i = \frac{TP_i}{TP_i + FP_i} \quad \text{where } TP_i + FP_i \neq 0$$

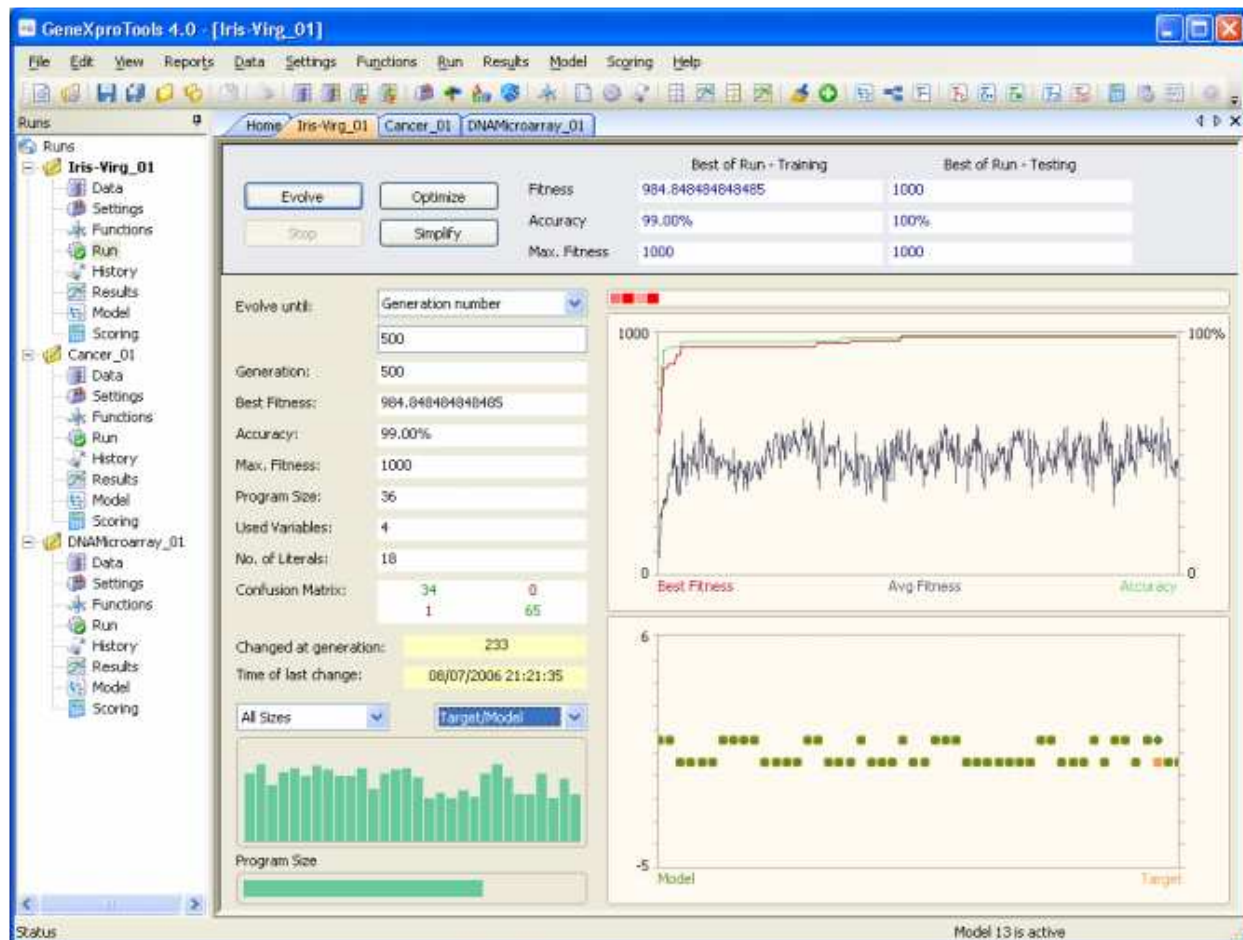
$$NPV_i = \frac{TN_i}{TN_i + FN_i} \quad \text{where } TN_i + FN_i \neq 0$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all classification problems, regardless of the fitness function, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



Thus, for evaluating the fitness f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot SSPN_i$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



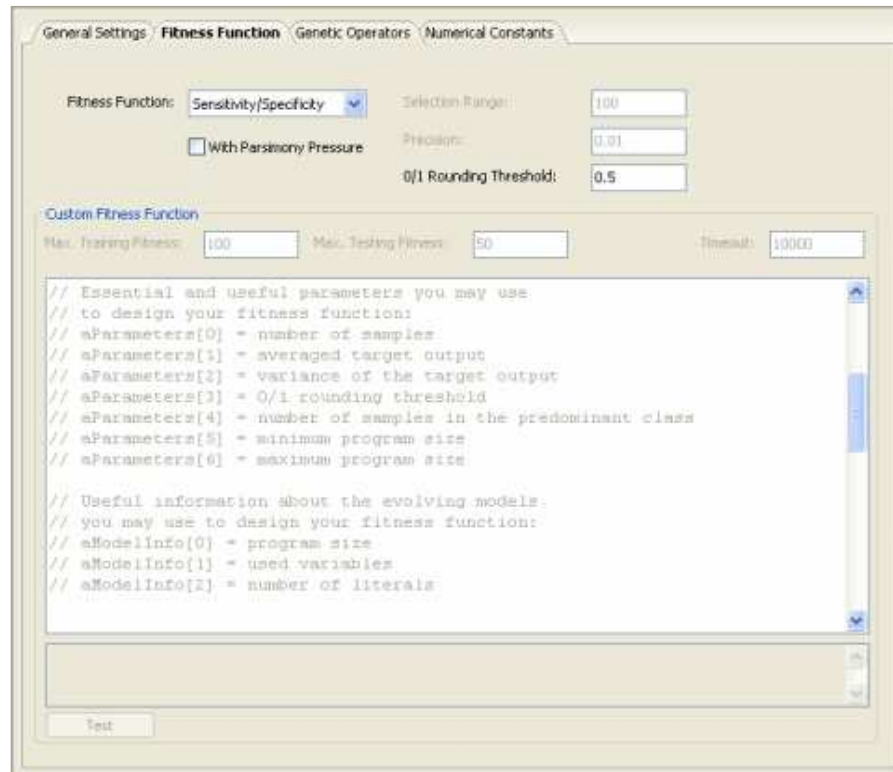
Choosing the Fitness Function

Mean Squared Error

GeneXproTools 4.0 implements the **Mean Squared Error** (MSE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **MSE** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [mean squared error](#).

The mean squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the MSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



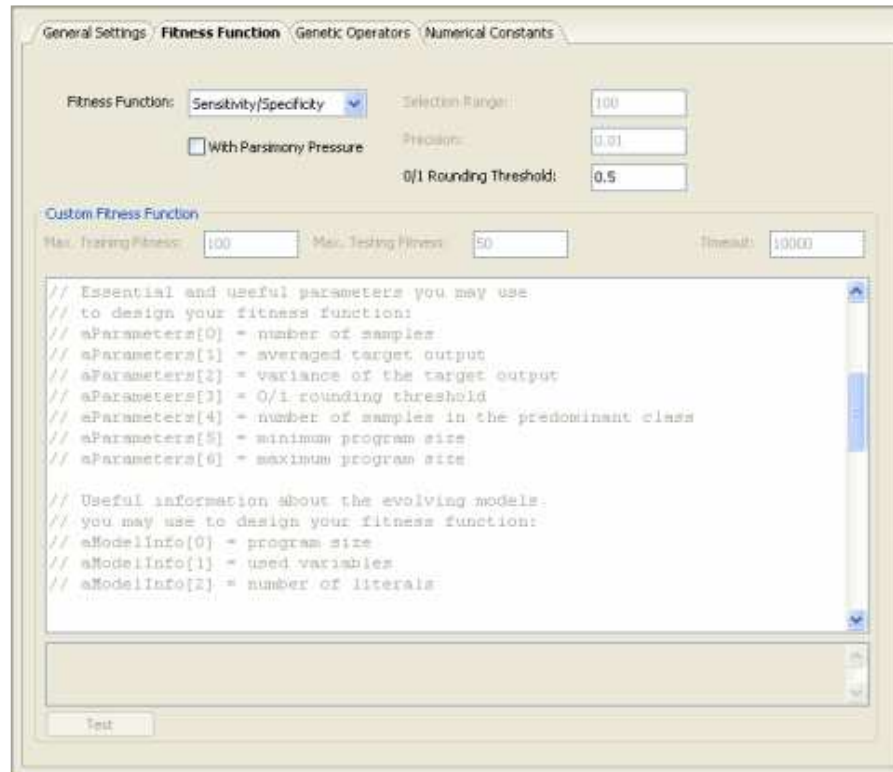
Choosing the Fitness Function

Root Mean Squared Error

GeneXproTools 4.0 implements the **Root Mean Squared Error** (RMSE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **RMSE** fitness function of GeneXproTools is, as expected, based on the standard [root mean squared error](#). By taking the square root of the [mean squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

The root mean squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the RMSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and

complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



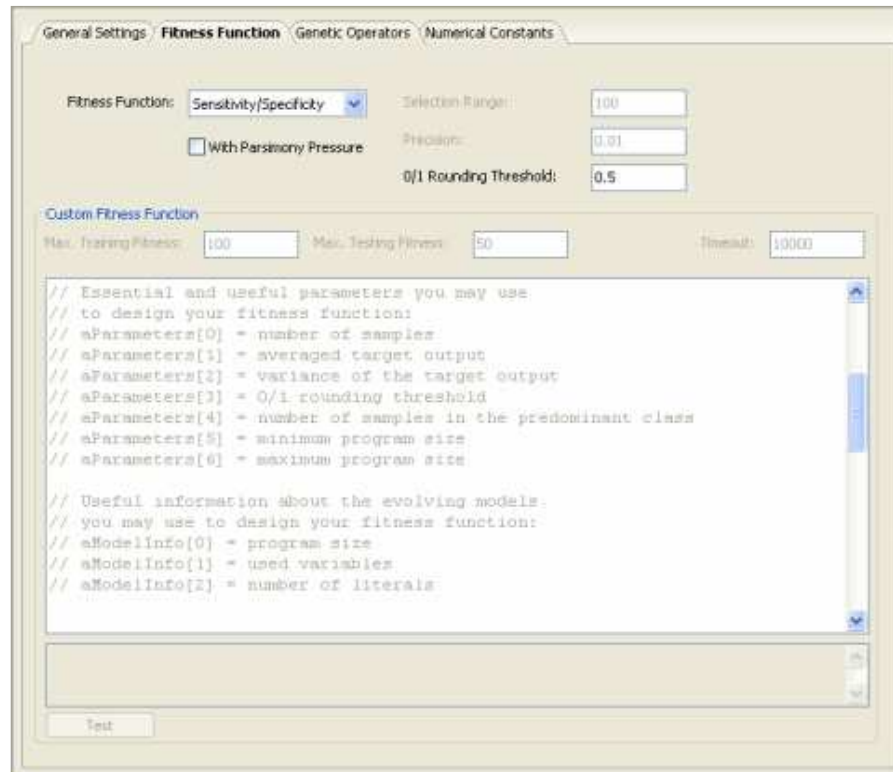
Choosing the Fitness Function

Mean Absolute Error

GeneXproTools 4.0 implements the **Mean Absolute Error** (MAE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **MAE** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [mean absolute error](#).

The mean absolute error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the MAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



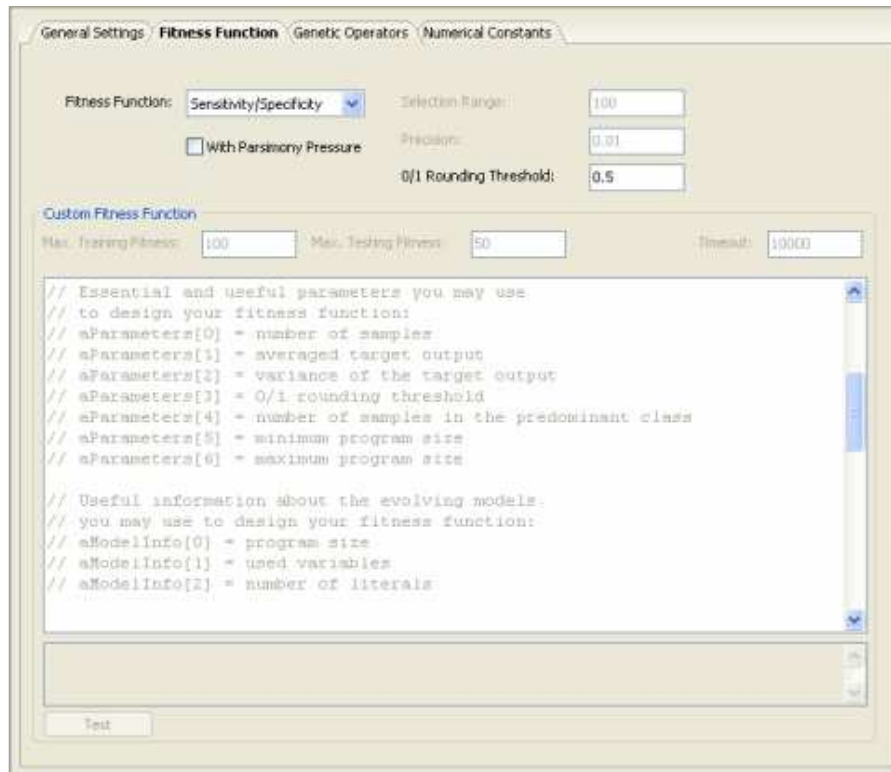
Choosing the Fitness Function

Relative Squared Error

GeneXproTools 4.0 implements the **Relative Squared Error** (RSE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **RSE** fitness function of GeneXproTools is, as expected, based on the standard [relative squared error](#).

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the relative squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

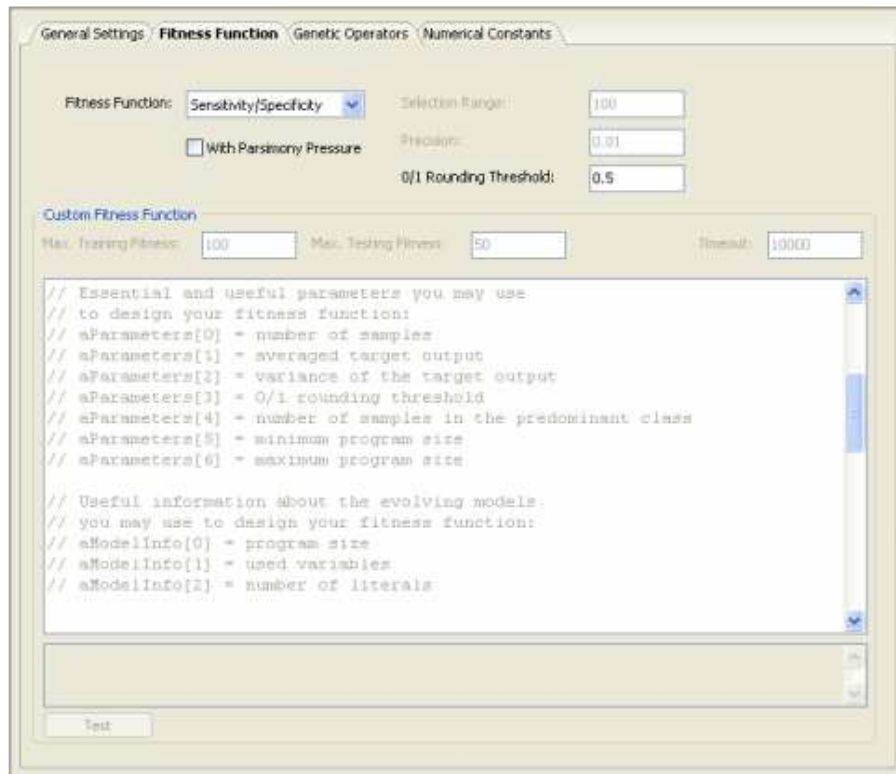
Root Relative Squared Error

GeneXproTools 4.0 implements the **Root Relative Squared Error** (RRSE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and

must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **RRSE** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [root relative squared error](#).

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the root relative squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RRSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

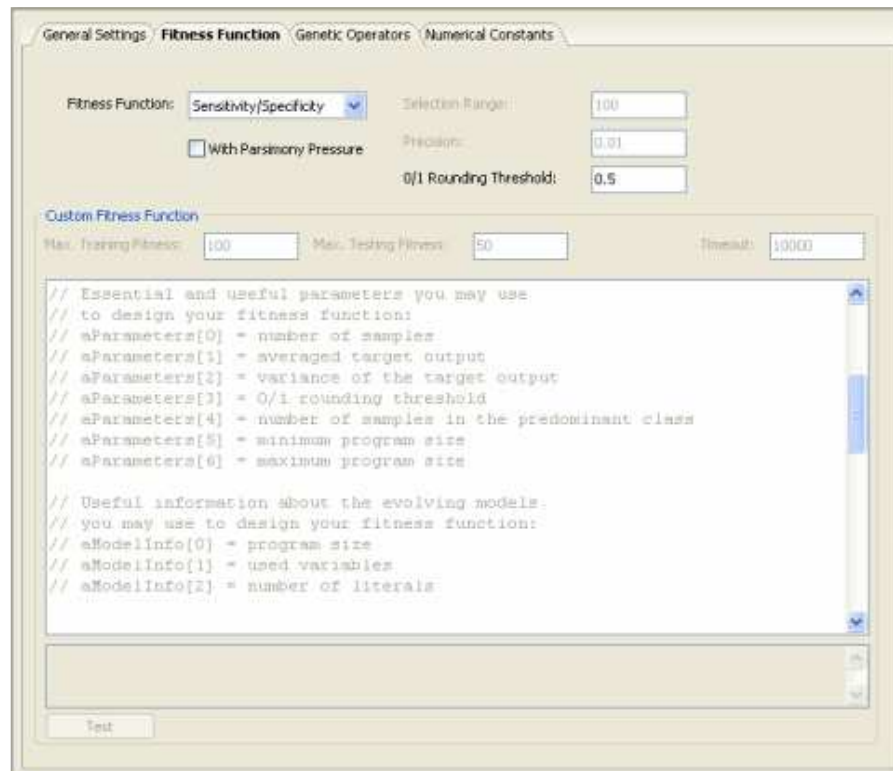
Relative Absolute Error

GeneXproTools 4.0 implements the **Relative Absolute Error** (RAE) fitness function both **with** and **without** parsimony pressure. The version **with parsimony pressure** puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning

algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **RAE** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [relative absolute error](#).

The relative absolute error is very similar to the [relative squared error](#) in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the relative absolute error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n |P_{(ij)} - T_j|}{\sum_{j=1}^n |T_j - \bar{T}|}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



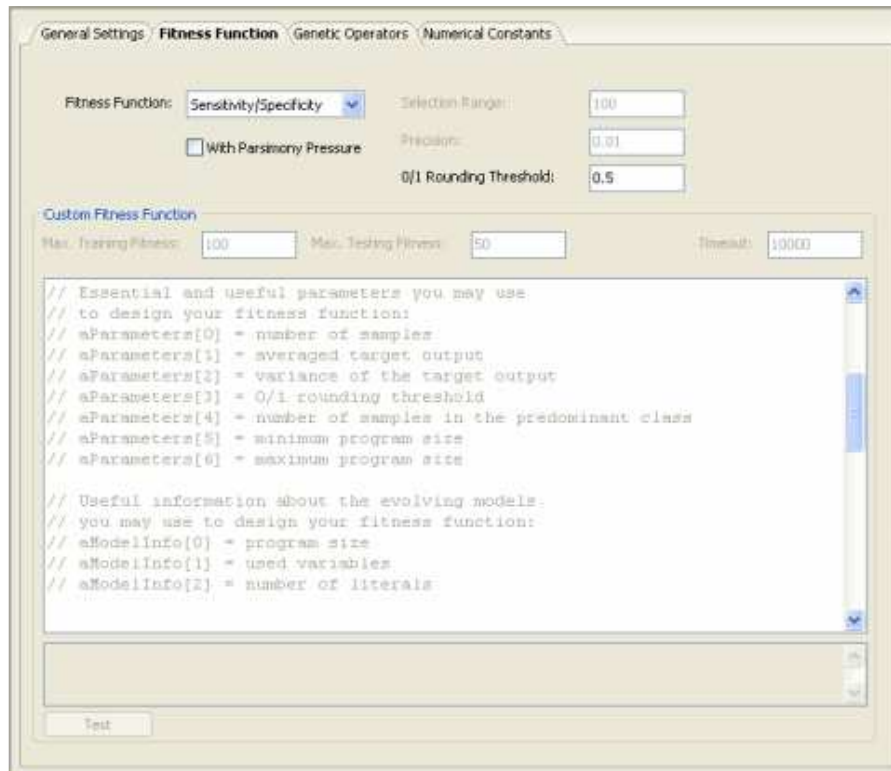
Choosing the Fitness Function

R-square

GeneXproTools 4.0 implements the **R-square** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **R-square** fitness function is, as expected, based on the standard [R-square](#), which returns the square of the Pearson product moment correlation coefficient.

The Pearson product moment correlation coefficient is a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between two data sets.

The Pearson product moment correlation coefficient R_i of an individual program i is evaluated by the equation:

$$R_i = \frac{n \sum_{j=1}^n (T_j P_{(ij)}) - \left(\sum_{j=1}^n T_j \right) \left(\sum_{j=1}^n P_{(ij)} \right)}{\sqrt{\left[n \sum_{j=1}^n T_j^2 - \left(\sum_{j=1}^n T_j \right)^2 \right] \left[n \sum_{j=1}^n P_{(ij)}^2 - \left(\sum_{j=1}^n P_{(ij)} \right)^2 \right]}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

and T_j is the target value for fitness case j .

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * R_i * R_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



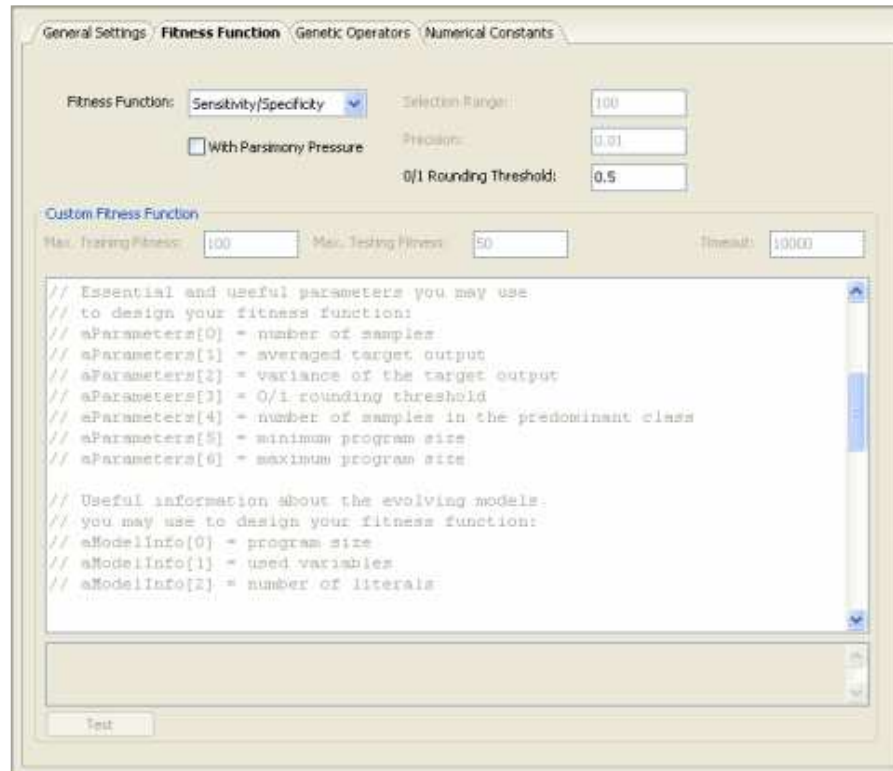
Choosing the Fitness Function

Correlation Coefficient

GeneXproTools 4.0 implements the **Correlation Coefficient** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.



The **Correlation Coefficient** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [correlation coefficient](#), which is a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between the predicted values and the target values.

The correlation coefficient C_i of an individual program i is evaluated by the equation:

$$C_i = \frac{Cov(T, P)}{\sigma_t \cdot \sigma_p}$$

where $Cov(T, P)$ is the covariance of the target and model outputs; and σ_t and σ_p are the corresponding standard deviations, which are given by:

$$Cov(T, P) = \frac{1}{n} \sum_{j=1}^n (T_j - \bar{T})(P_{(ij)} - \bar{P})$$

$$\sigma_t = \sqrt{\frac{\sum_{j=1}^n (T_j - \bar{T})^2}{n}}$$

$$\sigma_p = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - \bar{P})^2}{n}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n fitness cases or

sample cases); T_j is the target value for fitness case j ; and $\bar{T}$ and $\bar{P}$ are given by the formulas:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

$$\bar{P} = \frac{1}{n} \sum_{j=1}^n P_{(j)}$$

The correlation coefficient is confined to the range $[-1, 1]$. When $C_i = 1$, there is a perfect positive linear correlation between T and P , that is, they vary by the same amount. When $C_i = -1$, there is a perfect negative linear correlation between T and P , that is, they vary in opposite ways (when T increases, P decreases by the same amount). When $C_i = 0$, there is no correlation between T and P . Intermediate values describe partial correlations and the closer to 1 or -1 the better the model.

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * C_i * C_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

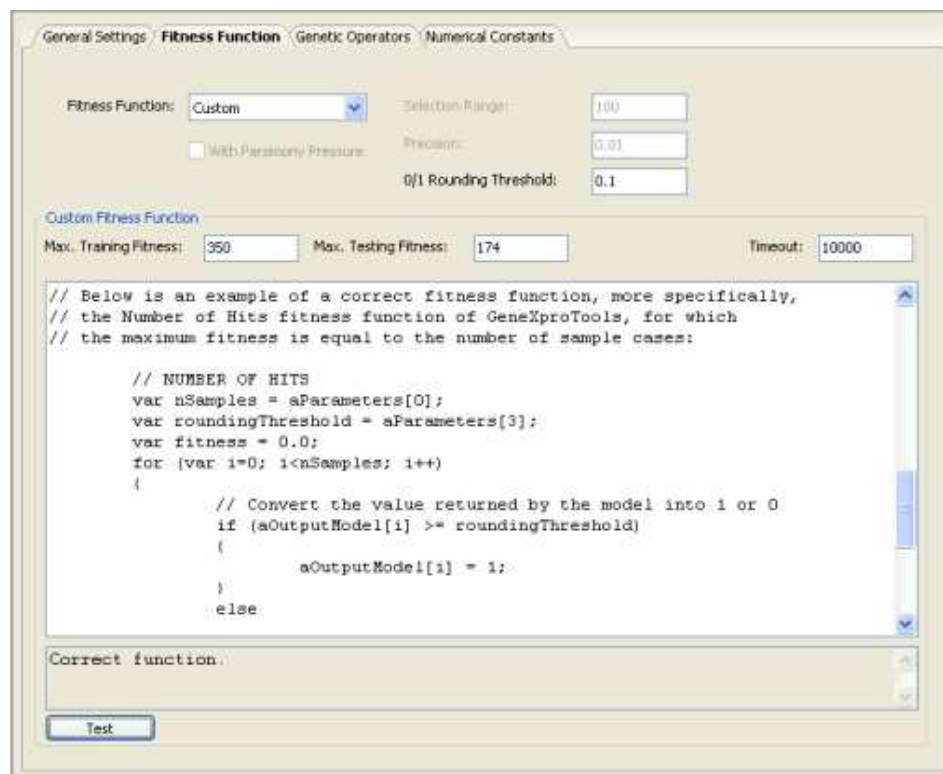
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$

Choosing the Fitness Function

User Defined Fitness Functions

GeneXproTools 4.0 comes equipped with 30 [built-in fitness functions](#) for Classification problems and, most probably, you will find among them a suitable fitness function for your problem. But if you want to try other fitness functions to model your data, GeneXproTools 4.0 allows you to design fitness functions of your own and use them to search the fitness landscape.

You can design your own fitness function using the Custom Fitness Function window.



For all classification problems, in order to be able to apply a particular fitness function, the learning algorithms of GeneXproTools 4.0 must convert the value returned by the evolved model into "1" or "0" using the **0/1 Rounding Threshold**. If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Thus, the 0/1 Rounding Threshold is an integral part of all fitness functions used for classification and must be appropriately set in the Settings Panel -> Fitness Function Tab.

GeneXproTools 4.0 gives you access to a wide set of essential and useful parameters you may use to design your fitness function:

- `aParameters[0]` = number of samples
- `aParameters[1]` = averaged target output
- `aParameters[2]` = variance of the target output
- `aParameters[3]` = 0/1 rounding threshold
- `aParameters[4]` = number of samples in the predominant class
- `aParameters[5]` = minimum program size
- `aParameters[6]` = maximum program size

In addition, GeneXproTools also gives you access to useful information about the evolving models that are essential for designing custom fitness functions with [parsimony pressure](#):

- aModelInfo[0] = program size
- aModelInfo[1] = used variables
- aModelInfo[2] = number of literals

The code for the custom fitness function must be in **JavaScript** and can be tested before evolving a model with it. Please remember that GeneXproTools 4.0 uses fitness proportionate selection and, therefore, fitness must increase with performance and only non-negative values are acceptable. Below is the sample code of a correct fitness function, which is identical to the [number of hits](#) fitness function of GeneXproTools 4.0.

```

////////////////////////////////////////
// All the values of the Target output
// are accessible through the array:
// aOutputTarget[0] = 0
// aOutputTarget[1] = 1
// aOutputTarget[2] = 0
// etc.

// All the values of the output of the Model
// are accessible through the array:
// aOutputModel[0] = 0
// aOutputModel[1] = 1
// aOutputModel[2] = 1
// etc.

// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[3] = 0/1 rounding threshold
// aParameters[4] = number of samples in the predominant class
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals

// Your custom fitness function must return a value, for example:
// return fitness;

// Below is an example of a correct fitness function, more specifically,
// the Number of Hits fitness function of GeneXproTools, for which
// the maximum fitness is equal to the number of sample cases:

// NUMBER OF HITS
var nSamples = aParameters[0];
var roundingThreshold = aParameters[3];
var fitness = 0.0;
for (var i=0; i<nSamples; i++)
{
    // Convert the value returned by the model into 1 or 0
    if (aOutputModel[i] >= roundingThreshold)
    {
        aOutputModel[i] = 1;
    }
}

```

```

    }
    else
    {
        aOutputModel[i] = 0;
    }
    // Evaluate fitness
    if (aOutputModel[i] == aOutputTarget[i])
    {
        fitness++;
    }
}

return fitness;

////////////////////////////////////

```

Obviously, in this case, maximum fitness equals 1000 both for the training and testing sets, and you must also feed this information into GeneXproTools through the Custom Fitness Function window in the boxes Max. Training Fitness and Max. Testing Fitness.



Choosing the Fitness Function

Fitness Functions for Time Series Prediction

For Time Series Prediction problems, the fitness functions of GeneXproTools 4.0 are organized as follows:

Fitness functions based on the absolute error:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)

Fitness functions based on the relative error:

- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)
- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)

Fitness functions based on correlation coefficients:

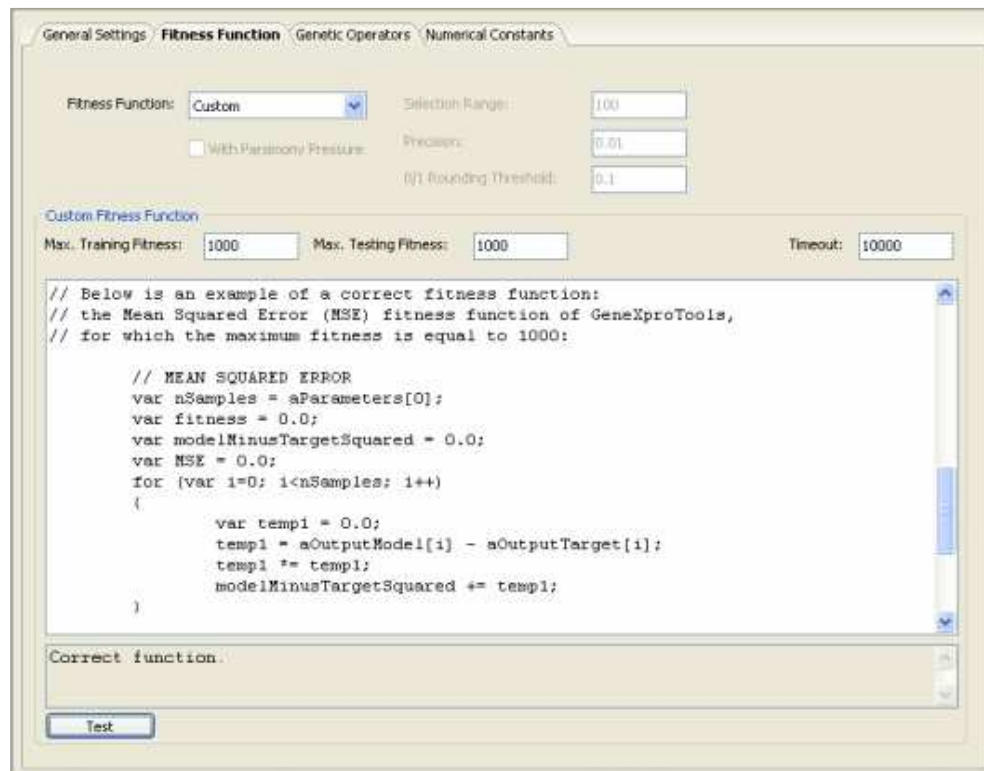
- [R-square](#)
- [Correlation Coefficient](#)

Fitness functions with parsimony pressure:

- And all the above with [parsimony pressure](#)

And user defined fitness functions:

- [Custom Fitness Function](#)

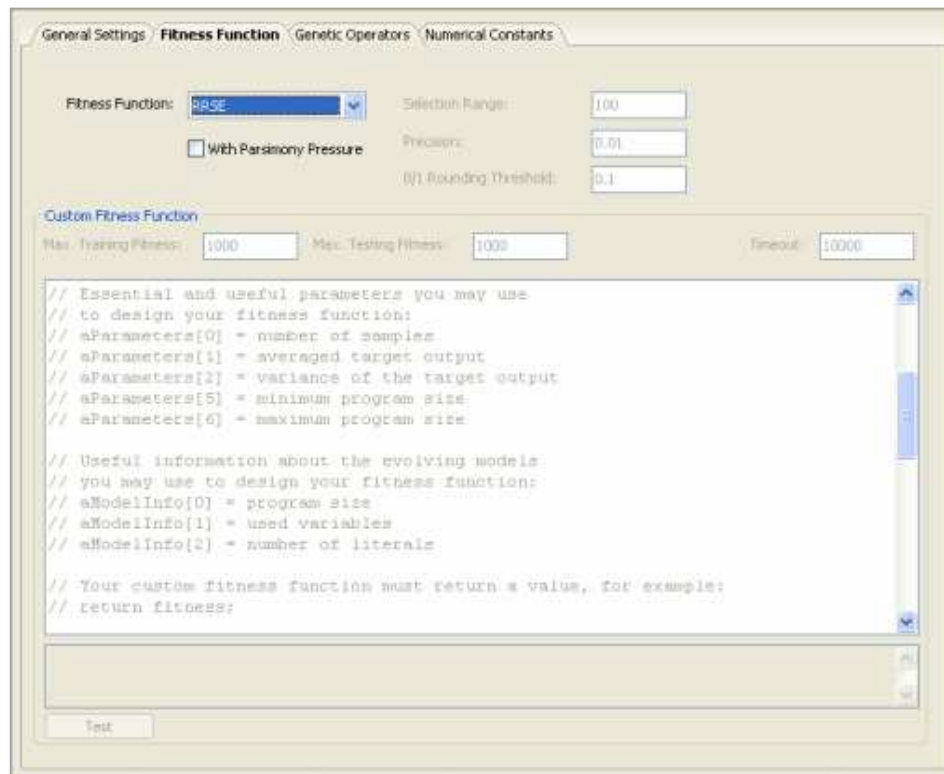


◀ | ▶

Choosing the Fitness Function

Fitness Functions and Parsimony Pressure

GeneXproTools allows you to select models based not only on their raw fitness but also on their size by choosing a fitness function with parsimony pressure. For that you just have to check the box With Parsimony Pressure on the Fitness Function Tab.



Designing parsimonious solutions makes sense not only in terms of efficiency but also clarity, and modelers always try to balance both these factors. The fitness functions with parsimony pressure of GeneXproTools allow you to achieve the same balance between efficiency and simplicity by favoring good solutions with more compact structures.

GeneXproTools has a total of 15 **fitness functions with parsimony pressure**, finely tuned to allow an efficient evolution:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)
- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)
- [R-square](#)
- [Correlation Coefficient](#)

In addition, GeneXproTools 4.0 allows you to play with program size and weight it as you see fit in your own [custom fitness functions](#).

Choosing the Fitness Function

Absolute Error Based Fitness Functions

GeneXproTools 4.0 has a total of 16 fitness functions based on the **absolute error**:

- [Absolute Error with Selection Range](#)
- [Absolute/Hits](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- All the above with [parsimony pressure](#)



Choosing the Fitness Function

Absolute Error with Selection Range

GeneXproTools 4.0 implements the **Absolute Error With Selection Range** (Absolute with SR) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **Absolute with SR** fitness function explores the idea of a selection range and a precision. The **selection range** is used as a limit for selection to operate, above which the performance of a program on a particular fitness case contributes nothing to its fitness. And the **precision** is the limit for improvement, as it allows the fine-tuning of the evolved solutions as accurately as possible.

Mathematically, the **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = \sum_{j=1}^n (R - |P_{(ij)} - T_j|)$$

where R is the selection range, $P_{(ij)}$ the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases) and T_j is the target value for fitness case j . Note that the absolute value term corresponds to the **absolute error**. This term is what is called the **precision** and if the error is smaller than or equal to the precision then the error becomes zero. Thus, for a good match, the absolute value term is zero and $f_i = f_{\max} = nR$.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = nR$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Absolute/Hits

GeneXproTools 4.0 implements the **Absolute/Hits** fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For some Function Finding problems it is important to evolve a model that performs well for all fitness cases within a certain absolute error (the **precision**) of the correct value.

For those cases, the **fitness** $f_{(ij)}$ of an individual program i for fitness case j is evaluated by the formula:

$$\text{If } E_{(ij)} \leq p, \text{ then } f_{(ij)} = 1; \text{ else } f_{(ij)} = 0$$

where p is the precision and $E_{(ij)}$ is the **absolute error** of an individual program i for fitness case j evaluated by the equation:

$$E_{(ij)} = |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j and T_j is the target value for fitness case j .

And overall, the **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = \sum_{j=1}^n f_{ij}$$

where n is the total number of fitness cases.

So, for this fitness function, maximum fitness $f_{\max}$ is given by:

$$f_{\max} = n$$

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = n$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Squared Error

GeneXproTools 4.0 implements the **Mean Squared Error** (MSE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **MSE** fitness function of GeneXproTools is, as expected, based on the standard [mean squared error](#), which, on its turn, is based on the **absolute error**.

The mean squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the MSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Mean Squared Error

GeneXproTools 4.0 implements the **Root Mean Squared Error** (RMSE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **RMSE** fitness function of GeneXproTools is, as expected, based on the standard [root mean squared error](#). By taking the square root of the [mean squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

The root mean squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the RMSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are

composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Absolute Error

GeneXproTools 4.0 implements the **Mean Absolute Error** (MAE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **MAE** fitness function of GeneXproTools is, as expected, based on the standard [mean absolute error](#), which, on its turn, is based on the **absolute error**.

The mean absolute error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the MAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal (the coefficient 1000 allows a fairer distribution of fitnesses for selection).

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Squared Error

GeneXproTools 4.0 implements the **Relative Squared Error** (RSE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **RSE** fitness function of GeneXproTools is, as expected, based on the standard [relative squared error](#), which, on its turn, is based on the **absolute error**.

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the relative squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Relative Squared Error

GeneXproTools 4.0 implements the **Root Relative Squared Error** (RRSE) fitness function both with

and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **RRSE** fitness function of GeneXproTools is, as expected, based on the standard [root relative squared error](#), which, on its turn, is based on the **absolute error**.

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the root relative squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RRSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Absolute Error

GeneXproTools 4.0 implements the **Relative Absolute Error** (RAE) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **RAE** fitness function of GeneXproTools is, as expected, based on the standard [relative absolute error](#), which, on its turn, is based on the **absolute error**.

The relative absolute error is very similar to the [relative squared error](#) in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the relative absolute error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n |P_{ij} - T_j|}{\sum_{j=1}^n |T_j - \bar{T}|}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Error Based Fitness Functions

GeneXproTools 4.0 has a total of 16 fitness functions based on the **relative error**:

- [Relative Error with Selection Range](#)
- [Relative/Hits](#)
- [rMSE](#) (relative MSE)

- [rRMSE](#) (relative RMSE)
- [rMAE](#) (relative MAE)
- [rRSE](#) (relative RSE)
- [rRRSE](#) (relative RRSE)
- [rRAE](#) (relative RAE)
- All the above with [parsimony pressure](#)



Choosing the Fitness Function

Relative Error with Selection Range

GeneXproTools 4.0 implements the **Relative Error With Selection Range** (Relative with SR) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **Relative with SR** fitness function explores the idea of a selection range and a precision. The **selection range** is used as a limit for selection to operate, above which the performance of a program on a particular fitness case contributes nothing to its fitness. And the **precision** is the limit for improvement, as it allows the fine-tuning of the evolved solutions as accurately as possible.

Mathematically, the **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = \sum_{j=1}^n \left(R - \left| \frac{P_{(ij)} - T_j}{T_j} \cdot 100 \right| \right)$$

where R is the selection range, $P_{(ij)}$ the value predicted by the individual program i for fitness case j (out of n fitness cases) and T_j is the target value for fitness case j . Note that the absolute value term corresponds to the **relative error**. This term is what is called the **precision** and if the error is smaller than or equal to the precision then the error becomes zero. Thus, for a good match the absolute value term is zero and $f_i = f_{\max} = nR$.

Note that when the target value for a particular fitness case is zero, the relative error is undefined but, for fitness evaluation purposes during training, GeneXproTools 4.0 salvages these cases so that they can also be used for fine-tuning solutions.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = nR$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative/Hits

GeneXproTools 4.0 implements the **Relative/Hits** fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

For some Time Series Prediction problems it is important to evolve a model that performs well for all fitness cases within a certain relative error (the **precision**) of the correct value.

For those cases, the **fitness** $f_{(ij)}$ of an individual program i for fitness case j is evaluated by the formula:

$$\text{If } E_{(ij)} \leq p, \text{ then } f_{(ij)} = 1; \text{ else } f_{(ij)} = 0$$

where p is the precision and $E_{(ij)}$ is the relative error of an individual program i for fitness case j evaluated by the equation:

$$E_{(ij)} = \left| \frac{P_{(ij)} - T_j}{T_j} \cdot 100 \right|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j and T_j is the target value for fitness case j .

And overall, the **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = \sum_{j=1}^n f_{(ij)}$$

where n is the total number of fitness cases.

So, for this fitness function, maximum fitness $f_{\max}$ is given by:

$$f_{\max} = n$$

Note that when the target value for a particular fitness case is zero, the relative error is undefined but, for fitness evaluation purposes during training, GeneXproTools 4.0 salvages these cases so that they can also be used for fine-tuning solutions.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = n$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Squared Error

GeneXproTools 4.0 implements the **Mean Squared Error** (rMSE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rMSE** fitness function of GeneXproTools is, as expected, based on the standard [mean squared error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The rMSE E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n \left(\frac{P_{(ij)} - T_j}{T_j} \right)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the rMSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Mean Squared Error

GeneXproTools 4.0 implements the **Root Mean Squared Error** (rRMSE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rRMSE** fitness function of GeneXproTools is, as expected, based on the standard [root mean squared error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure. By taking the square root of the [mean squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

The rRMSE E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n \left(\frac{P_{(ij)} - T_j}{T_j} \right)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the rRMSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the

linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Absolute Error

GeneXproTools 4.0 implements the **Mean Absolute Error** (rMAE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rMAE** fitness function of GeneXproTools is, as expected, based on the standard [mean absolute error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The rMAE E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n \left| \frac{P_{(ij)} - T_j}{T_j} \right|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the rMAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal (the coefficient 1000 allows a fairer distribution of fitnesses for selection).

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with

parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Squared Error

GeneXproTools 4.0 implements the **Relative Squared Error** (rRSE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rRSE** fitness function of GeneXproTools is, as expected, based on the standard [relative squared error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the rRSE E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n \left(\frac{P_{(j)} - T_j}{T_j} \right)^2}{\sum_{j=1}^n \left(\frac{T_j - \bar{T}}{\bar{T}} \right)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the rRSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Relative Squared Error

GeneXproTools 4.0 implements the **Root Relative Squared Error** (rRRSE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rRRSE** fitness function of GeneXproTools is, as expected, based on the standard [root relative squared error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the rRRSE E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^n \left(\frac{P_{ij} - T_j}{T_j} \right)^2}{\sum_{j=1}^n \left(\frac{T_j - \bar{T}}{\bar{T}} \right)^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the rRRSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Absolute Error

GeneXproTools 4.0 implements the **Relative Absolute Error** (rRAE, with the small "r" indicating that it is based on the relative error rather than the absolute) fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **rRAE** fitness function of GeneXproTools is, as expected, based on the standard [relative absolute error](#), which is usually based on the absolute error, but obviously the **relative error** can also be used in order to create a slightly different fitness measure.

The relative absolute error is very similar to the [relative squared error](#) in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the rRAE E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n \left| \frac{P_{(ij)} - T_j}{T_j} \right|}{\sum_{j=1}^n \left| \frac{T_j - \bar{T}}{\bar{T}} \right|}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the rRAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

R-square

GeneXproTools 4.0 implements the **R-square** fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **R-square** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [R-square](#), which returns the square of the Pearson product moment correlation coefficient.

The Pearson product moment correlation coefficient is a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between the predicted values and the target values.

The Pearson product moment correlation coefficient R_i of an individual program i is evaluated by the equation:

$$R_i = \frac{n \sum_{j=1}^n (T_j P_{(ij)}) - \left(\sum_{j=1}^n T_j \right) \left(\sum_{j=1}^n P_{(ij)} \right)}{\sqrt{\left[n \sum_{j=1}^n T_j^2 - \left(\sum_{j=1}^n T_j \right)^2 \right] \left[n \sum_{j=1}^n P_{(ij)}^2 - \left(\sum_{j=1}^n P_{(ij)} \right)^2 \right]}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases or sample cases); and T_j is the target value for fitness case j .

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * R_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



Choosing the Fitness Function

Correlation Coefficient

GeneXproTools 4.0 implements the **Correlation Coefficient** fitness function both with and without parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **Correlation Coefficient** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [correlation coefficient](#), which is a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between the predicted values and the target values.

The correlation coefficient C_i of an individual program i is evaluated by the equation:

$$C_i = \frac{Cov(T, P)}{\sigma_t \cdot \sigma_p}$$

where $Cov(T, P)$ is the covariance of the target and model outputs; and σ_t and σ_p are the corresponding standard deviations, which are given by:

$$Cov(T, P) = \frac{1}{n} \sum_{j=1}^n (T_j - \bar{T})(P_{(ij)} - \bar{P})$$

$$\sigma_t = \sqrt{\frac{\sum_{j=1}^n (T_j - \bar{T})^2}{n}}$$

$$\sigma_p = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - \bar{P})^2}{n}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n fitness cases or sample cases); T_j is the target value for fitness case j ; and $\bar{T}$ and $\bar{P}$ are given by the formulas:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

$$\bar{P} = \frac{1}{n} \sum_{j=1}^n P_{(j)}$$

The correlation coefficient is confined to the range $[-1, 1]$. When $C_i = 1$, there is a perfect positive linear correlation between T and P , that is, they vary by the same amount. When $C_i = -1$, there is a perfect negative linear correlation between T and P , that is, they vary in opposite ways (when T increases, P decreases by the same amount). When $C_i = 0$, there is no correlation between T and P . Intermediate values describe partial correlations and the closer to 1 or -1 the better the model.

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * C_i * C_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{500000000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.000000002 \cdot rf_{\max}$$



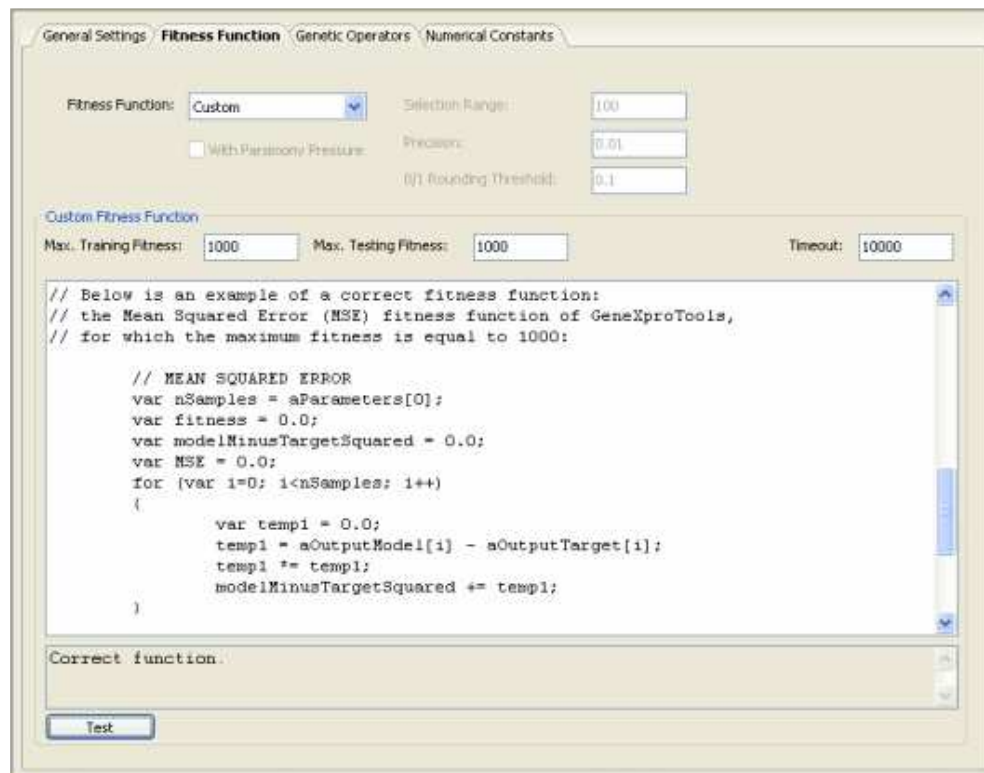
Choosing the Fitness Function

User Defined Fitness Functions

GeneXproTools 4.0 comes equipped with 36 [built-in fitness functions](#) for Time Series Prediction problems and, most probably, you will find among them a suitable fitness function for your problem.

But if you want to try other fitness functions to model your data, GeneXproTools 4.0 allows you to design fitness functions of your own and use them to search the fitness landscape.

You can design your own fitness function using the Custom Fitness Function window.



GeneXproTools 4.0 gives you access to a wide set of essential and useful parameters you may use to design your fitness function:

- `aParameters[0]` = number of samples
- `aParameters[1]` = averaged target output
- `aParameters[2]` = variance of the target output
- `aParameters[5]` = minimum program size
- `aParameters[6]` = maximum program size

In addition, GeneXproTools also gives you access to useful information about the evolving models that are essential for designing custom fitness functions with [parsimony pressure](#):

- `aModelInfo[0]` = program size
- `aModelInfo[1]` = used variables
- `aModelInfo[2]` = number of literals

The code for the custom fitness function must be in **JavaScript** and can be tested before evolving a model with it. Please remember that GeneXproTools 4.0 uses fitness proportionate selection and, therefore, fitness must increase with performance and only non-negative values are acceptable. Below is the sample code of a correct fitness function, which is identical to the [MSE fitness function](#) of GeneXproTools 4.0.

```
// All the values of the Target output
// are accessible through the array:
// aOutputTarget[0] = 2.549
```

```

// aOutputTarget[1] = 5.215
// etc.

// All the values of the output of the Model
// are accessible through the array:
// aOutputModel[0] = 78.6945
// aOutputModel[1] = 12.6421
// etc.

// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals

// Your custom fitness function must return a value, for example:
// return fitness;

// Below is an example of a correct fitness function:
// the Mean Squared Error (MSE) fitness function of GeneXproTools,
// for which the maximum fitness is equal to 1000:

// MEAN SQUARED ERROR
var nSamples = aParameters[0];
var fitness = 0.0;
var modelMinusTargetSquared = 0.0;
var MSE = 0.0;
for (var i=0; i<nSamples; i++)
{
    var temp1 = 0.0;
    temp1 = aOutputModel[i] - aOutputTarget[i];
    temp1 *= temp1;
    modelMinusTargetSquared += temp1;
}

MSE = modelMinusTargetSquared / nSamples;

if (MSE <= 0.000000001)
    MSE = 0.0;

fitness = (1/(1+MSE))*1000;

return fitness;

```

Obviously, in this case, maximum fitness equals 1000, both for the training and testing data, and you must also feed this information to GeneXproTools through the Custom Fitness Function window in the boxes Max. Training Fitness and Max. Testing Fitness.

Choosing the Fitness Function

Fitness Functions for Logic Synthesis

For Logic Synthesis problems you can select one of the 30 built-in fitness functions of GeneXproTools 4.0:

- [Number of Hits](#)
- [Hits with Penalty](#)
- [Accuracy](#)
- [Squared Accuracy](#)
- [Sensitivity/Specificity](#)
- [PPV/NPV](#)
- [SSPN](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)
- [R-square](#)
- [Correlation Coefficient](#)
- And all the above with [parsimony pressure](#)

Or create your own fitness function by choosing:

- [Custom Fitness Function](#)

The screenshot shows the 'Fitness Function' tab in the GeneXproTools 4.0 configuration window. The 'Fitness Function' dropdown is set to 'Custom'. The 'Selection Range' is 100, 'Precision' is 0.01, and '0/1 Rounding Threshold' is 0.1. The 'With Parsimony Pressure' checkbox is unchecked. The 'Custom Fitness Function' section shows 'Max. Training Fitness' as 120, 'Max. Testing Fitness' as 1928, and 'Timeout' as 10000. A text area contains a JavaScript example for counting hits. Below the text area, it says 'Correct function.' and there is a 'Test' button.

General Settings **Fitness Function** Genetic Operators Numerical Constants

Fitness Function: Custom Selection Range: 100
☐ With Parsimony Pressure Precision: 0.01
0/1 Rounding Threshold: 0.1

Custom Fitness Function

Max. Training Fitness: 120 Max. Testing Fitness: 1928 Timeout: 10000

```
// Below is an example of a correct fitness function, more specifically,  
// the Number of Hits fitness function of GeneXproTools, for which  
// the maximum fitness is equal to the number of sample cases:  
  
// NUMBER OF HITS  
var nSamples = aParameters[0];  
var fitness = 0.0;  
for (var i=0; i<nSamples; i++)  
{  
    // Evaluate fitness  
    if (aOutputModel[i] == aOutputTarget[i])  
    {  
        fitness++;  
    }  
}
```

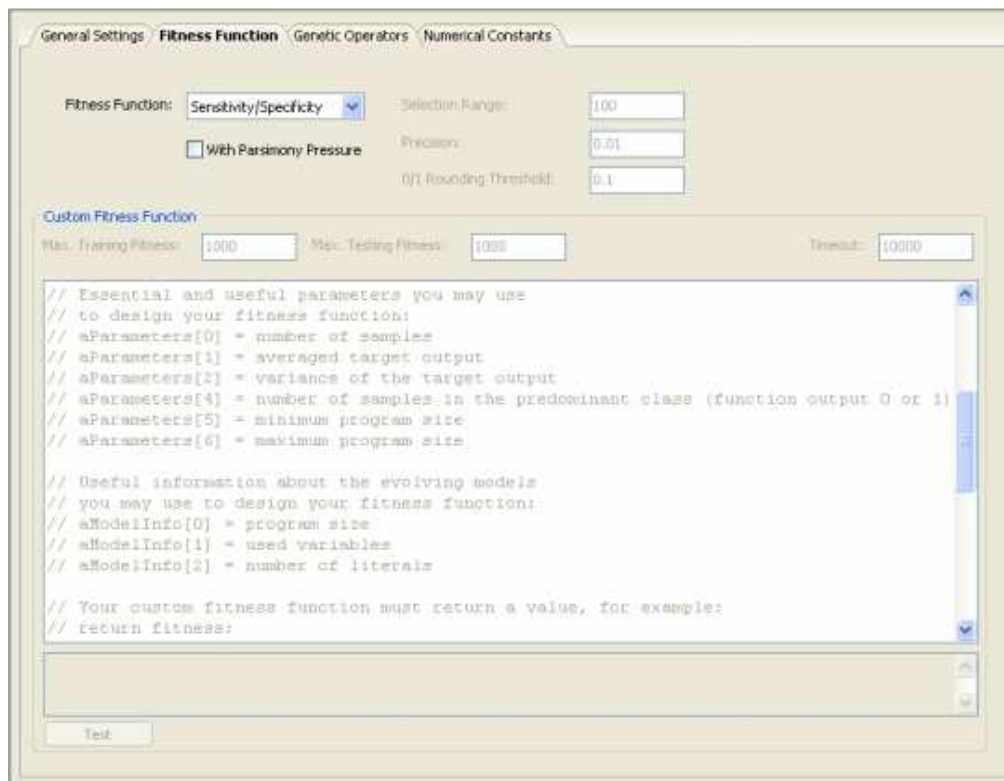
Correct function.

Test

Choosing the Fitness Function

Fitness Functions and Parsimony Pressure

GeneXproTools allows you to select models based not only on their raw fitness but also on their size by choosing a fitness function with parsimony pressure. For that you just have to check the box With Parsimony Pressure on the Fitness Function Tab.



Designing parsimonious solutions makes sense not only in terms of efficiency but also clarity, and modelers always try to balance both these factors. The fitness functions with parsimony pressure of GeneXproTools allow you to achieve the same balance between efficiency and simplicity by favoring good solutions with more compact structures.

GeneXproTools has a total of 15 **fitness functions with parsimony pressure**, finely tuned to allow an efficient evolution:

- [Number of Hits](#)
- [Hits with Penalty](#)
- [Accuracy](#)
- [Squared Accuracy](#)
- [Sensitivity/Specificity](#)
- [PPV/NPV](#)
- [SSPN](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)

- [RAE](#) (relative absolute error)
- [R-square](#)
- [Correlation Coefficient](#)

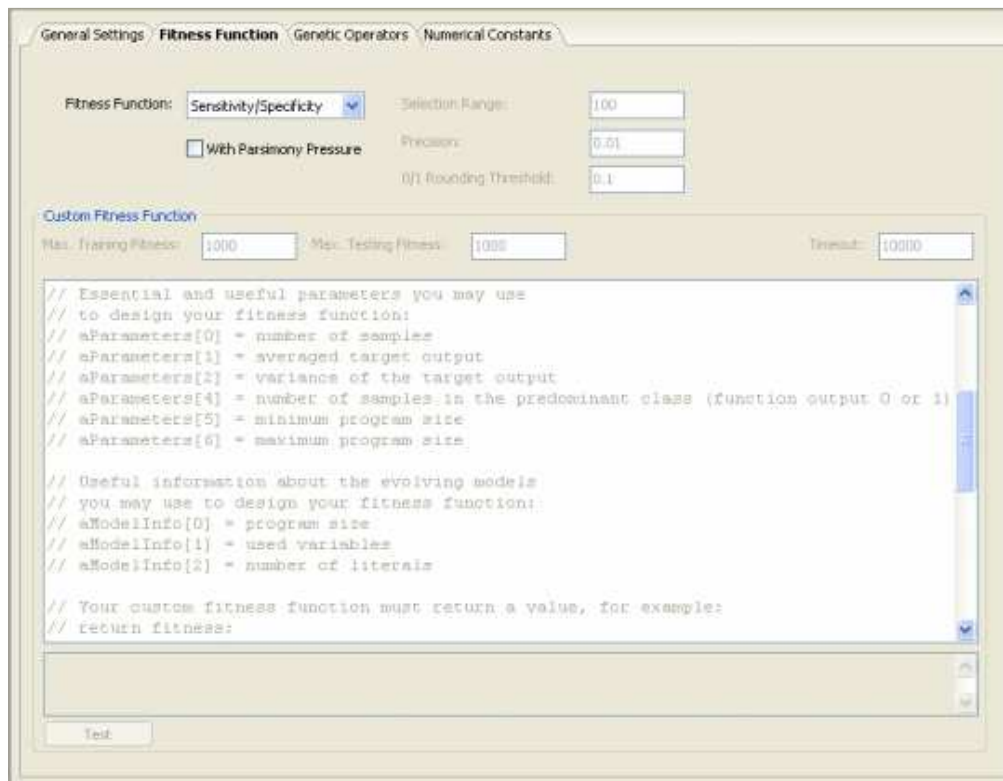
In addition, GeneXproTools 4.0 allows you to play with program size and weight it as you see fit in your own [custom fitness functions](#).



Choosing the Fitness Function

Number of Hits

GeneXproTools 4.0 implements the **Number of Hits** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **Number of Hits** fitness function is very simple and highly efficient, and is based on the number of samples correctly classified.

More formally, the fitness f_i of an individual program corresponds to the number of hits and is evaluated by the formula:

$$f_i = h$$

where h is the number of fitness cases correctly evaluated (number of hits).

So, for this fitness function, maximum fitness $f_{\max}$ is given by:

$$f_{\max} = n$$

where n is the number of fitness cases.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = n$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

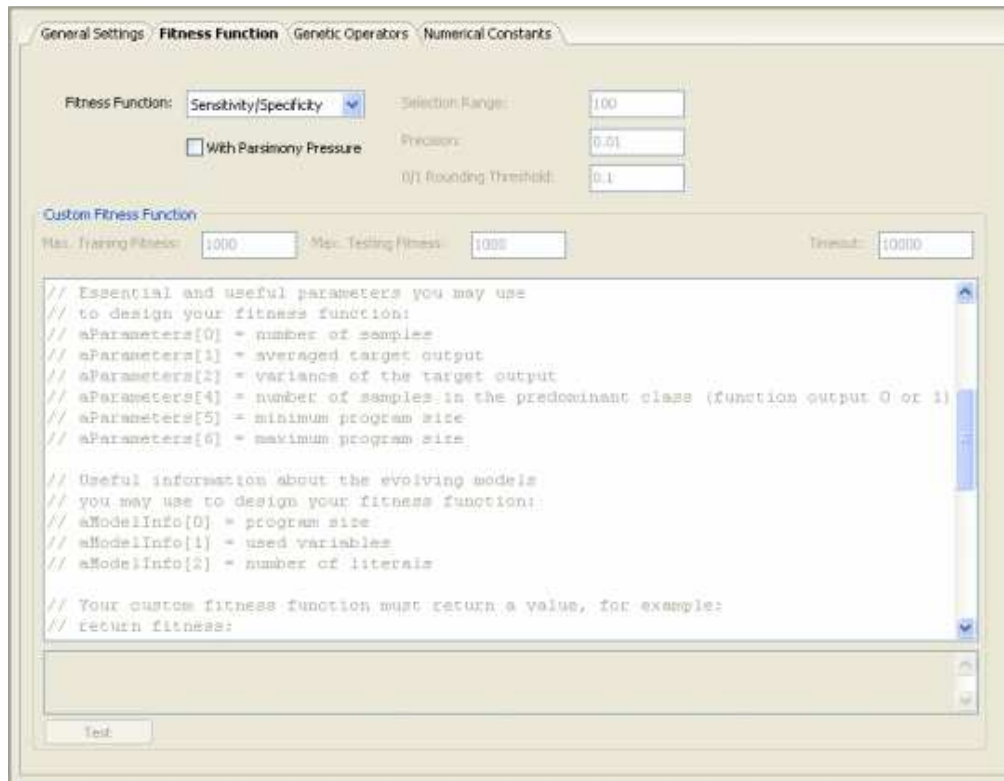
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Hits with Penalty

GeneXproTools 4.0 implements the **Hits with Penalty** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **Hits with Penalty** fitness function is very simple and highly efficient, and is based on the number of samples correctly classified.

More formally, the fitness f_i of an individual program corresponds to the number of hits and is evaluated by the formula:

$$\text{If } (TP_i = 0 \text{ OR } TN_i = 0), \text{ then } f_i = TP_i + TN_i; \text{ else } f_i = 0$$

where TP_i is the number of **true positives** and TN_i is the number of **true negatives**.

So, for this fitness function, maximum fitness $f_{\max}$ is given by:

$$f_{\max} = n$$

where n is the number of fitness cases.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = n$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

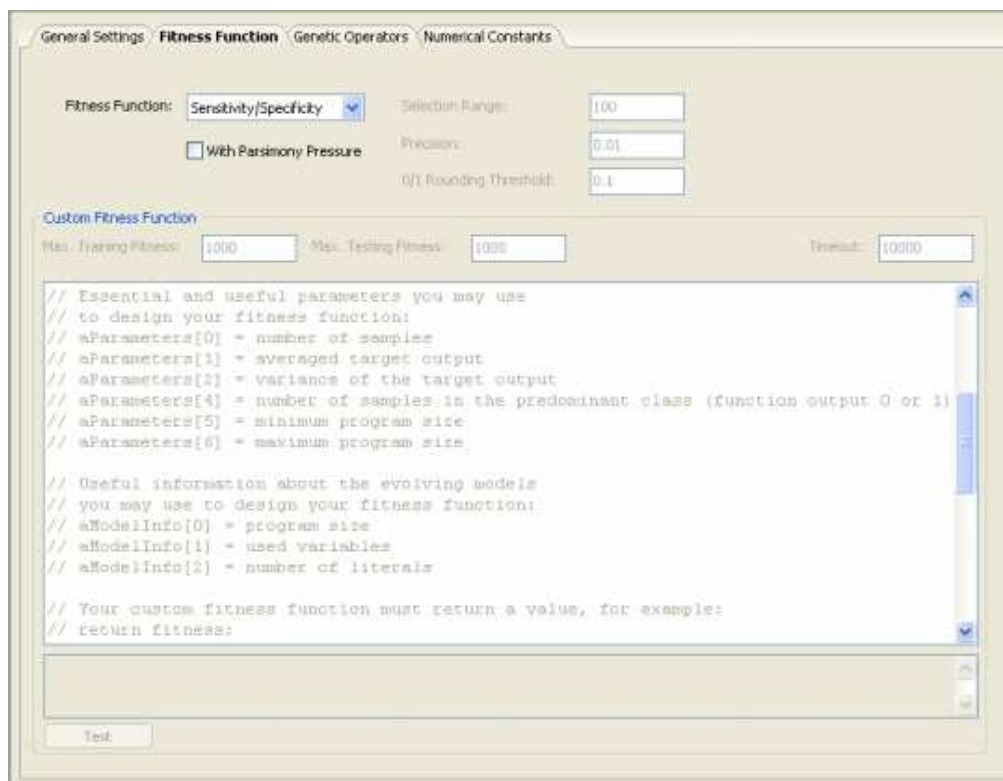
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Accuracy

GeneXproTools 4.0 implements the **Accuracy** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **Accuracy** fitness function of GeneXproTools is, as expected, based on the [classification accuracy](#).

The classification accuracy A_i of an individual program i depends on the number of fitness cases

correctly classified ([true positives](#) plus [true negatives](#)) and is evaluated by the formula:

$$A_i = \frac{t}{n}$$

where t is the number of sample cases correctly classified, and n is the total number of sample cases.

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * A_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

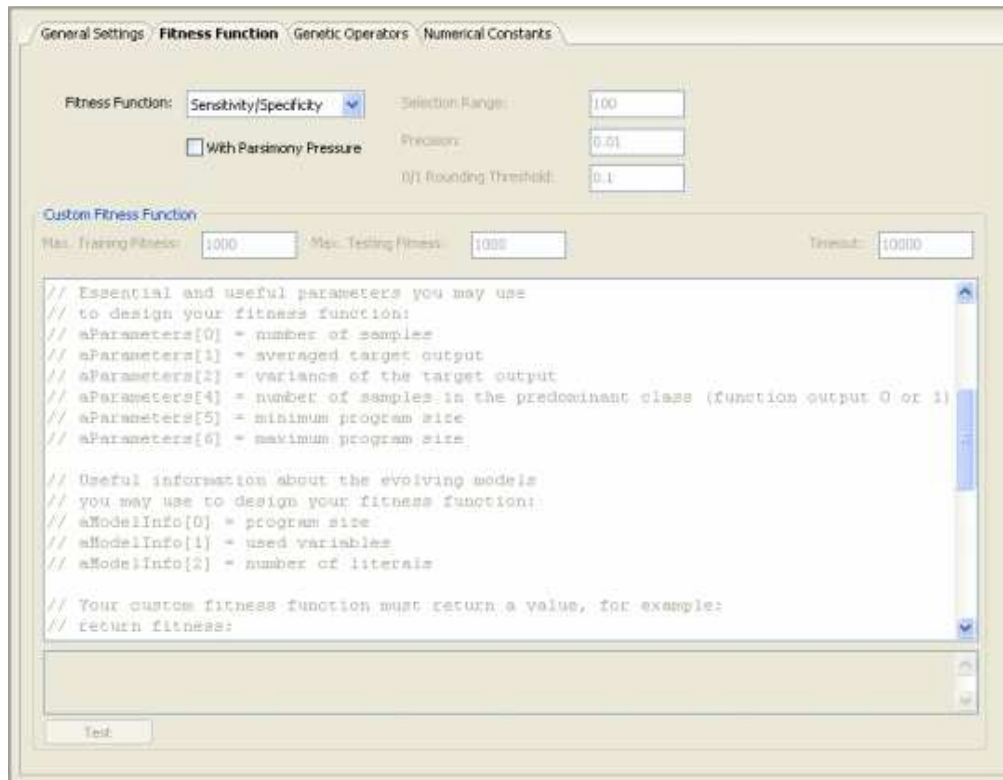
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Squared Accuracy

GeneXproTools 4.0 implements the **Squared Accuracy** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **Squared Accuracy** fitness function of GeneXproTools is, as expected, based on the [classification accuracy](#).

The classification accuracy A_i of an individual program i depends on the number of fitness cases correctly classified ([true positives](#) plus [true negatives](#)) and is evaluated by the formula:

$$A_i = \frac{t}{n}$$

where t is the number of sample cases correctly classified, and n is the total number of sample cases.

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * A_i * A_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

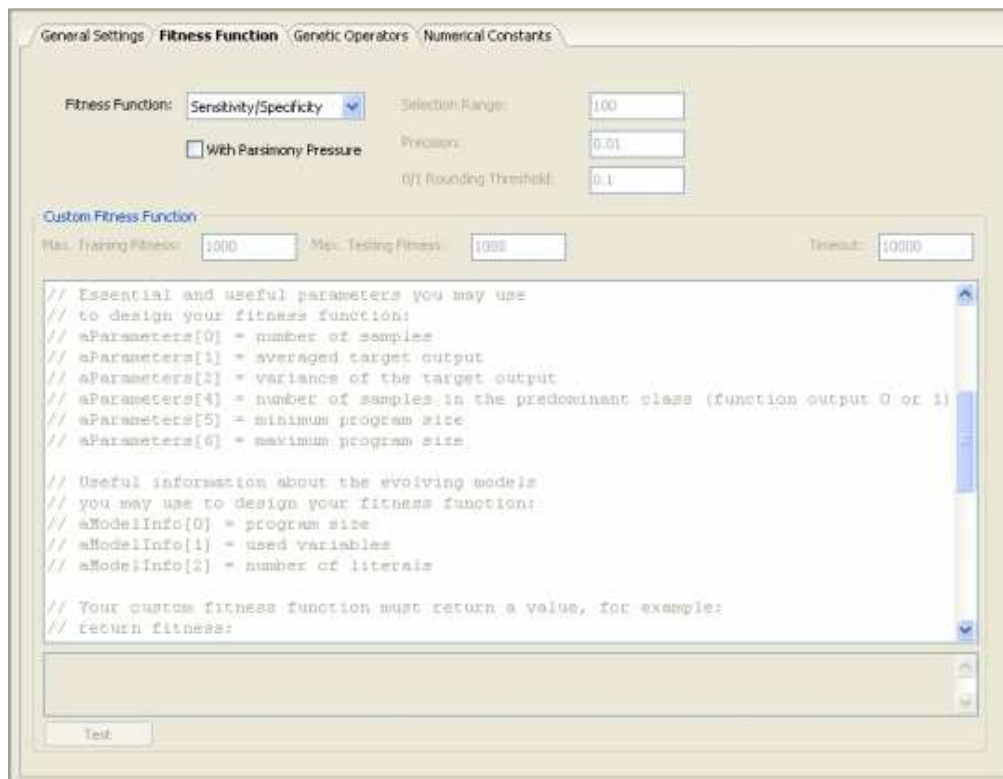
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Sensitivity/Specificity

GeneXproTools 4.0 implements the **Sensitivity/Specificity** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **Sensitivity/Specificity** fitness function of GeneXproTools 4.0 is, as expected, based both on the [sensitivity](#) and [specificity](#).

The sensitivity/specificity SS_i of an individual program i is evaluated by the equation:

$$SS_i = SE_i \cdot SP_i$$

where SE_i is the sensitivity and SP_i is the specificity of the individual program i , and are given by the formulas:

$$SE_i = \frac{TP_i}{TP_i + FN_i}$$

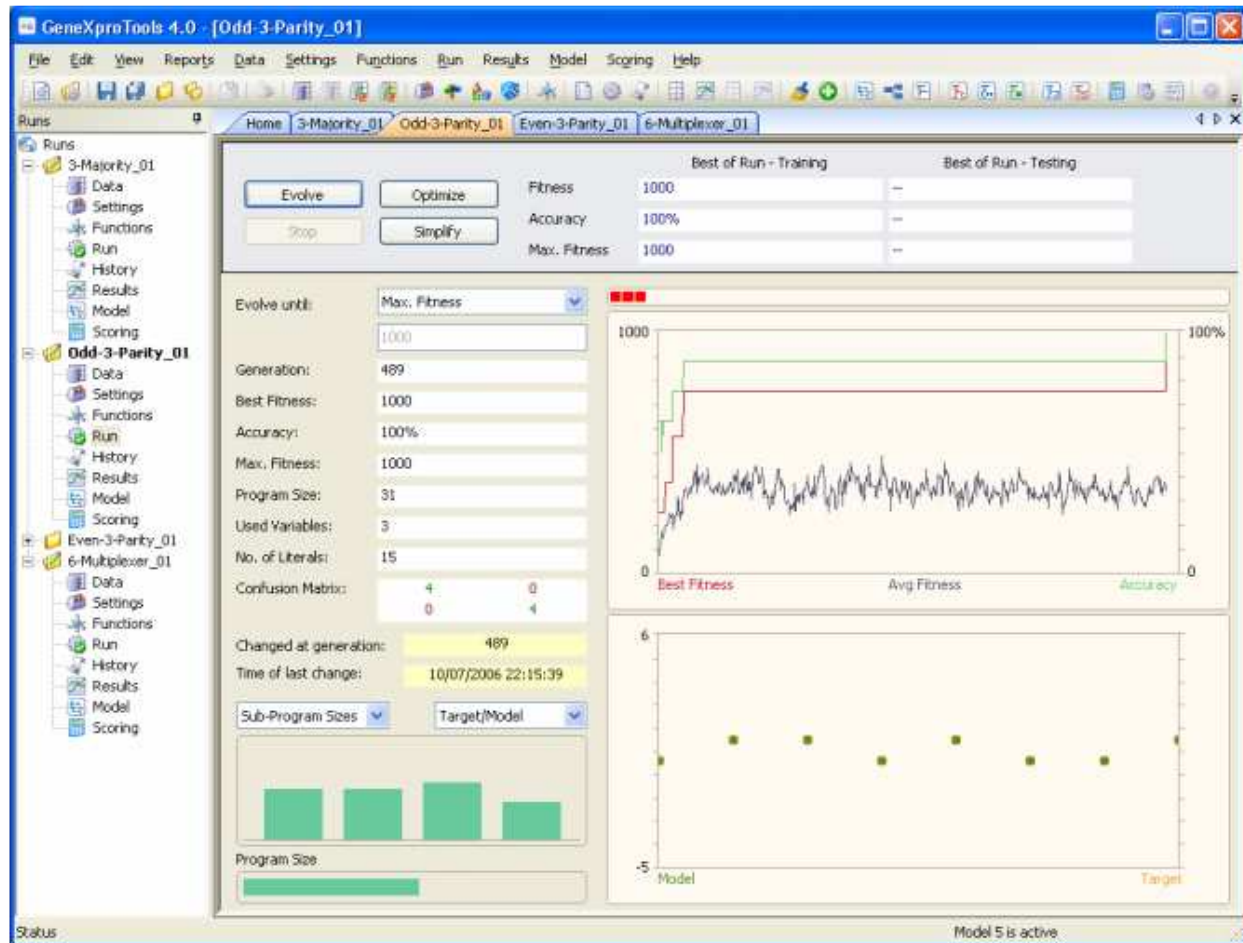
$$SP_i = \frac{TN_i}{TN_i + FP_i}$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all Logic Synthesis problems, regardless of the fitness function, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



Thus, for evaluating the fitness f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot SS_i$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

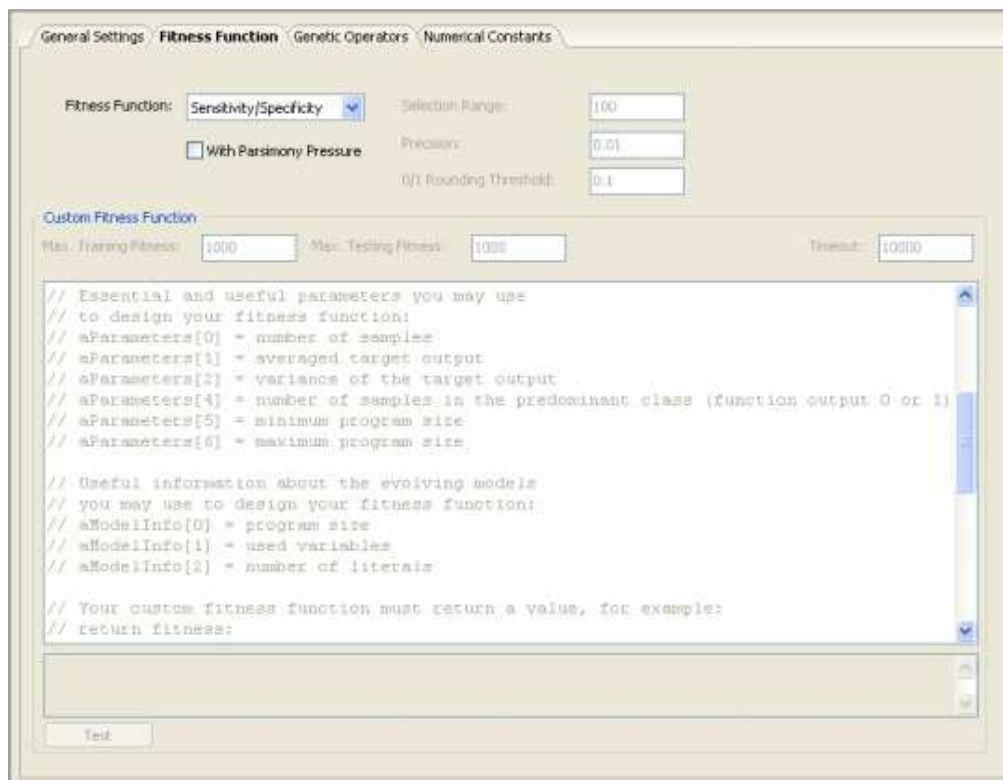
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

PPV/NPV

GeneXproTools 4.0 implements the **Positive Predictive Value / Negative Predictive Value** (PPV/NPV) fitness function both **with** and **without** parsimony pressure. The version **with parsimony pressure** puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **PPV/NPV** fitness function of GeneXproTools 4.0 is, as expected, based both on the **positive predictive value** (PPV) and **negative predictive value** (NPV).

The PPV/NPV PN_i of an individual program i is evaluated by the equation:

$$PN_i = PPV_i \cdot NPV_i$$

where PPV_i is the positive predictive value and NPV_i is the negative predictive value of the individual program i , and are given by the formulas:

$$PPV_i = \frac{TP_i}{TP_i + FP_i} \quad \text{where } TP_i + FP_i \neq 0$$

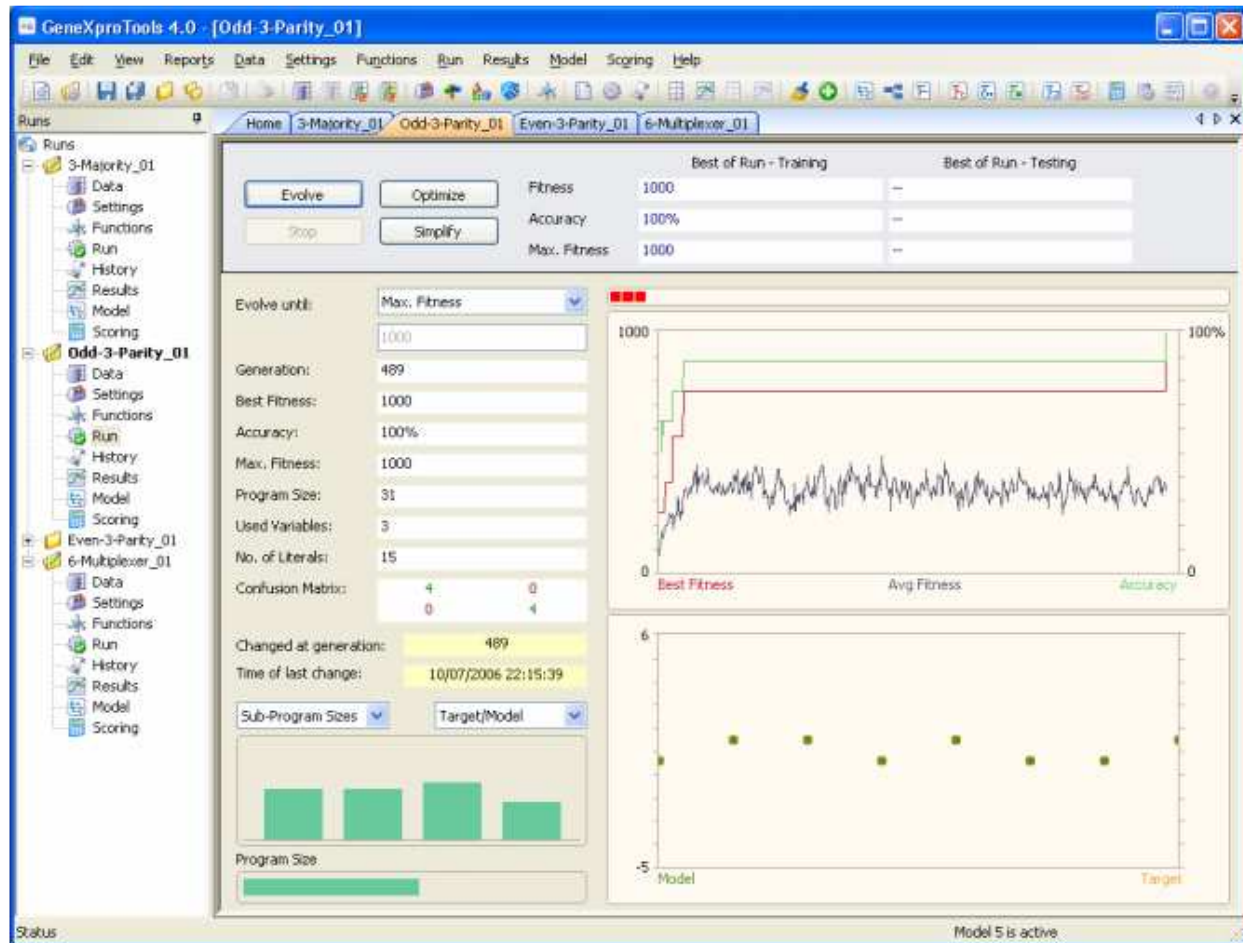
$$NPV_i = \frac{TN_i}{TN_i + FN_i} \quad \text{where } TN_i + FN_i \neq 0$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all Logic Synthesis problems, regardless of the fitness function, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



Thus, for evaluating the fitness f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot PN_i$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

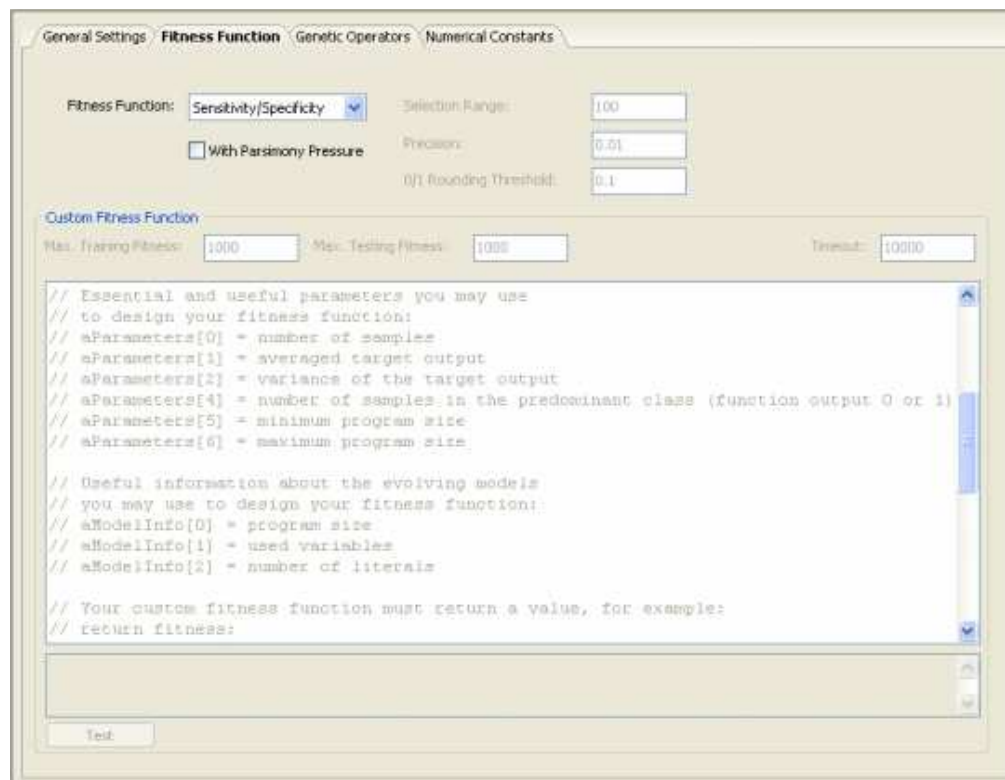
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Sensitivity/Specificity/PPV/NPV

GeneXproTools 4.0 implements the **Sensitivity/Specificity/PPV/NPV** (SSPN) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **SSPN** fitness function of GeneXproTools 4.0 is, as expected, based on all the four statistical indexes: [sensitivity](#), [specificity](#), [positive predictive value](#) (PPV), and [negative predictive value](#) (NPV).

The sensitivity/specificity/PPV/NPV $SSPN_i$ of an individual program i is evaluated by the equation:

$$SSPN_i = SE_i \cdot SP_i \cdot PPV_i \cdot NPV_i$$

where SE_i is the sensitivity, SP_i the specificity, PPV_i the positive predictive value, and NPV_i the negative predictive value of the individual program i , and are given by the formulas:

$$SE_i = \frac{TP_i}{TP_i + FN_i}$$

$$SP_i = \frac{TN_i}{TN_i + FP_i}$$

$$PPV_i = \frac{TP_i}{TP_i + FP_i} \quad \text{where } TP_i + FP_i \neq 0$$

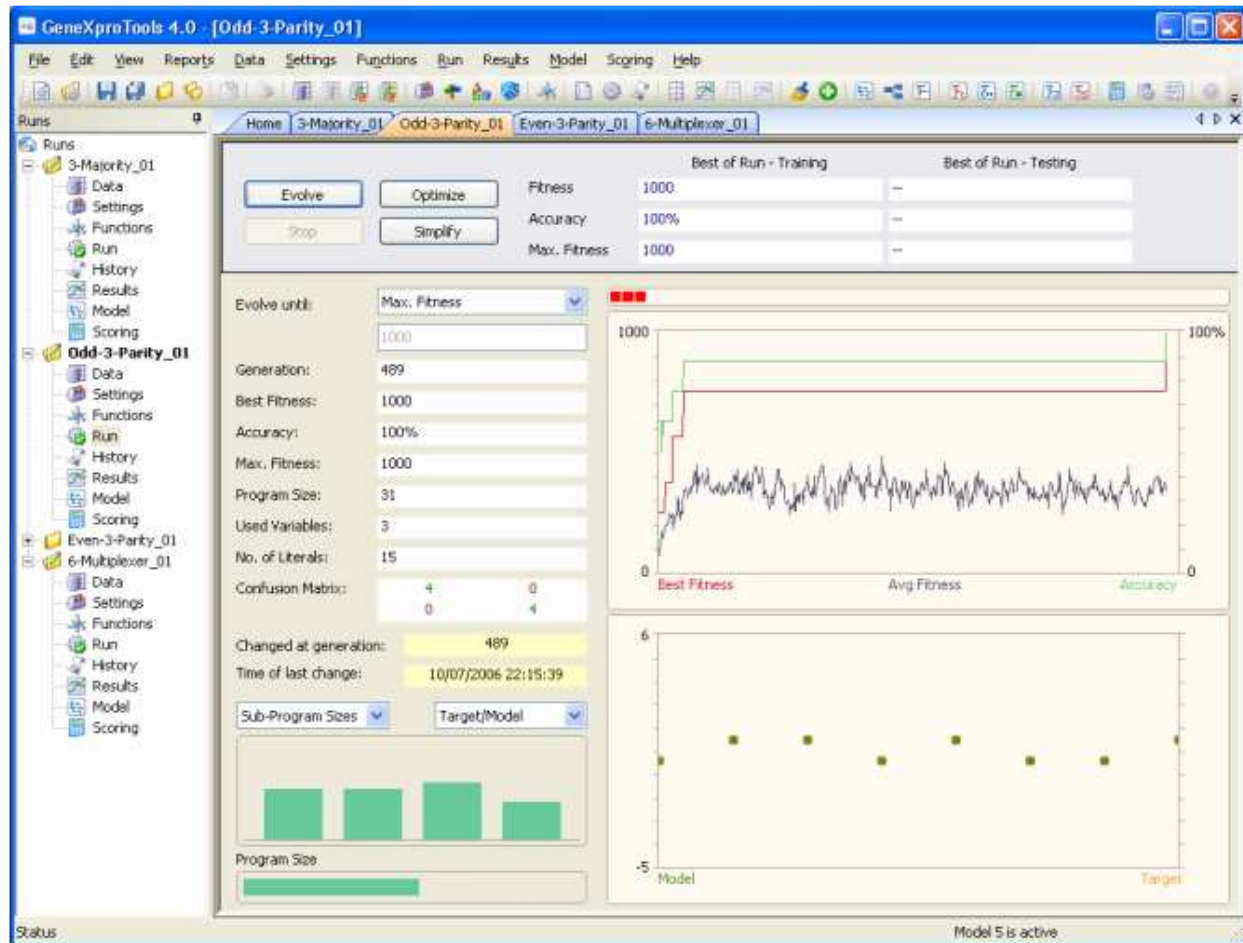
$$NPV_i = \frac{TN_i}{TN_i + FN_i} \quad \text{where } TN_i + FN_i \neq 0$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all Logic Synthesis problems, regardless of the fitness function, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



Thus, for evaluating the fitness f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot SSPN_i$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

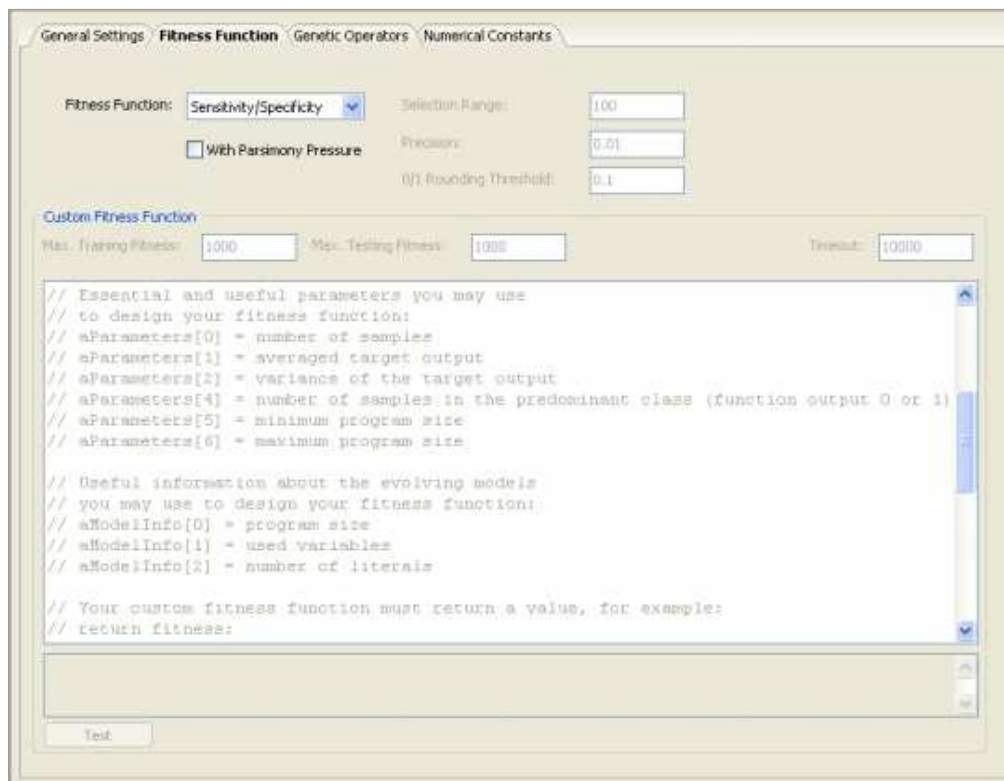
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Squared Error

GeneXproTools 4.0 implements the **Mean Squared Error** (MSE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **MSE** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [mean squared error](#).

The mean squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n (P_{ij} - T_j)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the MSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G(h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

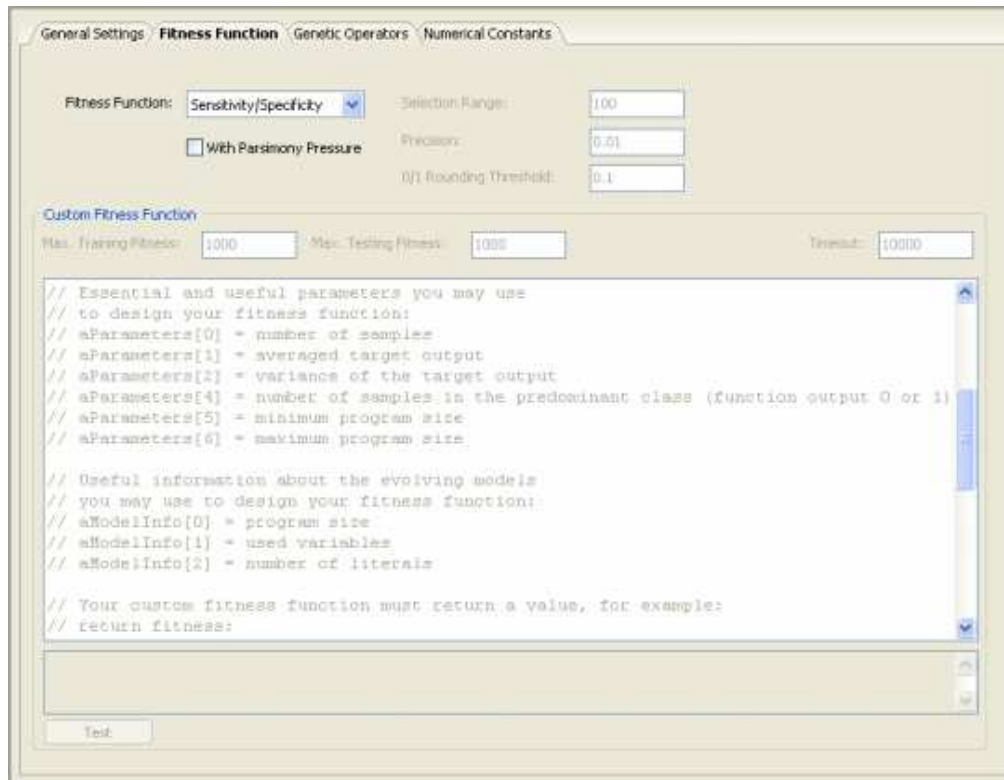
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Mean Squared Error

GeneXproTools 4.0 implements the **Root Mean Squared Error** (RMSE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **RMSE** fitness function of GeneXproTools is, as expected, based on the standard [root mean squared error](#). By taking the square root of the [mean squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

The root mean squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the RMSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

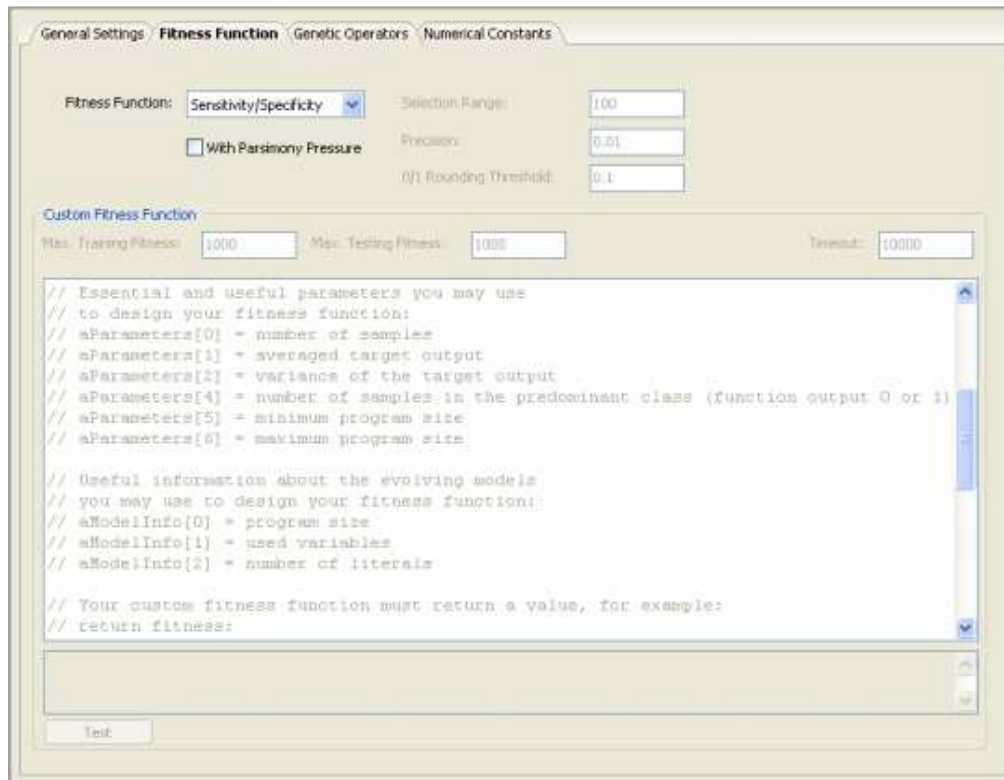
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Mean Absolute Error

GeneXproTools 4.0 implements the **Mean Absolute Error** (MAE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **MAE** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [mean absolute error](#).

The mean absolute error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the MAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

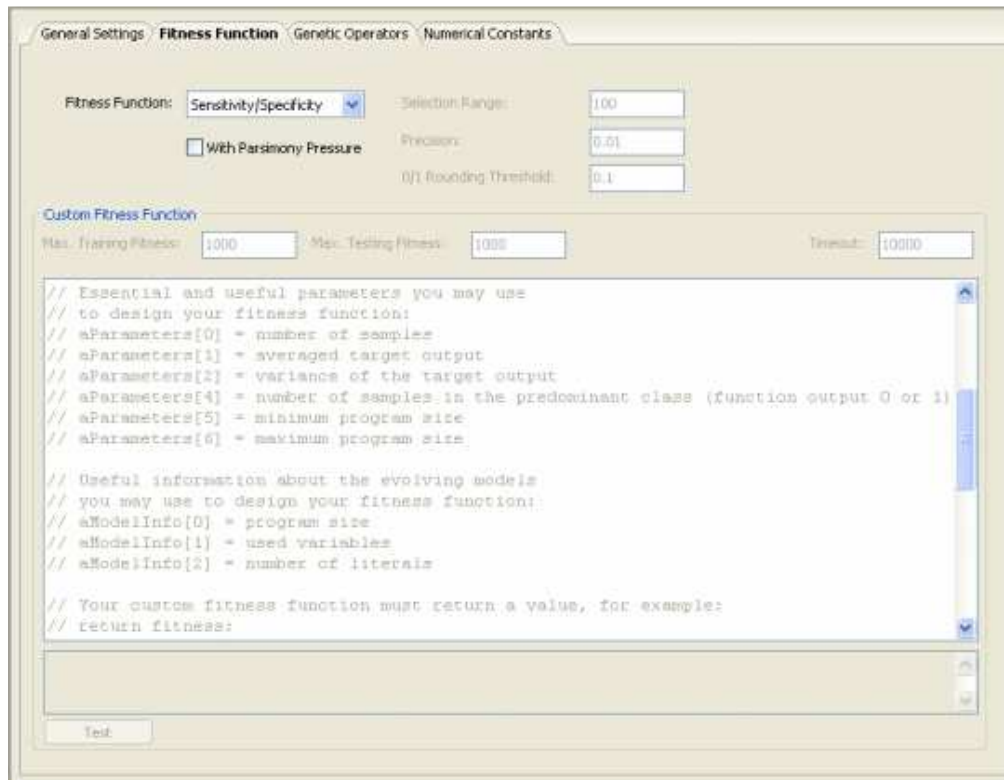
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Squared Error

GeneXproTools 4.0 implements the **Relative Squared Error** (RSE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **RSE** fitness function of GeneXproTools is, as expected, based on the standard [relative squared error](#).

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the relative squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of

fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

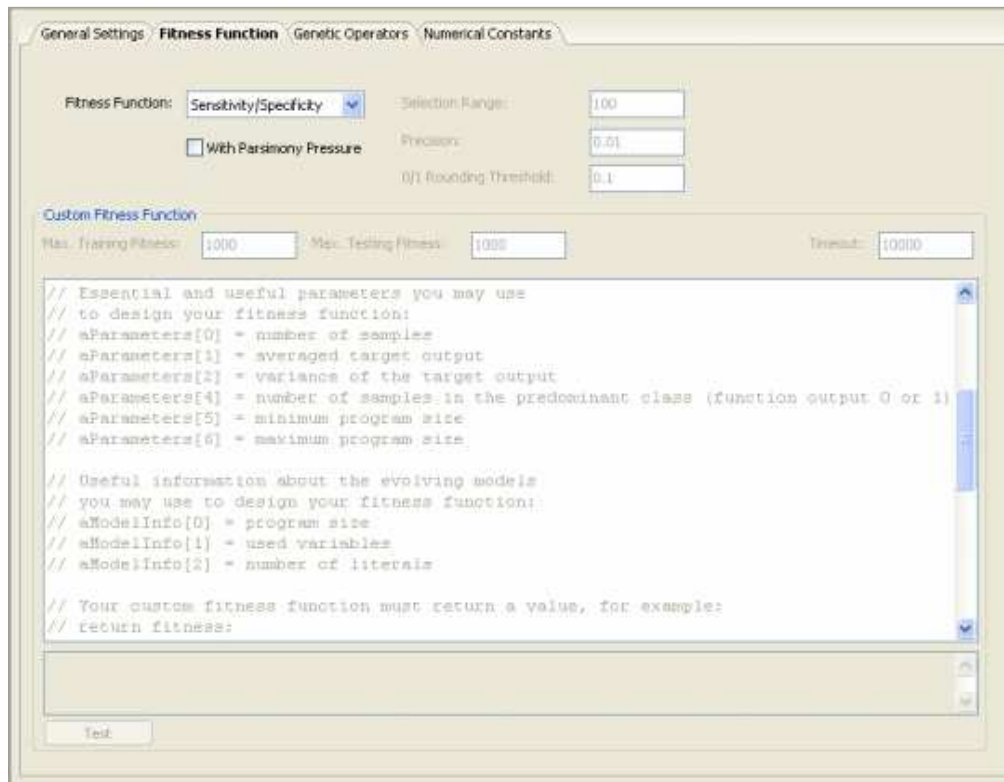
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Root Relative Squared Error

GeneXproTools 4.0 implements the **Root Relative Squared Error** (RRSE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **RRSE** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [root relative squared error](#).

The relative squared error is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the root relative squared error E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RRSE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

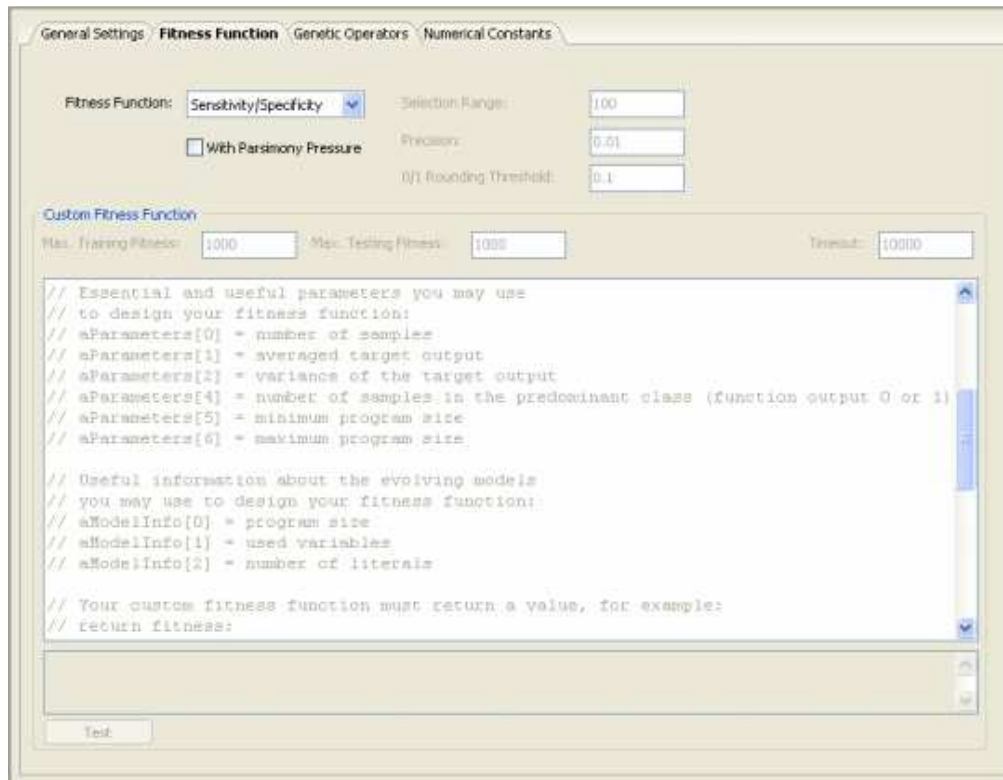
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Relative Absolute Error

GeneXproTools 4.0 implements the **Relative Absolute Error** (RAE) fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **RAE** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [relative absolute error](#).

The relative absolute error is very similar to the [relative squared error](#) in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the relative absolute error E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n |P_{(ij)} - T_j|}{\sum_{j=1}^n |T_j - \bar{T}|}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases);

T_j is the target value for fitness case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the RAE index ranges from 0 to infinity, with 0 corresponding to the ideal.

As it stands, E_i can not be used directly as fitness since, for fitness proportionate selection, the value of fitness must increase with efficiency.

Thus, for evaluating the **fitness** f_i of an individual program i , the following equation is used:

$$f_i = 1000 \cdot \frac{1}{1 + E_i}$$

which obviously ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

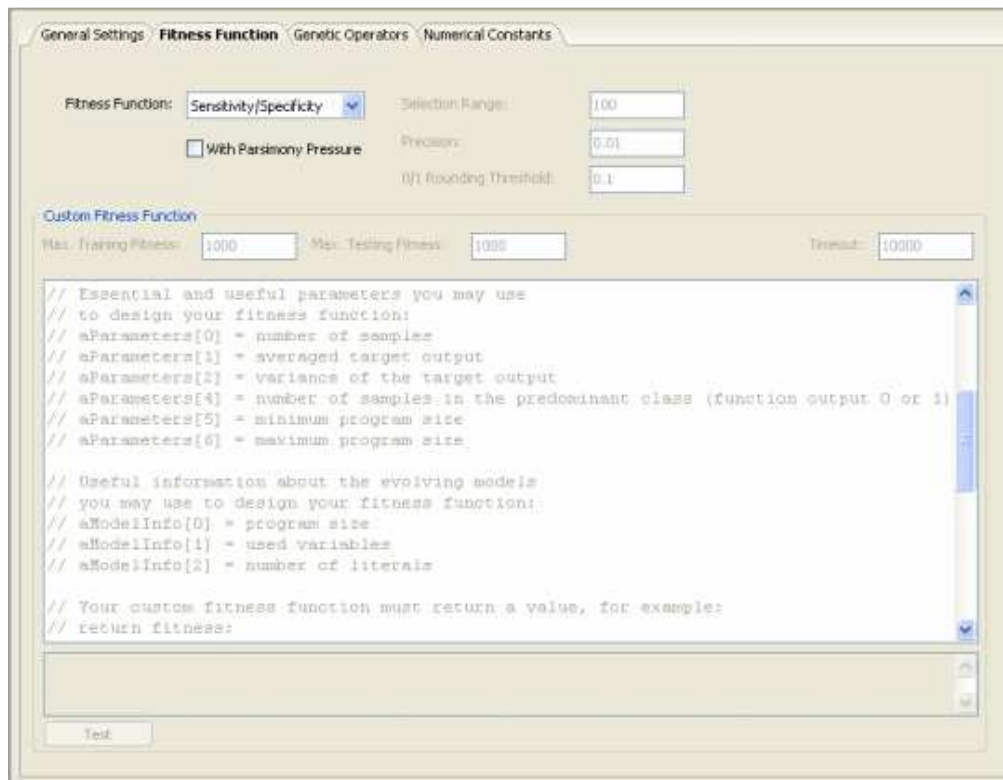
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

R-square

GeneXproTools 4.0 implements the **R-square** fitness function both **with** and **without** parsimony pressure. The version **with parsimony pressure** puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.



The **R-square** fitness function is, as expected, based on the standard [R-square](#), which returns the square of the Pearson product moment correlation coefficient.

The Pearson product moment correlation coefficient is a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between two data sets.

The Pearson product moment correlation coefficient R_i of an individual program i is evaluated by the equation:

$$R_i = \frac{n \sum_{j=1}^n (T_j P_{(ij)}) - \left(\sum_{j=1}^n T_j \right) \left(\sum_{j=1}^n P_{(ij)} \right)}{\sqrt{\left[n \sum_{j=1}^n T_j^2 - \left(\sum_{j=1}^n T_j \right)^2 \right] \left[n \sum_{j=1}^n P_{(ij)}^2 - \left(\sum_{j=1}^n P_{(ij)} \right)^2 \right]}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for fitness case j (out of n fitness cases); and T_j is the target value for fitness case j .

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * R_i * R_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$



Choosing the Fitness Function

Correlation Coefficient

GeneXproTools 4.0 implements the **Correlation Coefficient** fitness function both [with](#) and [without](#) parsimony pressure. The version [with parsimony pressure](#) puts a little pressure on the size of the evolving solutions, allowing the discovery of more compact models.

The **Correlation Coefficient** fitness function of GeneXproTools 4.0 is, as expected, based on the standard [correlation coefficient](#), which is a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between the predicted values and the target values.

The correlation coefficient C_i of an individual program i is evaluated by the equation:

$$C_i = \frac{Cov(T, P)}{\sigma_t \cdot \sigma_p}$$

where $Cov(T, P)$ is the covariance of the target and model outputs; and σ_t and σ_p are the corresponding standard deviations, which are given by:

$$Cov(T, P) = \frac{1}{n} \sum_{j=1}^n (T_j - \bar{T})(P_{(ij)} - \bar{P})$$

$$\sigma_t = \sqrt{\frac{\sum_{j=1}^n (T_j - \bar{T})^2}{n}}$$

$$\sigma_p = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - \bar{P})^2}{n}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n fitness cases or

sample cases); T_j is the target value for fitness case j ; and $\bar{T}$ and $\bar{P}$ are given by the formulas:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

$$\bar{P} = \frac{1}{n} \sum_{j=1}^n P_{(j)}$$

The correlation coefficient is confined to the range $[-1, 1]$. When $C_i = 1$, there is a perfect positive linear correlation between T and P , that is, they vary by the same amount. When $C_i = -1$, there is a perfect negative linear correlation between T and P , that is, they vary in opposite ways (when T increases, P decreases by the same amount). When $C_i = 0$, there is no correlation between T and P . Intermediate values describe partial correlations and the closer to 1 or -1 the better the model.

The **fitness** f_i of an individual program i is expressed by the equation:

$$f_i = 1000 * C_i * C_i$$

and therefore ranges from 0 to 1000, with 1000 corresponding to the ideal.

Its counterpart with **parsimony pressure**, uses this fitness measure f_i as **raw fitness** rf_i and complements it with a parsimony term.

Thus, in this case, raw maximum fitness $rf_{\max} = 1000$. And the overall fitness fpp_i (that is, fitness with parsimony pressure) is evaluated by the formula:

$$fpp_i = rf_i \cdot \left(1 + \frac{1}{5000} \cdot \frac{S_{\max} - S_i}{S_{\max} - S_{\min}} \right)$$

where S_i is the size of the program, $S_{\max}$ and $S_{\min}$ represent, respectively, maximum and minimum program sizes and are evaluated by the formulas:

$$S_{\max} = G (h + t)$$

$$S_{\min} = G$$

where G is the number of genes, and h and t are the head and tail sizes (note that, for simplicity, the linking function was not taken into account). Thus, when $rf_i = rf_{\max}$ and $S_i = S_{\min}$ (highly improbable, though, as this can only happen for very simple functions as this means that all the sub-ETs are composed of just one node), $fpp_i = fpp_{\max}$, with $fpp_{\max}$ evaluated by the formula:

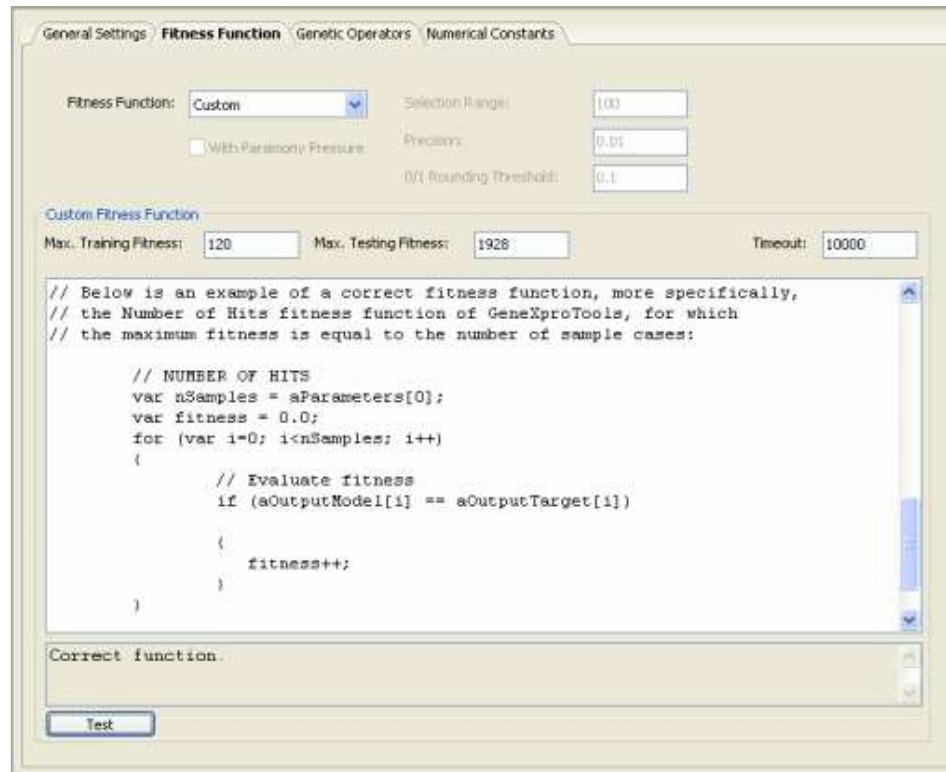
$$fpp_{\max} = 1.0002 \cdot rf_{\max}$$

Choosing the Fitness Function

User Defined Fitness Functions

GeneXproTools 4.0 comes equipped with 30 [built-in fitness functions](#) for Logic Synthesis problems and, most probably, you will find among them a suitable fitness function for your problem. But if you want to try other fitness functions to model your data, GeneXproTools 4.0 allows you to design fitness functions of your own and use them to search the fitness landscape.

You can design your own fitness function using the Custom Fitness Function window.



GeneXproTools 4.0 gives you access to a wide set of essential and useful parameters you may use to design your fitness function:

- `aParameters[0]` = number of samples
- `aParameters[1]` = averaged target output
- `aParameters[2]` = variance of the target output
- `aParameters[4]` = number of samples in the predominant class (function output 0 or 1)
- `aParameters[5]` = minimum program size
- `aParameters[6]` = maximum program size

In addition, GeneXproTools also gives you access to useful information about the evolving models that are essential for designing custom fitness functions with [parsimony pressure](#):

- `aModelInfo[0]` = program size
- `aModelInfo[1]` = used variables
- `aModelInfo[2]` = number of literals

The code for the custom fitness function must be in **JavaScript** and can be tested before evolving a model with it. Please remember that GeneXproTools 4.0 uses fitness proportionate selection and,

therefore, fitness must increase with performance and only non-negative values are acceptable. Below is the sample code of a correct fitness function, which is identical to the [number of hits](#) fitness function of GeneXproTools 4.0.

```

////////////////////////////////////////
// All the values of the Target output
// are accessible through the array:
// aOutputTarget[0] = 0
// aOutputTarget[1] = 1
// aOutputTarget[2] = 0
// etc.

// All the values of the output of the Model
// are accessible through the array:
// aOutputModel[0] = 0
// aOutputModel[1] = 1
// aOutputModel[2] = 1
// etc.

// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[4] = number of samples in the predominant class (function output 0
or 1)
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals

// Your custom fitness function must return a value, for example:
// return fitness;

// Below is an example of a correct fitness function, more specifically,
// the Number of Hits fitness function of GeneXproTools, for which
// the maximum fitness is equal to the number of sample cases:

// NUMBER OF HITS
var nSamples = aParameters[0];
var fitness = 0.0;
for (var i=0; i<nSamples; i++)
{
    // Evaluate fitness
    if (aOutputModel[i] == aOutputTarget[i])
    {
        fitness++;
    }
}

return fitness;

////////////////////////////////////////

```

Obviously, in this case, maximum fitness equals 1000 both for the training and testing sets, and you must also feed this information into GeneXproTools through the Custom Fitness Function window in the

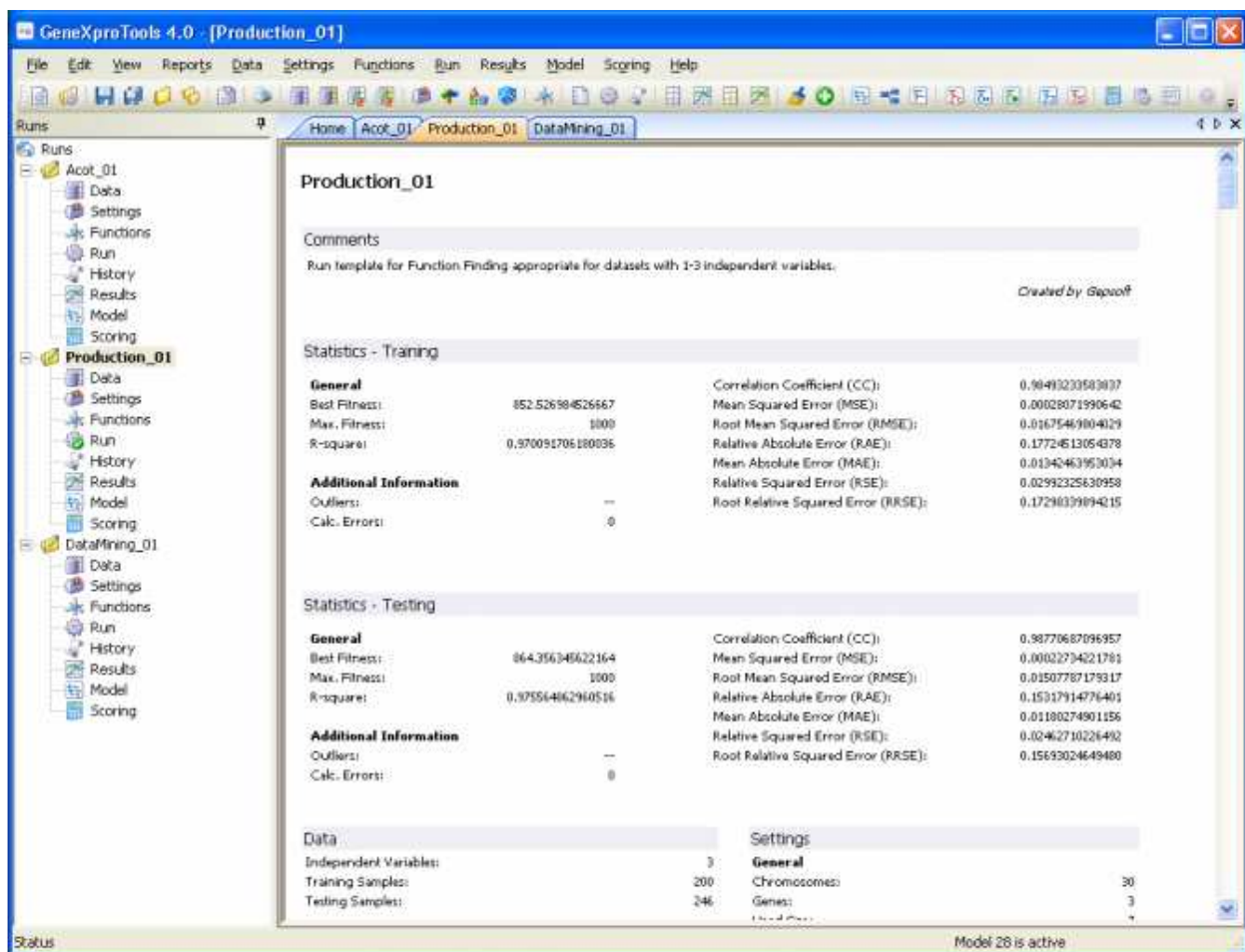
boxes Max. Training Fitness and Max. Testing Fitness.



Analyzing GeneXproTools Models Statistically

This chapter describes all the statistical functions used for evaluating the performance of:

- [Function finding models.](#)
- [Classification models.](#)
- [Time series prediction models.](#)
- [Logic synthesis models.](#)

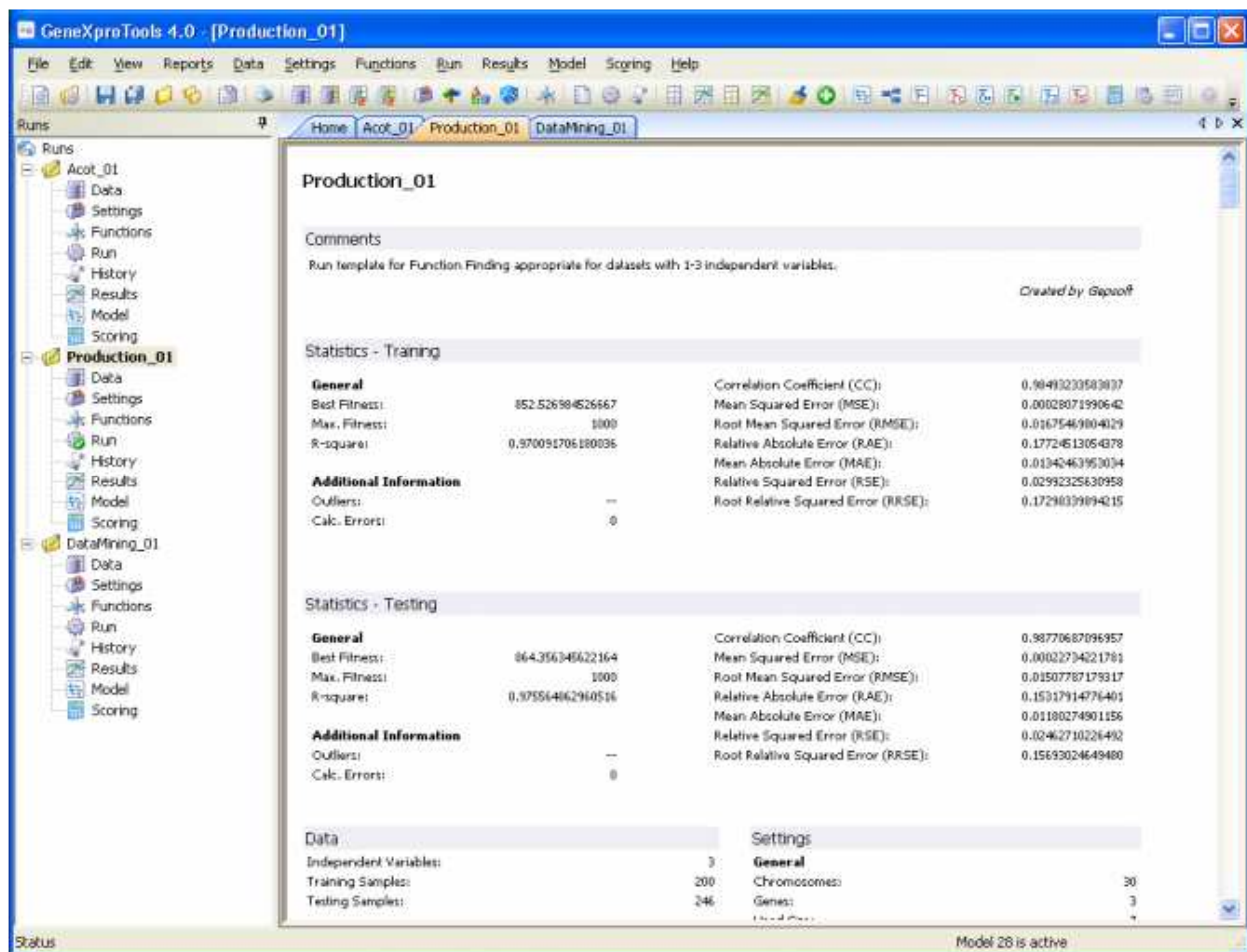


Analyzing GeneXproTools Models Statistically

Statistical Analysis of Function Finding Models

For Function Finding models, GeneXproTools 4.0 enables you to evaluate easily and without leaving the GeneXproTools modeling environment, the following statistical functions both for training and testing data:

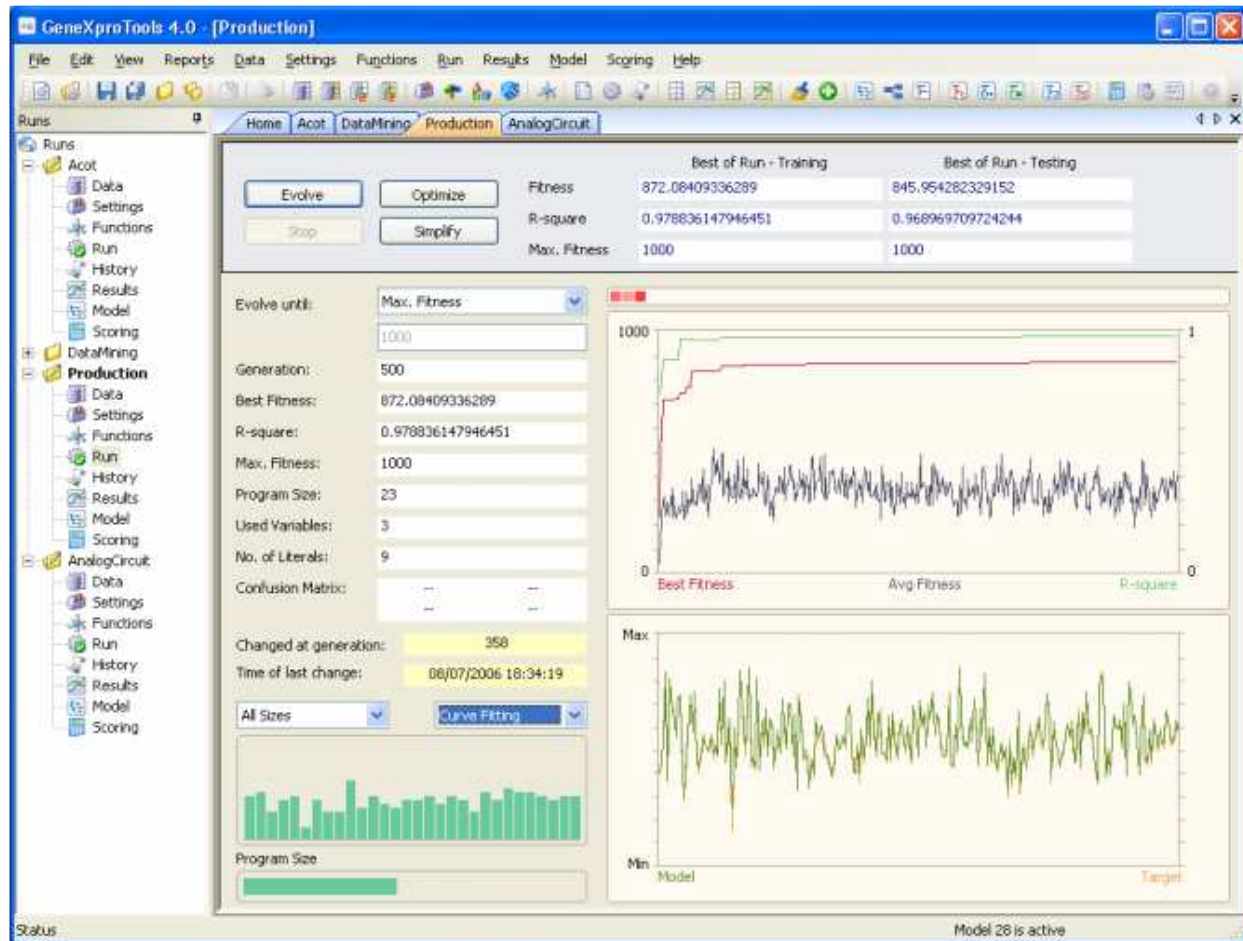
- [R-square](#)
- [Correlation Coefficient](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)



Analyzing GeneXproTools Models Statistically

R-square

The **R-square** returns the square of the Pearson product moment correlation coefficient, which consists of a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between two data sets. During a run, GeneXproTools updates continuously the R-square of the evolving models and plots it in a chart in the [Run Panel](#).

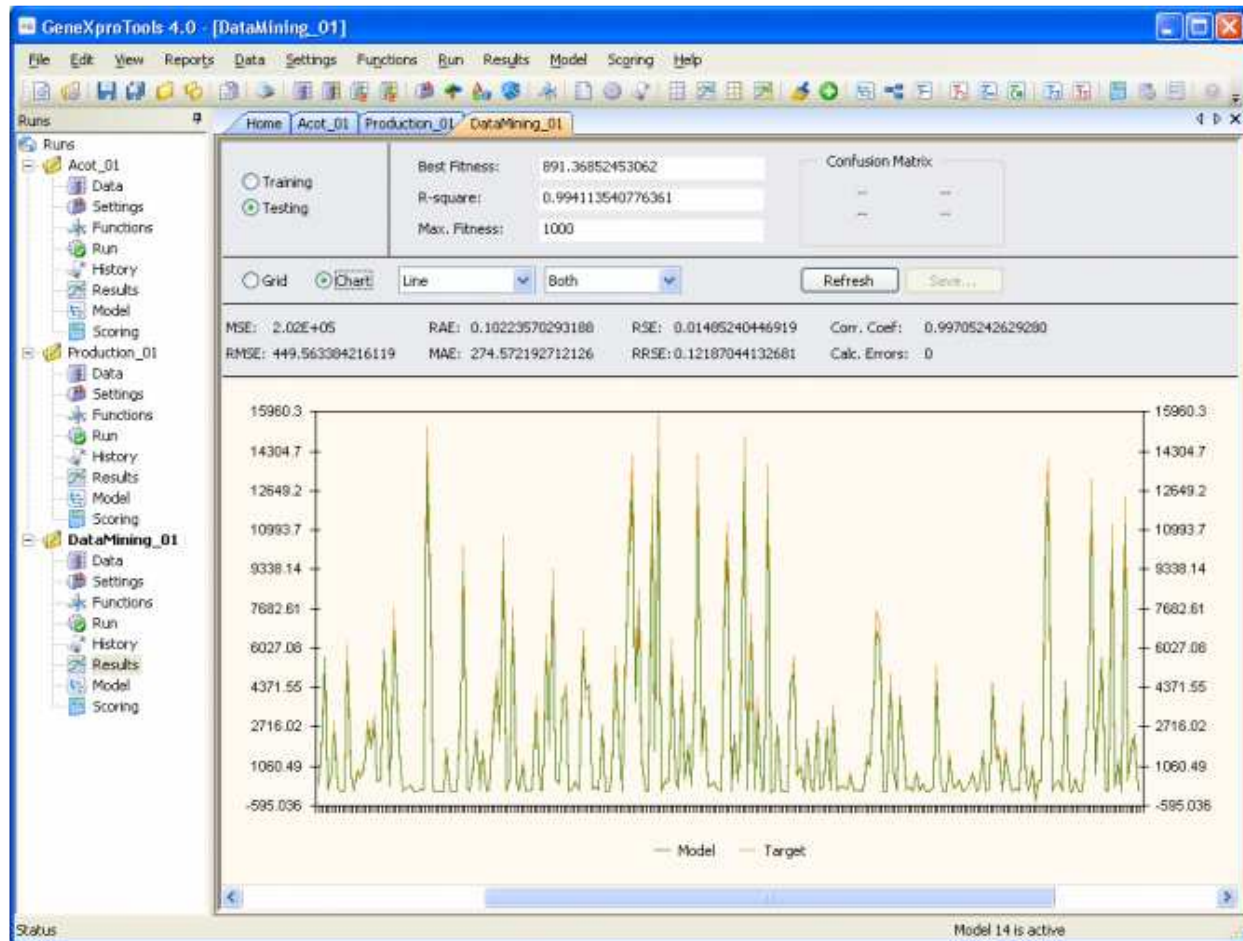


The Pearson product moment correlation coefficient R_i of an individual program i is evaluated by the equation:

$$R_i = \frac{n \sum_{j=1}^n (T_j P_{(ij)}) - \left(\sum_{j=1}^n T_j \right) \left(\sum_{j=1}^n P_{(ij)} \right)}{\sqrt{\left[n \sum_{j=1}^n T_j^2 - \left(\sum_{j=1}^n T_j \right)^2 \right] \left[n \sum_{j=1}^n P_{(ij)}^2 - \left(\sum_{j=1}^n P_{(ij)} \right)^2 \right]}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

To evaluate the **R-square** of your model on the testing set, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Correlation Coefficient

The **correlation coefficient** C_i of an individual program i is evaluated by the equation:

$$C_i = \frac{Cov(T, P)}{\sigma_t \cdot \sigma_p}$$

where $Cov(T, P)$ is the **covariance** of the target and model outputs; and σ_t and σ_p are the corresponding **standard deviations**, which are given by:

$$Cov(T, P) = \frac{1}{n} \sum_{j=1}^n (T_j - \bar{T})(P_{(j)} - \bar{P})$$

$$\sigma_t = \sqrt{\frac{\sum_{j=1}^n (T_j - \bar{T})^2}{n}}$$

$$\sigma_p = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - \bar{P})^2}{n}}$$

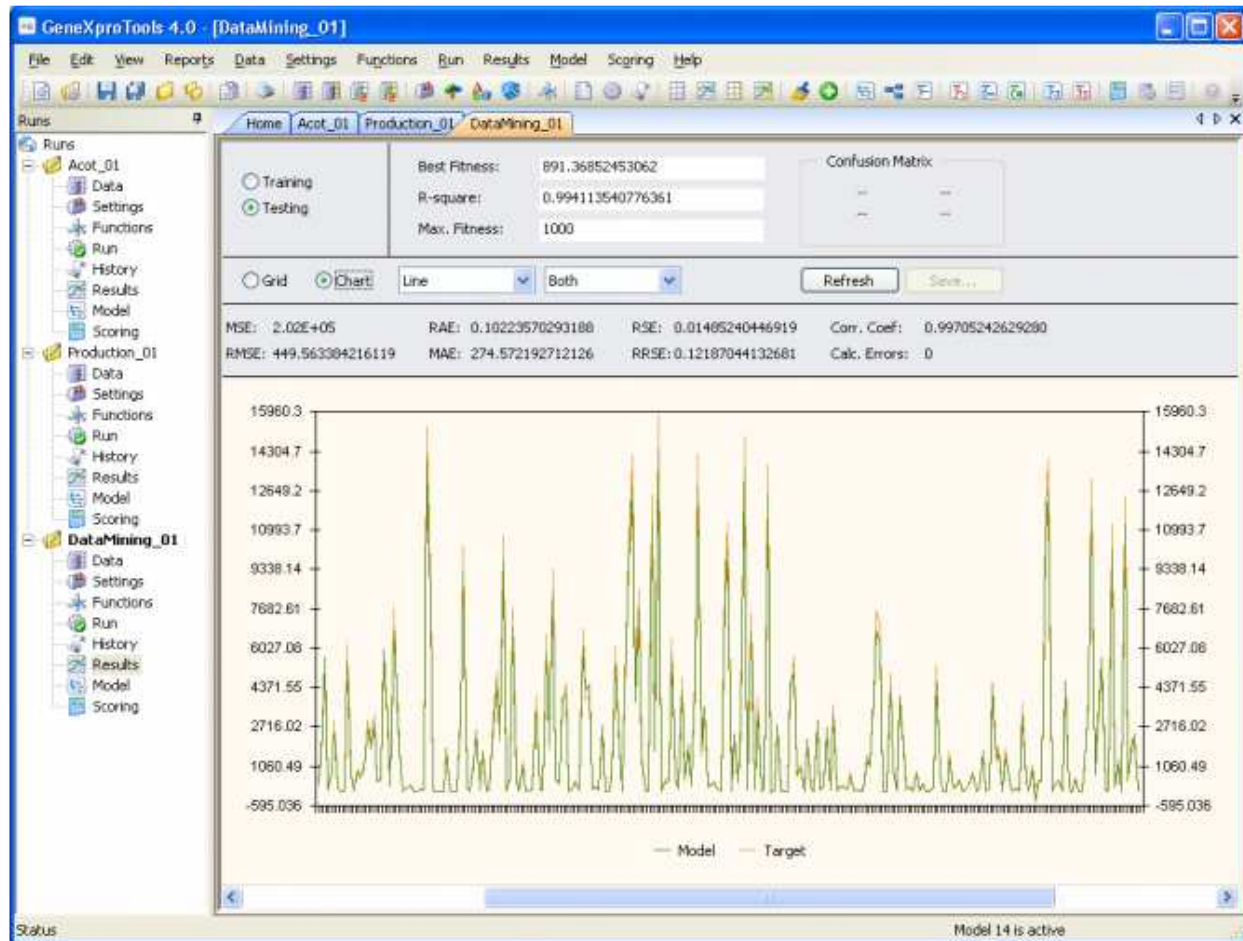
where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ and $\bar{P}$ are given by the formulas:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

$$\bar{P} = \frac{1}{n} \sum_{j=1}^n P_{(ij)}$$

The correlation coefficient is confined to the range [-1, 1]. When $C_i = 1$, there is a perfect positive linear correlation between T and P , that is, they vary by the same amount. When $C_i = -1$, there is a perfect negative linear correlation between T and P , that is, they vary in opposite ways (when T increases, P decreases by the same amount). When $C_i = 0$, there is no correlation between T and P . Intermediate values describe partial correlations and the closer to 1 or -1 the better the model.

To evaluate the **correlation coefficient** of your model both on the training and testing sets, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Mean Squared Error

The **mean squared error** E_i of an individual program i is evaluated by the equation:

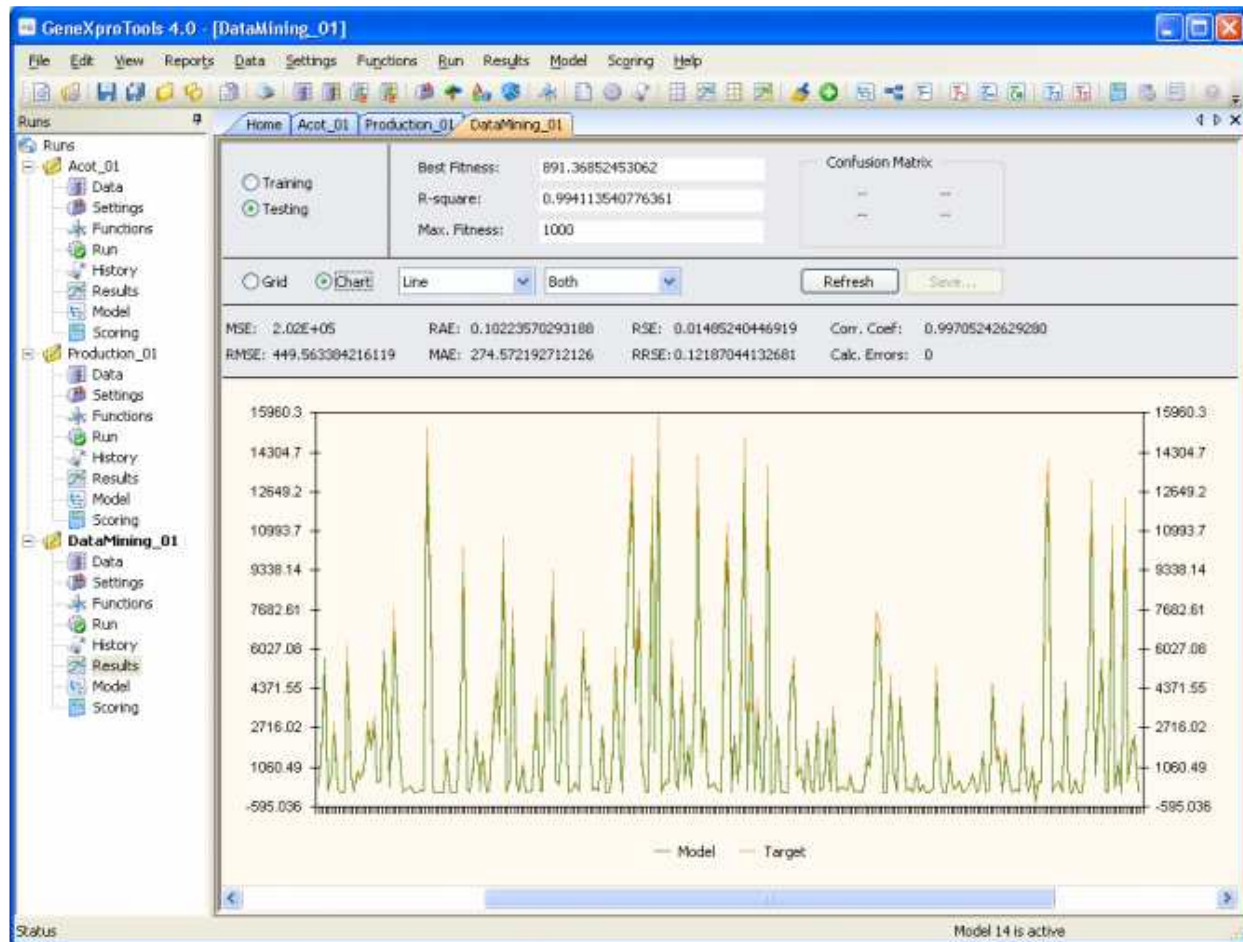
$$E_i = \frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **MSE** of your model both on the training and testing sets, you just have to go to the

Results Panel after a run.



◀ | ▶

Analyzing GeneXproTools Models Statistically

Root Mean Squared Error

The **root mean squared error** E_i of an individual program i is evaluated by the equation:

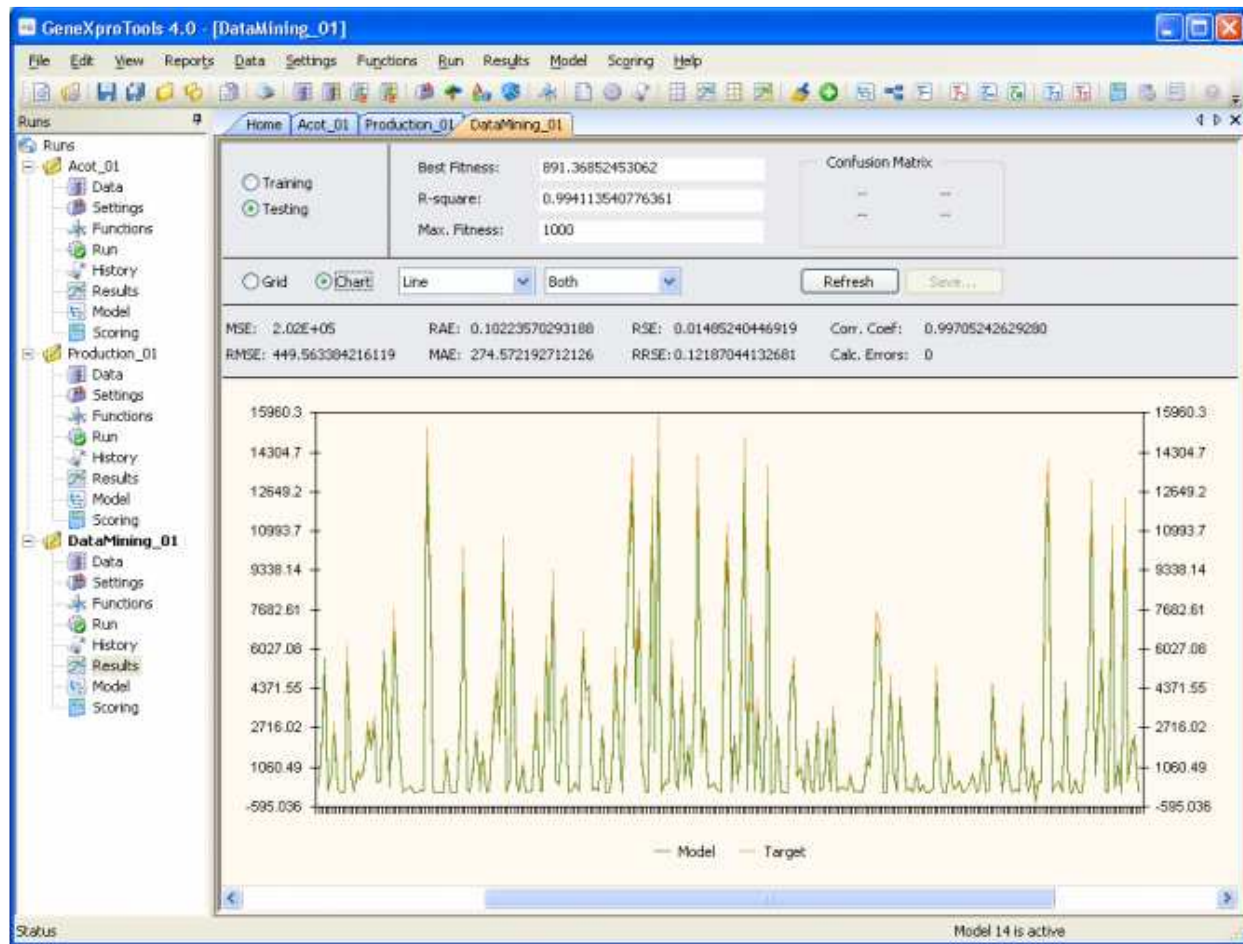
$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to

the ideal.

To evaluate the **RMSE** of your model both on the training and testing sets, you just have to go to the [Results Panel](#) after a run.



◀ | ▶

Analyzing GeneXproTools Models Statistically

Mean Absolute Error

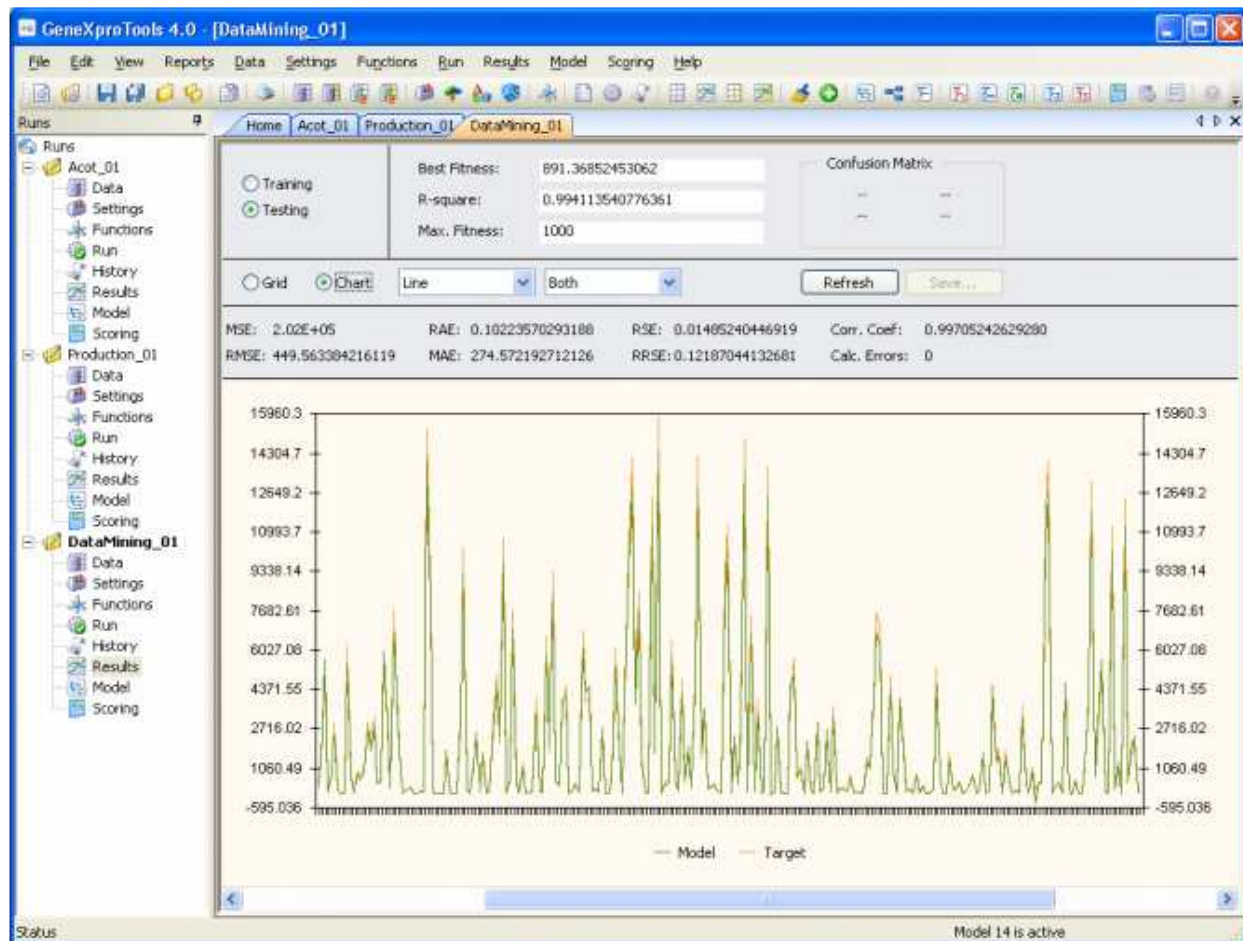
The **mean absolute error** E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **MAE** of your model both on the training and testing sets, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Relative Squared Error

The **relative squared error** is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the **relative squared error** E_i of an individual program i is evaluated by the equation:

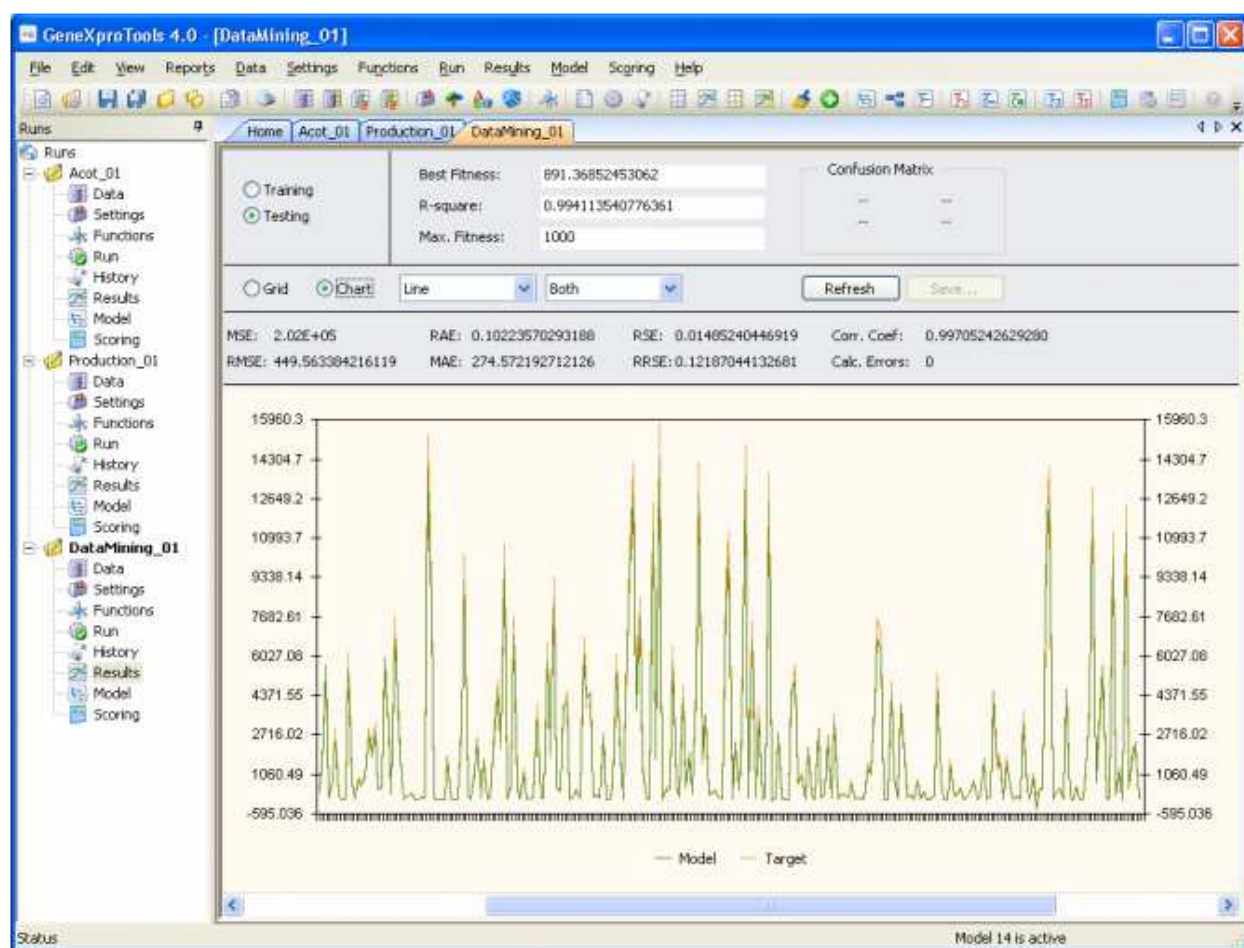
$$E_i = \frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RSE** of your model both on the training and testing sets, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Root Relative Squared Error

The **root relative squared error** is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the **root relative squared error** E_i of an individual program i is evaluated by the equation:

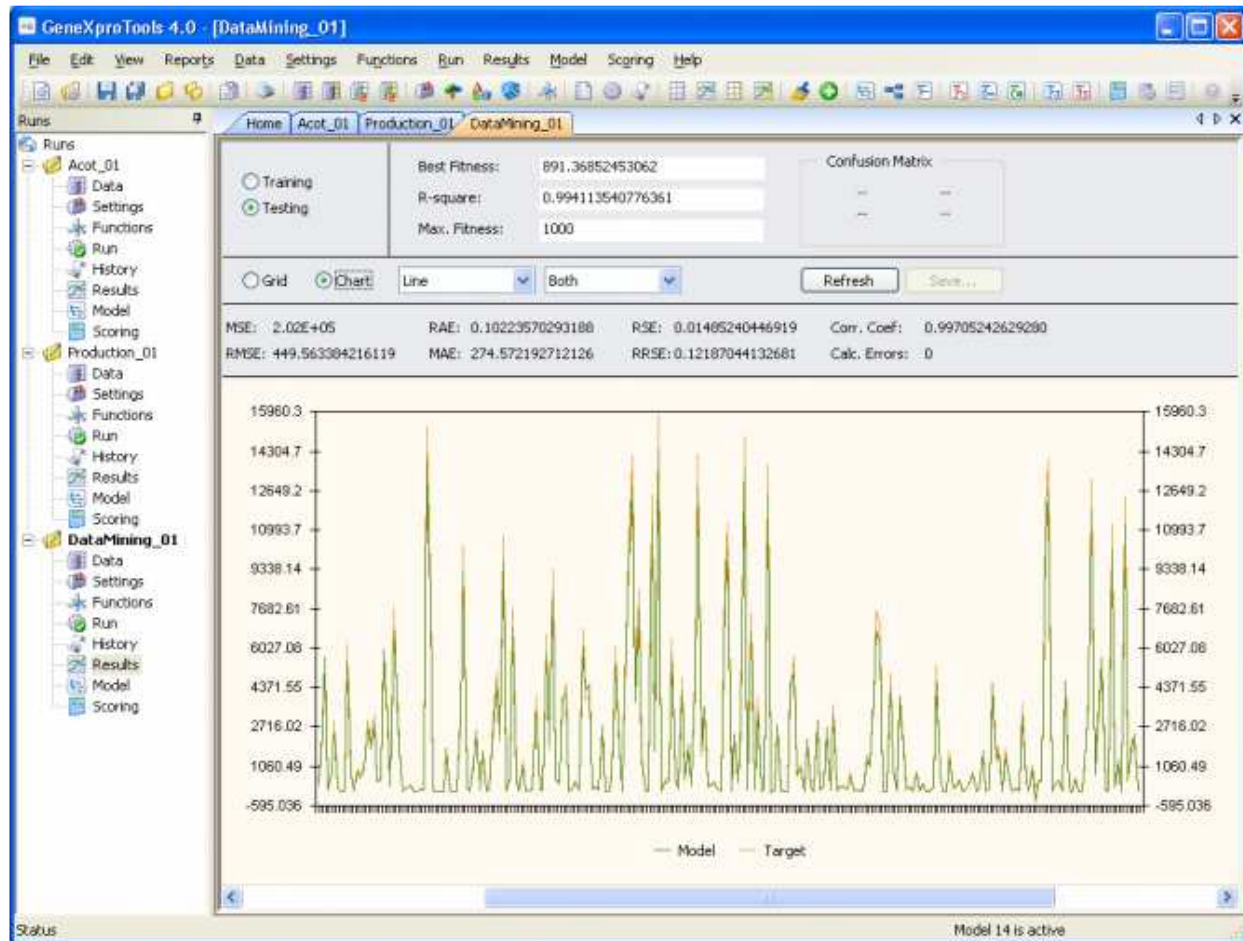
$$E_i = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RRSE** of your model both on the training and testing sets, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Relative Absolute Error

The **relative absolute error** is very similar to the [relative squared error](#) in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the **relative absolute error** E_i of an individual program i is evaluated by the equation:

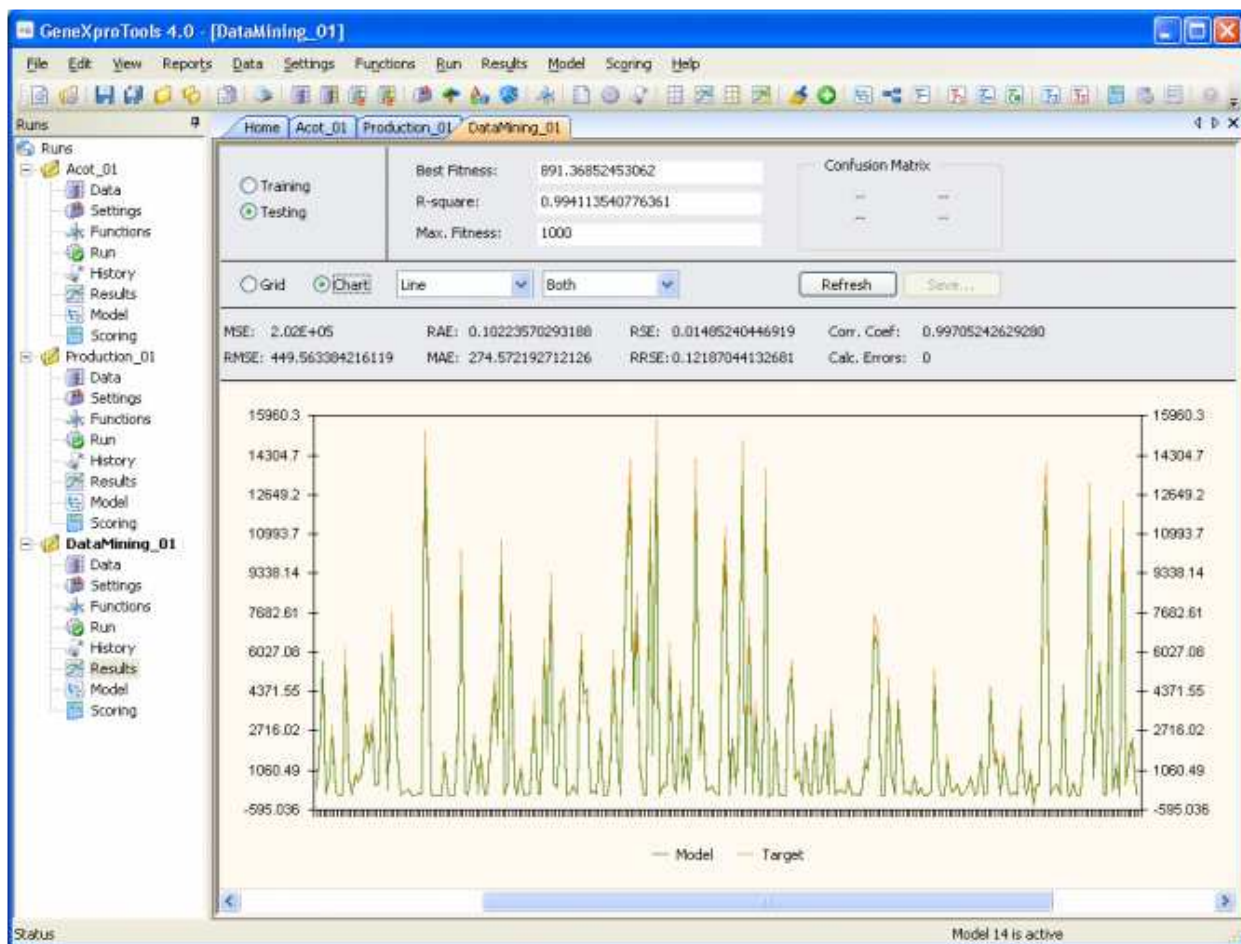
$$E_i = \frac{\sum_{j=1}^n |P_{ij} - T_j|}{\sum_{j=1}^n |T_j - \bar{T}|}$$

where P_{ij} is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RAE** of your model both on the training and testing sets, you just have to go to the [Results Panel](#) after a run.

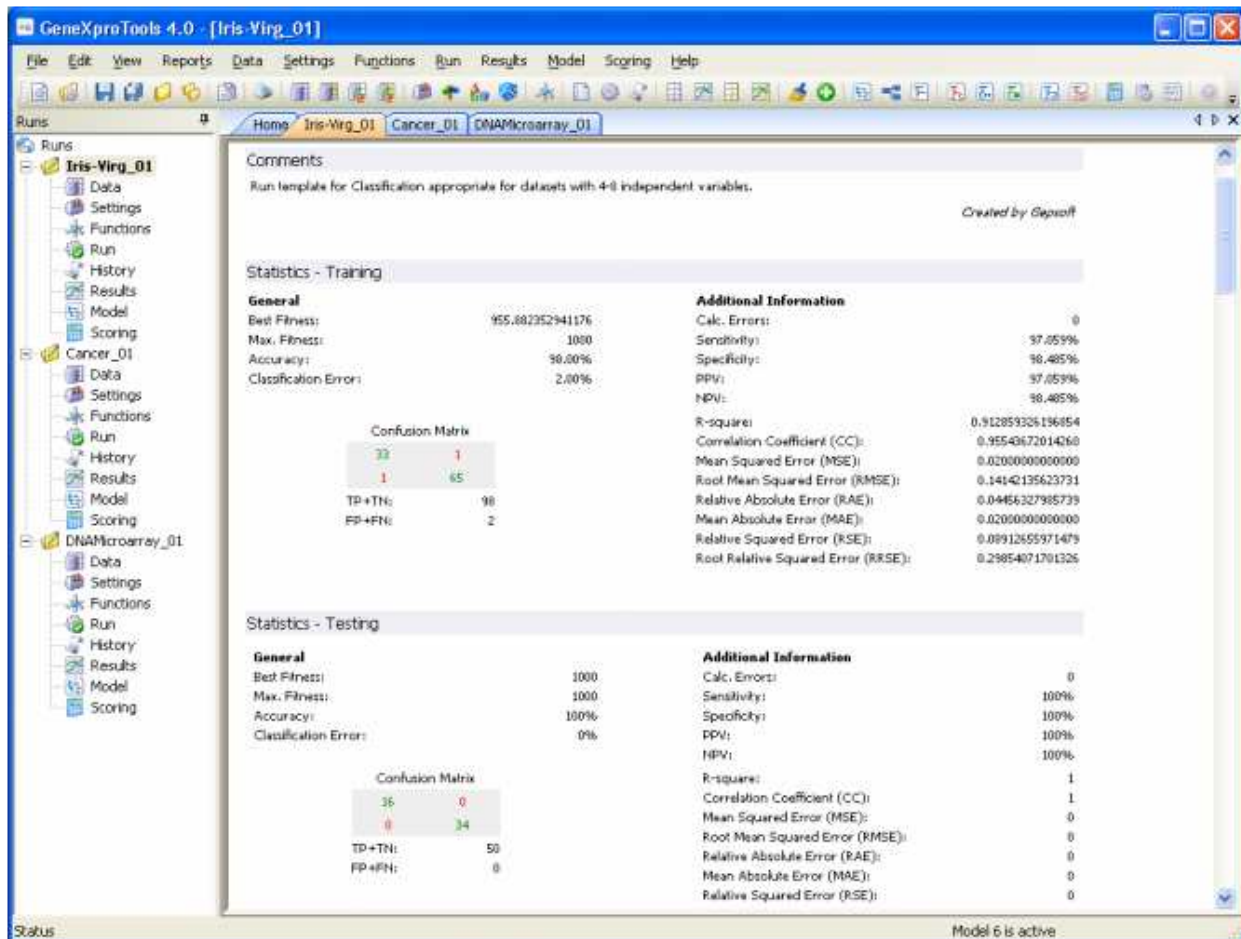


Analyzing GeneXproTools Models Statistically

Statistical Analysis of Classification Models

For Classification models, GeneXproTools 4.0 enables you to evaluate easily and without leaving the GeneXproTools modeling environment, the following statistical functions both for training and testing data:

- [Classification Error](#)
- [Classification Accuracy](#)
- [Confusion Matrix](#)
- [Sensitivity](#)
- [Specificity](#)
- [PPV](#) (positive predictive value)
- [NPV](#) (negative predictive value)
- [R-square](#)
- [Correlation Coefficient](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)





Analyzing GeneXproTools Models Statistically

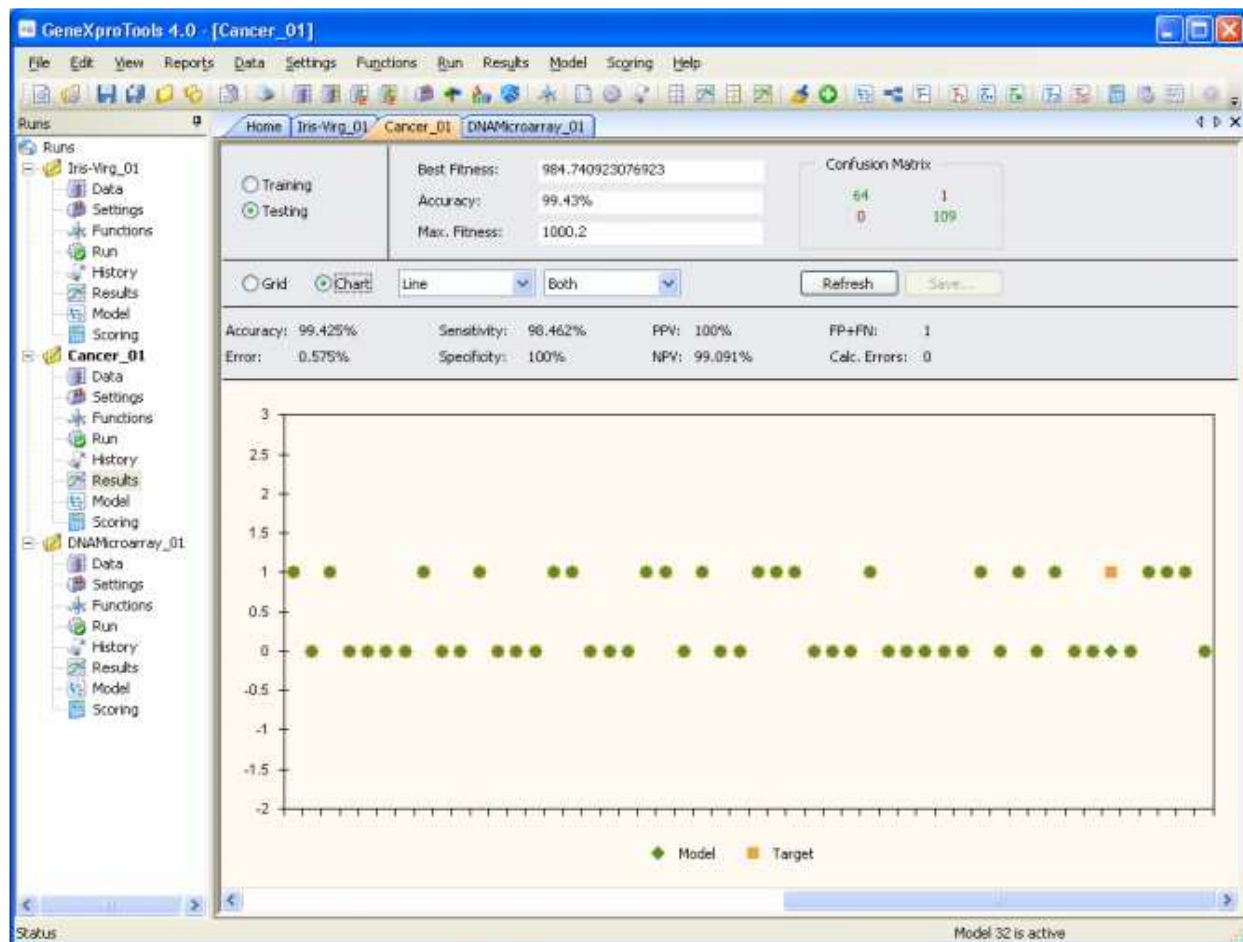
Classification Error

The **classification error** E_i of an individual program i depends on the number of samples incorrectly classified (**false positives** plus **false negatives**) and is evaluated by the formula:

$$E_i = \frac{f}{n} \cdot 100$$

where f is the number of sample cases incorrectly classified, and n is the total number of sample cases.

To evaluate the **classification error** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

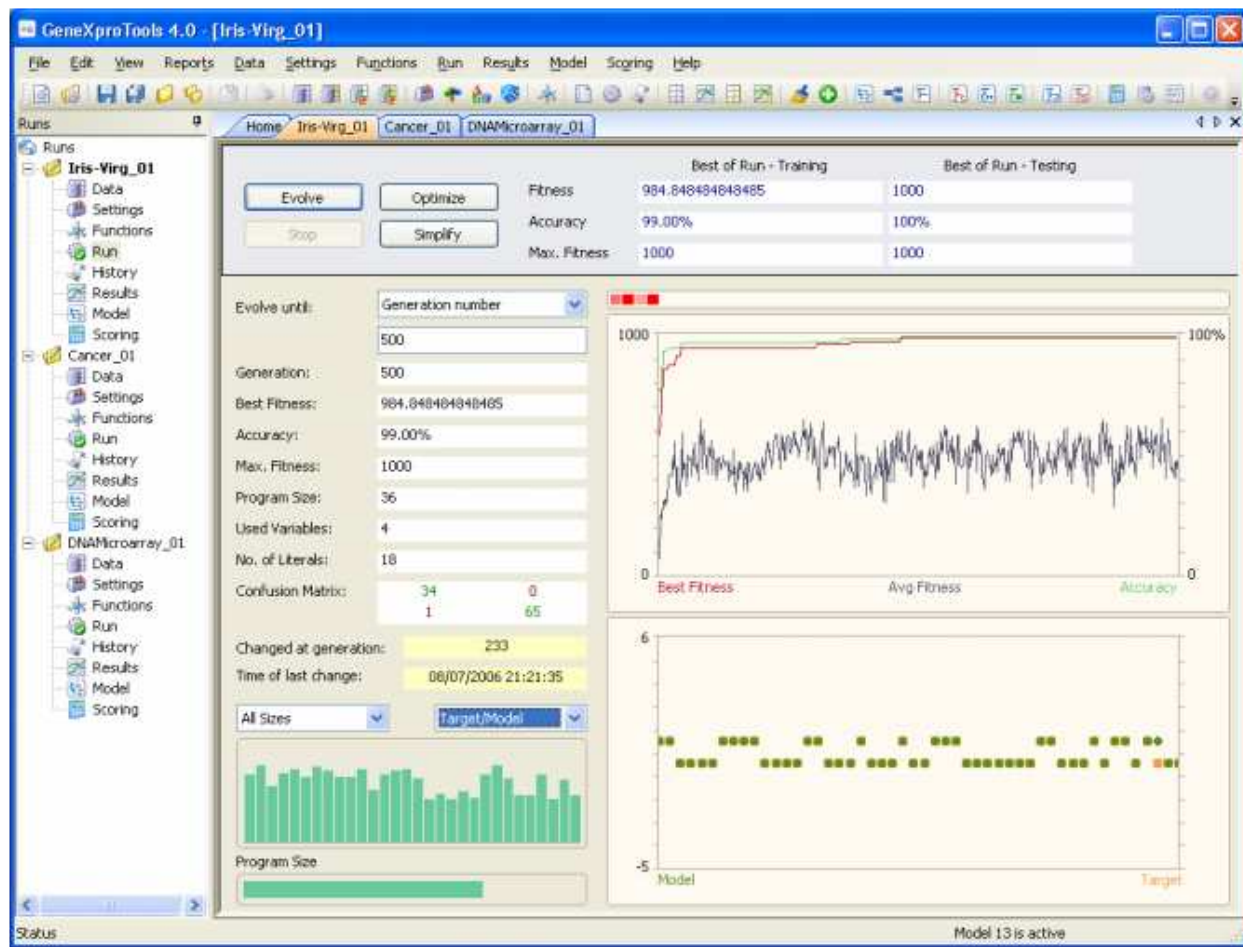
Classification Accuracy

The **classification accuracy** A_i of an individual program i depends on the number of samples correctly classified (**true positives** plus **true negatives**) and is evaluated by the formula:

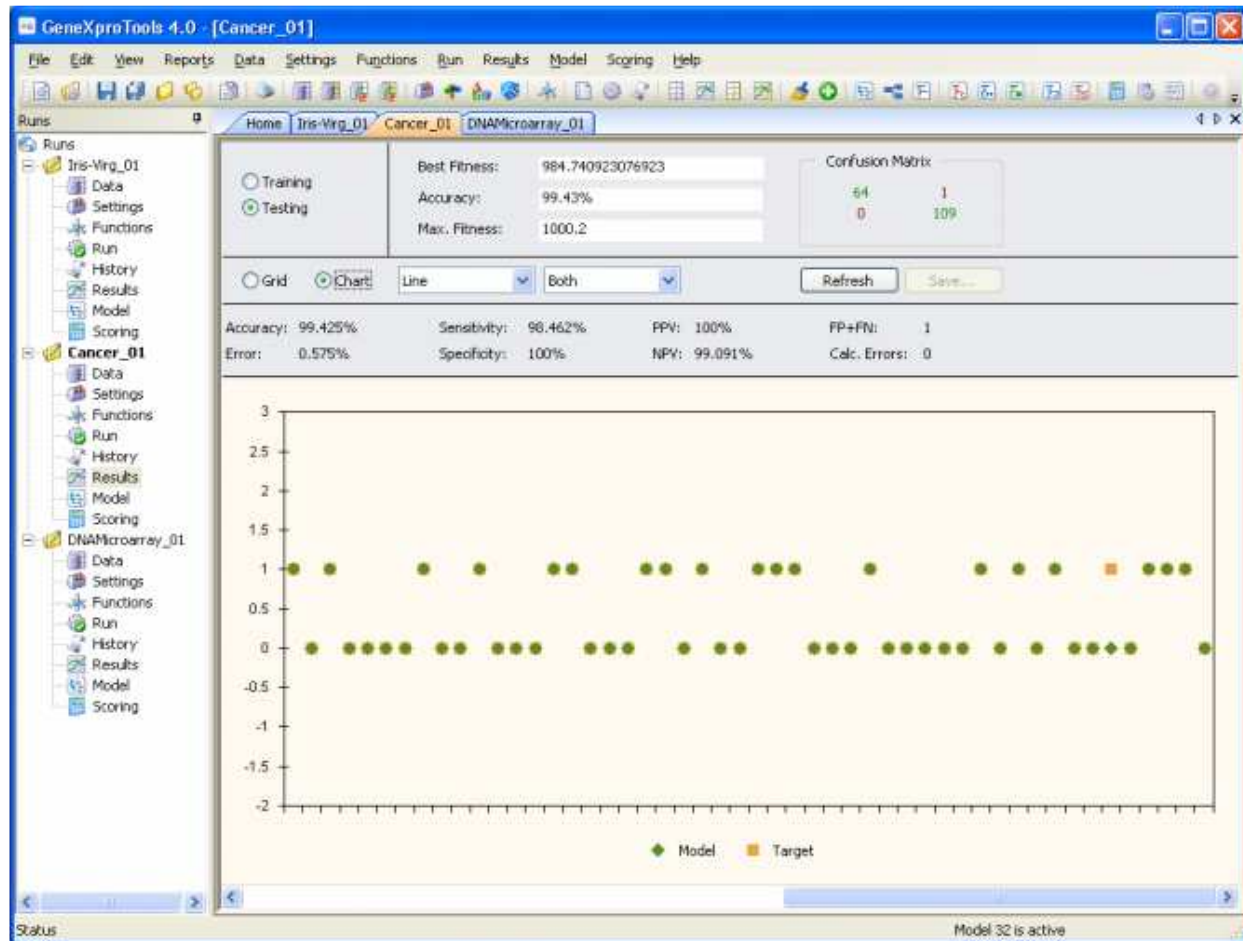
$$A_i = \frac{t}{n} \cdot 100$$

where t is the number of sample cases correctly classified, and n is the total number of sample cases.

During a run, GeneXproTools updates continuously the classification accuracy of the evolving models and plots it in a chart in the [Run Panel](#).



And to evaluate the **classification accuracy** of your model on the testing data, after a run you just have to go to the [Results Panel](#).



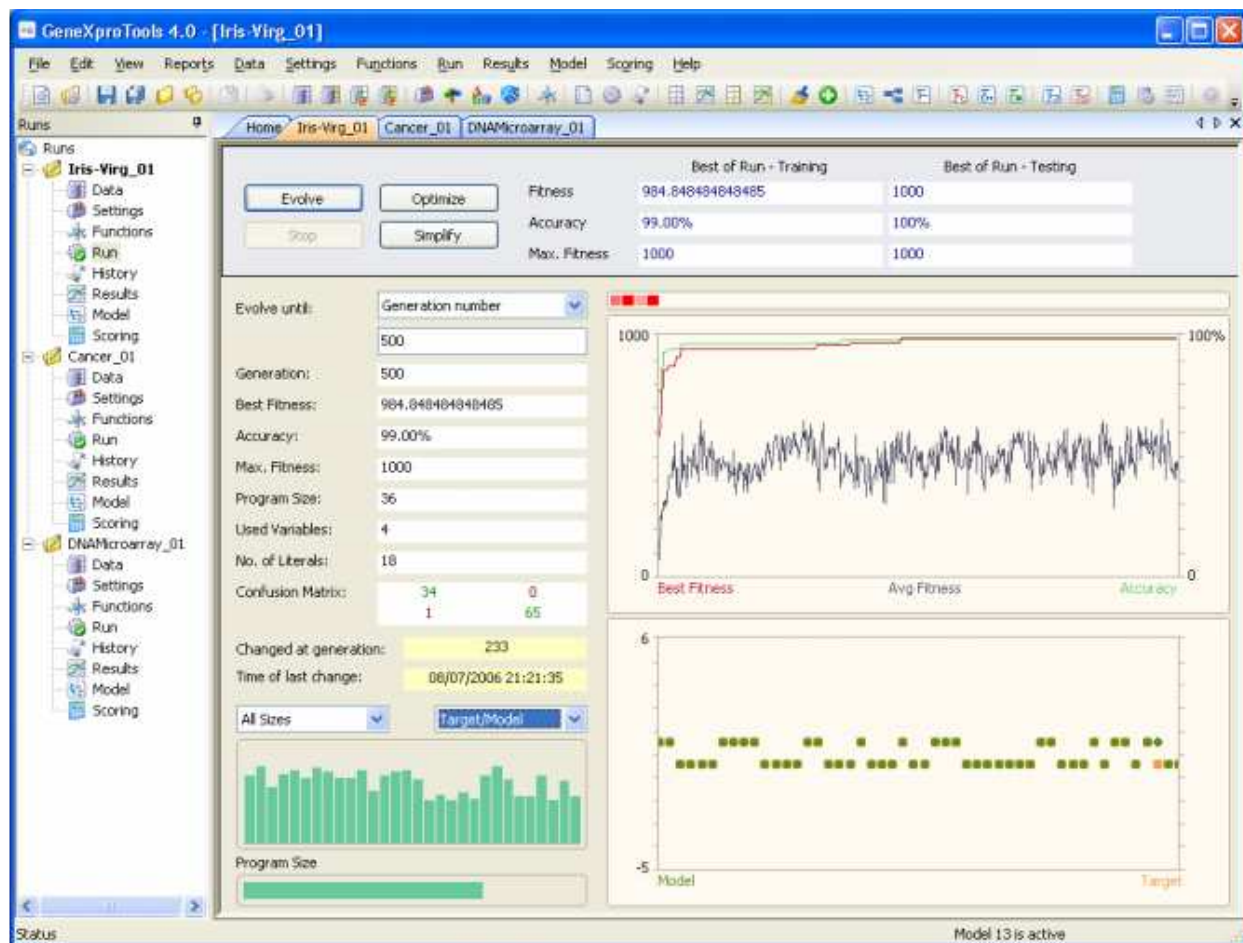
Analyzing GeneXproTools Models Statistically

Confusion Matrix

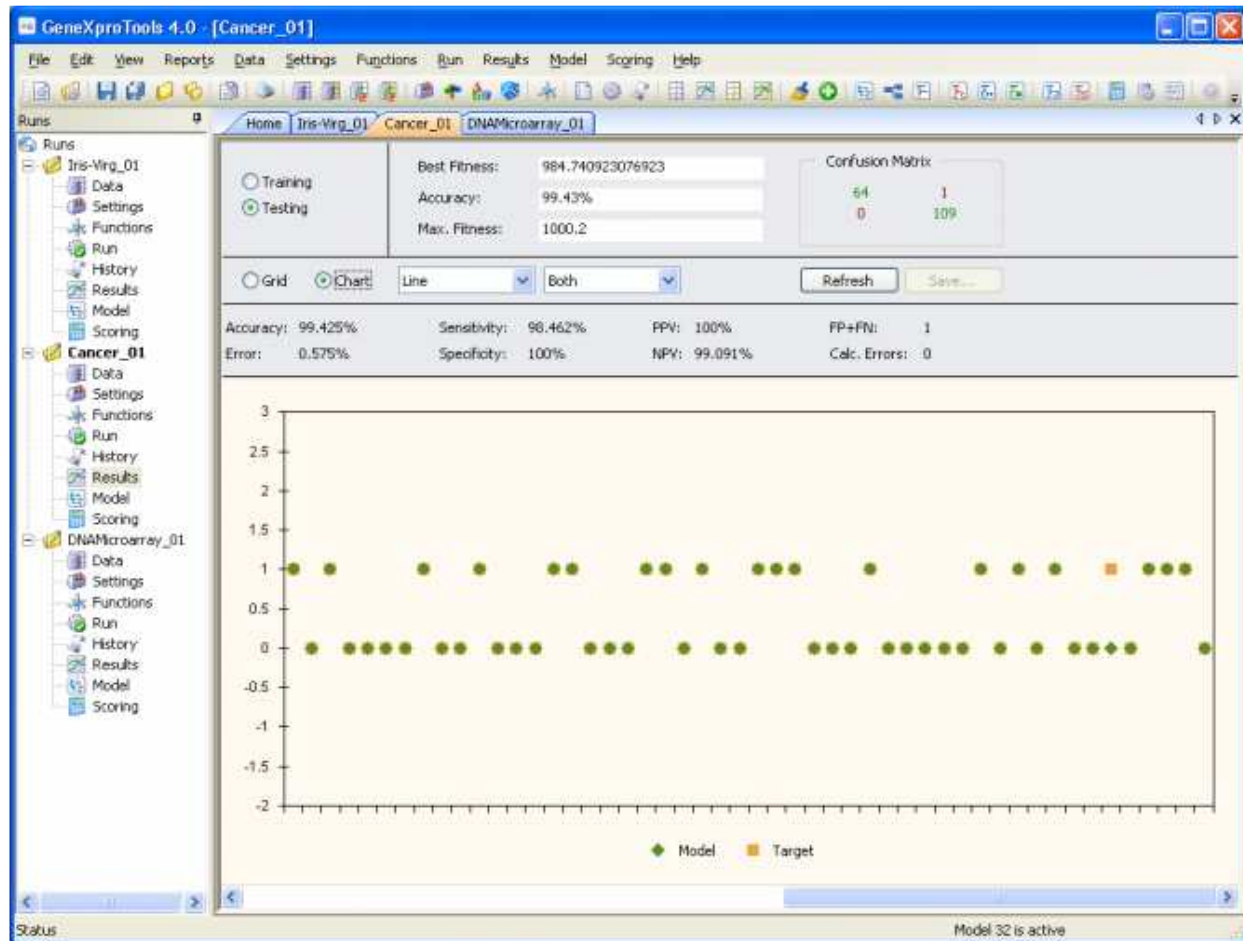
True positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all classification problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **confusion matrix** of your model on the testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Sensitivity

This indicator comes from the medical domain and consists of the probability of the diagnostic test finding disease among those who have the disease or the proportion of people with disease who have a positive test result.

More formally, the **sensitivity** SE_i of an individual program i is evaluated by the equation:

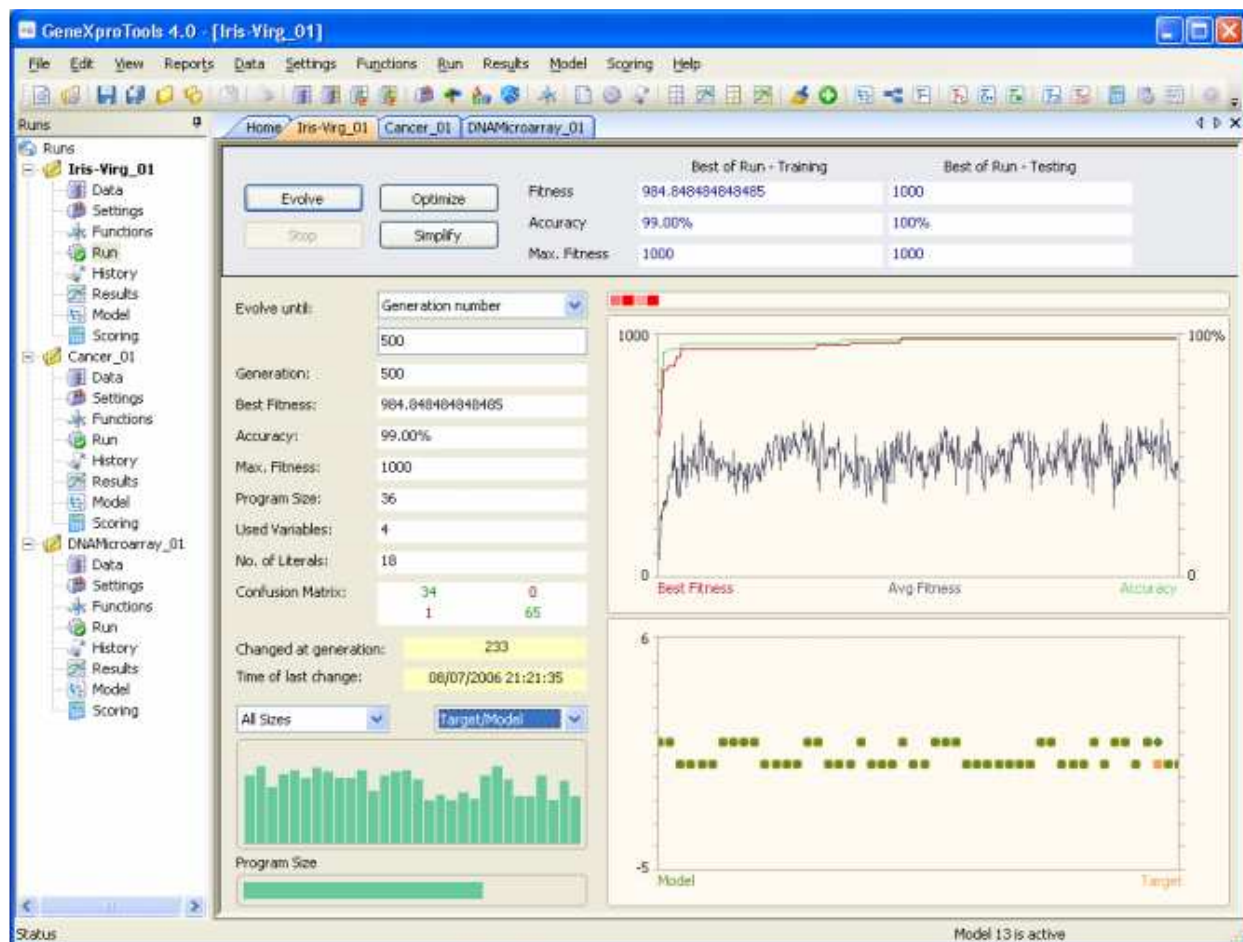
$$SE_i = \frac{TP_i}{TP_i + FN_i}$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

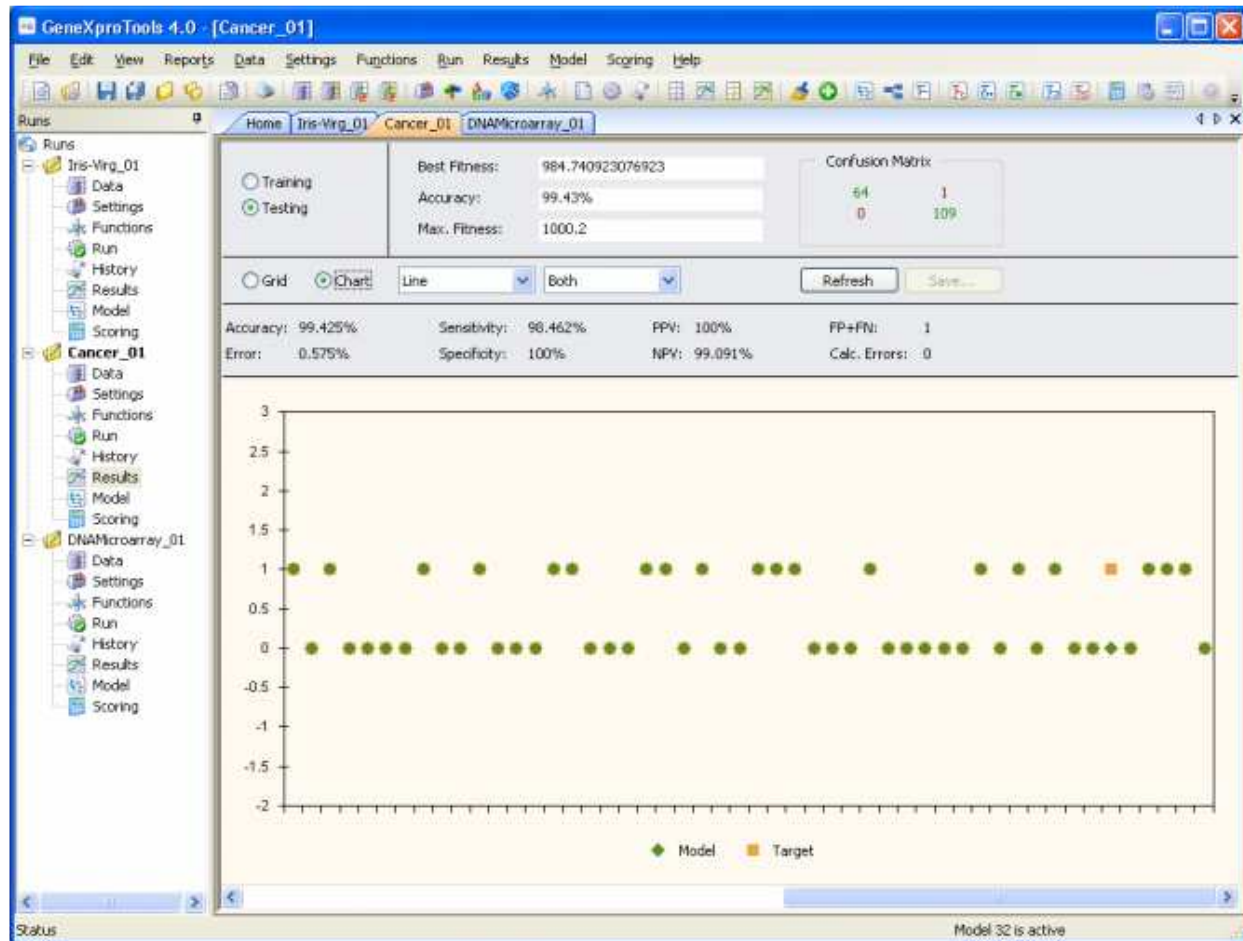
TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all classification problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **sensitivity** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Specificity

This indicator comes from the medical domain and consists of the probability of the diagnostic test finding no disease among those who do not have the disease or the proportion of people free of a disease who have a negative test.

More formally, the **specificity** SP_i of an individual program i is evaluated by the equation:

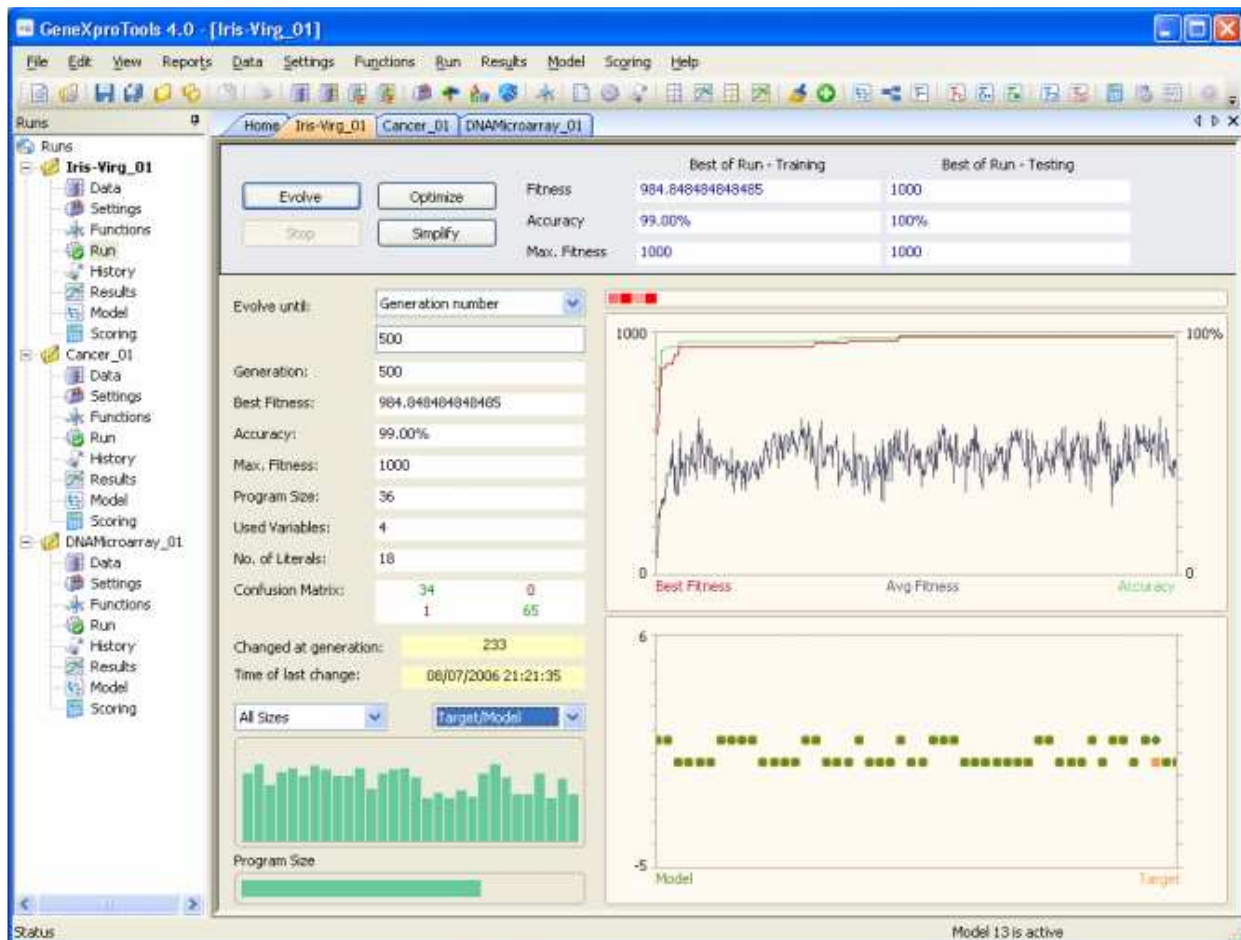
$$SP_i = \frac{TN_i}{TN_i + FP_i}$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

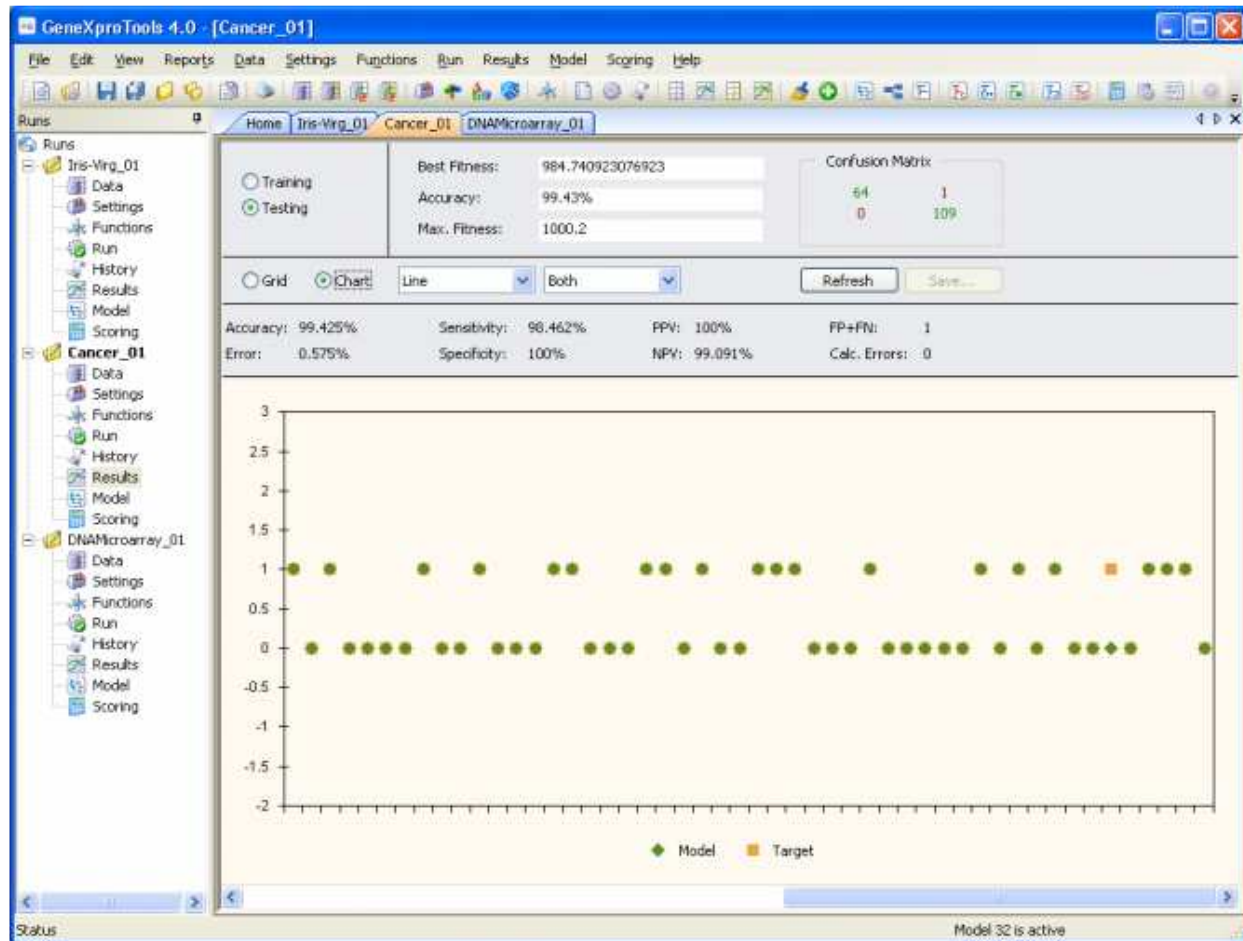
TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all classification problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **specificity** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Positive Predictive Value

This indicator comes from the medical domain and consists of the percentage of people with a positive diagnostic test result who actually have the disease.

More formally, the **positive predictive value** PPV_i of an individual program i is evaluated by the equation:

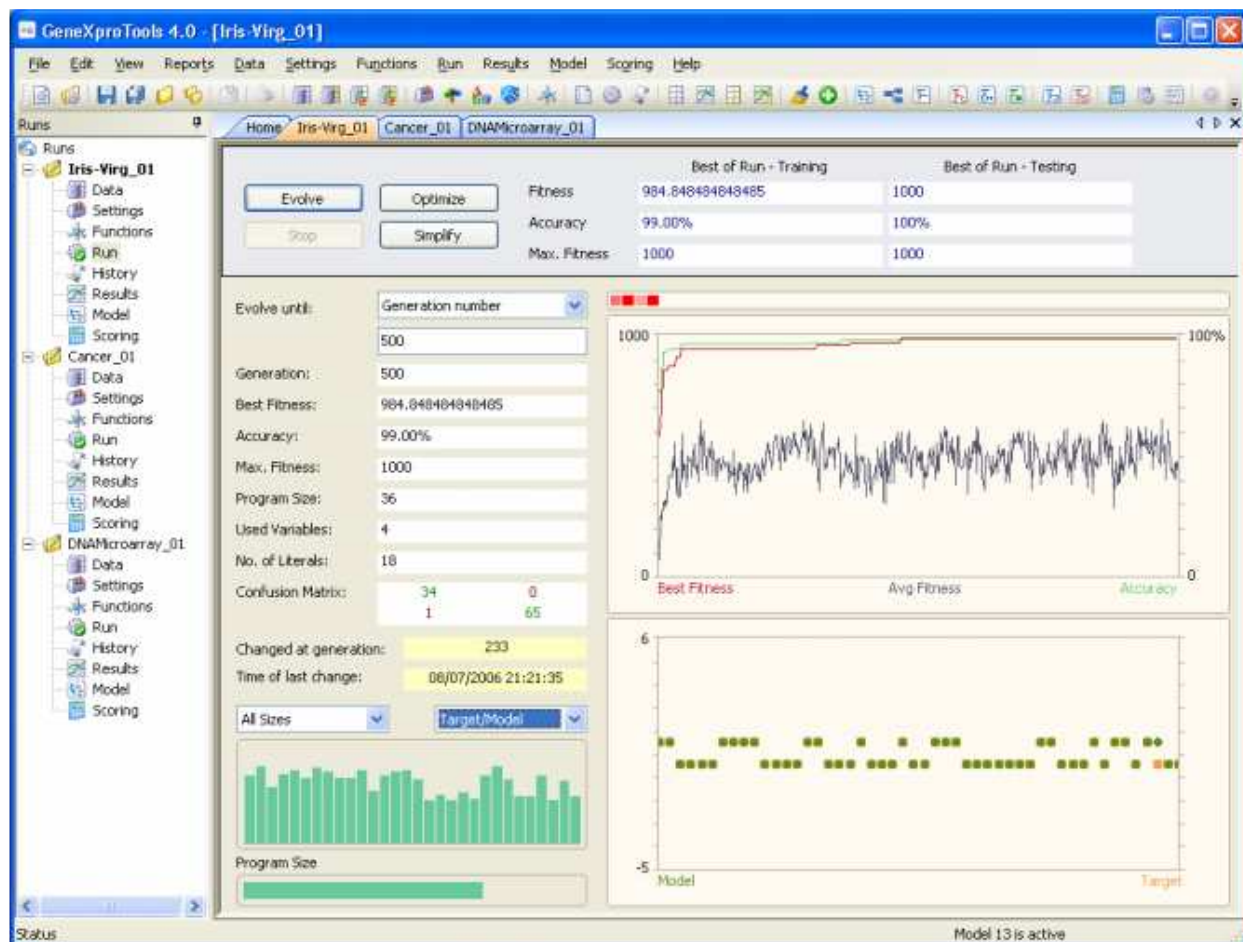
$$PPV_i = \frac{TP_i}{TP_i + FP_i} \text{ where } TP_i + FP_i \neq 0$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

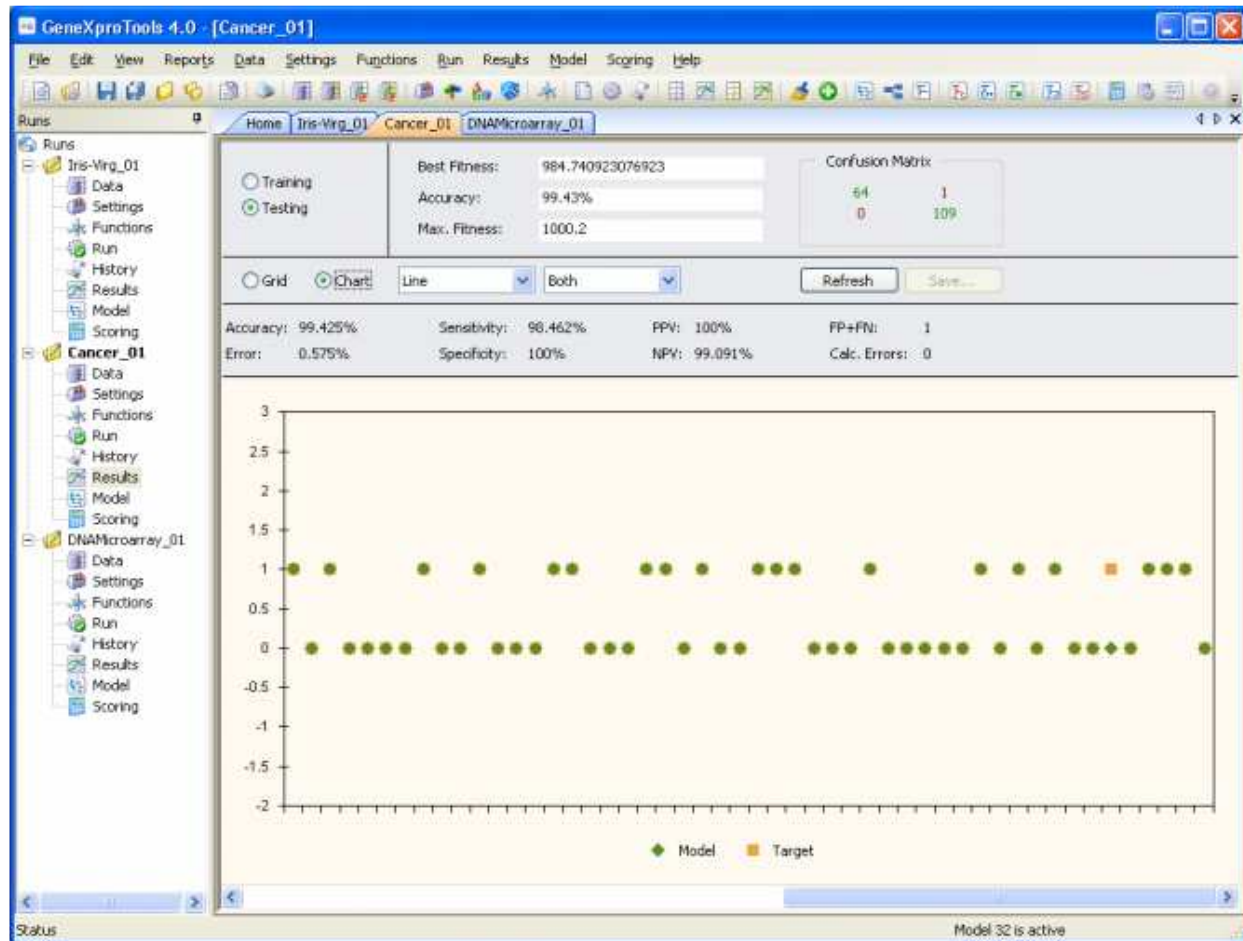
TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all classification problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **PPV** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Negative Predictive Value

This indicator comes from the medical domain and consists of the percentage of people with a negative diagnostic test who do not have the disease.

More formally, the **negative predictive value** NPV_i of an individual program i is evaluated by the equation:

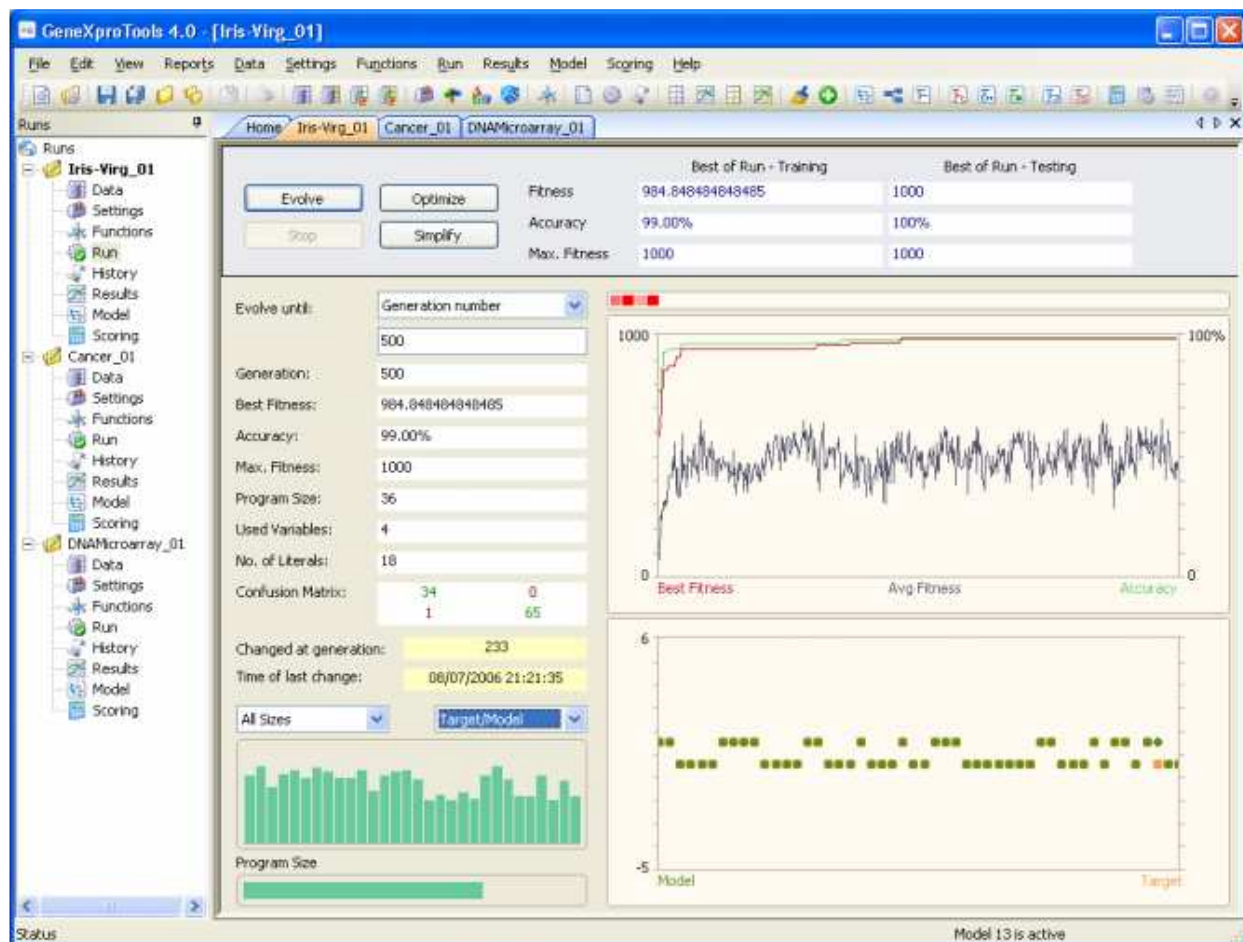
$$NPV_i = \frac{TN_i}{TN_i + FN_i} \text{ where } TN_i + FN_i \neq 0$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

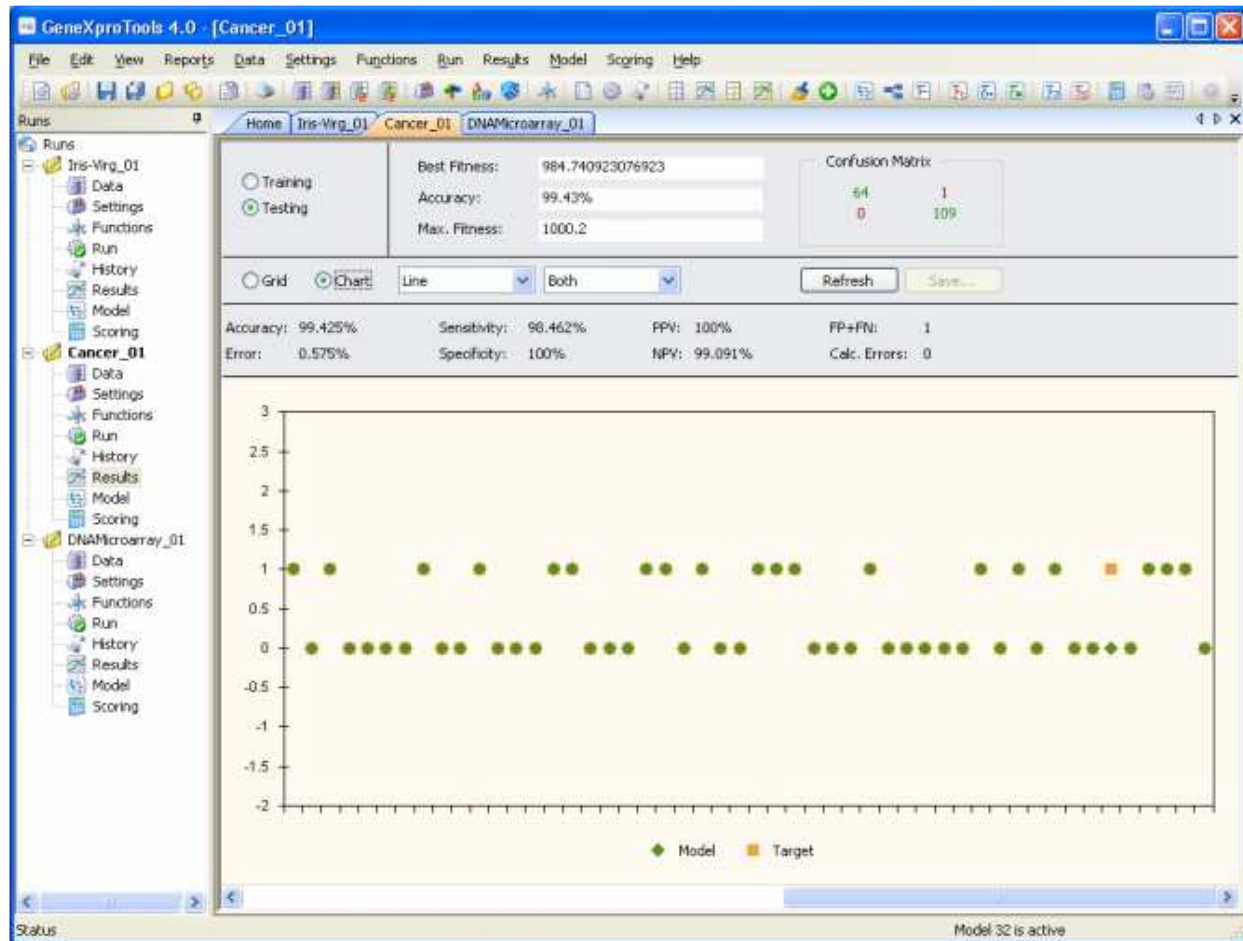
TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all classification problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **NPV** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

R-square

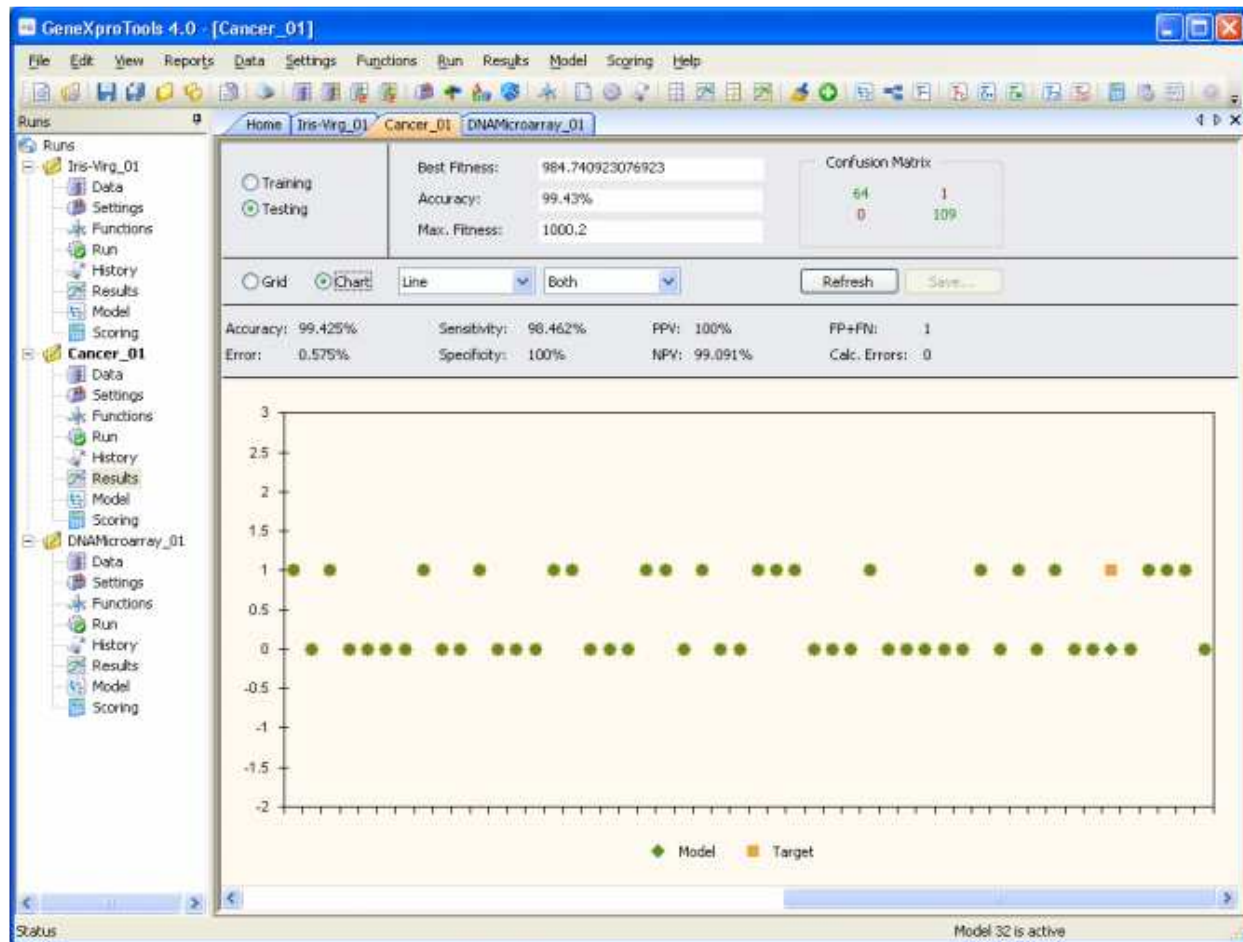
The **R-square** returns the square of the Pearson product moment correlation coefficient, which consists of a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between two data sets.

The Pearson product moment correlation coefficient R_i of an individual program i is evaluated by the equation:

$$R_i = \frac{n \sum_{j=1}^n (T_j P_{(i)}) - \left(\sum_{j=1}^n T_j \right) \left(\sum_{j=1}^n P_{(i)} \right)}{\sqrt{\left[n \sum_{j=1}^n T_j^2 - \left(\sum_{j=1}^n T_j \right)^2 \right] \left[n \sum_{j=1}^n P_{(i)}^2 - \left(\sum_{j=1}^n P_{(i)} \right)^2 \right]}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

To evaluate the **R-square** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Analyzing GeneXproTools Models Statistically

Correlation Coefficient

The **correlation coefficient** C_i of an individual program i is evaluated by the equation:

$$C_i = \frac{Cov(T, P)}{\sigma_i \cdot \sigma_p}$$

where $Cov(T,P)$ is the **covariance** of the target and model outputs; and σ_t and σ_p are the corresponding **standard deviations**, which are given by:

$$Cov(T,P) = \frac{1}{n} \sum_{j=1}^n (T_j - \bar{T})(P_{(ij)} - \bar{P})$$

$$\sigma_t = \sqrt{\frac{\sum_{j=1}^n (T_j - \bar{T})^2}{n}}$$

$$\sigma_p = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - \bar{P})^2}{n}}$$

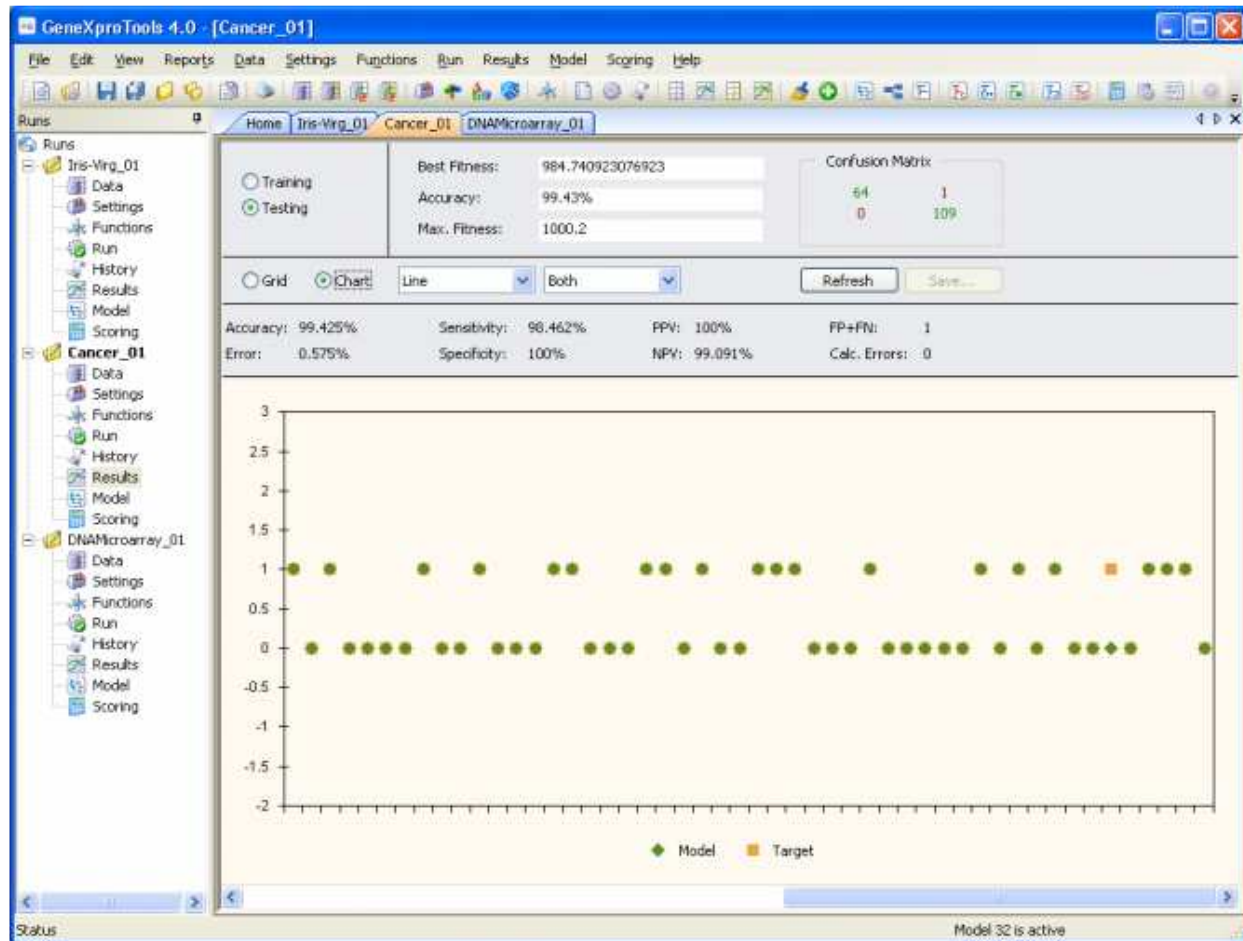
where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ and $\bar{P}$ are given by the formulas:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

$$\bar{P} = \frac{1}{n} \sum_{j=1}^n P_{(ij)}$$

The correlation coefficient is confined to the range $[-1, 1]$. When $C_i = 1$, there is a perfect positive linear correlation between T and P , that is, they vary by the same amount. When $C_i = -1$, there is a perfect negative linear correlation between T and P , that is, they vary in opposite ways (when T increases, P decreases by the same amount). When $C_i = 0$, there is no correlation between T and P . Intermediate values describe partial correlations and the closer to 1 or -1 the better the model.

To evaluate the **correlation coefficient** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Analyzing GeneXproTools Models Statistically

Mean Squared Error

The **mean squared error** E_i of an individual program i is evaluated by the equation:

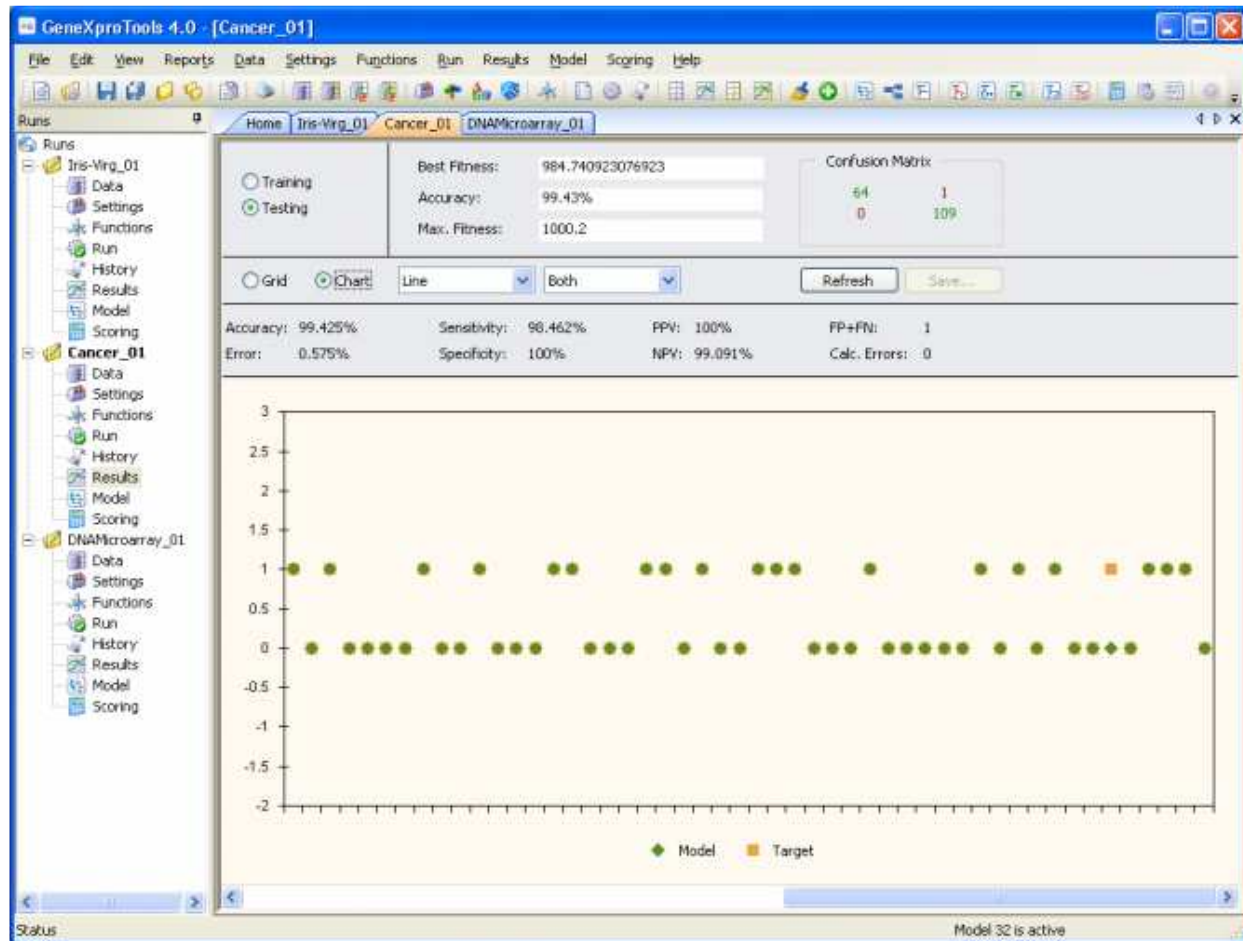
$$E_i = \frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **MSE** of your model both on the training and testing data, you just have to go to the

[Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Analyzing GeneXproTools Models Statistically

Root Mean Squared Error

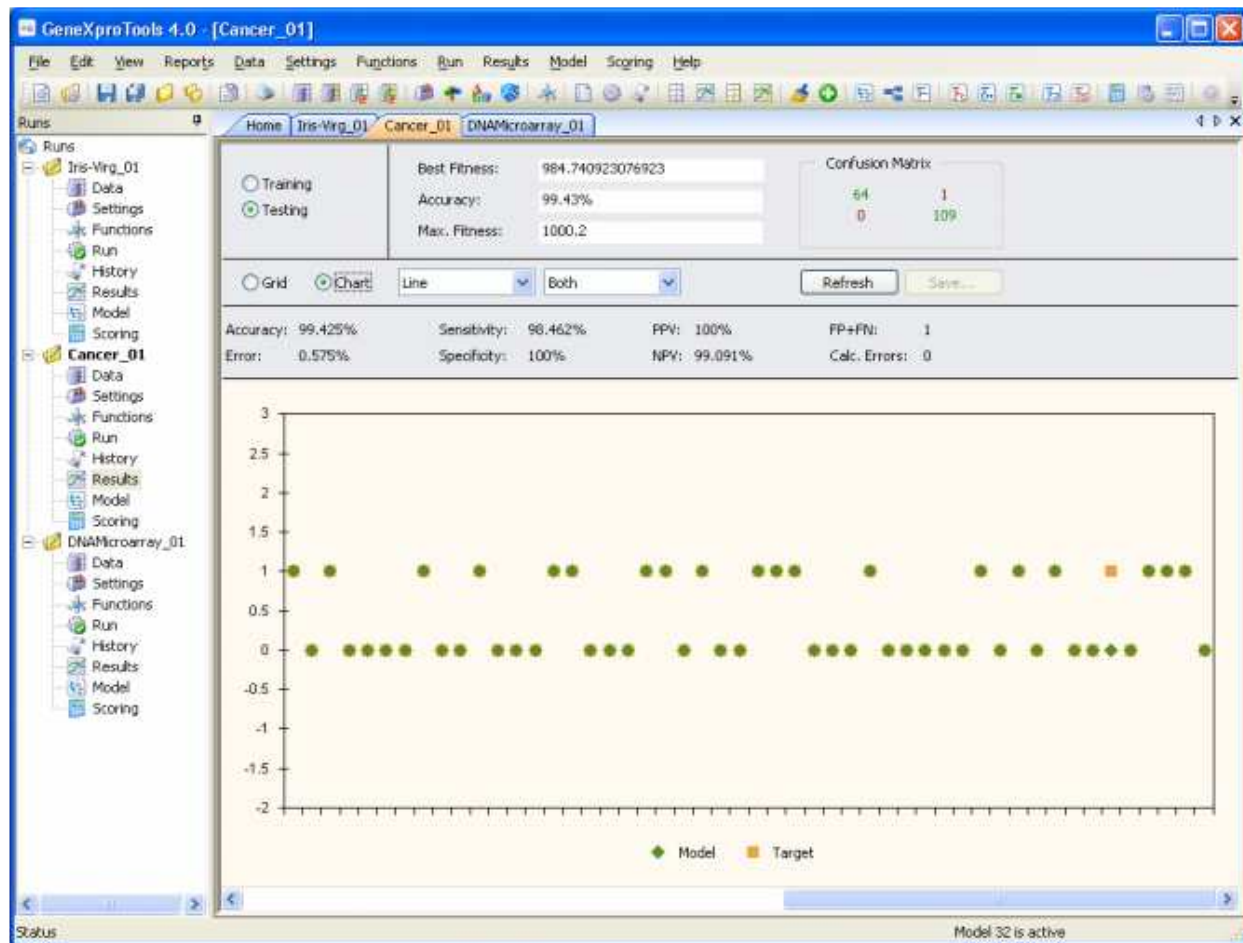
The **root mean squared error** E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RMSE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



◀ | ▶

Analyzing GeneXproTools Models Statistically

Mean Absolute Error

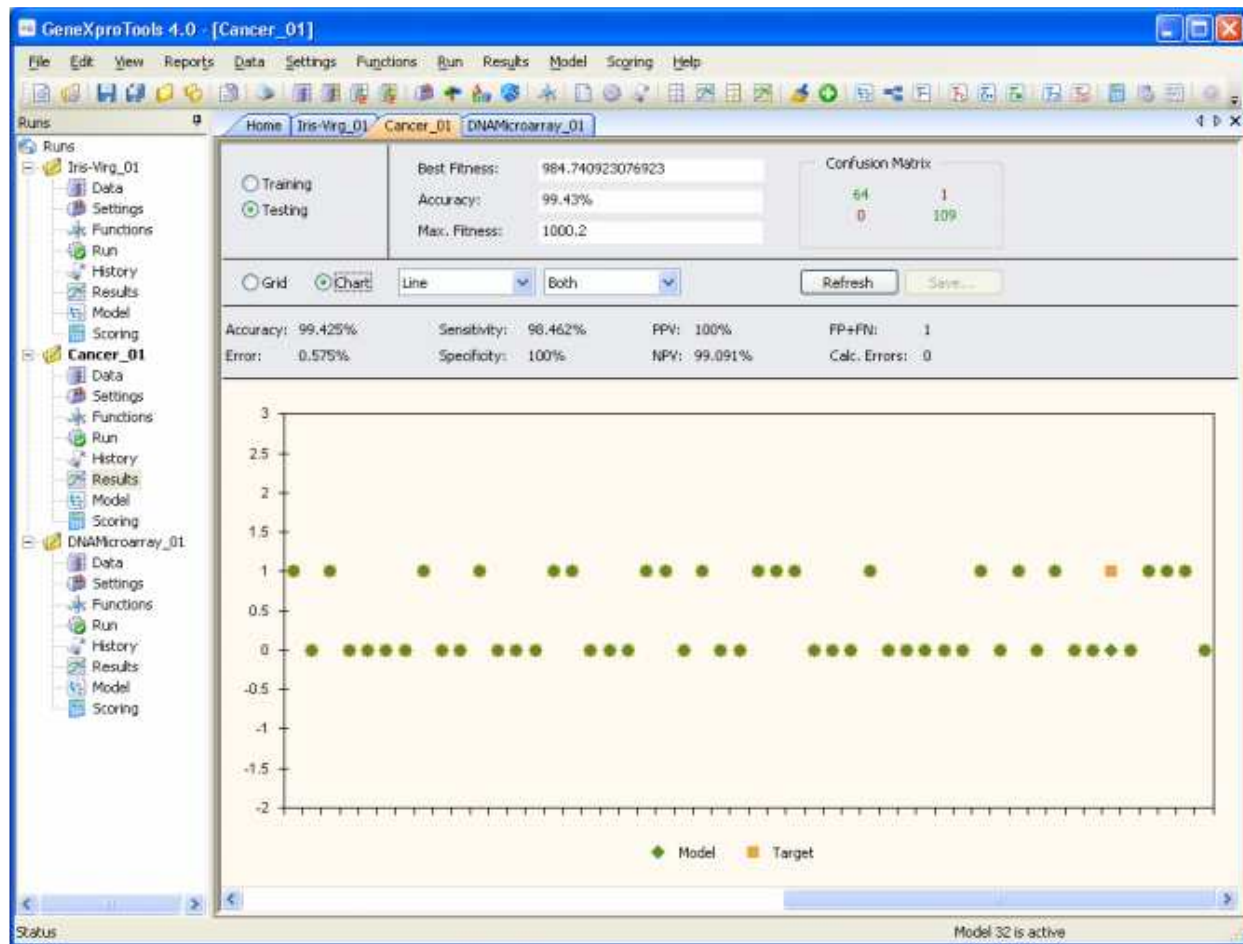
The **mean absolute error** E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **MAE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



◀ | ▶

Analyzing GeneXproTools Models Statistically

Relative Squared Error

The **relative squared error** is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the **relative squared error** E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}$$

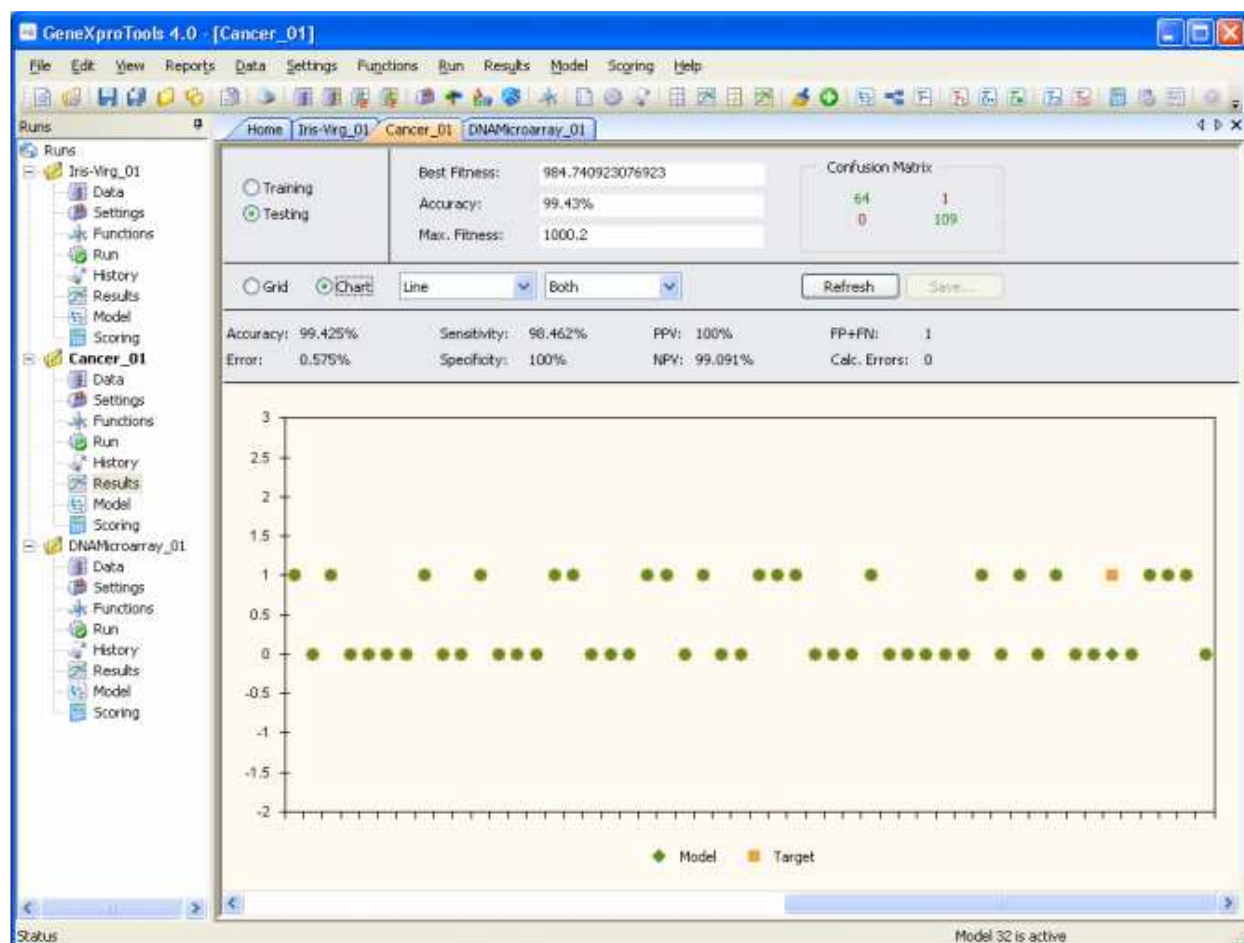
where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases);

T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RSE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).





Analyzing GeneXproTools Models Statistically

Root Relative Squared Error

The **root relative squared error** is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the **root relative squared error** E_i of an individual program i is evaluated by the equation:

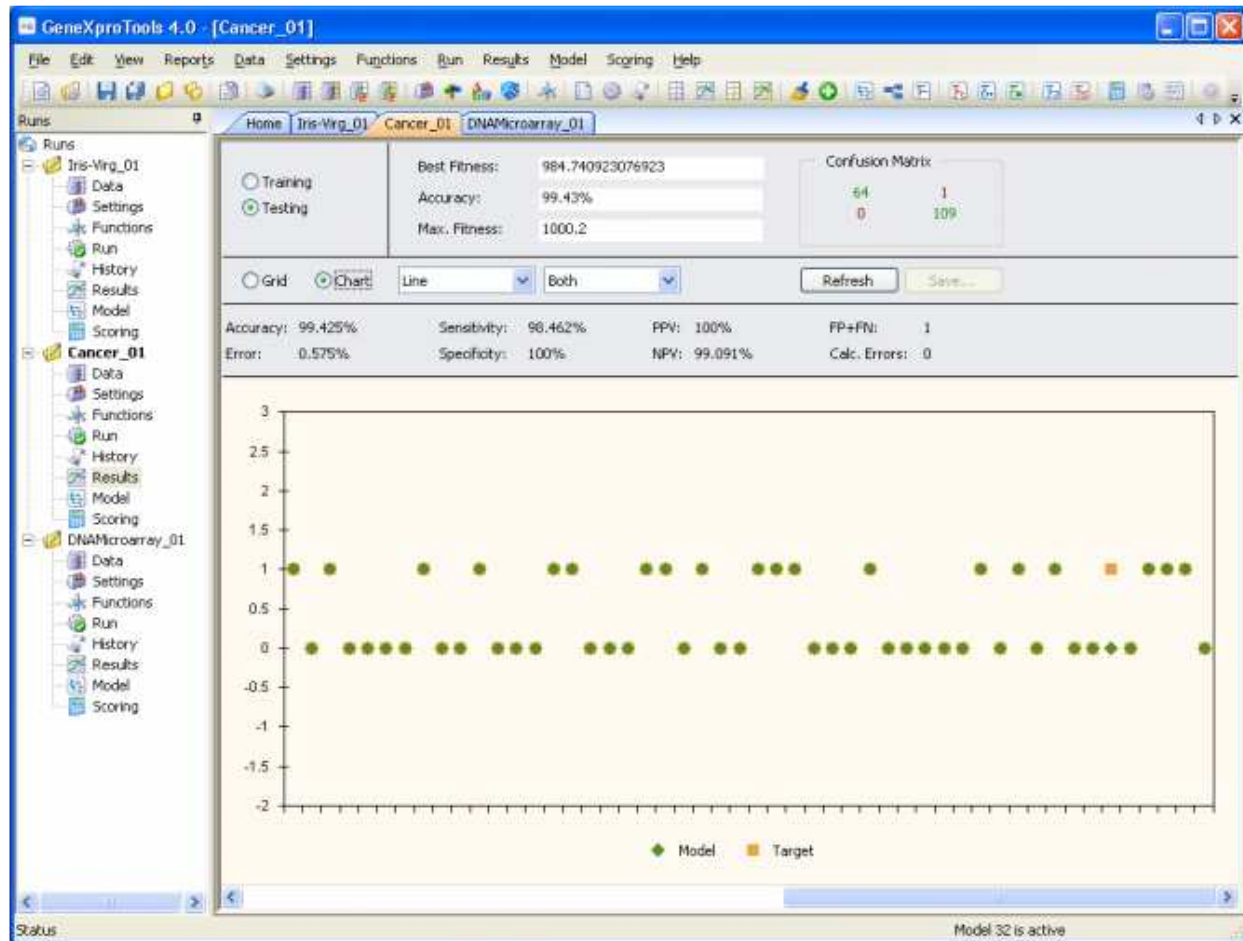
$$E_i = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RRSE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



◀ | ▶

Analyzing GeneXproTools Models Statistically

Relative Absolute Error

The **relative absolute error** is very similar to the **relative squared error** in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the **relative absolute error** E_i of an individual program i is evaluated by the equation:

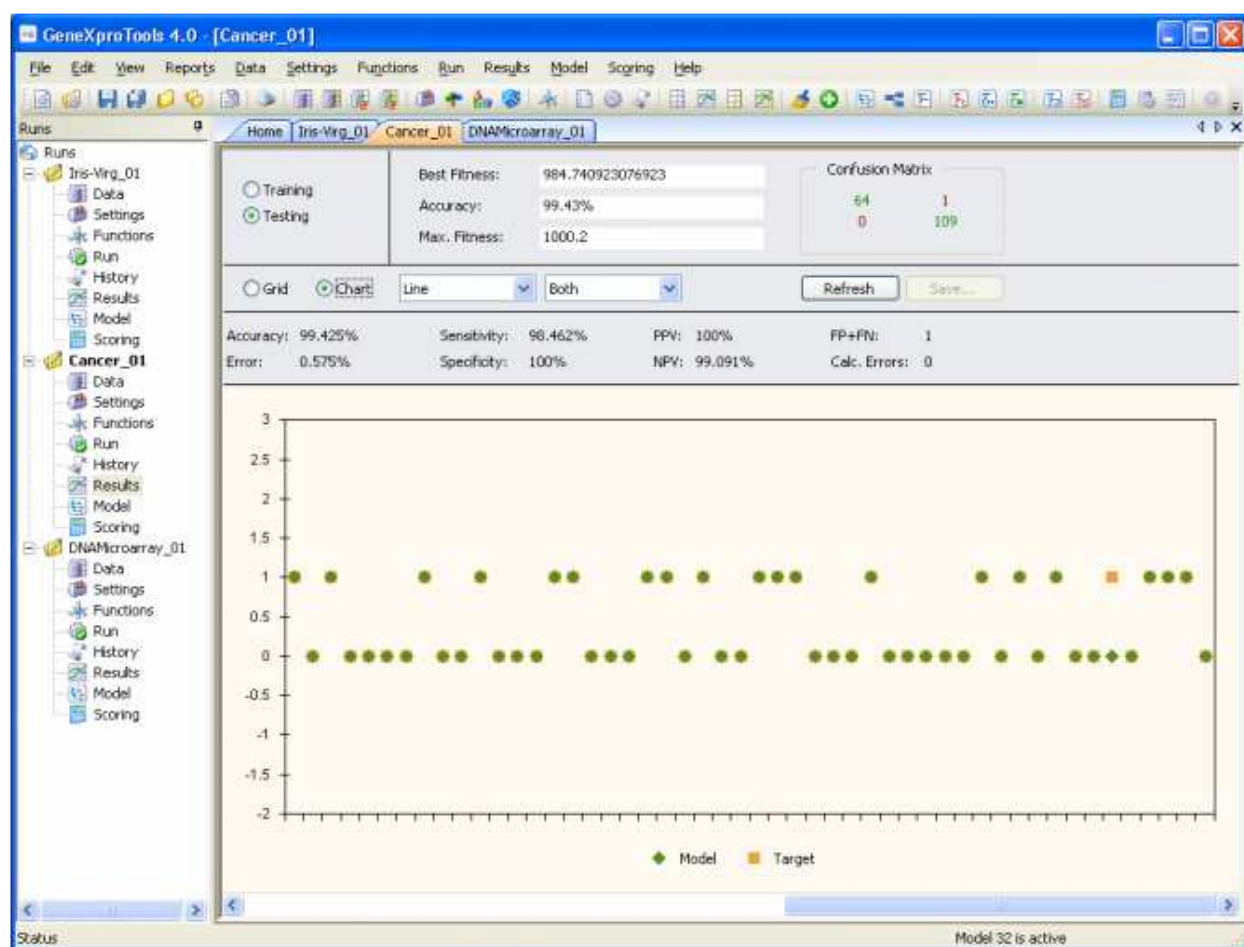
$$E_i = \frac{\sum_{j=1}^n |P_{ij} - T_j|}{\sum_{j=1}^n |T_j - \bar{T}|}$$

where P_{ij} is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RAE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).

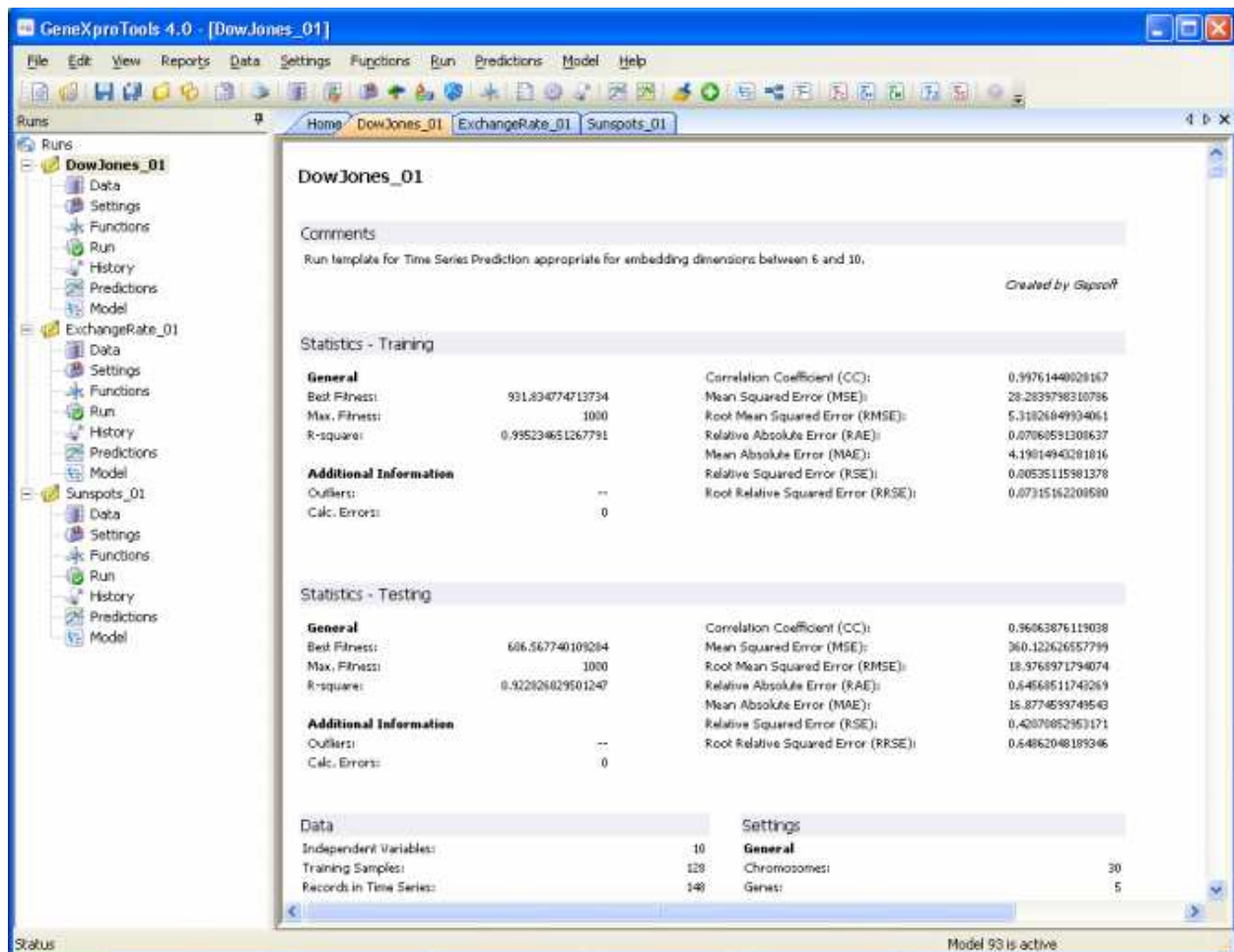


Analyzing GeneXproTools Models Statistically

Statistical Analysis of Time Series Prediction Models

For Time Series Prediction models, GeneXproTools 4.0 enables you to evaluate easily and without leaving the GeneXproTools modeling environment, the following statistical functions both for training and recursive testing:

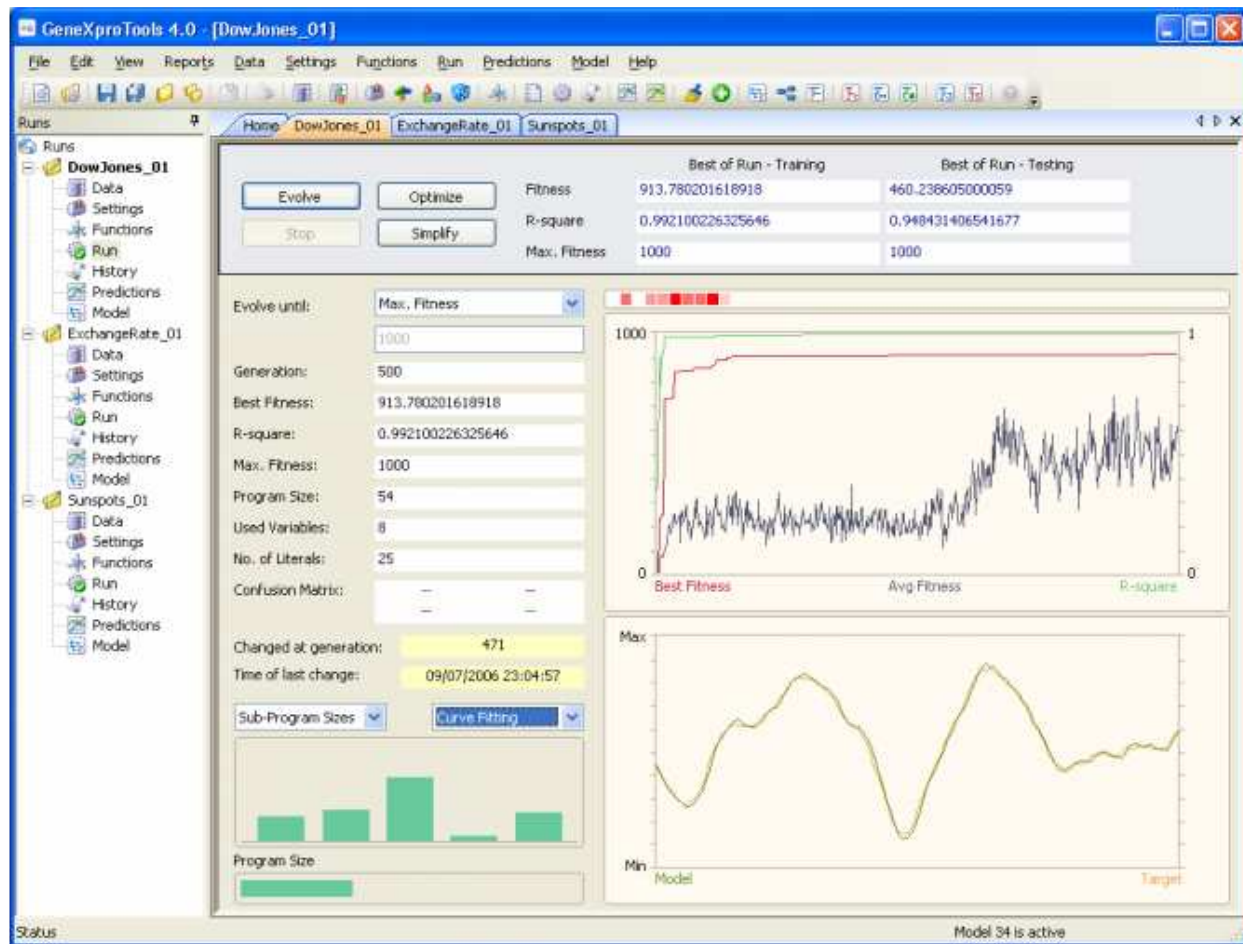
- [R-square](#)
- [Correlation Coefficient](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)



Analyzing GeneXproTools Models Statistically

R-square

The **R-square** returns the square of the Pearson product moment correlation coefficient, which consists of a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between two data sets. During a run, GeneXproTools updates continuously the R-square of the evolving models and plots it in a chart in the [Run Panel](#).

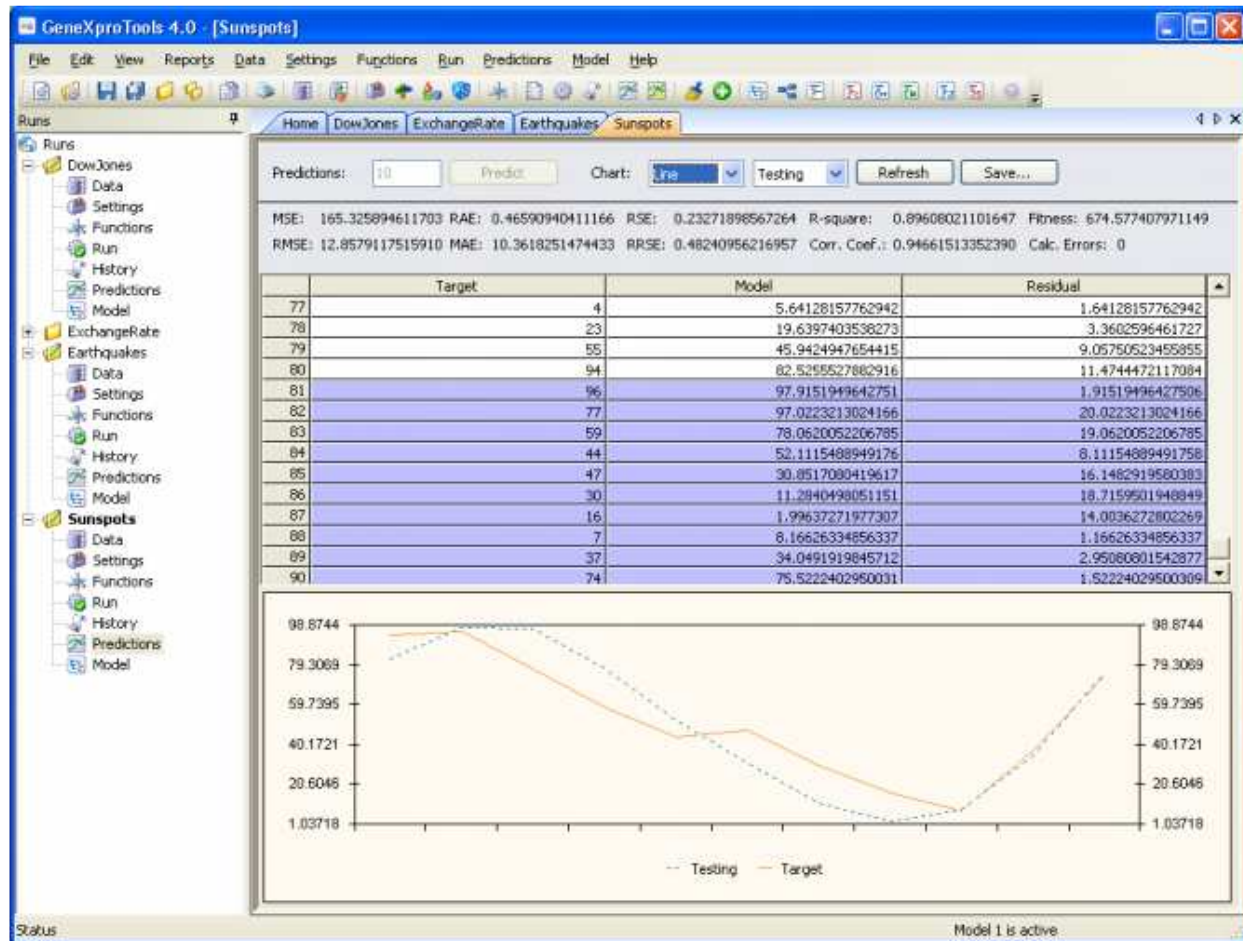


The Pearson product moment correlation coefficient R_i of an individual program i is evaluated by the equation:

$$R_i = \frac{n \sum_{j=1}^n (T_j P_{(ij)}) - \left(\sum_{j=1}^n T_j \right) \left(\sum_{j=1}^n P_{(ij)} \right)}{\sqrt{\left[n \sum_{j=1}^n T_j^2 - \left(\sum_{j=1}^n T_j \right)^2 \right] \left[n \sum_{j=1}^n P_{(ij)}^2 - \left(\sum_{j=1}^n P_{(ij)} \right)^2 \right]}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

To evaluate the **R-square** of your model on the testing set, you just have to go to the [Predictions Panel](#) after a run.



◀ | ▶

Analyzing GeneXproTools Models Statistically

Correlation Coefficient

The **correlation coefficient** C_i of an individual program i is evaluated by the equation:

$$C_i = \frac{Cov(T, P)}{\sigma_t \cdot \sigma_p}$$

where $Cov(T, P)$ is the **covariance** of the target and model outputs; and σ_t and σ_p are the corresponding **standard deviations**, which are given by:

$$Cov(T, P) = \frac{1}{n} \sum_{j=1}^n (T_j - \bar{T})(P_{(ij)} - \bar{P})$$

$$\sigma_t = \sqrt{\frac{\sum_{j=1}^n (T_j - \bar{T})^2}{n}}$$

$$\sigma_p = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - \bar{P})^2}{n}}$$

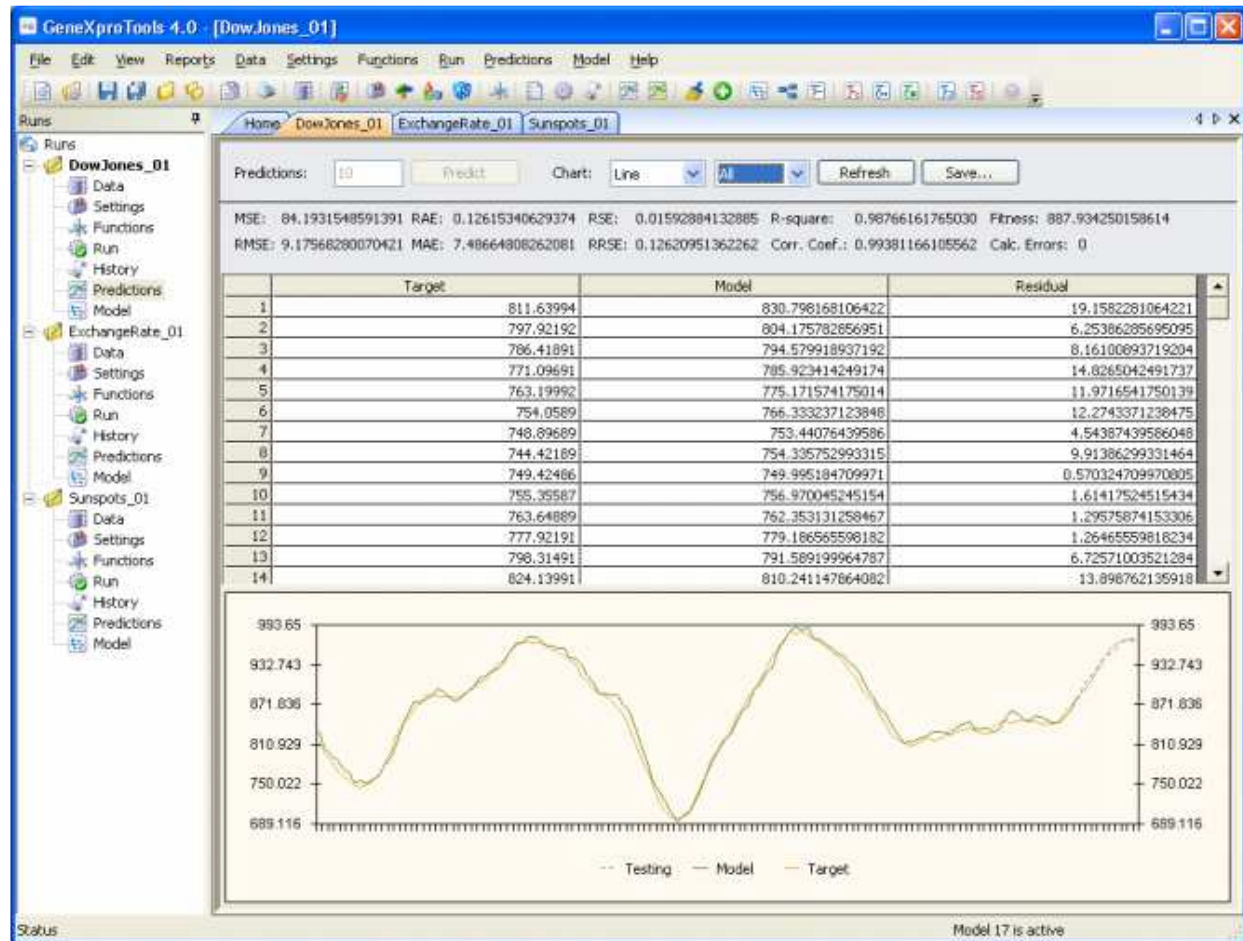
where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ and $\bar{P}$ are given by the formulas:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

$$\bar{P} = \frac{1}{n} \sum_{j=1}^n P_{(ij)}$$

The correlation coefficient is confined to the range [-1, 1]. When $C_i = 1$, there is a perfect positive linear correlation between T and P , that is, they vary by the same amount. When $C_i = -1$, there is a perfect negative linear correlation between T and P , that is, they vary in opposite ways (when T increases, P decreases by the same amount). When $C_i = 0$, there is no correlation between T and P . Intermediate values describe partial correlations and the closer to 1 or -1 the better the model.

To evaluate the **correlation coefficient** of your model both on the training and testing data, you just have to go to the [Predictions Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Mean Squared Error

The **mean squared error** E_i of an individual program i is evaluated by the equation:

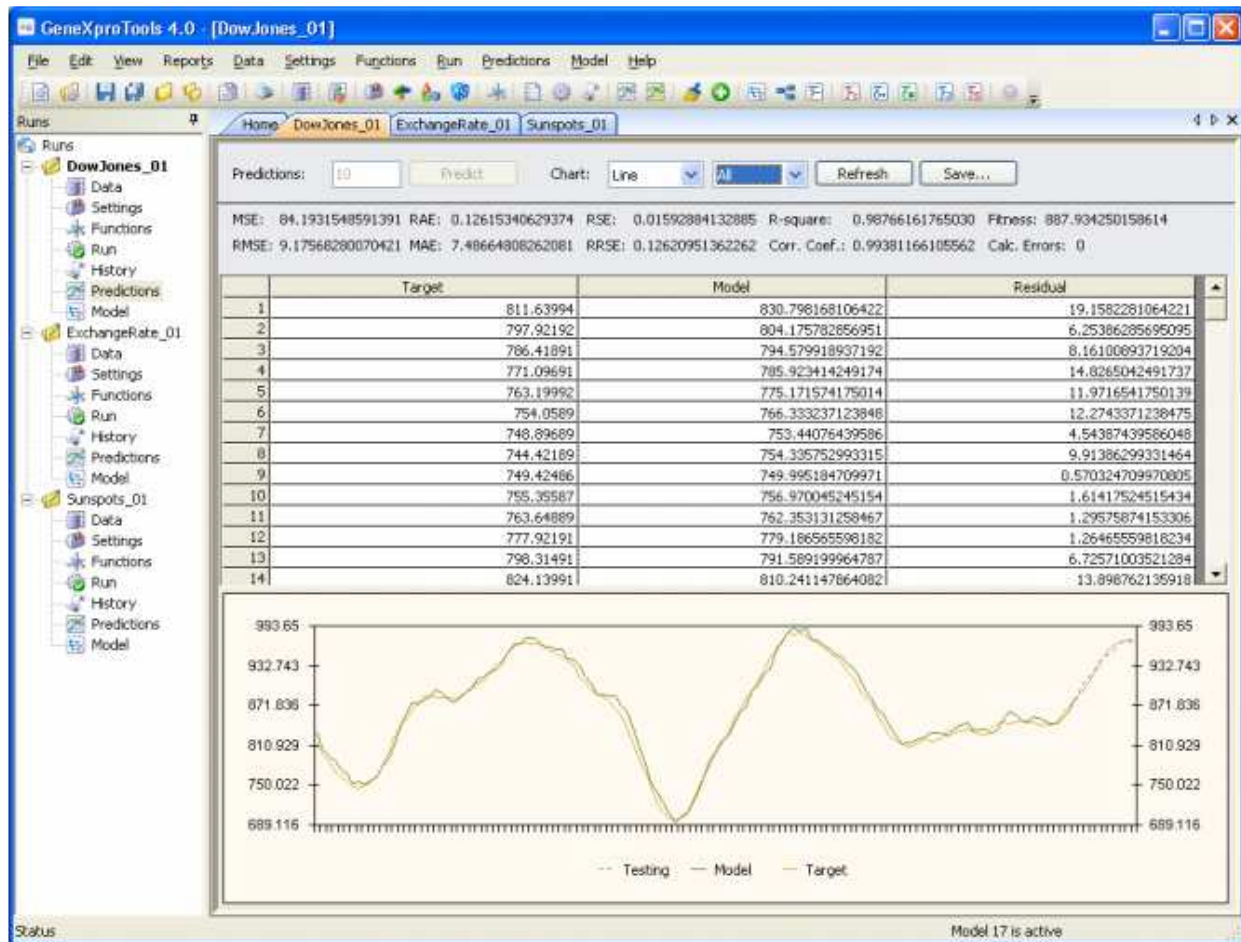
$$E_i = \frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **MSE** of your model both on the training and testing data, you just have to go to the

Predictions Panel after a run.



◀ | ▶

Analyzing GeneXproTools Models Statistically

Root Mean Squared Error

The **root mean squared error** E_i of an individual program i is evaluated by the equation:

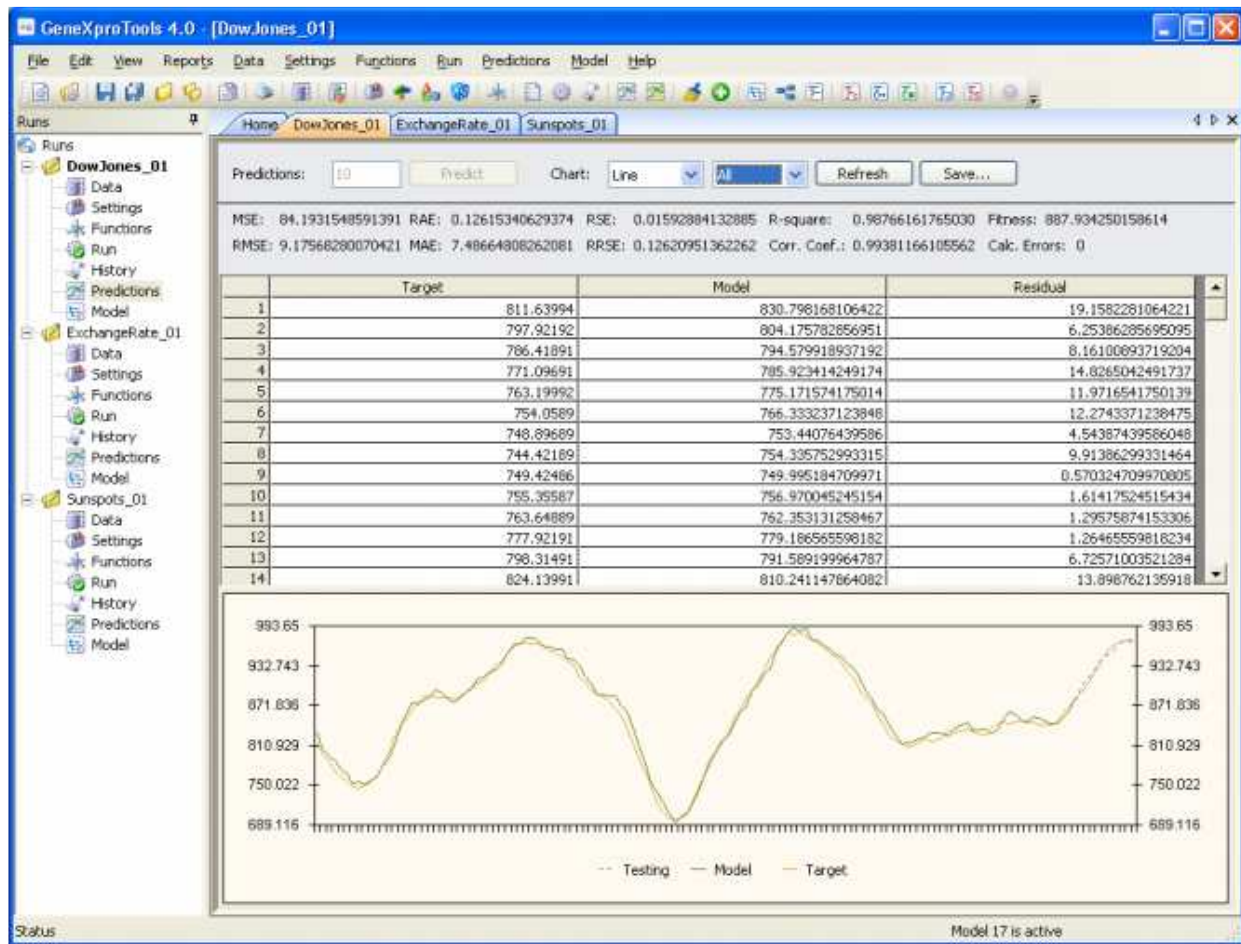
$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to

the ideal.

To evaluate the **RMSE** of your model both on the training and testing data, you just have to go to the [Predictions Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Mean Absolute Error

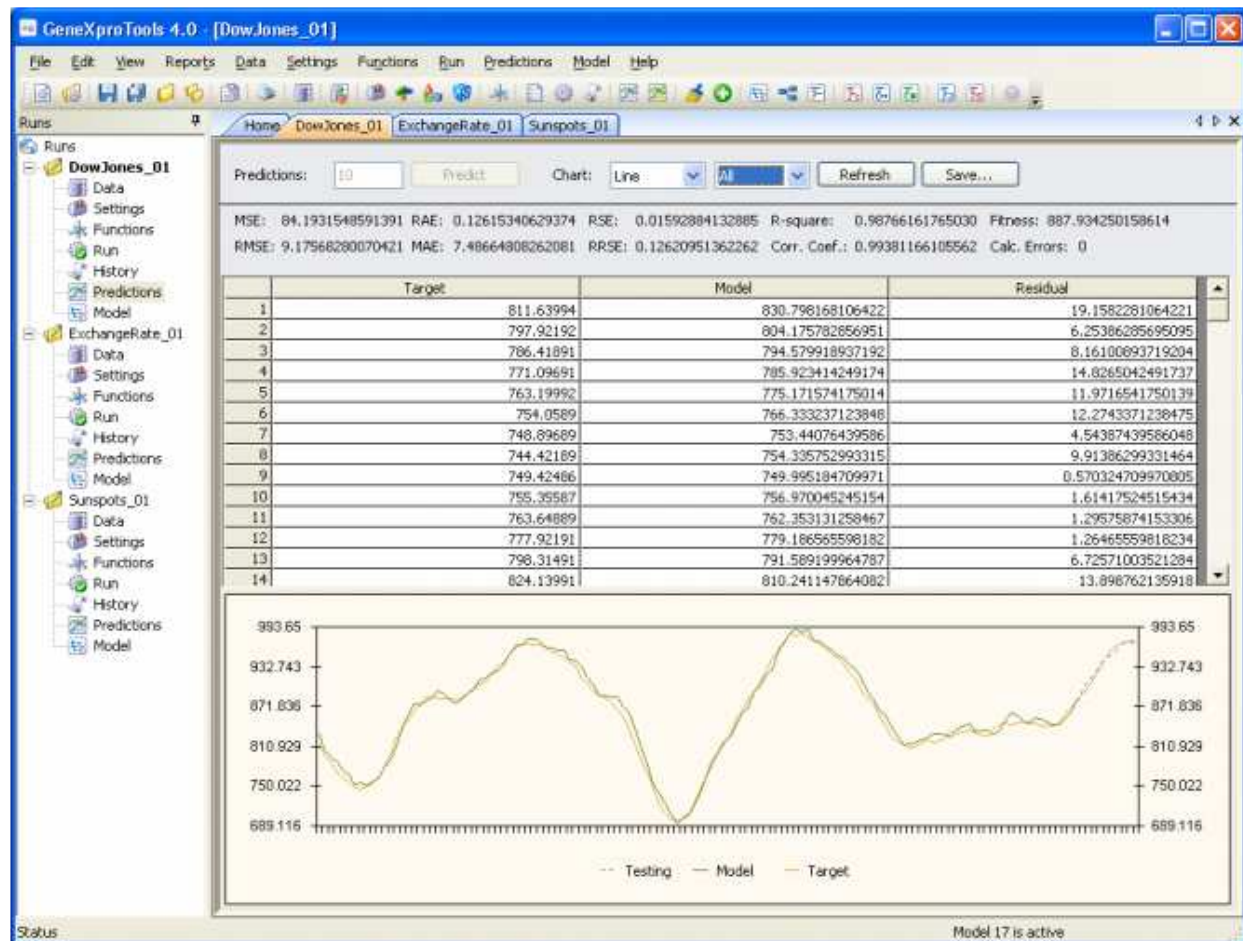
The **mean absolute error** E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **MAE** of your model both on the training and testing data, you just have to go to the **Predictions Panel** after a run.



Analyzing GeneXproTools Models Statistically

Relative Squared Error

The **relative squared error** is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the **relative squared error** E_i of an individual program i is evaluated by the equation:

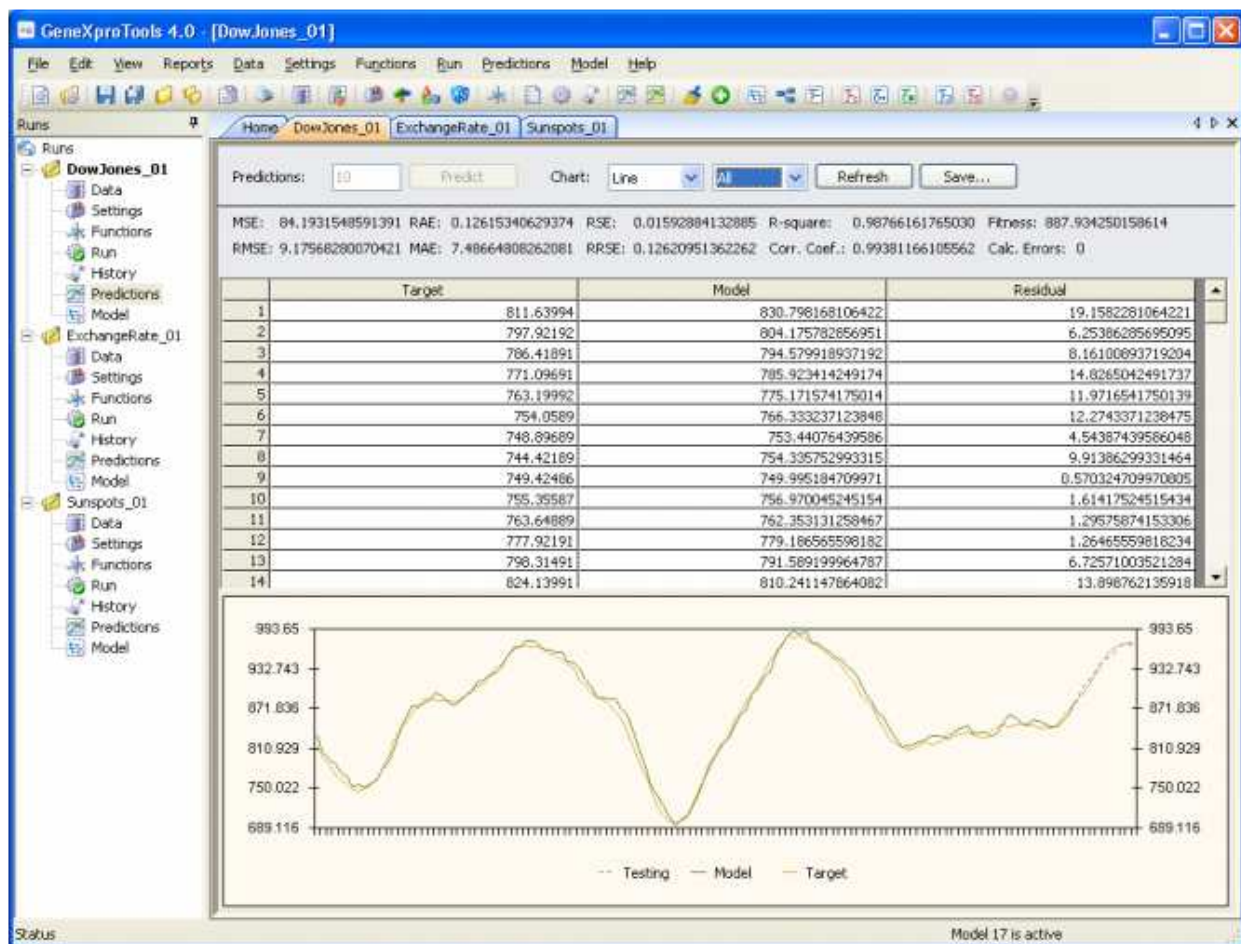
$$E_i = \frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RSE** of your model both on the training and testing data, you just have to go to the [Predictions Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Root Relative Squared Error

The **root relative squared error** is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the **root relative squared error** E_i of an individual program i is evaluated by the equation:

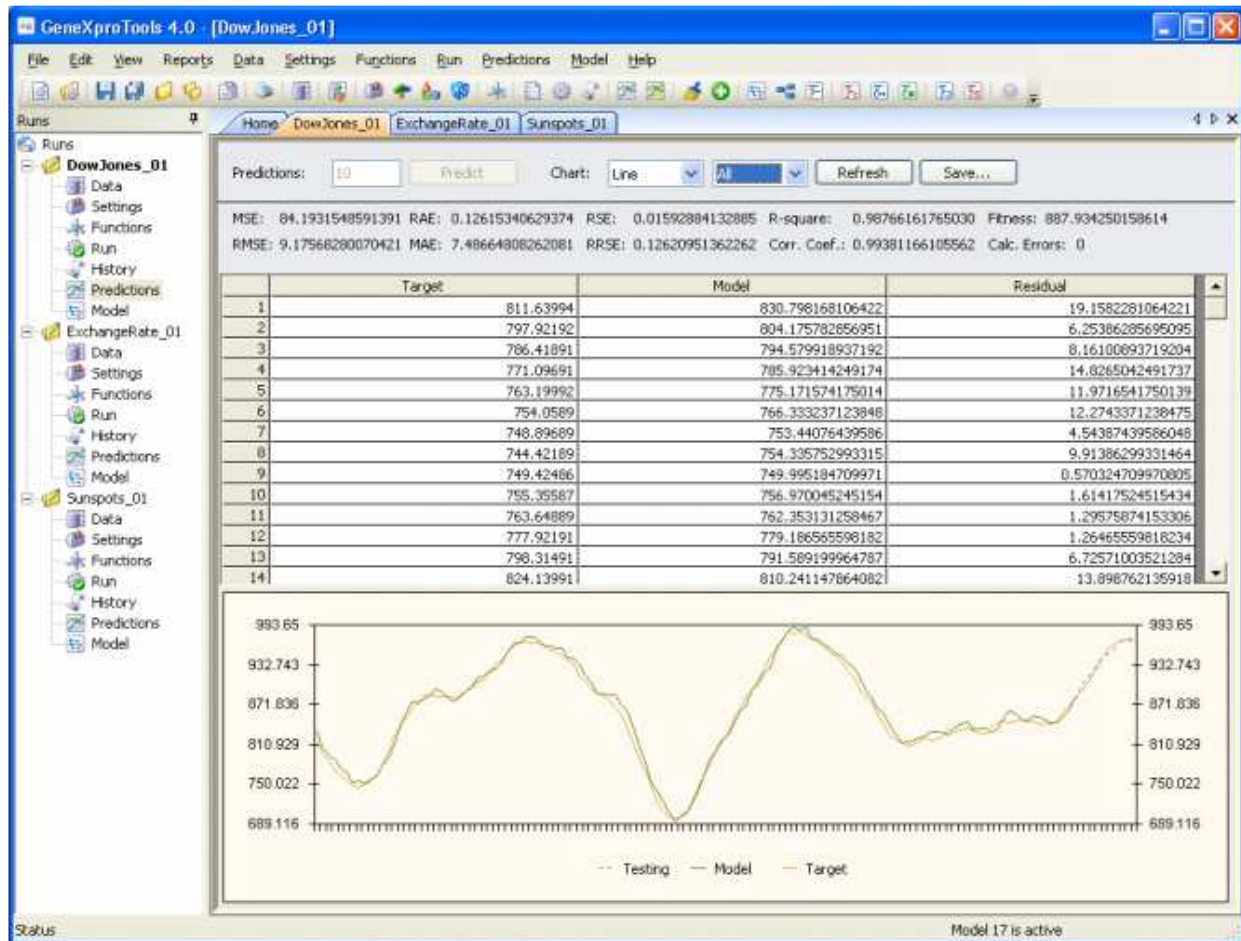
$$E_i = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RRSE** of your model both on the training and testing data, you just have to go to the [Predictions Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Relative Absolute Error

The **relative absolute error** is very similar to the **relative squared error** in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the **relative absolute error** E_i of an individual program i is evaluated by the equation:

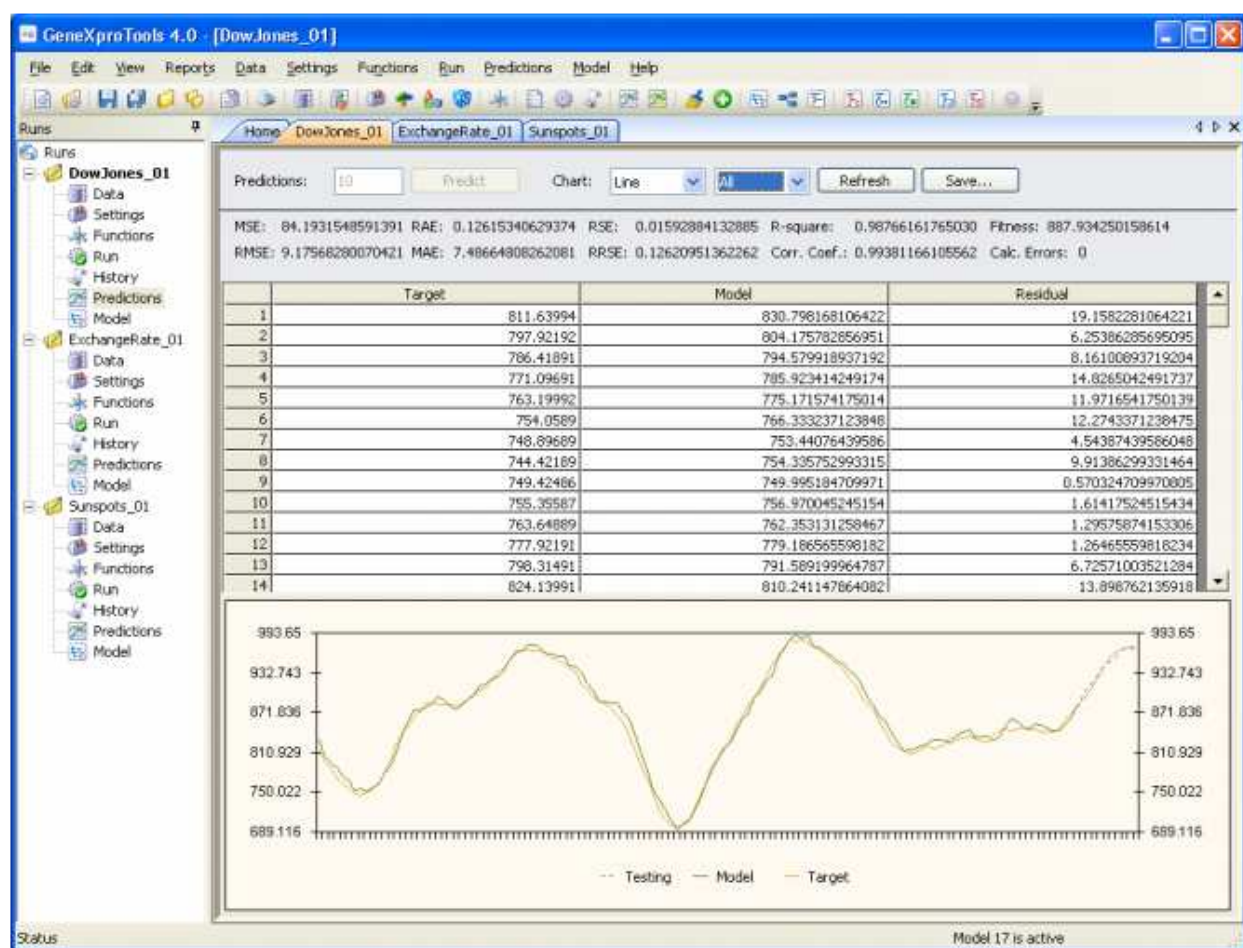
$$E_i = \frac{\sum_{j=1}^n |P_{ij} - T_j|}{\sum_{j=1}^n |T_j - \bar{T}|}$$

where P_{ij} is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RAE** of your model both on the training and testing data, you just have to go to the [Predictions Panel](#) after a run.

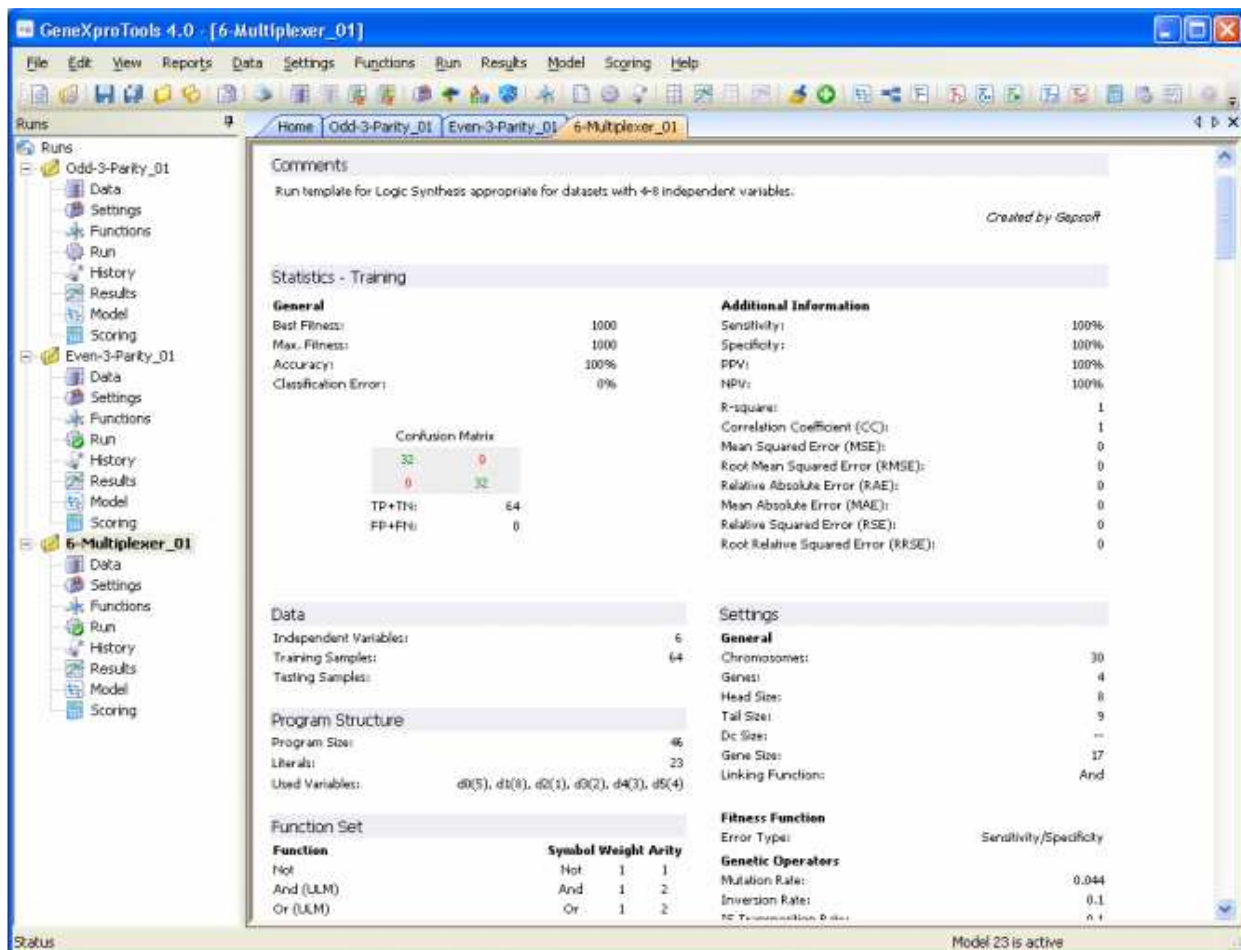


Analyzing GeneXproTools Models Statistically

Statistical Analysis of Logic Synthesis Models

For Logic Synthesis models, GeneXproTools 4.0 enables you to evaluate easily and without leaving the GeneXproTools modeling environment, the following statistical functions both for training and testing data:

- [Classification Error](#)
- [Classification Accuracy](#)
- [Confusion Matrix](#)
- [Sensitivity](#)
- [Specificity](#)
- [PPV](#) (positive predictive value)
- [NPV](#) (negative predictive value)
- [R-square](#)
- [Correlation Coefficient](#)
- [MSE](#) (mean squared error)
- [RMSE](#) (root mean squared error)
- [MAE](#) (mean absolute error)
- [RSE](#) (relative squared error)
- [RRSE](#) (root relative squared error)
- [RAE](#) (relative absolute error)





Analyzing GeneXproTools Models Statistically

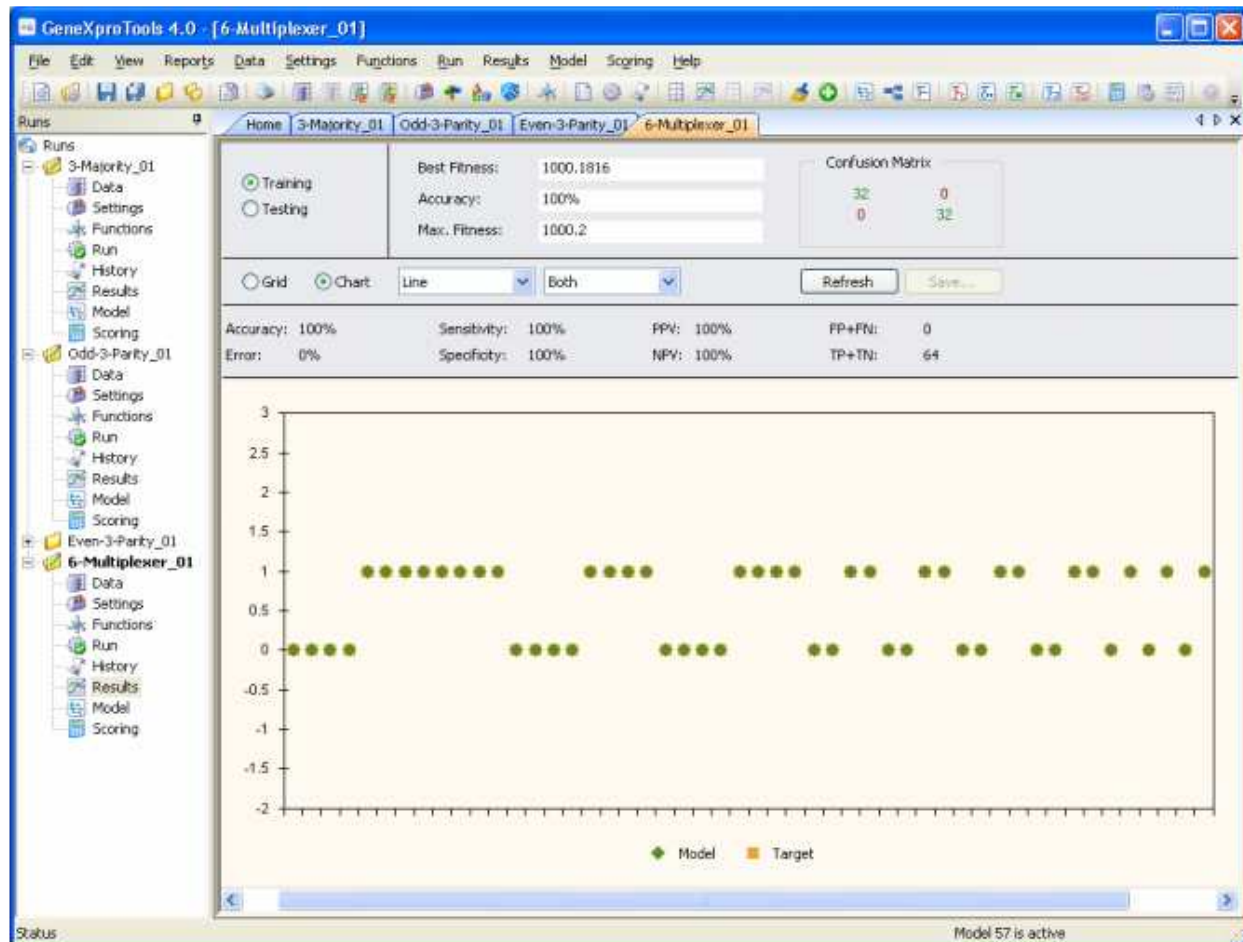
Classification Error

The **classification error** E_i of an individual program i depends on the number of samples incorrectly classified (**false positives** plus **false negatives**) and is evaluated by the formula:

$$E_i = \frac{f}{n} \cdot 100$$

where f is the number of sample cases incorrectly classified, and n is the total number of sample cases.

To evaluate the **classification error** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

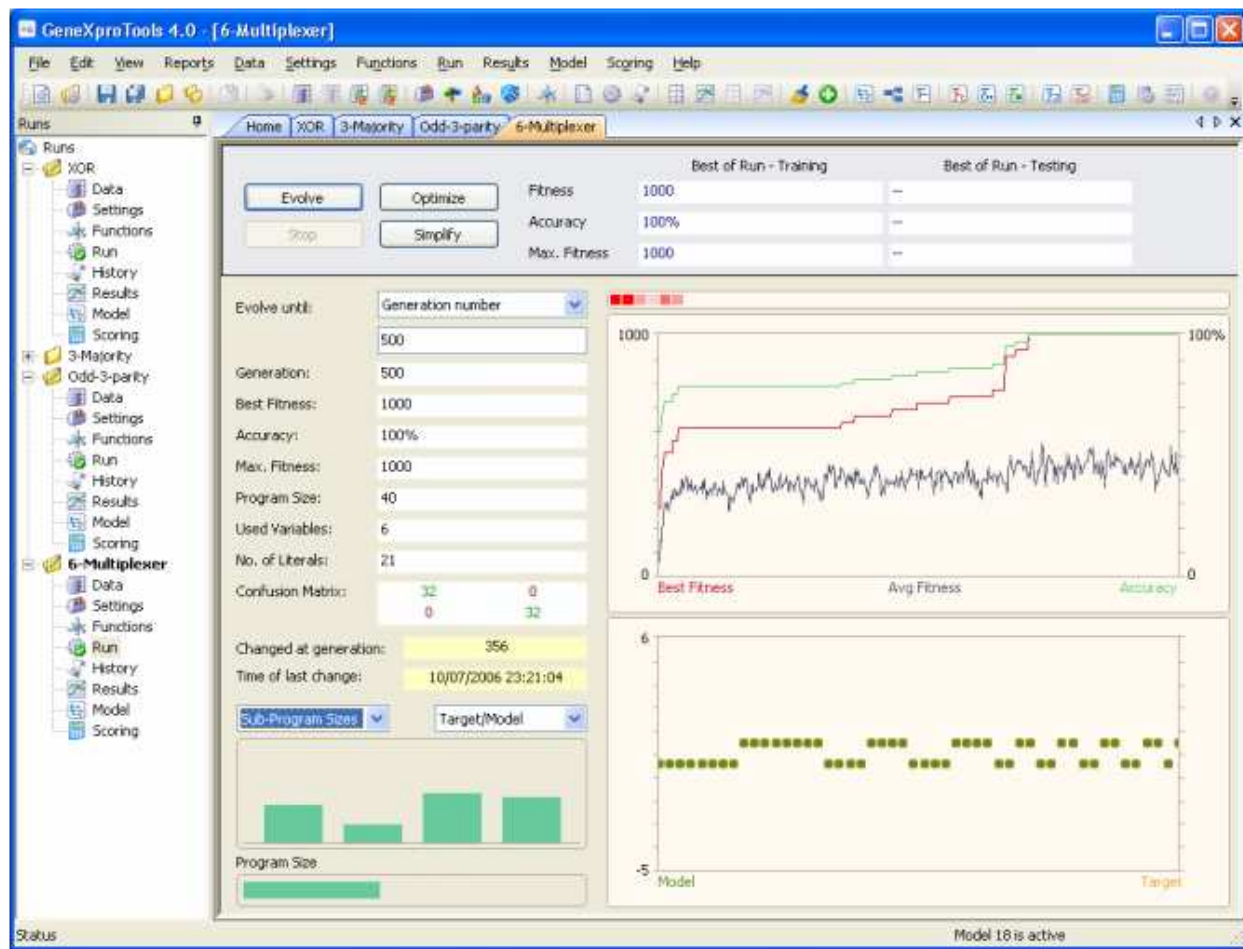
Classification Accuracy

The **classification accuracy** A_i of an individual program i depends on the number of samples correctly classified (**true positives** plus **true negatives**) and is evaluated by the formula:

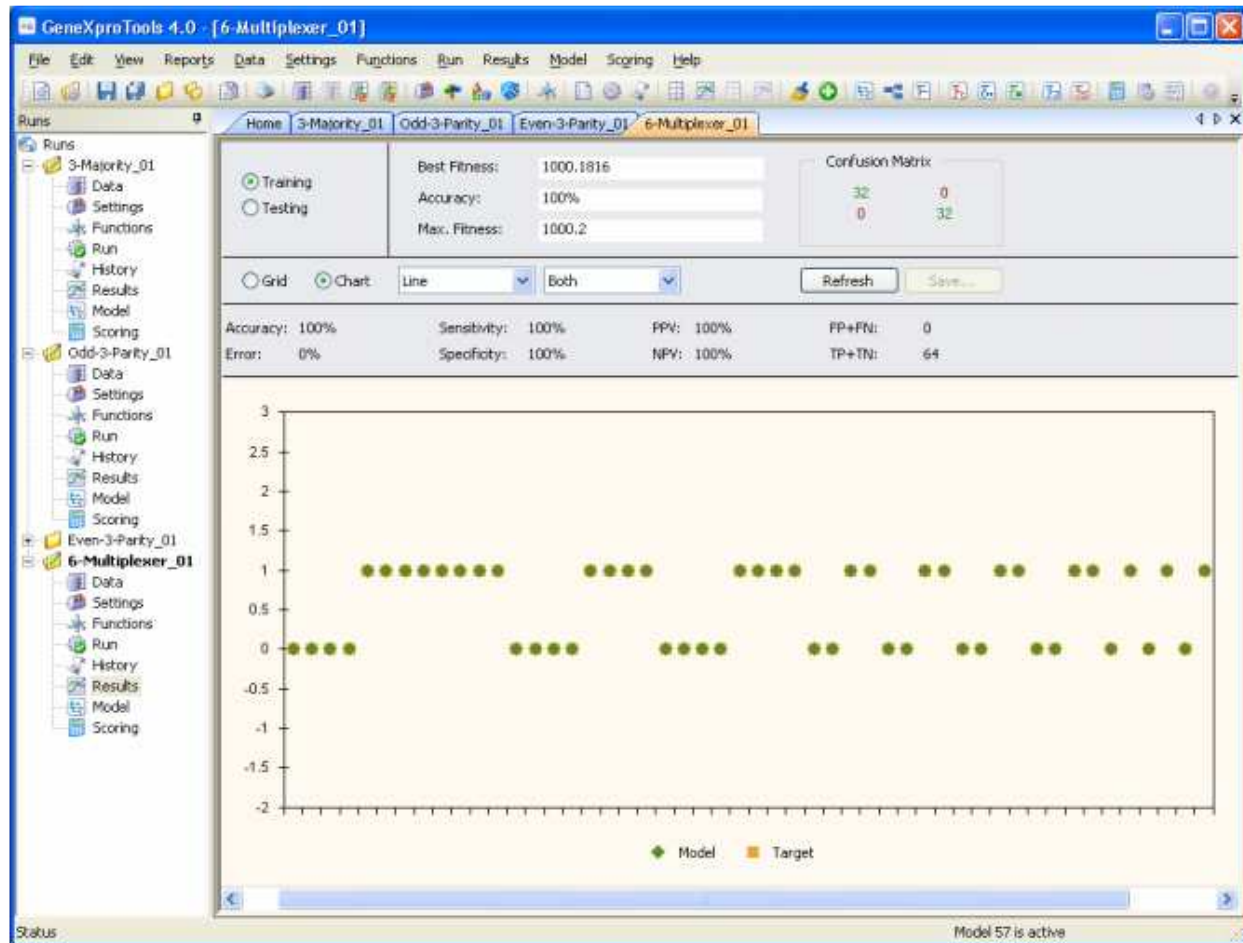
$$A_i = \frac{t}{n} \cdot 100$$

where t is the number of sample cases correctly classified, and n is the total number of sample cases.

During a run, GeneXproTools updates continuously the classification accuracy of the evolving models and plots it in a chart in the [Run Panel](#).



And to evaluate the **classification accuracy** of your model on the testing data, after a run you just have to go to the [Results Panel](#).



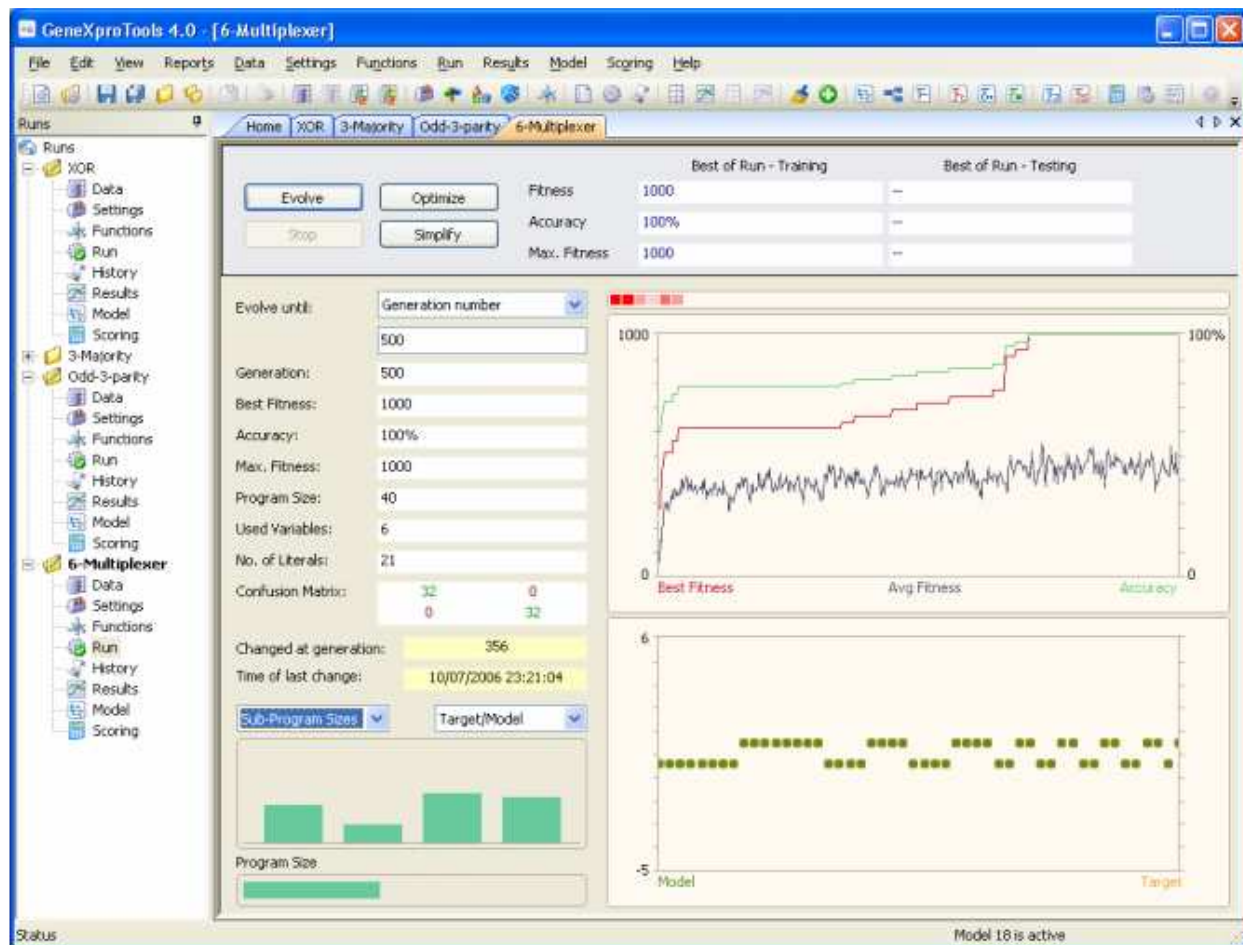
Analyzing GeneXproTools Models Statistically

Confusion Matrix

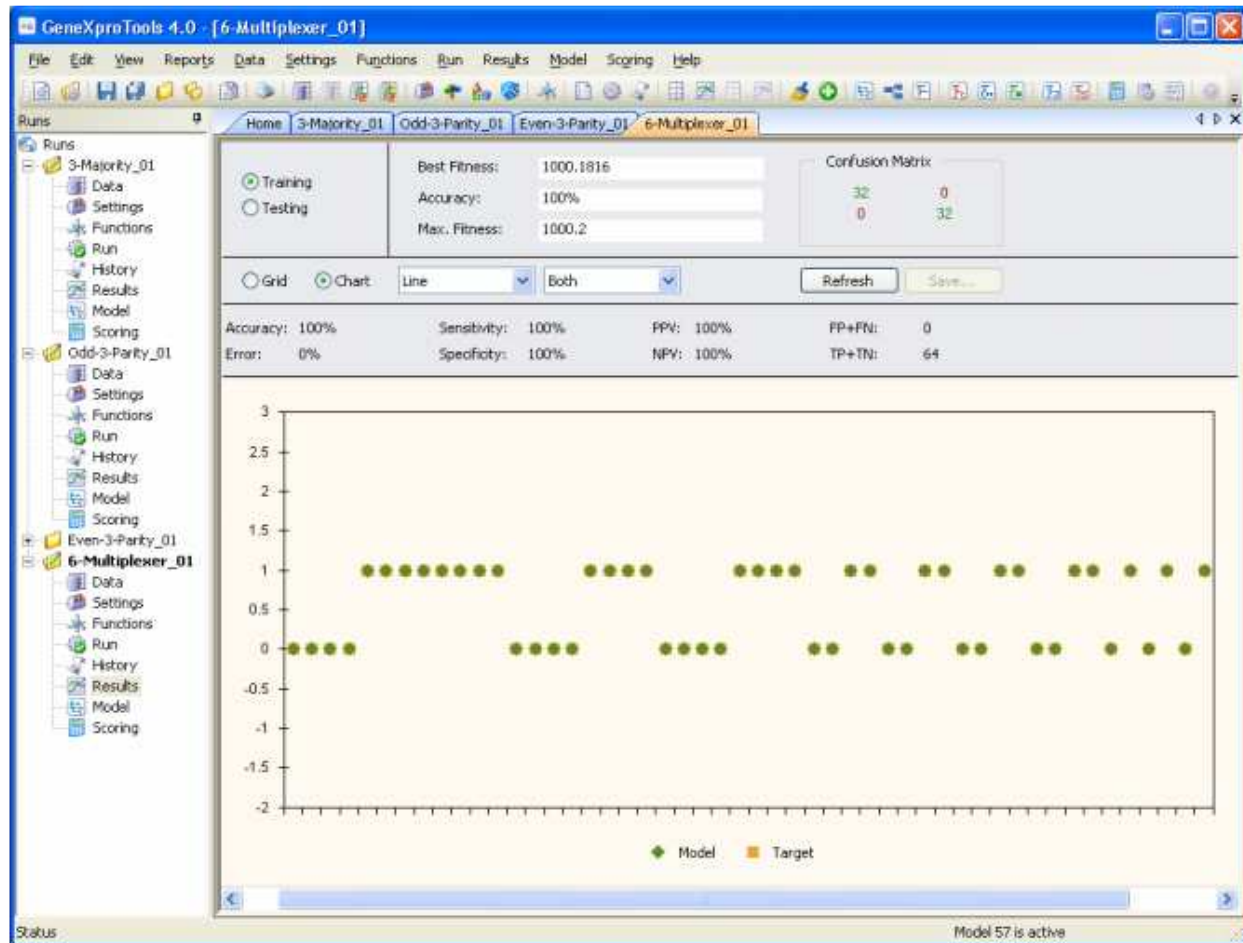
True positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("yes") and "0" ("no"). A **false positive** is when the outcome is incorrectly classified as "yes" (or "positive"), when it is in fact "no" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all Logic Synthesis problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **confusion matrix** of your model on the testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Sensitivity

This indicator comes from the medical domain and consists of the probability of the diagnostic test finding disease among those who have the disease or the proportion of people with disease who have a positive test result.

More formally, the **sensitivity** SE_i of an individual program i is evaluated by the equation:

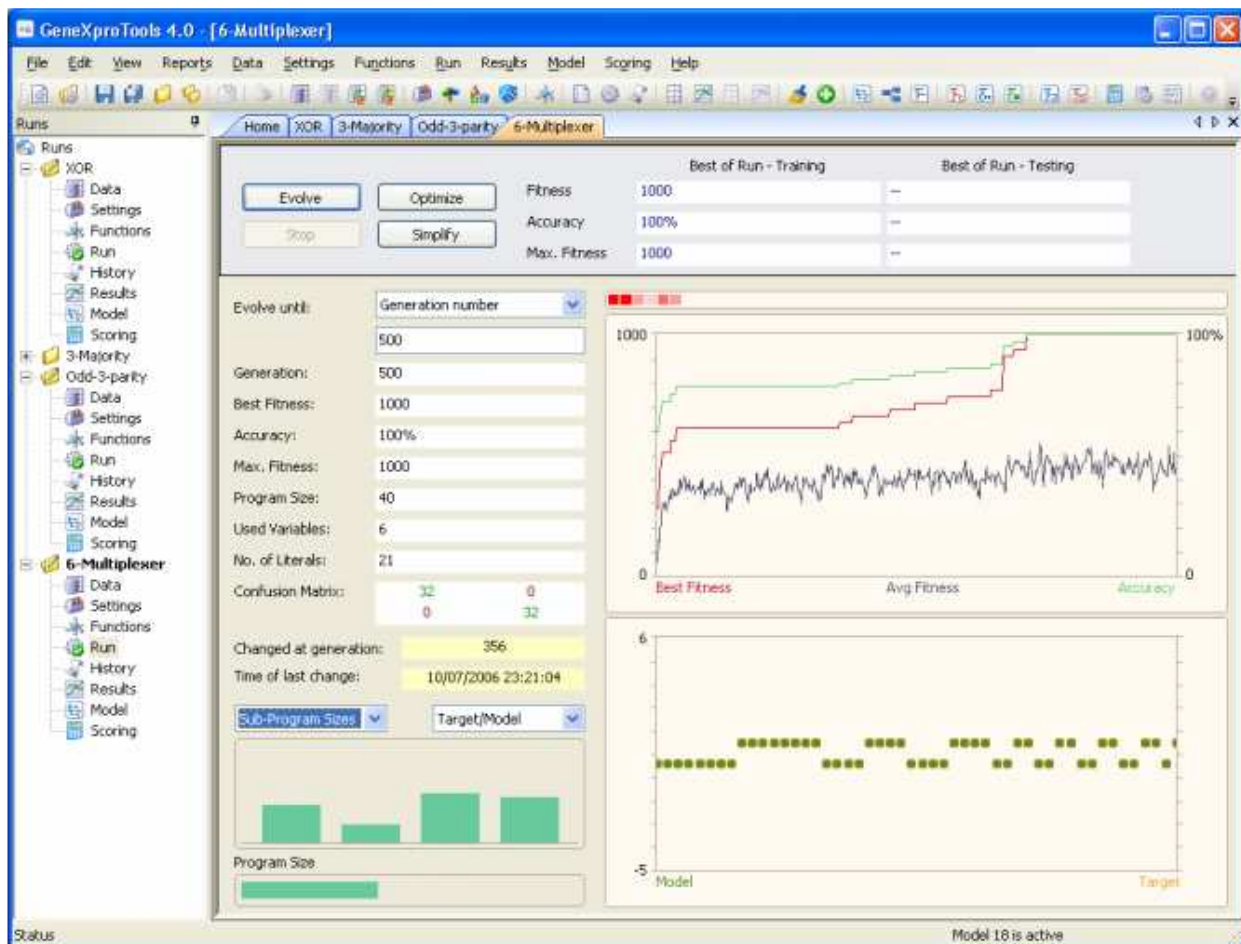
$$SE_i = \frac{TP_i}{TP_i + FN_i}$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

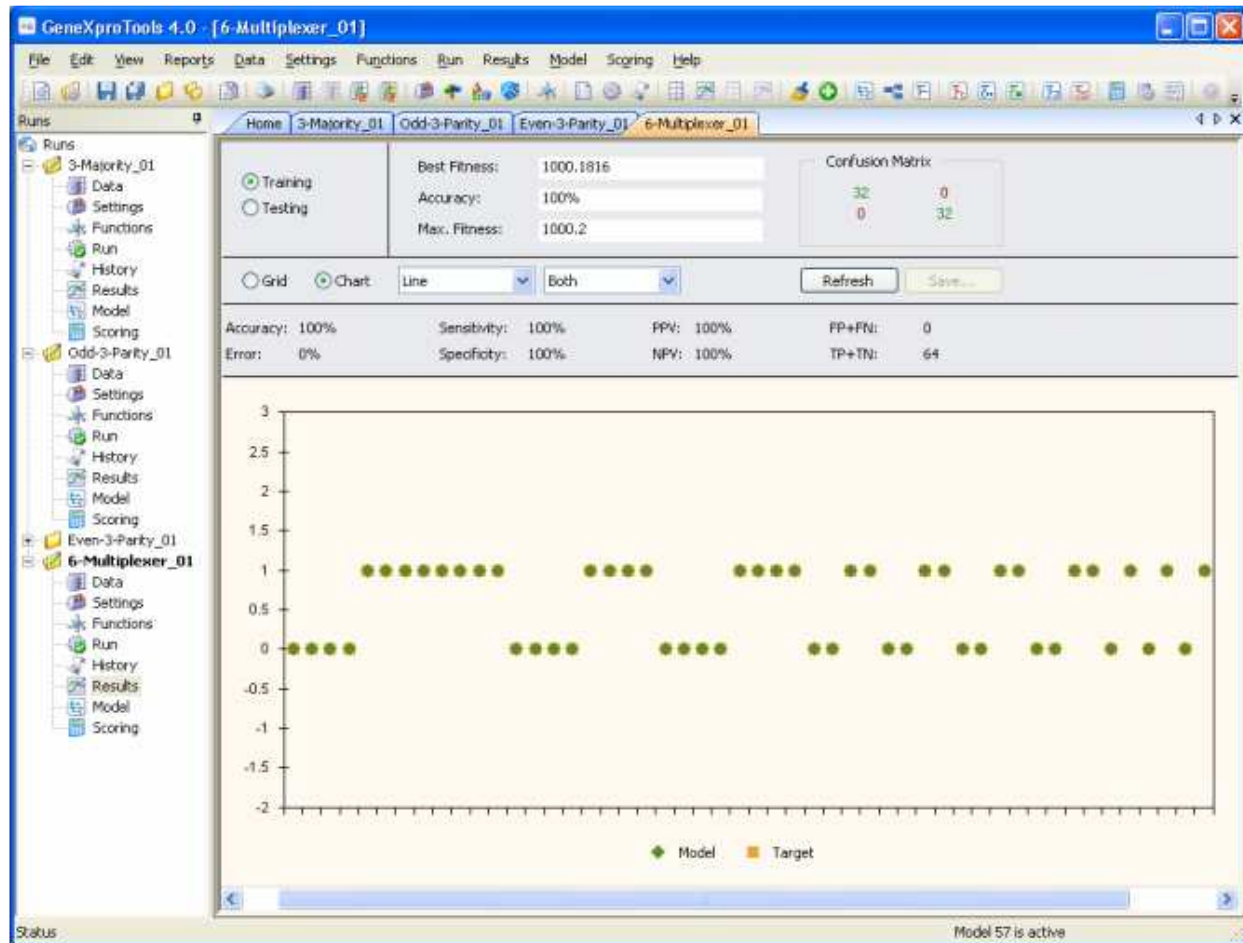
TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("true") and "0" ("false"). A **false positive** is when the outcome is incorrectly classified as "true" (or "positive"), when it is in fact "false" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all Logic Synthesis problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **sensitivity** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Specificity

This indicator comes from the medical domain and consists of the probability of the diagnostic test finding no disease among those who do not have the disease or the proportion of people free of a disease who have a negative test.

More formally, the **specificity** SP_i of an individual program i is evaluated by the equation:

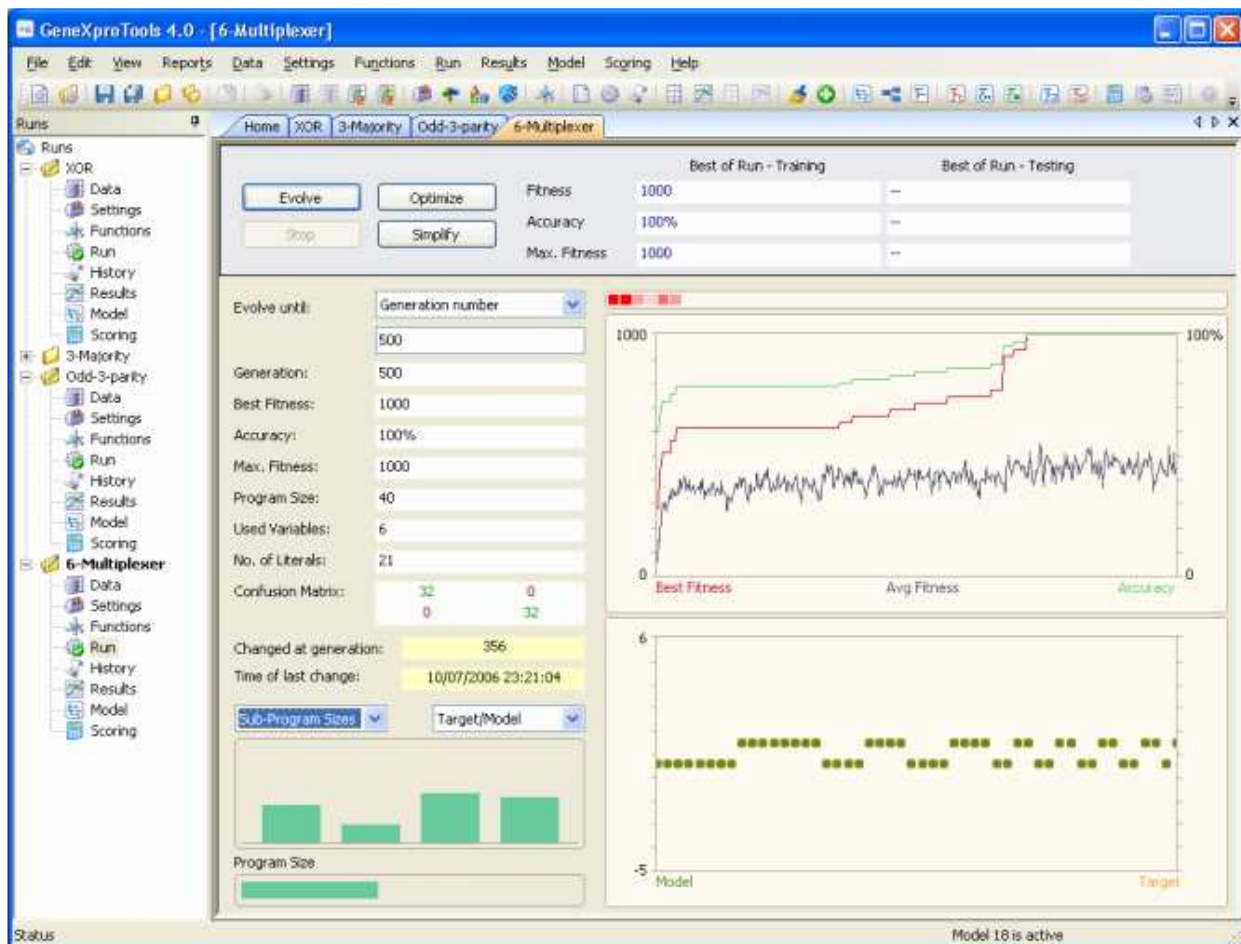
$$SP_i = \frac{TN_i}{TN_i + FP_i}$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

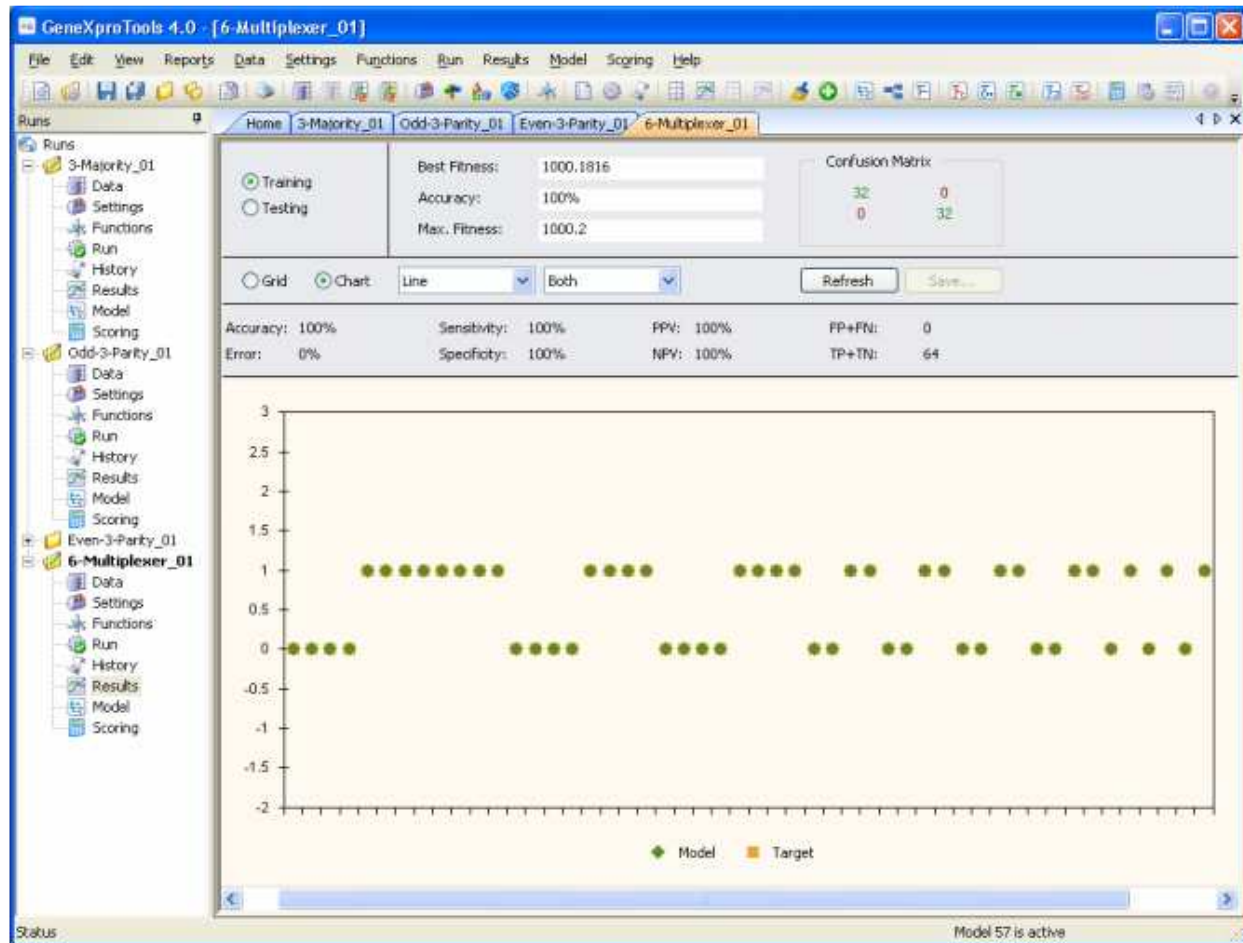
TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("true") and "0" ("false"). A **false positive** is when the outcome is incorrectly classified as "true" (or "positive"), when it is in fact "false" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all Logic Synthesis problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **specificity** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Positive Predictive Value

This indicator comes from the medical domain and consists of the percentage of people with a positive diagnostic test result who actually have the disease.

More formally, the **positive predictive value** PPV_i of an individual program i is evaluated by the equation:

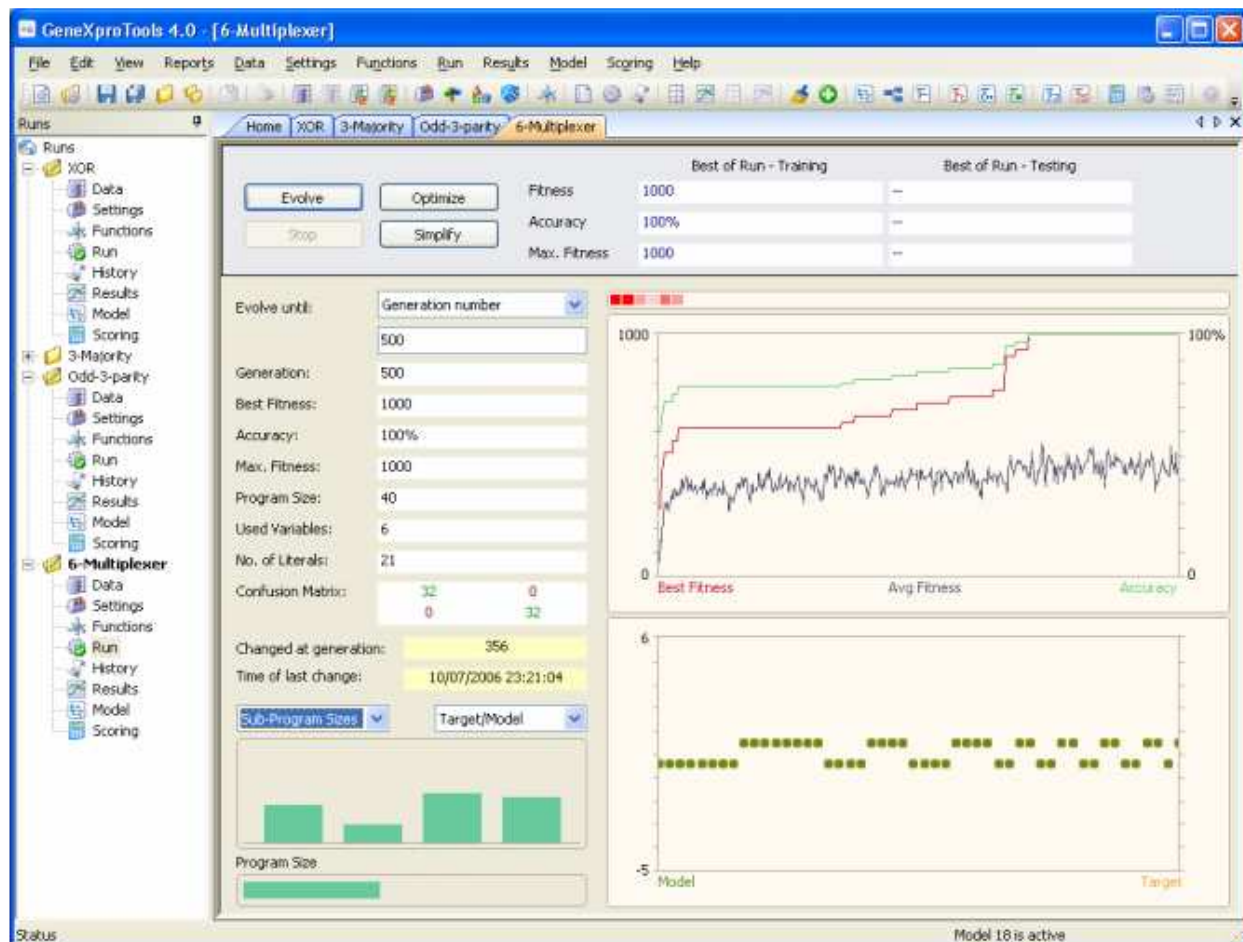
$$PPV_i = \frac{TP_i}{TP_i + FP_i} \text{ where } TP_i + FP_i \neq 0$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

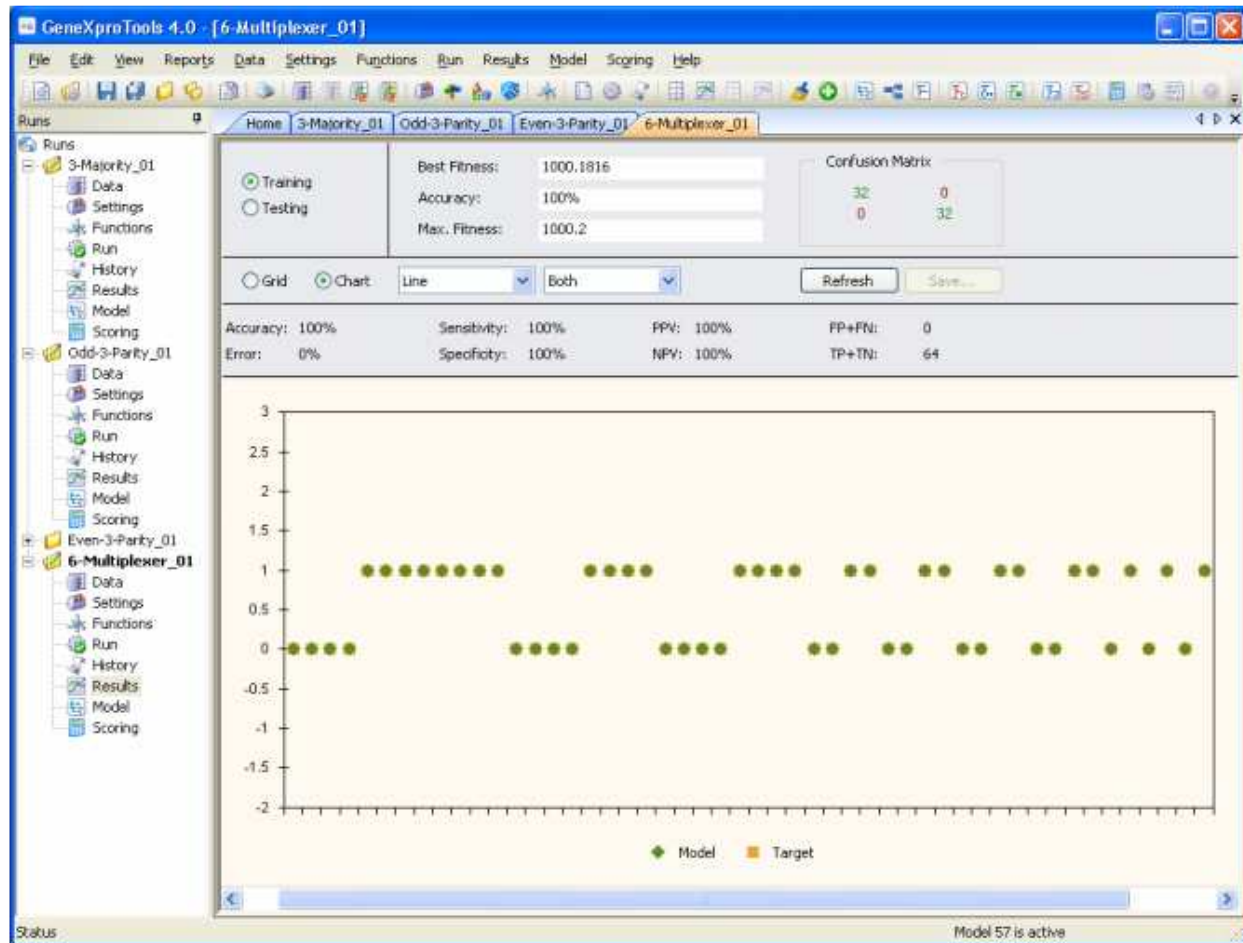
TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("true") and "0" ("false"). A **false positive** is when the outcome is incorrectly classified as "true" (or "positive"), when it is in fact "false" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all Logic Synthesis problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **PPV** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

Negative Predictive Value

This indicator comes from the medical domain and consists of the percentage of people with a negative diagnostic test who do not have the disease.

More formally, the **negative predictive value** NPV_i of an individual program i is evaluated by the equation:

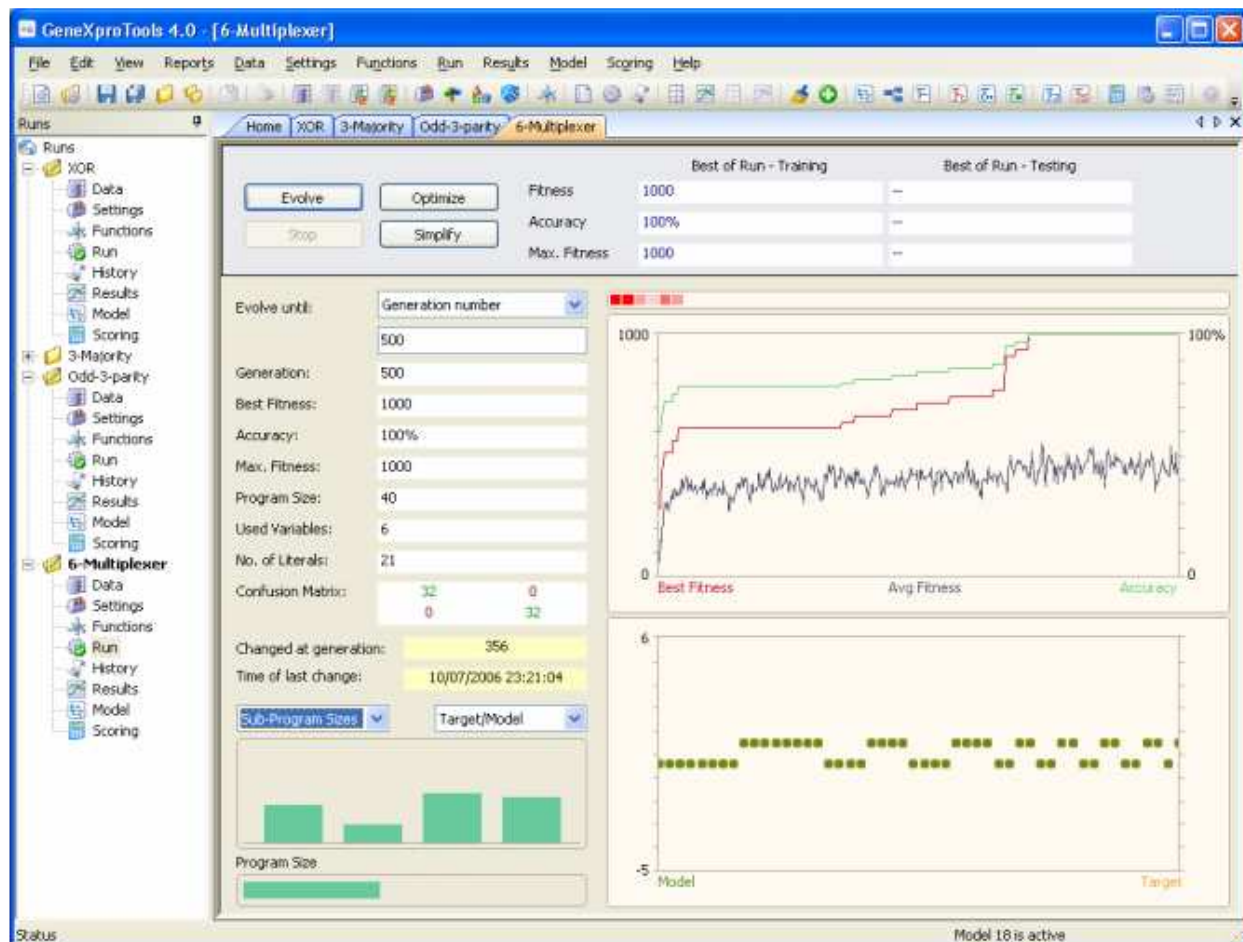
$$NPV_i = \frac{TN_i}{TN_i + FN_i} \text{ where } TN_i + FN_i \neq 0$$

where TP_i , TN_i , FP_i , and FN_i represent, respectively, the number of **true positives**, **true negatives**, **false positives**, and **false negatives**.

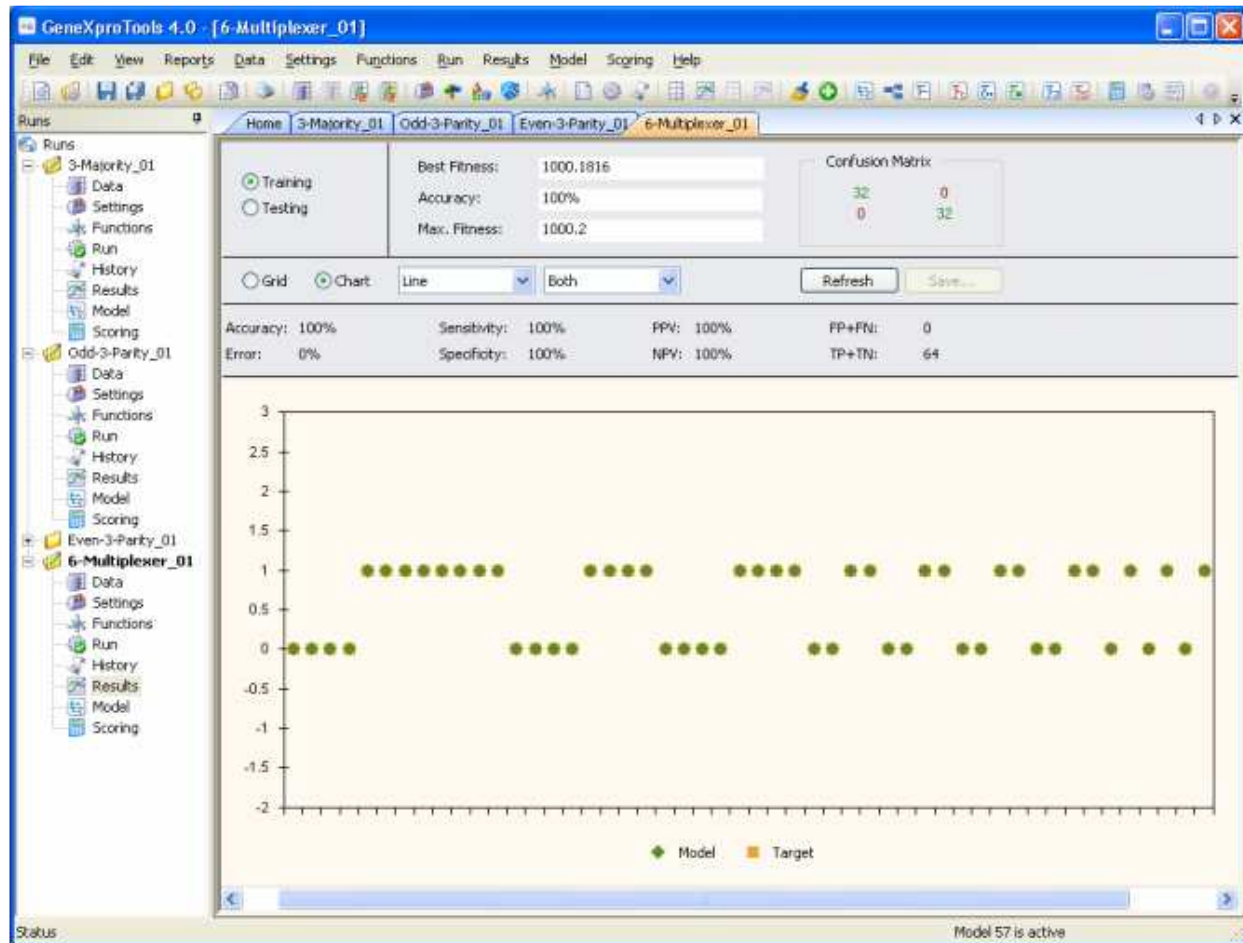
TP_i , TN_i , FP_i , and FN_i are the four different possible outcomes of a single prediction for a two-class case with classes "1" ("true") and "0" ("false"). A **false positive** is when the outcome is incorrectly classified as "true" (or "positive"), when it is in fact "false" (or "negative"). A **false negative** is when the outcome is incorrectly classified as negative when it is in fact positive. **True positives** and **true negatives** are obviously correct classifications.

Keeping track of all these possible outcomes is such an error-prone activity, that they are usually shown in what is called a **confusion matrix**. And for all Logic Synthesis problems, GeneXproTools 4.0 shows the confusion matrix for all the evolved models and updates it continuously during the run.

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN



And to evaluate the **NPV** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run.



Analyzing GeneXproTools Models Statistically

R-square

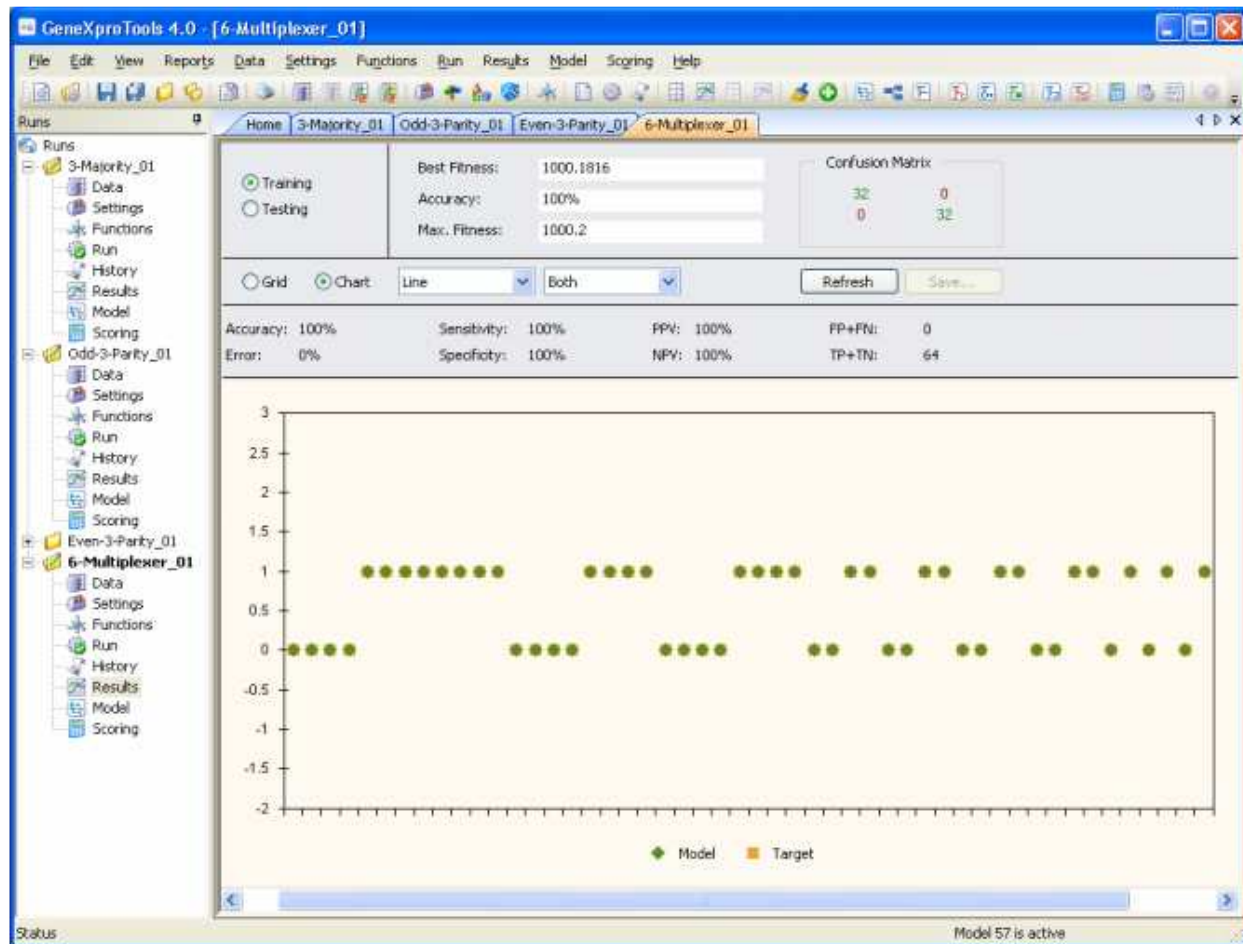
The **R-square** returns the square of the Pearson product moment correlation coefficient, which consists of a dimensionless index that ranges from -1 to 1 and reflects the extent of a linear relationship between two data sets.

The Pearson product moment correlation coefficient R_i of an individual program i is evaluated by the equation:

$$R_i = \frac{n \sum_{j=1}^n (T_j P_{(i)}) - \left(\sum_{j=1}^n T_j \right) \left(\sum_{j=1}^n P_{(i)} \right)}{\sqrt{\left[n \sum_{j=1}^n T_j^2 - \left(\sum_{j=1}^n T_j \right)^2 \right] \left[n \sum_{j=1}^n P_{(i)}^2 - \left(\sum_{j=1}^n P_{(i)} \right)^2 \right]}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

To evaluate the **R-square** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Analyzing GeneXproTools Models Statistically

Correlation Coefficient

The **correlation coefficient** C_i of an individual program i is evaluated by the equation:

$$C_i = \frac{Cov(T, P)}{\sigma_i \cdot \sigma_p}$$

where $Cov(T,P)$ is the **covariance** of the target and model outputs; and σ_t and σ_p are the corresponding **standard deviations**, which are given by:

$$Cov(T,P) = \frac{1}{n} \sum_{j=1}^n (T_j - \bar{T})(P_{(ij)} - \bar{P})$$

$$\sigma_t = \sqrt{\frac{\sum_{j=1}^n (T_j - \bar{T})^2}{n}}$$

$$\sigma_p = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - \bar{P})^2}{n}}$$

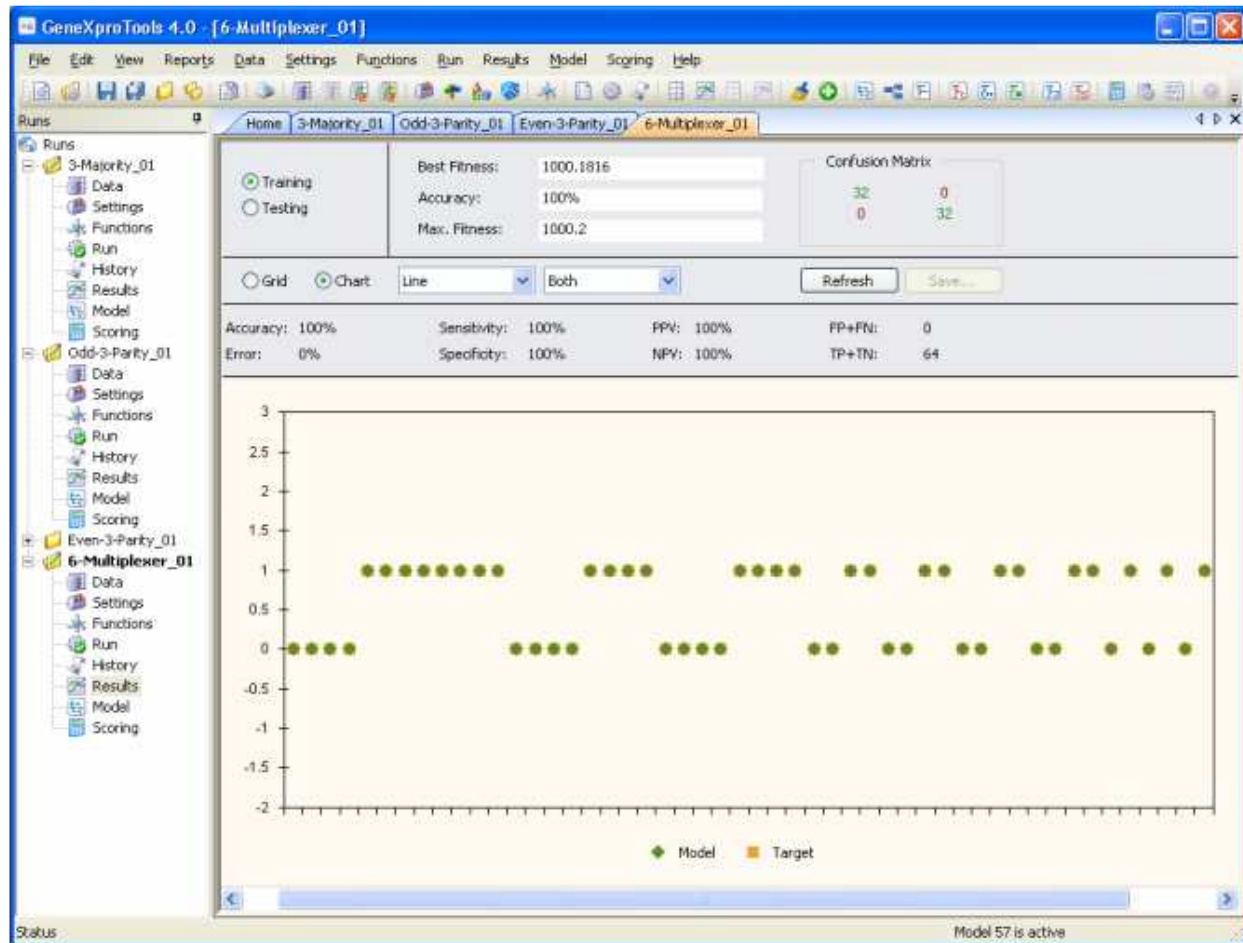
where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ and $\bar{P}$ are given by the formulas:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

$$\bar{P} = \frac{1}{n} \sum_{j=1}^n P_{(ij)}$$

The correlation coefficient is confined to the range $[-1, 1]$. When $C_i = 1$, there is a perfect positive linear correlation between T and P , that is, they vary by the same amount. When $C_i = -1$, there is a perfect negative linear correlation between T and P , that is, they vary in opposite ways (when T increases, P decreases by the same amount). When $C_i = 0$, there is no correlation between T and P . Intermediate values describe partial correlations and the closer to 1 or -1 the better the model.

To evaluate the **correlation coefficient** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Analyzing GeneXproTools Models Statistically

Mean Squared Error

The **mean squared error** E_i of an individual program i is evaluated by the equation:

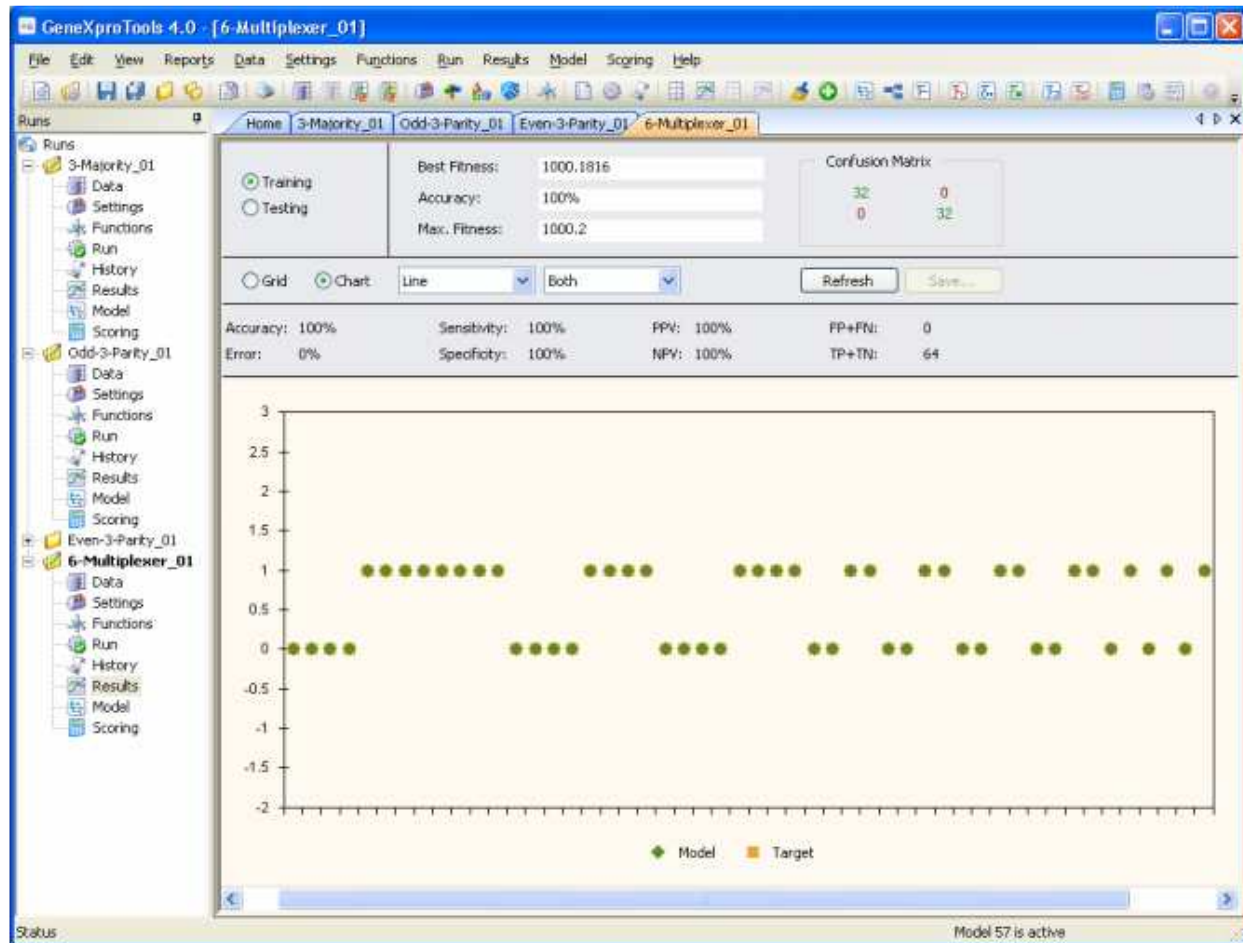
$$E_i = \frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **MSE** of your model both on the training and testing data, you just have to go to the

[Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



◀ | ▶

Analyzing GeneXproTools Models Statistically

Root Mean Squared Error

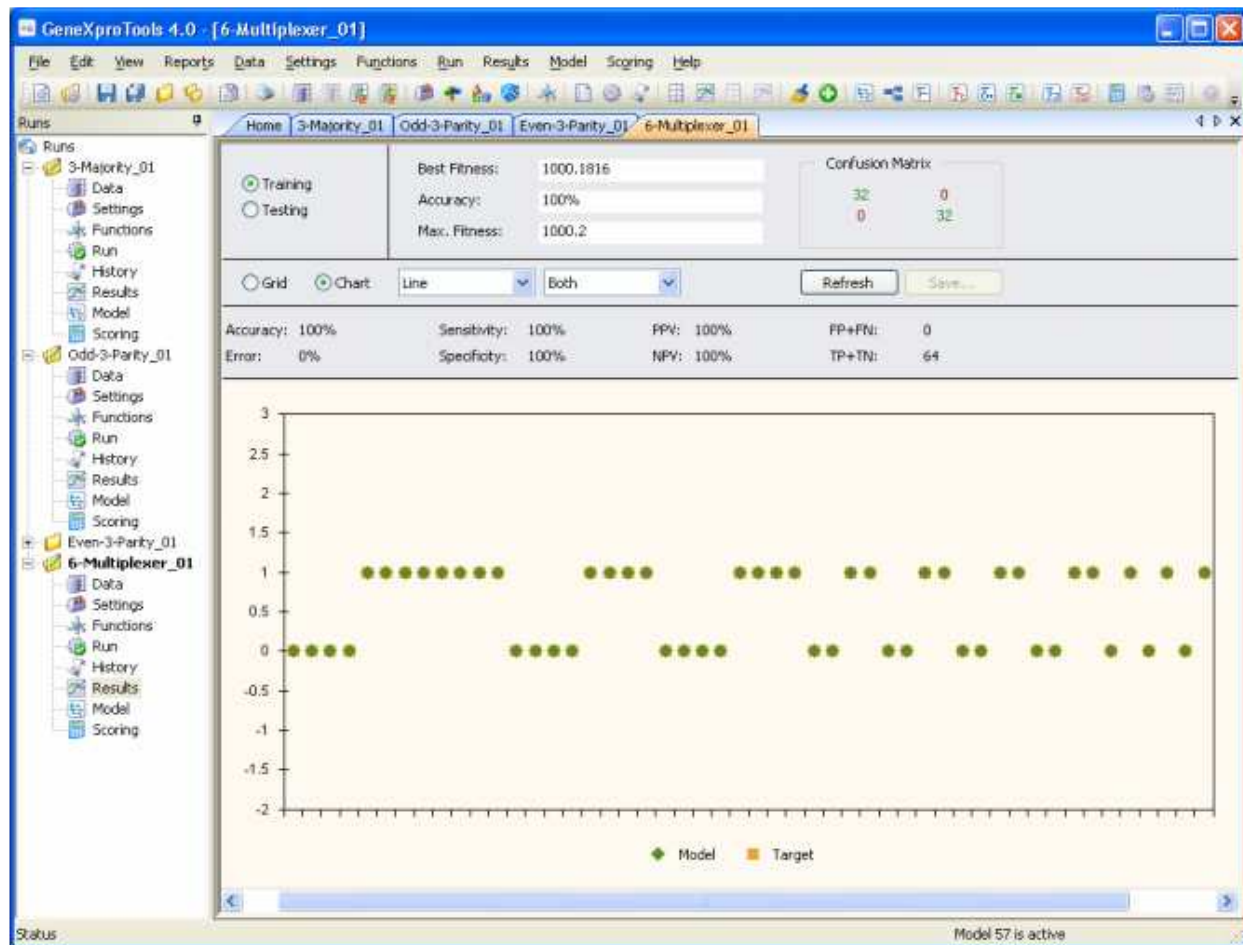
The **root mean squared error** E_i of an individual program i is evaluated by the equation:

$$E_i = \sqrt{\frac{1}{n} \sum_{j=1}^n (P_{(ij)} - T_j)^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RMSE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Analyzing GeneXproTools Models Statistically

Mean Absolute Error

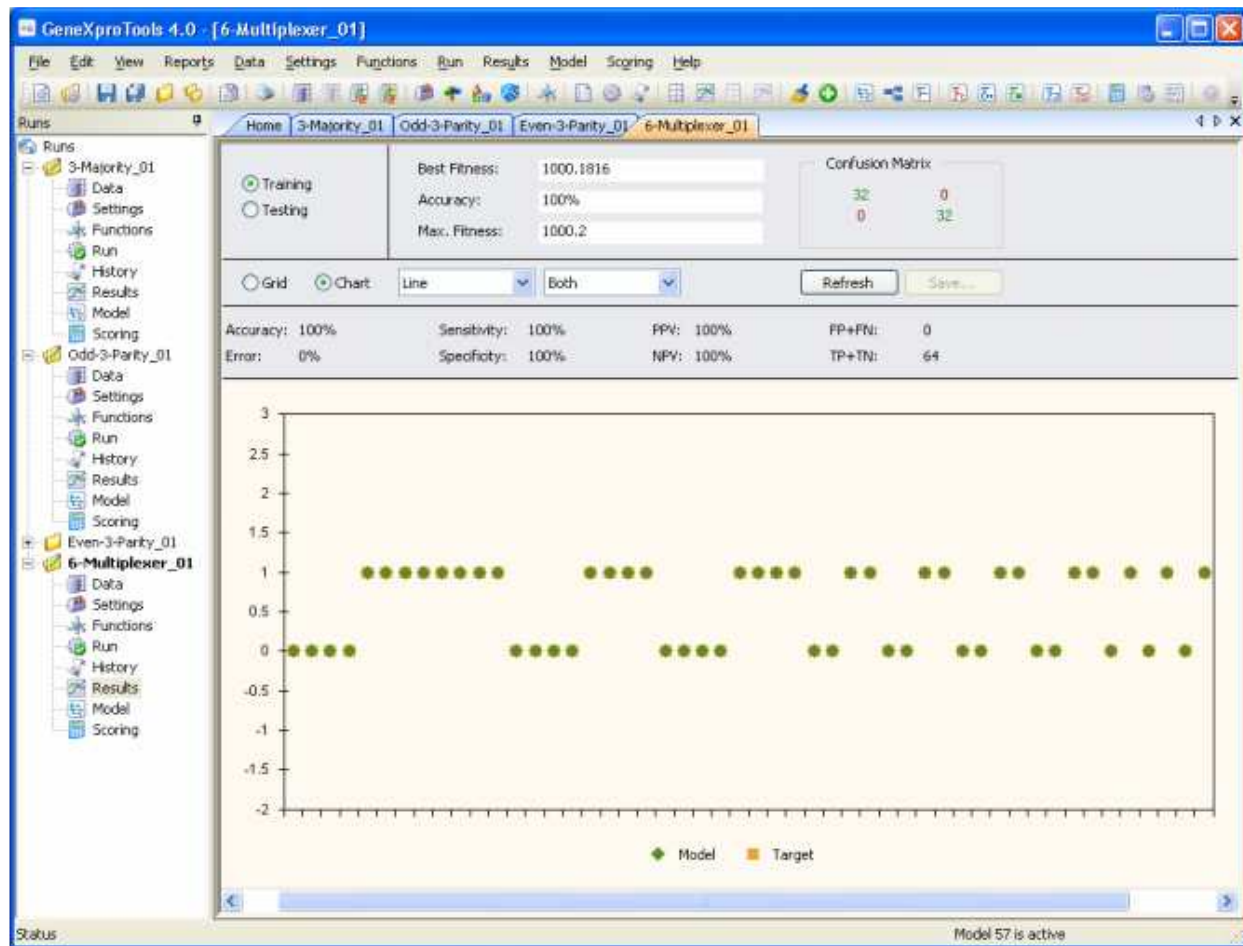
The **mean absolute error** E_i of an individual program i is evaluated by the equation:

$$E_i = \frac{1}{n} \sum_{j=1}^n |P_{(ij)} - T_j|$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); and T_j is the target value for sample case j .

For a perfect fit, $P_{(ij)} = T_j$ and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **MAE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Analyzing GeneXproTools Models Statistically

Relative Squared Error

The **relative squared error** is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor.

Mathematically, the **relative squared error** E_i of an individual program i is evaluated by the equation:

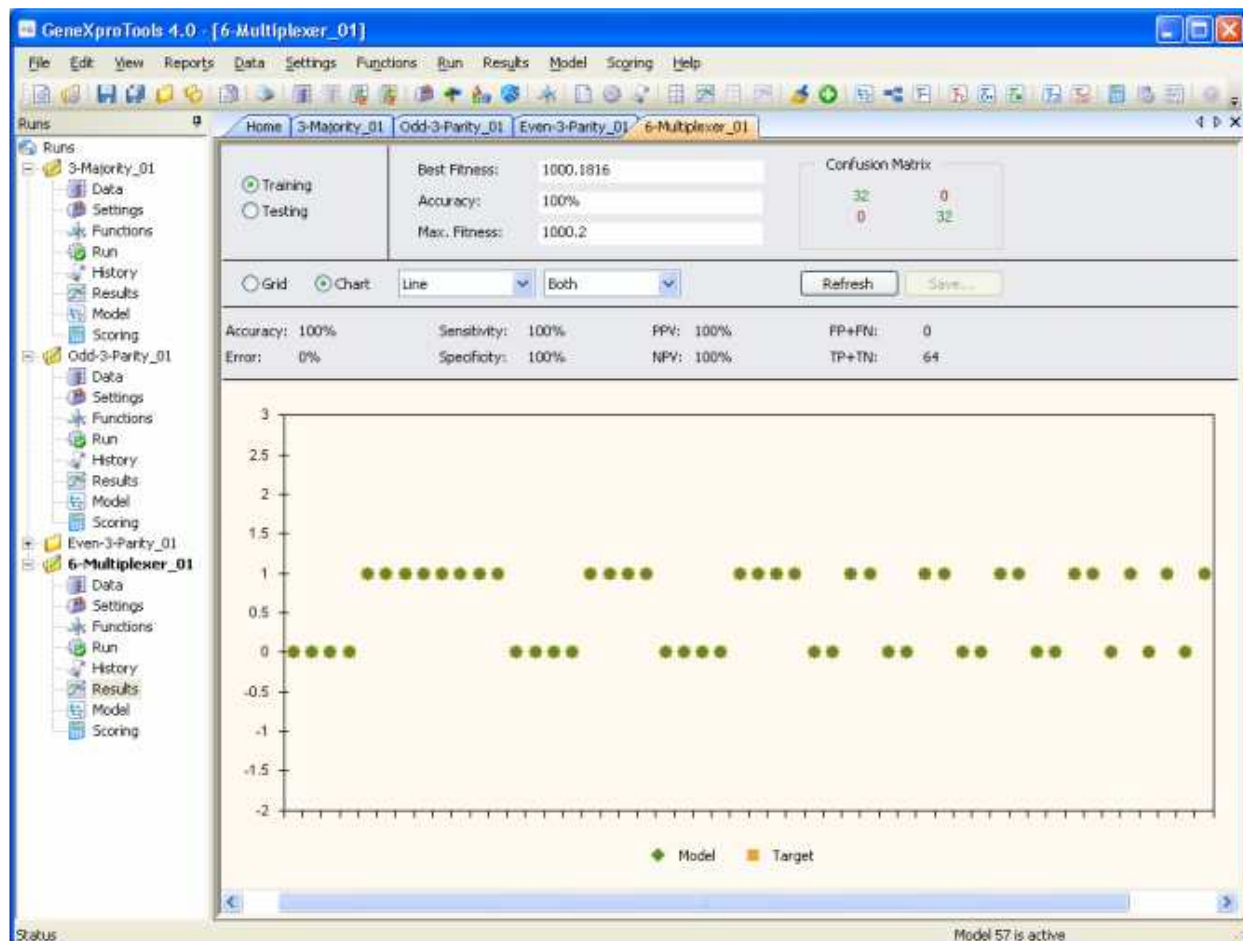
$$E_i = \frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RSE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).





Analyzing GeneXproTools Models Statistically

Root Relative Squared Error

The **root relative squared error** is relative to what it would have been if a simple predictor had been used. More specifically, this simple predictor is just the average of the actual values. Thus, the relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the simple predictor. By taking the square root of the [relative squared error](#) one reduces the error to the same dimensions as the quantity being predicted.

Mathematically, the **root relative squared error** E_i of an individual program i is evaluated by the equation:

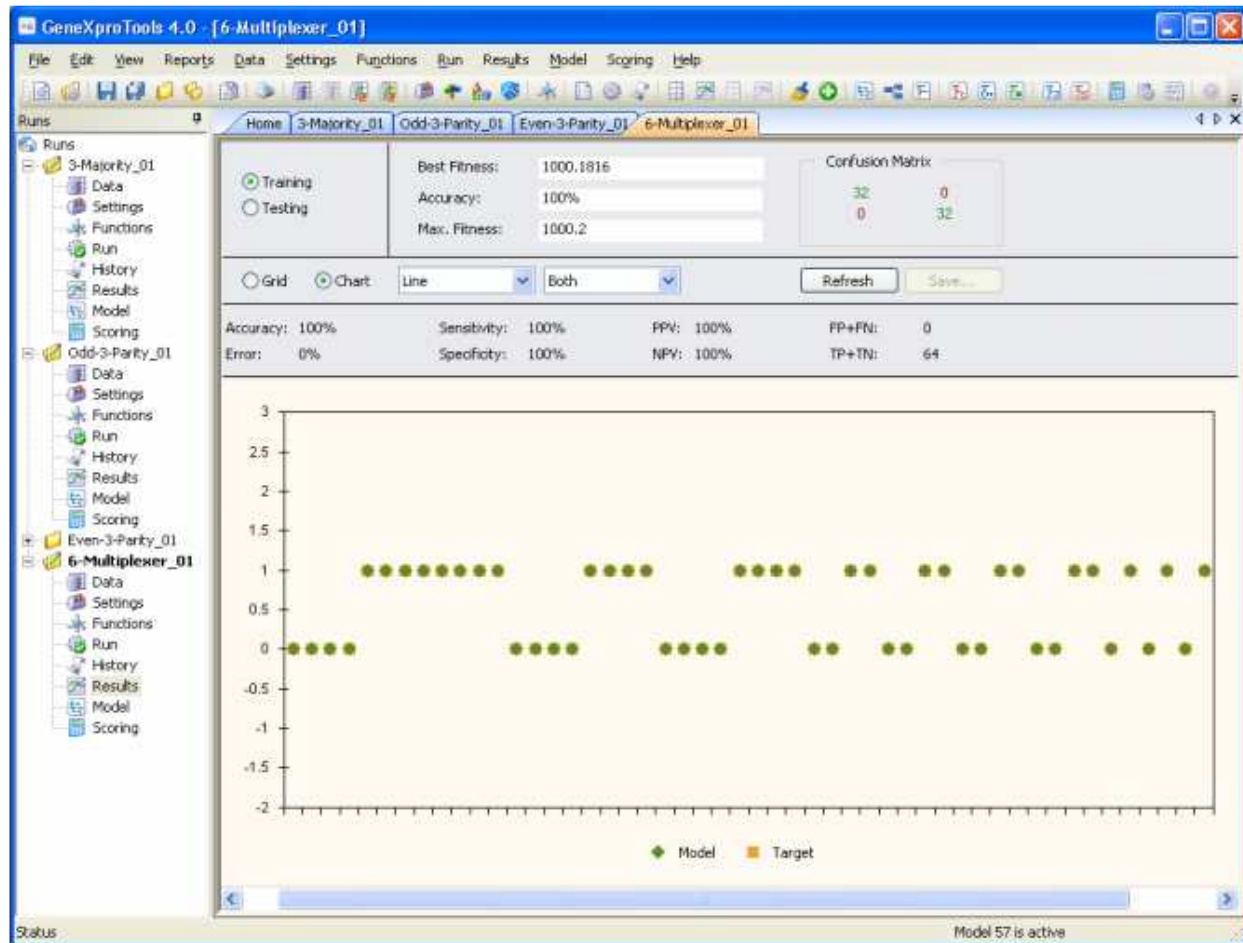
$$E_i = \sqrt{\frac{\sum_{j=1}^n (P_{(ij)} - T_j)^2}{\sum_{j=1}^n (T_j - \bar{T})^2}}$$

where $P_{(ij)}$ is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

To evaluate the **RRSE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Analyzing GeneXproTools Models Statistically

Relative Absolute Error

The **relative absolute error** is very similar to the [relative squared error](#) in the sense that it is also relative to a simple predictor, which is just the average of the actual values. In this case, though, the error is just the total absolute error instead of the total squared error. Thus, the relative absolute error takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.

Mathematically, the **relative absolute error** E_i of an individual program i is evaluated by the equation:

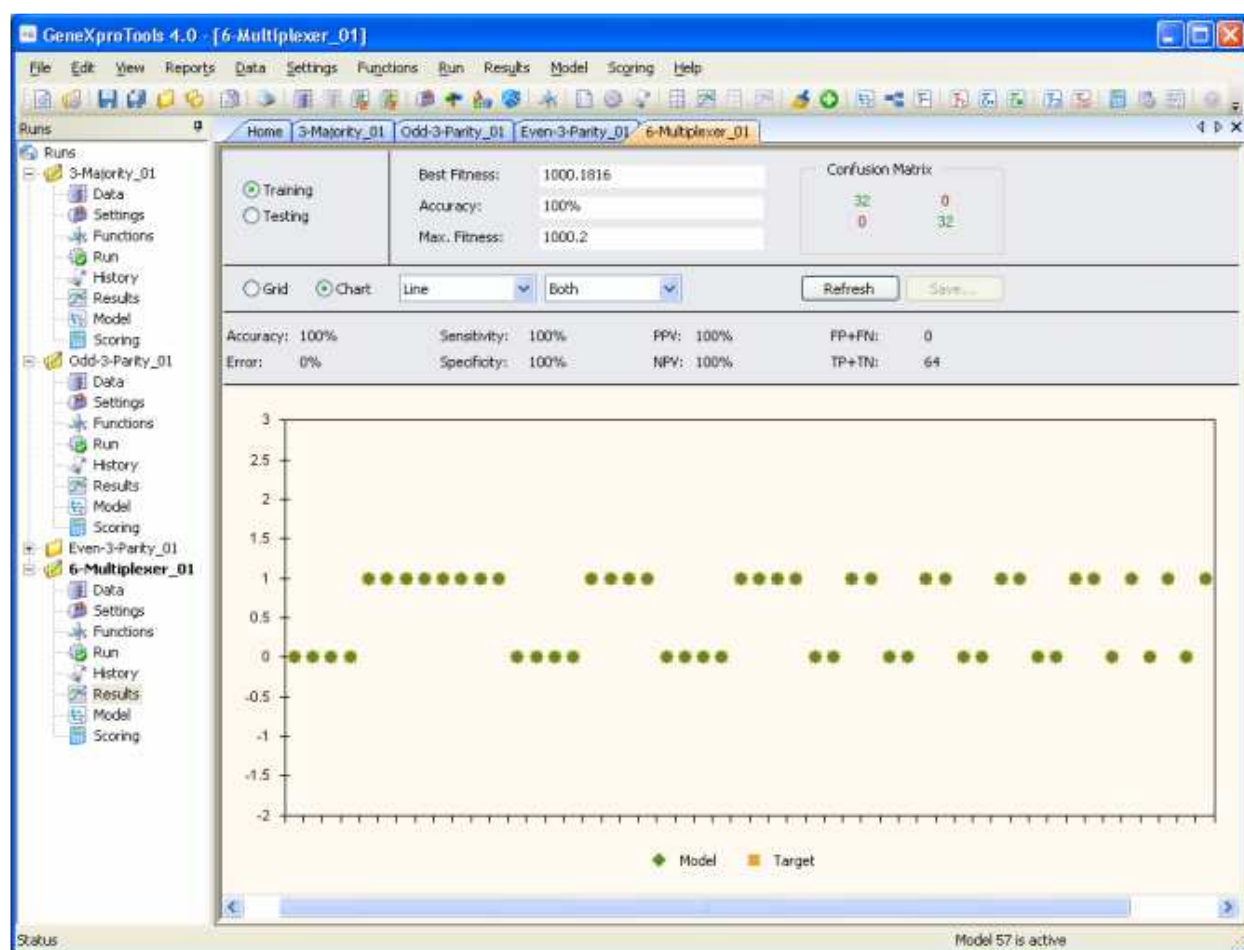
$$E_i = \frac{\sum_{j=1}^n |P_{ij} - T_j|}{\sum_{j=1}^n |T_j - \bar{T}|}$$

where P_{ij} is the value predicted by the individual program i for sample case j (out of n sample cases); T_j is the target value for sample case j ; and $\bar{T}$ is given by the formula:

$$\bar{T} = \frac{1}{n} \sum_{j=1}^n T_j$$

For a perfect fit, the numerator is equal to 0 and $E_i = 0$. So, the E_i index ranges from 0 to infinity, with 0 corresponding to the ideal.

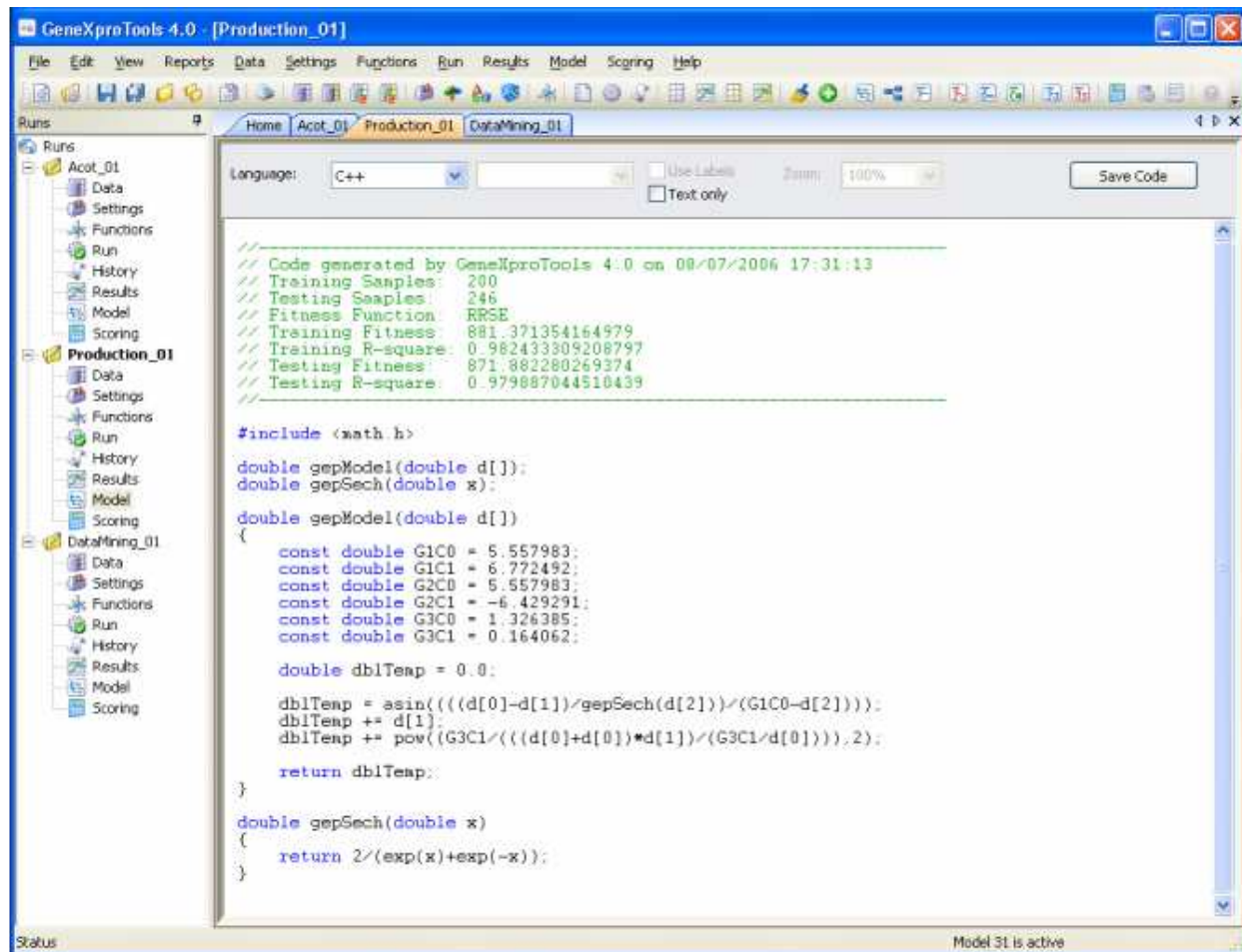
To evaluate the **RAE** of your model both on the training and testing data, you just have to go to the [Results Panel](#) after a run and, although it is not shown there, it is also evaluated there and kept for your future reference in the [Report Panel](#).



Generating Code Automatically

In this chapter you'll learn about:

- The endogenous code of GeneXproTools 4.0.
- Expression trees and the built-in parser of GeneXproTools 4.0.
- The built-in grammars of GeneXproTools 4.0.
- The built-in logical systems of GeneXproTools 4.0 for automatic code generation.
- User defined grammars.

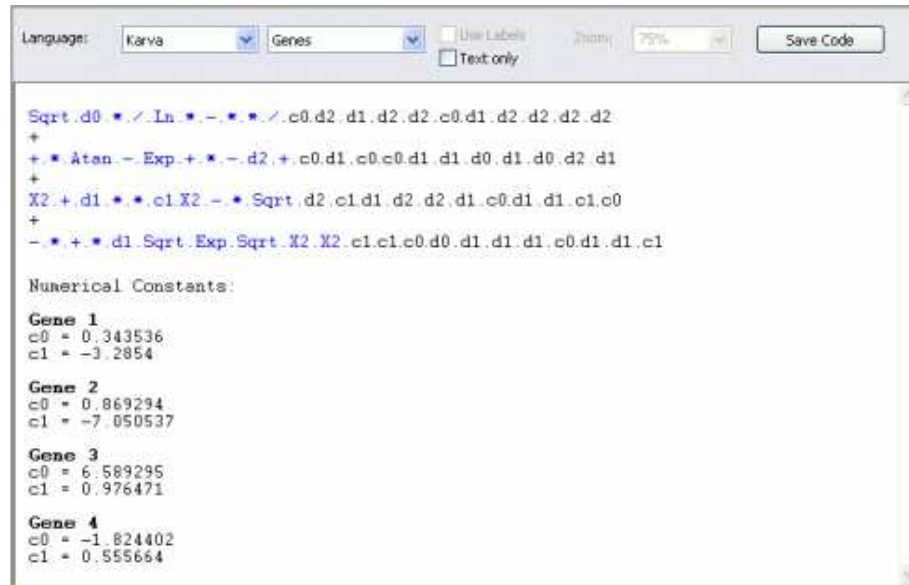


Generating Code Automatically

Endogenous Karva Code

Karva code is the native language of [Gene Expression Programming](#) (GEP), the algorithm used by GeneXproTools 4.0 to create complex and accurate models.

To see your model in **Karva notation**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Karva.



The screenshot shows the GeneXproTools interface with the Language dropdown set to 'Karva' and the Genes dropdown set to 'Genes'. The main text area displays a complex Karva expression and numerical constants for four genes.

```

Sqrt.d0 * / In * - * * / .c0.d2 d1.d2.d2.c0.d1.d2.d2.d2.d2
+
+ * Atan - Exp + * - .d2 + .c0.d1.c0.c0.d1 d1.d0.d1.d0.d2.d1
+
X2 + .d1 * * .c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * .d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

```

And by choosing K-expressions in the Language list box, GeneXproTools shows you only the elements that are part of the model. In both cases, either Genes or K-expressions, the head elements are shown in blue.



The screenshot shows the GeneXproTools interface with the Language dropdown set to 'Karva' and the K-expressions dropdown set to 'K-expressions'. The main text area displays the same Karva expression and numerical constants as the previous screenshot, but with the head elements highlighted in blue.

```

Sqrt.d0
+
+ * Atan - Exp + * - .d2 + .c0.d1.c0.c0.d1 d1.d0
+
X2 + .d1 * * .c1.X2 - * Sqrt.d2.c1.d1.d2
+
- * + * .d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

```

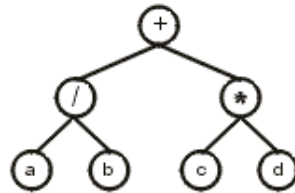
Karva notation was developed specifically for [Gene Expression Programming](#) and consists of a universal way of representing any mathematical or logical expression that can be represented as a tree. Here we are going to present it briefly, but you can learn more about it in the chapter [The Architectures of GeneXproTools Learning Algorithms](#).

The chromosomes of Gene Expression Programming code for one or more [K-expressions](#) (from Karva

notation), and each K-expression is encoded in a **gene**. Genes and K-expressions are very easy to interpret. For example, the gene:

```
0123456
+/*abcd
```

can be represented as a diagram or **expression tree** (ET):



The **translation** of a gene into its corresponding ET is straightforward: The first element in the gene (position 0) corresponds to the root of the ET; then, below that node, are attached as many nodes as there are arguments to that function (two, in this case); then these nodes are filled consecutively with the elements in the gene (in this case, positions 1 and 2), and so forth. The process is repeated until a line composed of only terminals is formed (in this case, the third line).

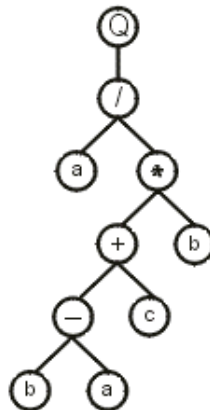
More formally, the gene and ET above can be represented by the expression:

$$\frac{a}{b} + cd$$

Usually the genes evolved by GEP are more complex than the gene presented above, with **noncoding regions** at the end. Consider the following gene (the **head** is shown in blue):

```
01234567890123456
Q/a*+b-cbabaccbac
```

where "Q" represents the square root function. This gene has a **head** of length 8 and a **tail** of length 9. Note that the head contains both functions and terminals, whereas the tail contains exclusively terminals. The translation of such genes is done exactly as in the previous example, obtaining:



Note that, in this case, not all the elements in the gene were used to construct the ET, as the

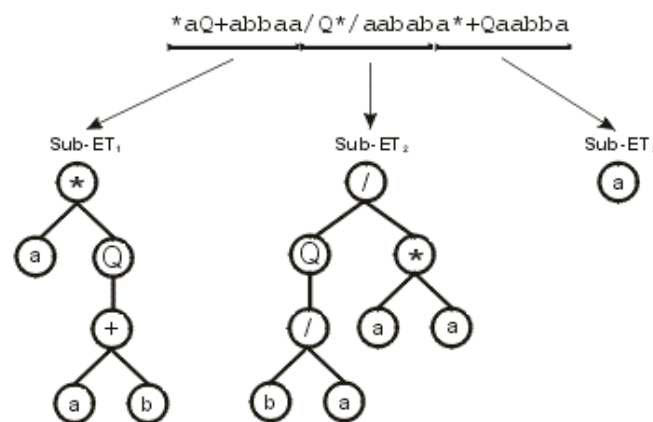
translation ends whenever a line containing only terminals is formed. In this case, the gene ends at position 16 whereas the K-expression ends at position 9.

Furthermore, GEP chromosomes are usually **multigenic**, and each gene codes for a sub-ET or sub-program. After translation, the sub-ETs are linked by a particular **linking function**: addition, subtraction, multiplication, or division for all kinds of mathematical models and And, Or, Nand, Nor, Xor, Nxor, Less Than, Greater Than, Less Or Equal, or Greater Or Equal for logical expressions.

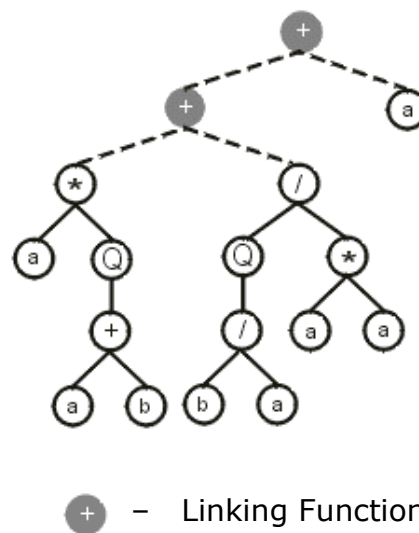
For example, the following chromosome composed of three genes (position 0 indicates the beginning of each gene):

```
012345678012345678012345678
*aQ+abbaa/Q*/aababa*+Qaabba
```

encodes the following sub-ETs:



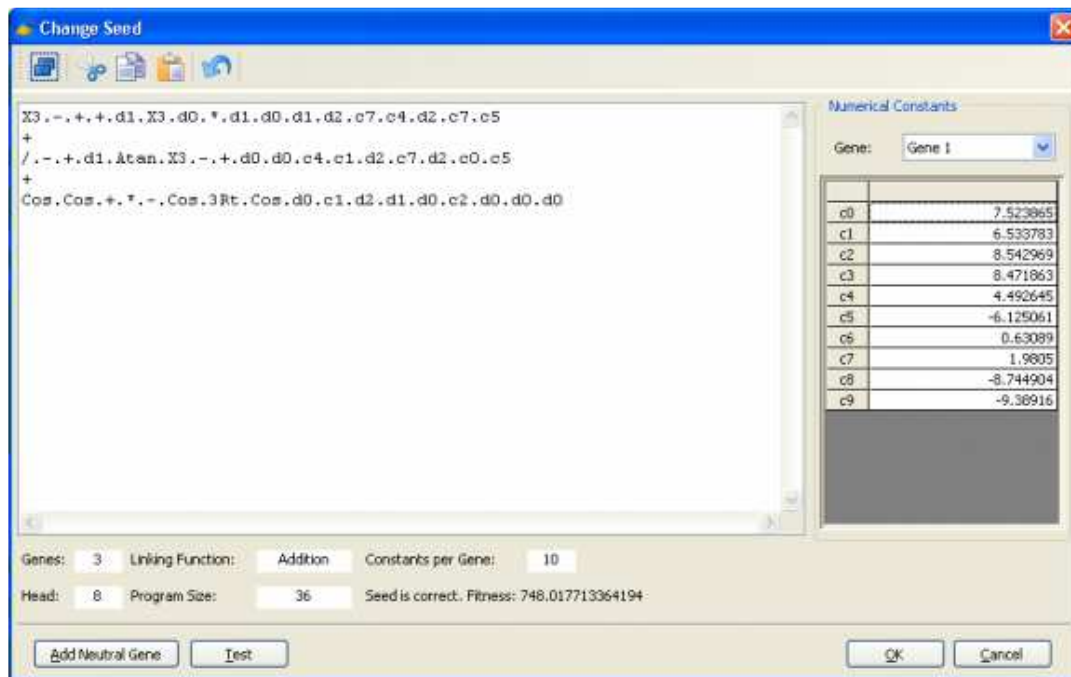
Then the sub-ETs are afterwards linked by one of the available linking functions. For instance, if the linking function were addition, then the following program would be obtained:



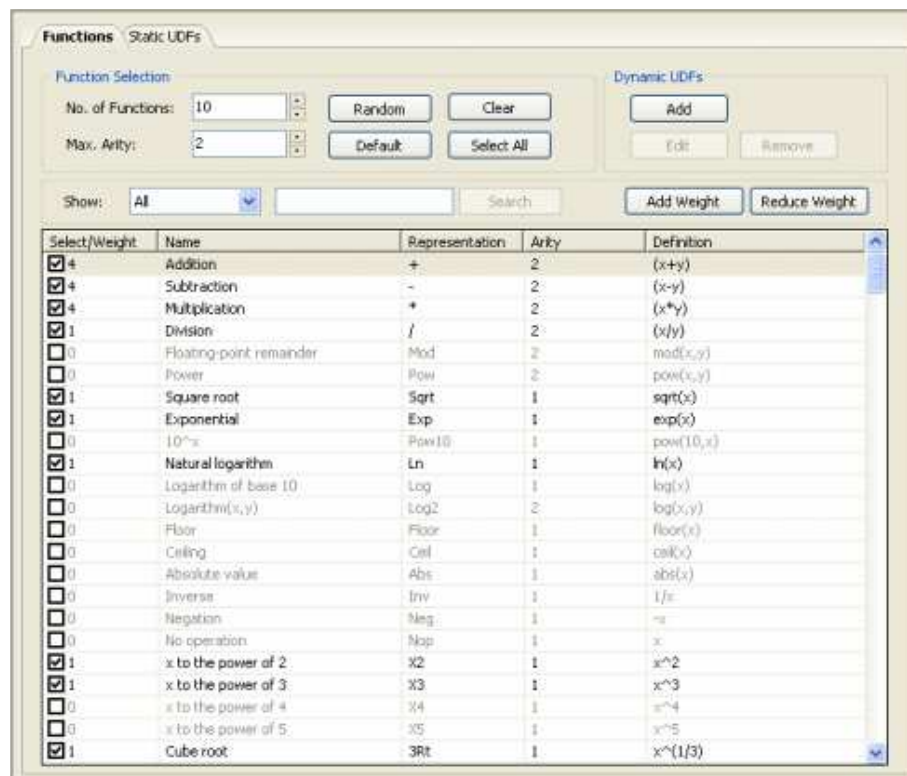
Note that the ET above can be easily linearized into a single **K-expression**:

```
01234567890123456
++a*/aQQ*+/aaabba
```

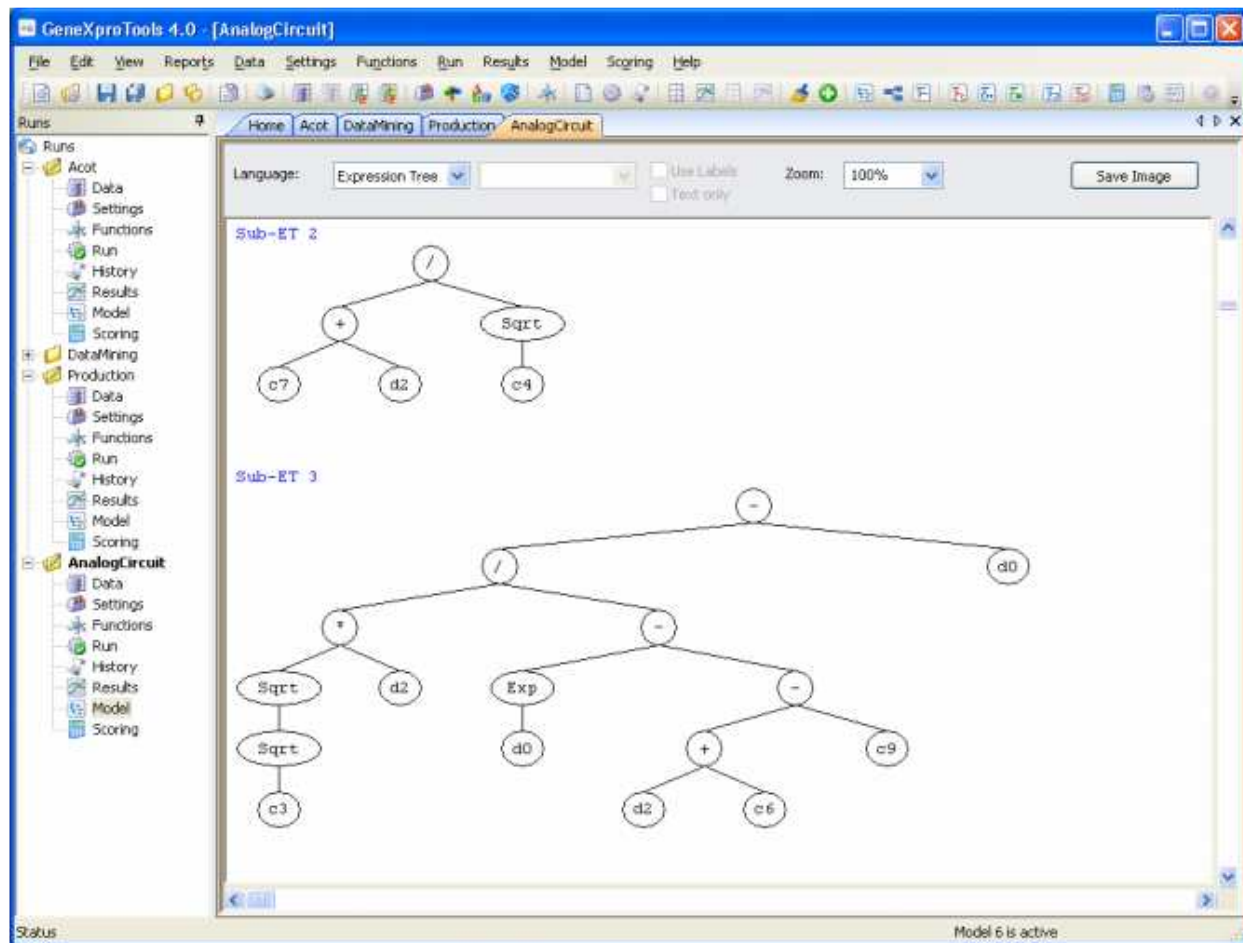
These manipulations are important in order to fully explore all the GeneXproTools features, especially the [Change Seed method](#), which allows you to change or create a particular program and then use this program to breed even better programs.



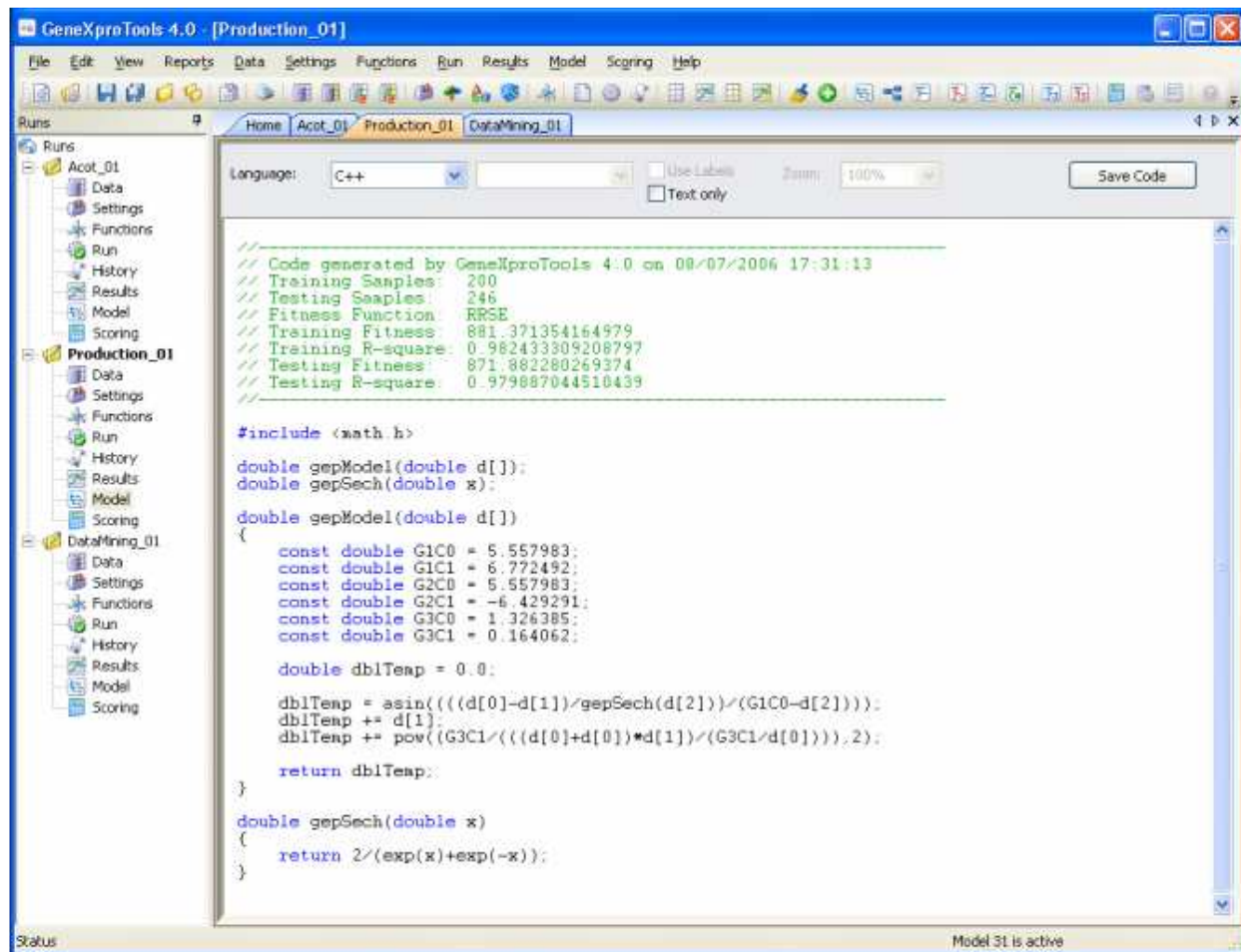
Obviously you'll need to be familiar with the symbols used to represent all the 278 [built-in mathematical functions](#) and the 258 [built-in logical functions](#) of GeneXproTools 4.0 in Karva if you wish to create seed programs with them. But they are quite easy to learn as most of the times their names obey common rules of naming mathematical/logical functions in programming languages.



So, GeneXproTools 4.0 evolves models in its native Karva code, but the models evolved by GeneXproTools in its native language can be automatically parsed into visually appealing [expression trees](#), allowing a quicker and more complete understanding of their mathematical/logical intricacies.



Additionally, GeneXproTools 4.0 offers a total of 16 [built-in grammars](#) so that the models evolved by GeneXproTools in its native language can be automatically translated into the most commonly used programming languages (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL). This code can then be used in other applications.



And finally, GeneXproTools 4.0 allows also the design of [User Defined Grammars](#) so that the models evolved by GeneXproTools in its native language can be automatically translated into the programming language of your choice if you happen to prefer a programming language not already covered by the 16 [built-in grammars](#) of GeneXproTools 4.0 (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL).



Generating Code Automatically

Expression Trees

GeneXproTools 4.0 evolves models in its native [Karva code](#), but the models evolved by GeneXproTools in its native language can be automatically parsed into visually appealing **expression trees**, allowing a quicker and more complete understanding of their mathematical/logical intricacies.

Language: Karva Genes Use Labels Zoom: 75% Save Code
☐ Text only

```

Sqrt.d0 * ./Ln * - * ./c0.d2.d1.d2.d2.c0.d1.d2.d2.d2
+
+ * Atan - Exp + * - .d2 + c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + d1 * * c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

```

Numerical Constants:

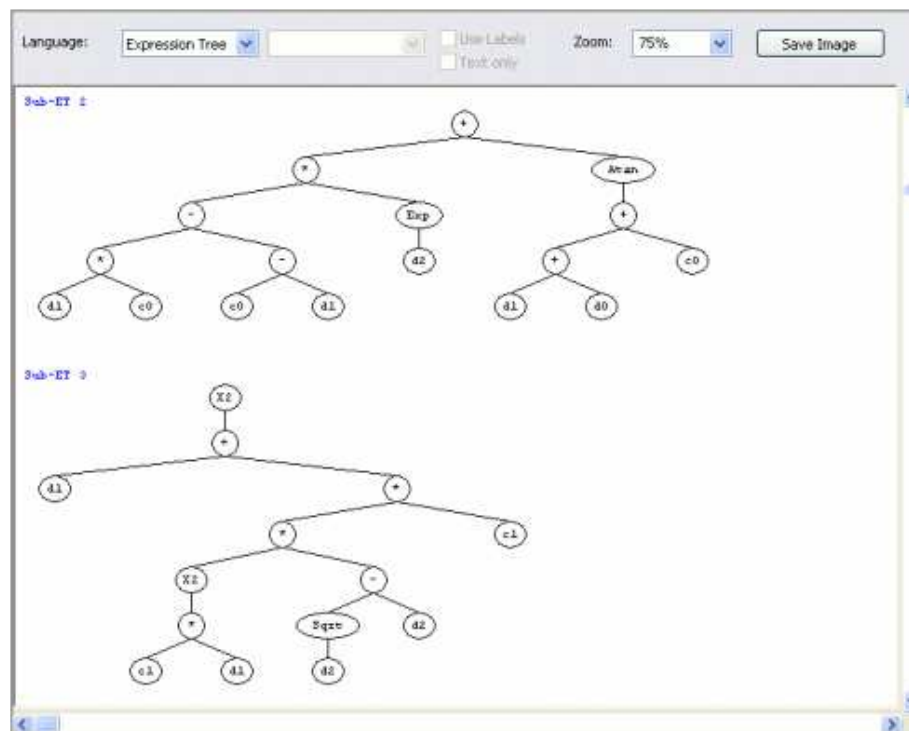
Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

To see a tree rendition of your model, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Expression Tree.



These expression trees can be copied to the clipboard or saved to disk.



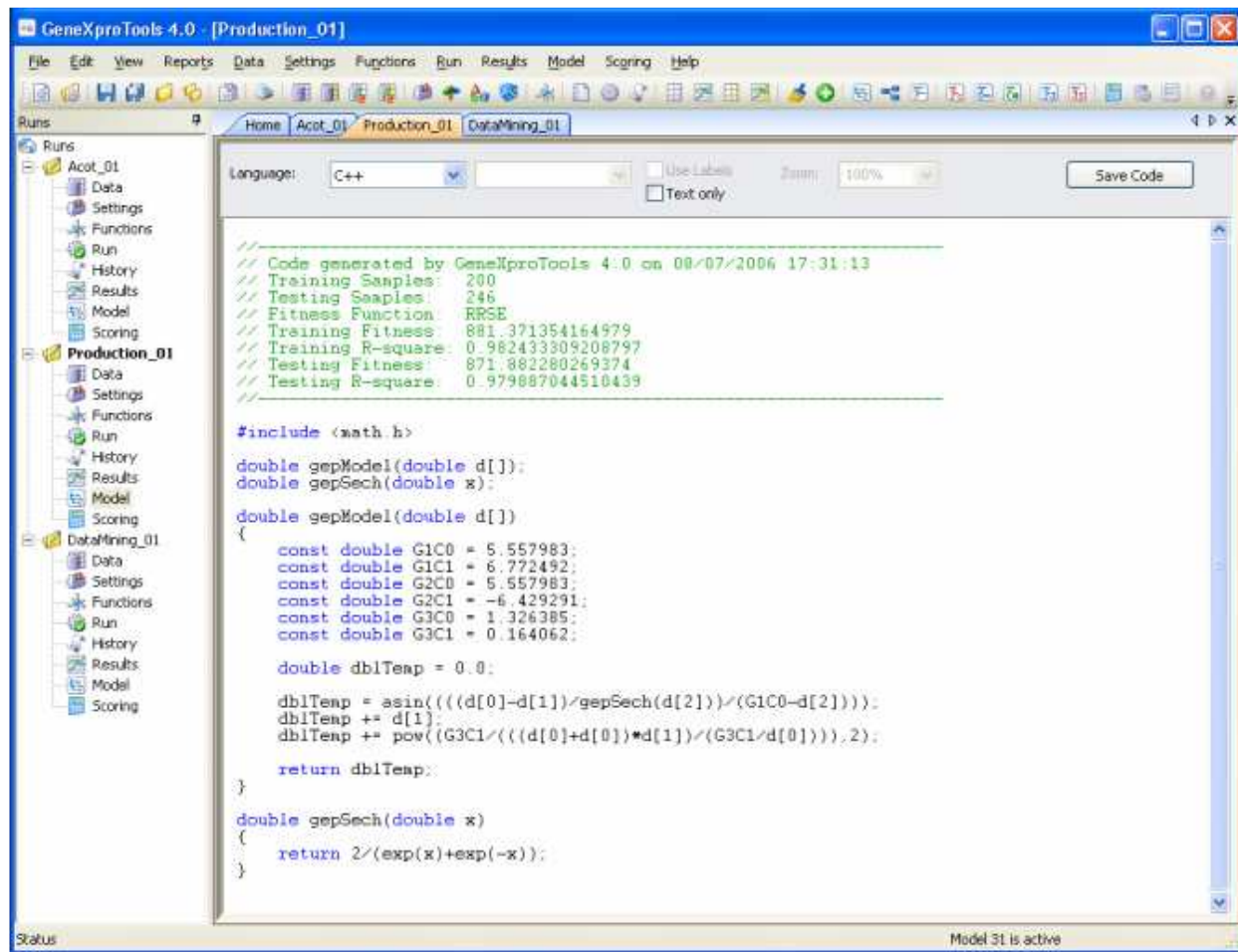
Generating Code Automatically

Built-in Grammars

GeneXproTools 4.0 offers a total of 16 built-in grammars so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the most commonly used programming languages:

- [Ada](#)
- [C](#)
- [C++](#)
- [C#](#)
- [Fortran](#)
- [Java](#)
- [Java Script](#)
- [Matlab](#)
- [Pascal](#)
- [Perl](#)
- [PHP](#)
- [Python](#)
- [Visual Basic](#)
- [VB.Net](#)
- [Verilog](#)
- [VHDL](#)

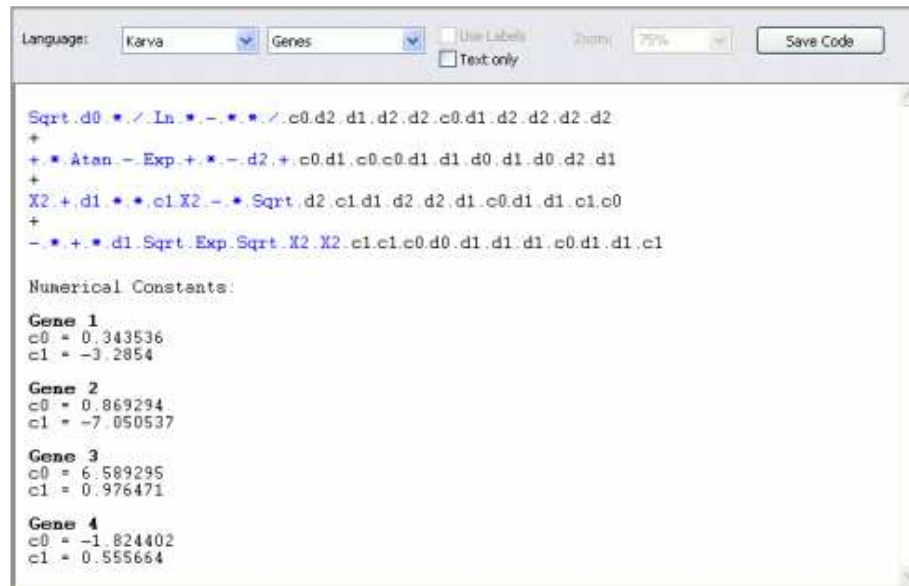
Other grammars can be easily designed using this wide set of built-in grammars as reference, as all of them can be accessed and modified by the user. As an example, you can take a look at the C++ Math Grammar of GeneXproTools [here](#) and at the VHDL Boolean All Gates Grammar [here](#).



Generating Code Automatically

Ada

For mathematical models, GeneXproTools 4.0 uses the **Ada Math grammar** to generate **Ada code** from its native **Karva code**, whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Ada grammar**.



Language: Karva Genes ☐ Use Labels Zoom: 75% ☐ Text only

```

Sqrt.d0 * / In * - * * / .c0.d2 d1.d2.d2.c0.d1.d2.d2.d2
+
* * .Atan - Exp.+ * - .d2+.c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2+.d1.*.*.c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * .d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

```

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

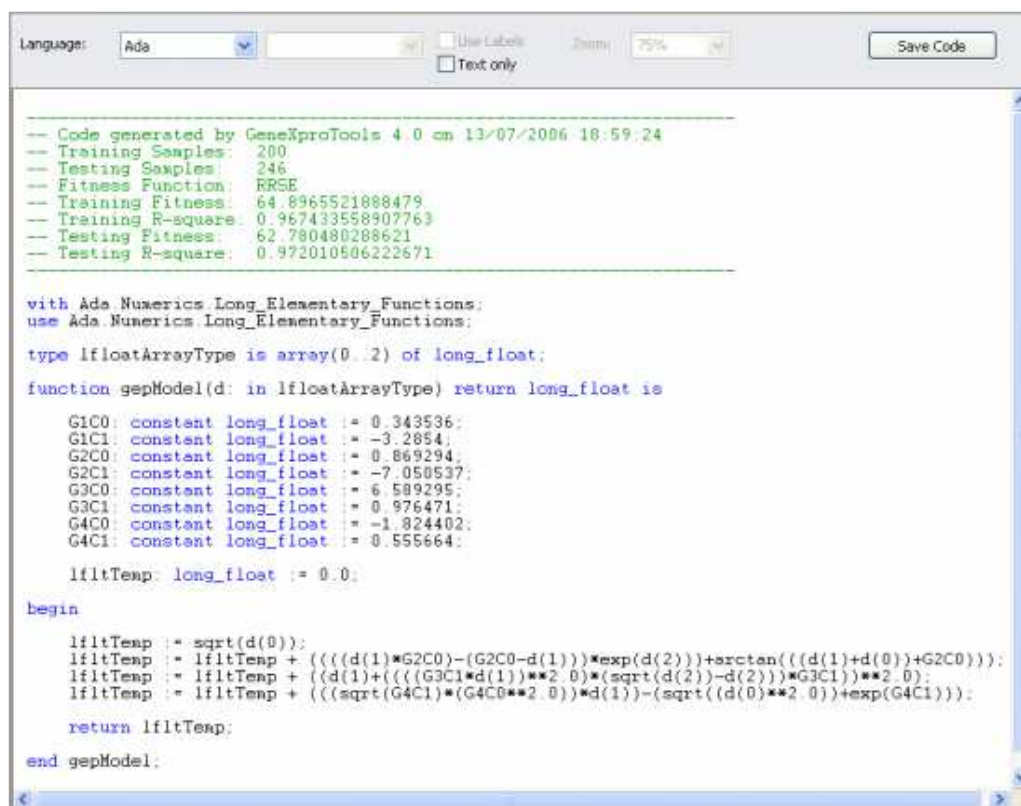
Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Ada. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **Ada**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Ada.



Language: Ada ☐ Use Labels Zoom: 75% ☐ Text only

```

-- Code generated by GeneXproTools 4.0 on 13/07/2006 18:59:24
-- Training Samples: 200
-- Testing Samples: 246
-- Fitness Function: RRSE
-- Training Fitness: 64.8965521888479
-- Training R-square: 0.967433558907763
-- Testing Fitness: 62.780480288621
-- Testing R-square: 0.972010506222671

with Ada.Numerics.Long_Elementary_Functions;
use Ada.Numerics.Long_Elementary_Functions;

type lfloatArrayType is array(0..2) of long_float;

function gepModel(d: in lfloatArrayType) return long_float is

  G1C0: constant long_float := 0.343536;
  G1C1: constant long_float := -3.2854;
  G2C0: constant long_float := 0.869294;
  G2C1: constant long_float := -7.050537;
  G3C0: constant long_float := 6.589295;
  G3C1: constant long_float := 0.976471;
  G4C0: constant long_float := -1.824402;
  G4C1: constant long_float := 0.555664;

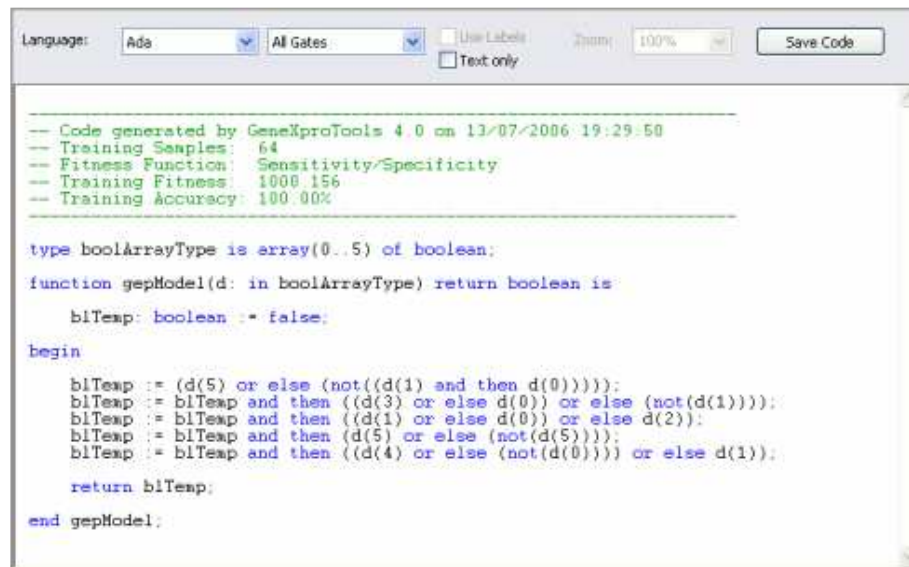
  lfltTemp: long_float := 0.0;

begin
  lfltTemp := sqrt(d(0));
  lfltTemp := lfltTemp + (((d(1)*G2C0)-(G2C0-d(1)))*exp(d(2)))+arctan(((d(1)+d(0))+G2C0));
  lfltTemp := lfltTemp + ((d(1)+(((G3C1*d(1))**2.0)*(sqrt(d(2))-d(2)))*G3C1)**2.0);
  lfltTemp := lfltTemp + (((sqrt(G4C1)*(G4C0**2.0))*d(1))-(sqrt((d(0)**2.0)+exp(G4C1))));

  return lfltTemp;
end gepModel;

```

And to translate your logic circuits into **Ada**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Ada and All Gates on the Language list boxes.



```

Language: Ada All Gates ☐ Use Labels Zoom: 100% 
☐ Text only

-- Code generated by GeneXproTools 4.0 on 13/07/2006 19:29:50
-- Training Samples: 64
-- Fitness Function: Sensitivity/Specificity
-- Training Fitness: 1000.156
-- Training Accuracy: 100.00%

type boolArrayType is array(0..5) of boolean;

function gepModel(d: in boolArrayType) return boolean is
    biTemp: boolean := false;
begin
    biTemp := (d(5) or else (not((d(1) and then d(0)))));
    biTemp := biTemp and then ((d(3) or else d(0)) or else (not(d(1))));
    biTemp := biTemp and then ((d(1) or else d(0)) or else d(2));
    biTemp := biTemp and then (d(5) or else (not(d(5))));
    biTemp := biTemp and then ((d(4) or else (not(d(0)))) or else d(1));
    return biTemp;
end gepModel;

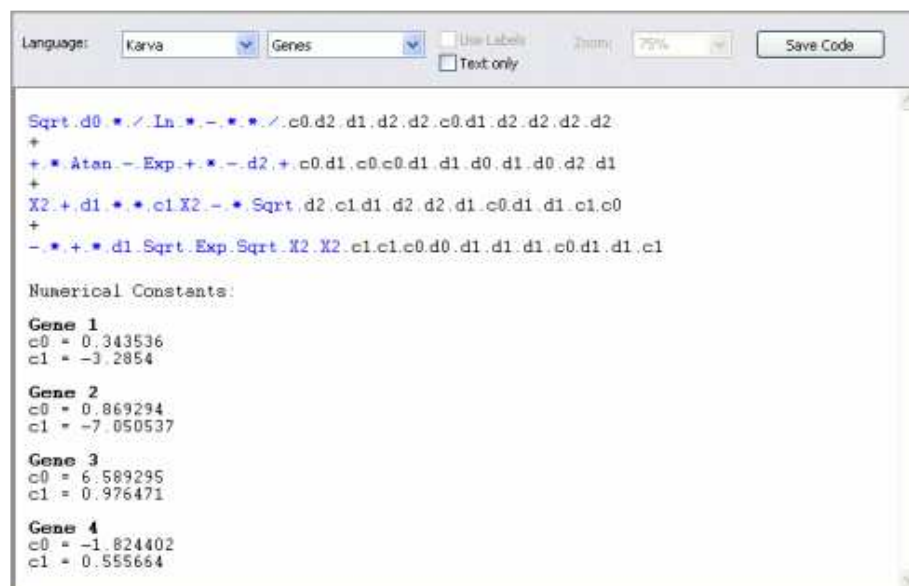
```



Generating Code Automatically

C

For mathematical models, GeneXproTools 4.0 uses the **C Math grammar** to generate **C code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates C grammar**.



```

Language: Karva Genes ☐ Use Labels Zoom: 75% 
☐ Text only

Sqrt.d0 * ./ Ln * - * ./ c0.d2.d1.d2.d2.c0.d1.d2.d2.d2
+
+ * Atan - Exp + * - d2 + c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + d1 * . c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

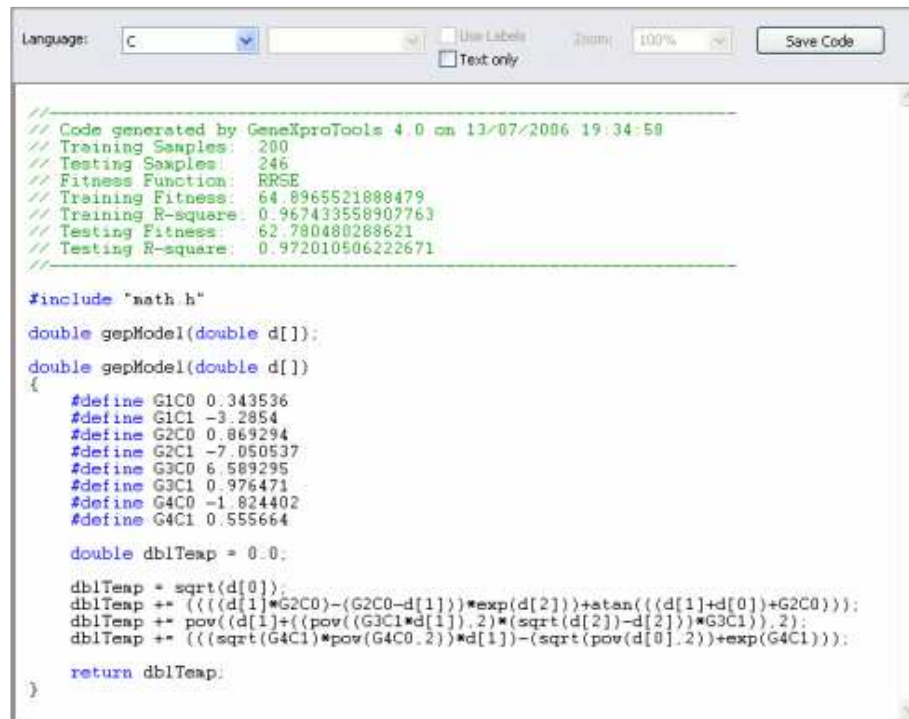
Gene 4
c0 = -1.824402
c1 = 0.555664

```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into C. The code can then be copied to the clipboard or saved to disk. This code

can be freely distributed in source or binary form.

To translate your mathematical models into **C code**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select C.



```

Language: C
Use Labels
Zoom: 100%
Save Code
Text only

// Code generated by GeneXproTools 4.0 on 13/07/2006 19:34:59
// Training Samples: 200
// Testing Samples: 246
// Fitness Function: RRSE
// Training Fitness: 64.8965521888479
// Training R-square: 0.967433558907763
// Testing Fitness: 62.780480288621
// Testing R-square: 0.972010506222671
//

#include "math.h"

double gepModel(double d[]);
double gepModel(double d[])
{
    #define G1C0 0.343536
    #define G1C1 -3.2854
    #define G2C0 0.869294
    #define G2C1 -7.050537
    #define G3C0 6.589295
    #define G3C1 0.976471
    #define G4C0 -1.824402
    #define G4C1 0.555664

    double dblTemp = 0.0;

    dblTemp = sqrt(d[0]);
    dblTemp += (((d[1]*G2C0)-(G2C0-d[1]))*exp(d[2]))+atan(((d[1]+d[0])+G2C0));
    dblTemp += pow((d[1]+((pow((G3C1*d[1]),2)*(sqrt(d[2])-d[2]))*G3C1)),2);
    dblTemp += (((sqrt(G4C1)*pow(G4C0,2))*d[1])-(sqrt(pow(d[0],2))+exp(G4C1)));

    return dblTemp;
}
  
```

And to translate your logic circuits into **C code**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both C and All Gates on the Language list boxes.



```

// Code generated by GeneXproTools 4.0 on 13/07/2006 20:11:13
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%
//

int gepModel(int d[]);
int gepModel(int d[])
{
    #define TRUE 1
    #define FALSE 0

    int varTemp = FALSE;

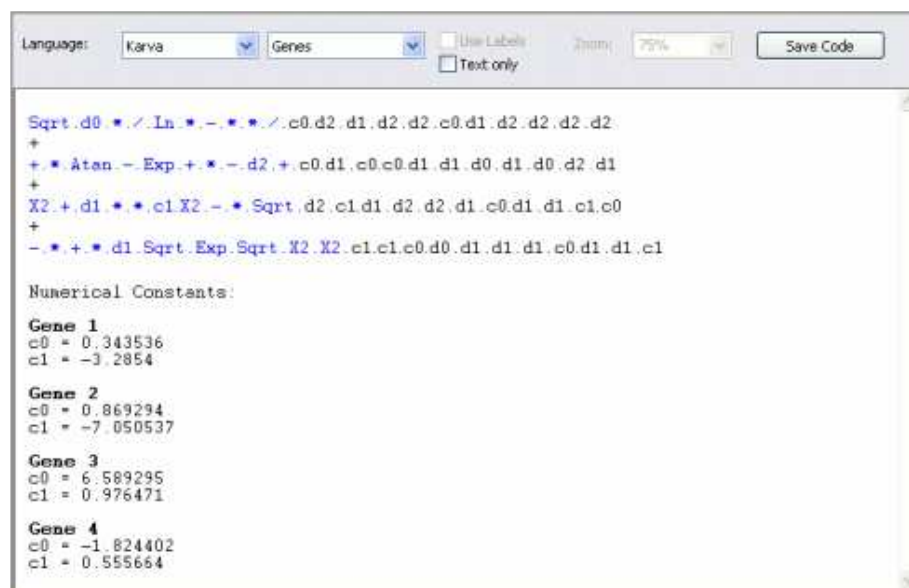
    varTemp = (d[5] || (!((d[1] && d[0]))));
    varTemp = varTemp && ((d[3] || d[0]) || (!d[1]));
    varTemp = varTemp && ((d[1] || d[0]) || d[2]);
    varTemp = varTemp && (d[5] || (!d[5]));
    varTemp = varTemp && ((d[4] || (!d[0])) || d[1]);

    return varTemp;
}
  
```

Generating Code Automatically

C++

For mathematical models, GeneXproTools 4.0 uses the **C++ Math grammar** to generate **C++ code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates C++ grammar**.



```

Sqrt.d0 * / In * - * * / .c0.d2 d1.d2.d2.c0.d1.d2.d2.d2
+
+ * .Aten - Exp.+ * - .d2.+c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2.+d1.*.*.c1.X2.-.*.Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
-.*.+.*.d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

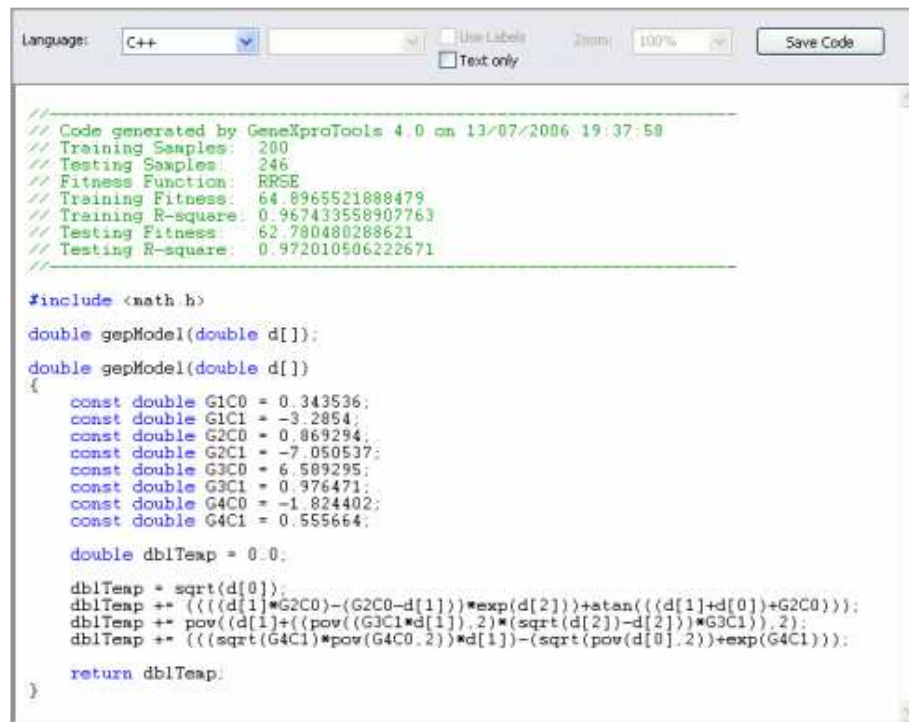
Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into C++. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **C++**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select C++.



The screenshot shows the GeneXproTools interface with the Language dropdown set to C++. The code area displays the following C++ code:

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 19:37:59
// Training Samples: 200
// Testing Samples: 246
// Fitness Function: RRSE
// Training Fitness: 64.8965521888479
// Training R-square: 0.967433558907763
// Testing Fitness: 62.780480288621
// Testing R-square: 0.972010506222671
//

#include <math.h>

double gepModel(double d[]);
double gepModel(double d[])
{
    const double G1C0 = 0.343536;
    const double G1C1 = -3.2854;
    const double G2C0 = 0.869294;
    const double G2C1 = -7.050537;
    const double G3C0 = 6.589295;
    const double G3C1 = 0.976471;
    const double G4C0 = -1.824402;
    const double G4C1 = 0.555664;

    double dblTemp = 0.0;

    dblTemp = sqrt(d[0]);
    dblTemp += (((d[1]*G2C0)-(G2C0-d[1]))*exp(d[2]))+atan(((d[1]+d[0])+G2C0));
    dblTemp += pow((d[1]+(pow((G3C1*d[1]),2)*(sqrt(d[2])-d[2]))*G3C1)),2);
    dblTemp += (((sqrt(G4C1)*pow(G4C0,2))*d[1])-(sqrt(pow(d[0],2)+exp(G4C1))));

    return dblTemp;
}
```

And to translate your logic circuits into **C++**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both C++ and All Gates on the Language list boxes.



The screenshot shows the GeneXproTools interface with the Language dropdown set to C++ and the All Gates dropdown selected. The code area displays the following C++ code:

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:14:30
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.15%
// Training Accuracy: 100.00%
//

bool gepModel(bool d[]);
bool gepModel(bool d[])
{
    bool blTemp = false;

    blTemp = (d[5] || (((d[1] && d[0]))));
    blTemp = blTemp && ((d[3] || d[0]) || ((d[1])));
    blTemp = blTemp && ((d[1] || d[0]) || d[2]);
    blTemp = blTemp && (d[5] || ((d[5])));
    blTemp = blTemp && ((d[4] || ((d[0])) || d[1]));

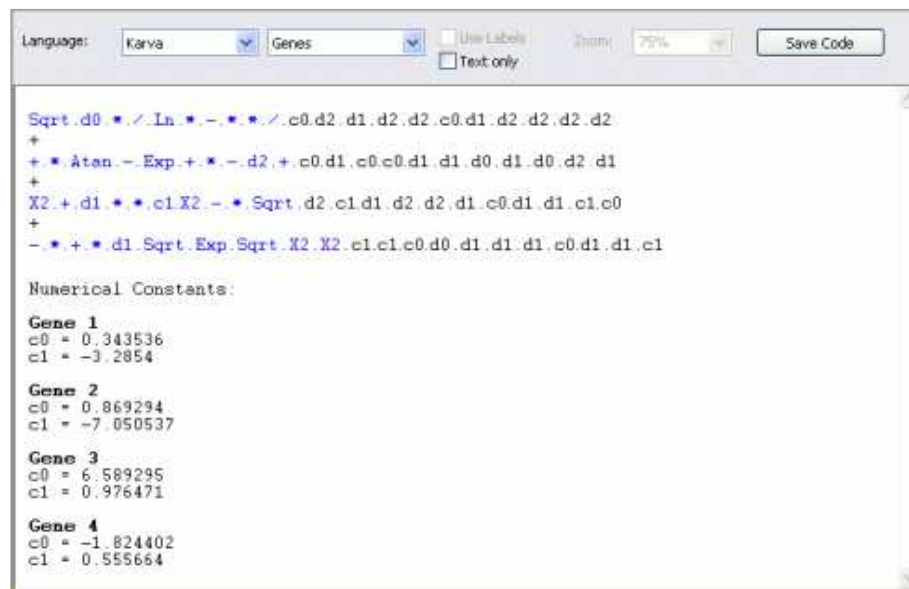
    return blTemp;
}
```

◀ | ▶

Generating Code Automatically

C#

For mathematical models, GeneXproTools 4.0 uses the **C# Math grammar** to generate **C# code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates C# grammar**.



The screenshot shows the GeneXproTools 4.0 interface with the 'Language' dropdown set to 'Karva'. The main text area contains the following Karva code and numerical constants:

```

Sqrt.d0 * / In * - * * / .c0.d2 d1.d2.d2.c0.d1.d2.d2.d2
+
* * .Aten - Exp + * - .d2 + .c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + .d1 * * .c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * .d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

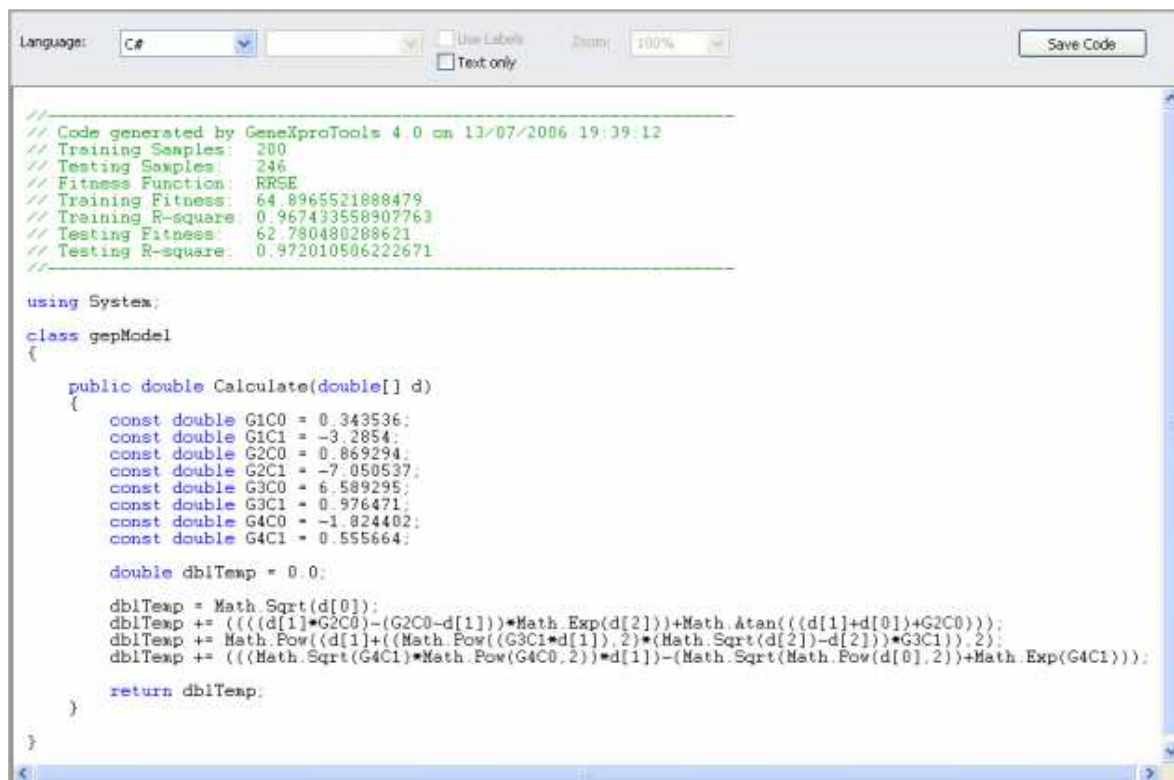
Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664
  
```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into C#. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **C#**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select C#.



The screenshot shows the GeneXproTools 4.0 interface with the 'Language' dropdown set to 'C#'. The main text area contains the following C# code generated from the Karva code:

```

// Code generated by GeneXproTools 4.0 on 13/07/2006 19:39:12
// Training Samples: 200
// Testing Samples: 246
// Fitness Function: RRSE
// Training Fitness: 64.8965521888479
// Training R-square: 0.967433558907763
// Testing Fitness: 62.780480288621
// Testing R-square: 0.972010506222671
//

using System;

class gepModel
{
    public double Calculate(double[] d)
    {
        const double G1C0 = 0.343536;
        const double G1C1 = -3.2854;
        const double G2C0 = 0.869294;
        const double G2C1 = -7.050537;
        const double G3C0 = 6.589295;
        const double G3C1 = 0.976471;
        const double G4C0 = -1.824402;
        const double G4C1 = 0.555664;

        double dblTemp = 0.0;

        dblTemp = Math.Sqrt(d[0]);
        dblTemp += (((d[1]*G2C0)-(G2C0-d[1]))*Math.Exp(d[2]))+Math.Atan(((d[1]+d[0])+G2C0));
        dblTemp += Math.Pow((d[1]+((Math.Pow((G3C1*d[1]), 2)*(Math.Sqrt(d[2])-d[2]))*G3C1)), 2);
        dblTemp += (((Math.Sqrt(G4C1)*Math.Pow(G4C0, 2))*d[1])-(Math.Sqrt(Math.Pow(d[0], 2))+Math.Exp(G4C1)));

        return dblTemp;
    }
}
  
```


And to translate your logic circuits into **C#**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both C# and All Gates on the Language list boxes.

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:15:47
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.15%
// Training Accuracy: 100.00%

using System;
class gepModel
{
    public bool Calculate(bool[] d)
    {
        bool blTemp = false;

        blTemp = (d[5] || (!((d[1] && d[0]))));
        blTemp = blTemp && ((d[3] || d[0]) || (!d[1]));
        blTemp = blTemp && ((d[1] || d[0]) || d[2]);
        blTemp = blTemp && (d[5] || (!d[5]));
        blTemp = blTemp && ((d[4] || (!d[0])) || d[1]);

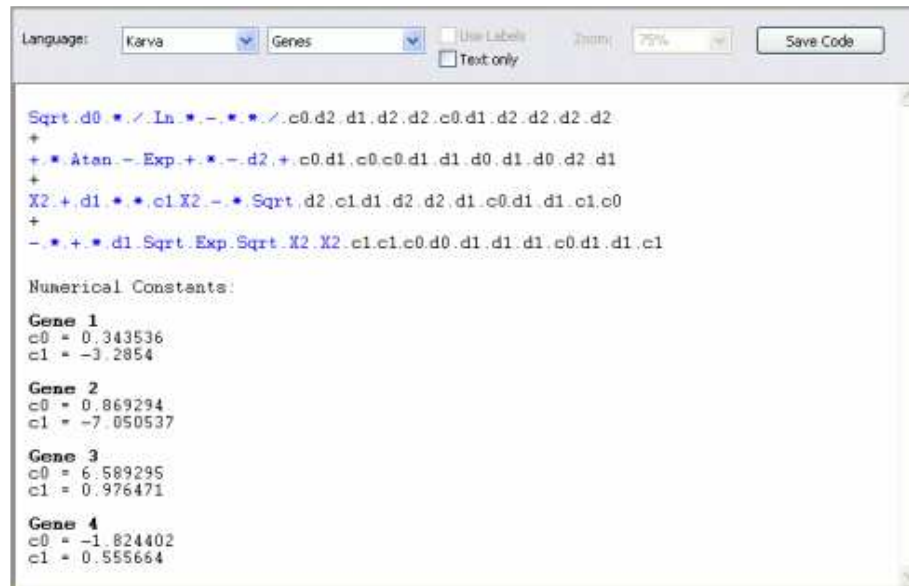
        return blTemp;
    }
}
```

◀ | ▶

Generating Code Automatically

Fortran

For mathematical models, GeneXproTools 4.0 uses the **Fortran Math grammar** to generate **Fortran code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Fortran grammar**.



Language: Karva Genes ☐ Use Labels Zoom: 75% ☐ Text only

```

Sqrt.d0 * ./Ln * - * ./c0.d2.d1.d2.d2.c0.d1.d2.d2.d2
+
+ * Atan - Exp + * - .d2 + c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + d1 * * .c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * .d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

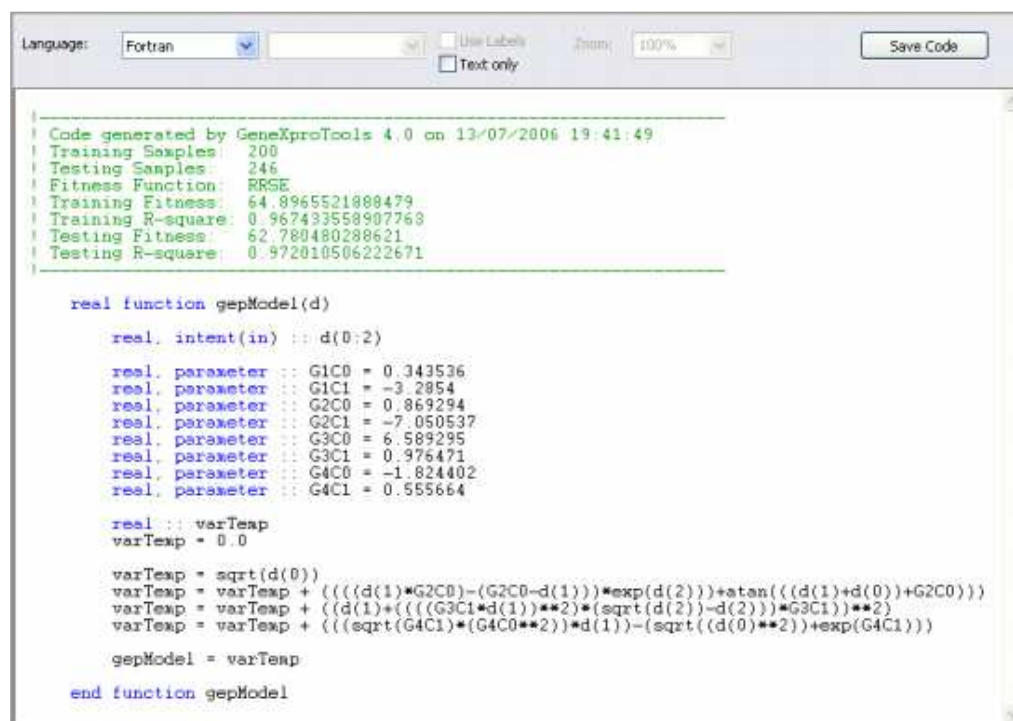
Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Fortran. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **Fortran**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Fortran.



Language: Fortran ☐ Use Labels Zoom: 100% ☐ Text only

```

! Code generated by GeneXproTools 4.0 on 13/07/2006 19:41:49
! Training Samples: 200
! Testing Samples: 246
! Fitness Function: RRSE
! Training Fitness: 64.8965521888479
! Training R-square: 0.967433558997763
! Testing Fitness: 62.780480288621
! Testing R-square: 0.972010506222671

real function gepModel(d)
    real, intent(in) :: d(0:2)

    real, parameter :: G1C0 = 0.343536
    real, parameter :: G1C1 = -3.2854
    real, parameter :: G2C0 = 0.869294
    real, parameter :: G2C1 = -7.050537
    real, parameter :: G3C0 = 6.589295
    real, parameter :: G3C1 = 0.976471
    real, parameter :: G4C0 = -1.824402
    real, parameter :: G4C1 = 0.555664

    real :: varTemp
    varTemp = 0.0

    varTemp = sqrt(d(0))
    varTemp = varTemp + (((d(1)*G2C0)-(G2C0-d(1)))*exp(d(2)))+atan(((d(1)+d(0))+G2C0)))
    varTemp = varTemp + ((d(1)+(((G3C1*d(1))**2)*(sqrt(d(2))-d(2)))*G3C1))**2)
    varTemp = varTemp + (((sqrt(G4C1)*(G4C0**2))*d(1))-(sqrt((d(0)**2))+exp(G4C1)))

    gepModel = varTemp
end function gepModel

```

And to translate your logic circuits into **Fortran**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Fortran and All Gates on the Language list boxes.

```

Language: Fortran All Gates Use Labels Zoom: 100% Save Code
Text only

Code generated by GeneXproTools 4.0 on 13/07/2006 20:17:18
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

logical function gepModel(d)
    logical, intent(in) :: d(0:5)
    logical :: varTemp
    varTemp = .false.

    varTemp = (d(5) .or. (.not.((d(1) .and. d(0)))))
    varTemp = varTemp .and. ((d(3) .or. d(0)) .or. (.not. (d(1))))
    varTemp = varTemp .and. ((d(1) .or. d(0)) .or. d(2))
    varTemp = varTemp .and. (d(5) .or. (.not. (d(5))))
    varTemp = varTemp .and. ((d(4) .or. (.not. (d(0)))) .or. d(1))

    gepModel = varTemp
end function gepModel
  
```

◀ | ▶

Generating Code Automatically

Java

For mathematical models, GeneXproTools 4.0 uses the **Java Math grammar** to generate **Java code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Java grammar**.

```

Language: Karva Genes Use Labels Zoom: 75% Save Code
Text only

Sqrt.d0 * ./Ln * - * ./c0.d2.d1.d2.d2.c0.d1.d2.d2.d2
+
+ * Atan - Exp + * - d2 + c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + d1 * ./c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

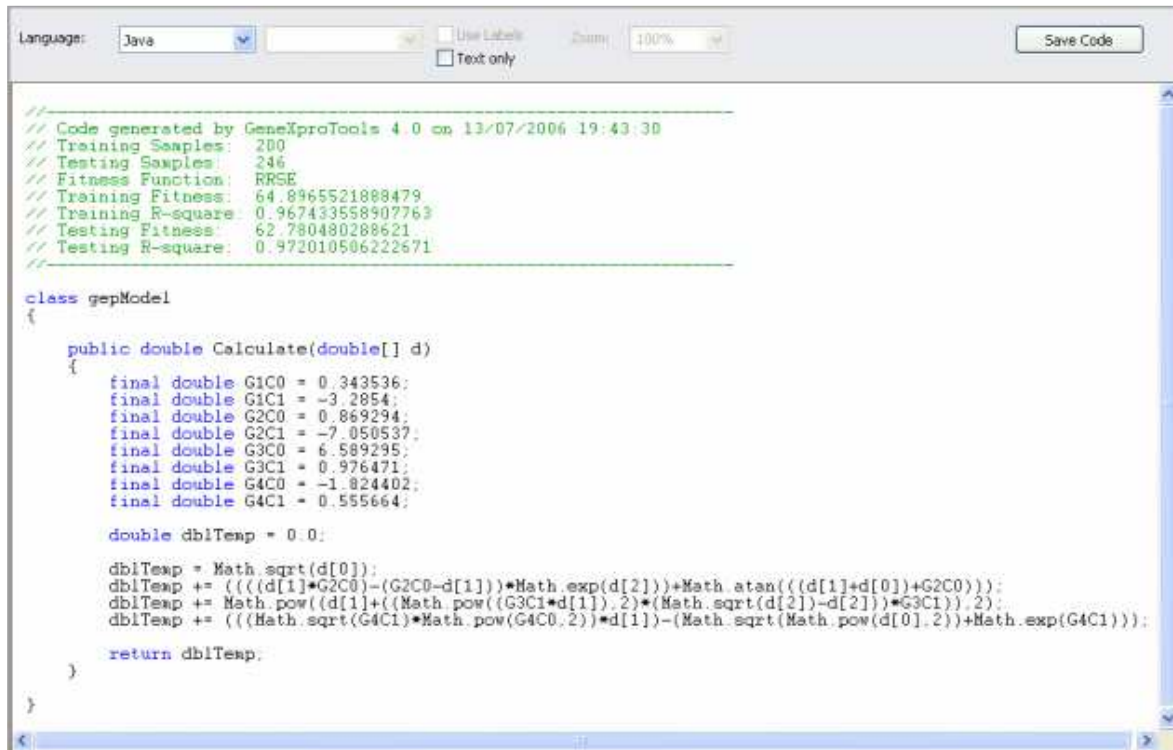
Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555654
  
```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Java. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **Java**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Java.



The screenshot shows the GeneXproTools interface with the Language dropdown set to 'Java'. The code area displays the following Java code:

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 19:43:30
// Training Samples: 200
// Testing Samples: 246
// Fitness Function: RRSE
// Training Fitness: 64.8965521888479
// Training R-square: 0.967433558907763
// Testing Fitness: 62.780480288621
// Testing R-square: 0.972010506222671

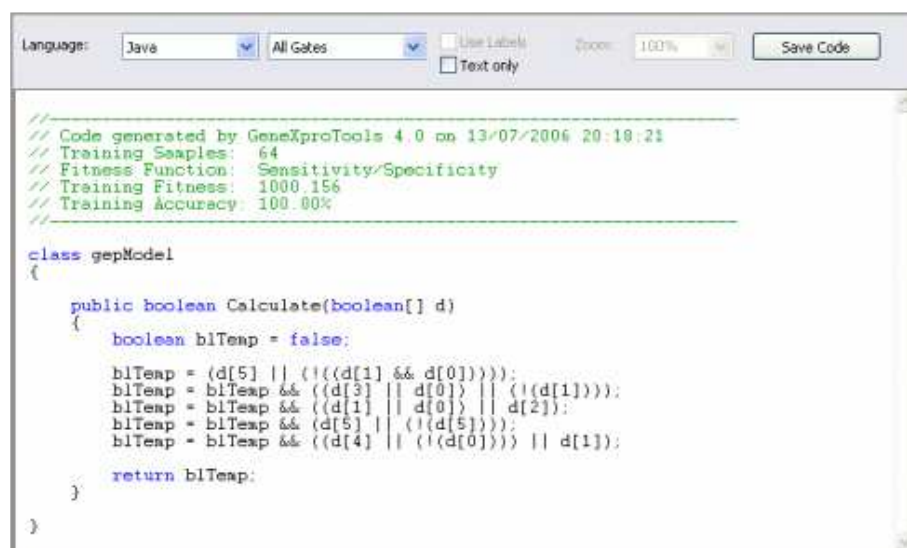
class gepModel
{
    public double Calculate(double[] d)
    {
        final double G1C0 = 0.343536;
        final double G1C1 = -3.2854;
        final double G2C0 = 0.869294;
        final double G2C1 = -7.050537;
        final double G3C0 = 6.589295;
        final double G3C1 = 0.976471;
        final double G4C0 = -1.824402;
        final double G4C1 = 0.555664;

        double dblTemp = 0.0;

        dblTemp = Math.sqrt(d[0]);
        dblTemp += (((d[1]*G2C0)-(G2C0-d[1]))*Math.exp(d[2]))+Math.atan(((d[1]+d[0])+G2C0)));
        dblTemp += Math.pow((d[1]+((Math.pow((G3C1*d[1]),2)*(Math.sqrt(d[2])-d[2]))*G3C1)),2);
        dblTemp += (((Math.sqrt(G4C1)*Math.pow(G4C0,2))*d[1])-(Math.sqrt(Math.pow(d[0],2))+Math.exp(G4C1)));

        return dblTemp;
    }
}
```

And to translate your logic circuits into **Java**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Java and All Gates on the Language list boxes.



The screenshot shows the GeneXproTools interface with the Language dropdown set to 'Java' and the All Gates dropdown selected. The code area displays the following Java code:

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:18:21
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%

class gepModel
{
    public boolean Calculate(boolean[] d)
    {
        boolean blTemp = false;

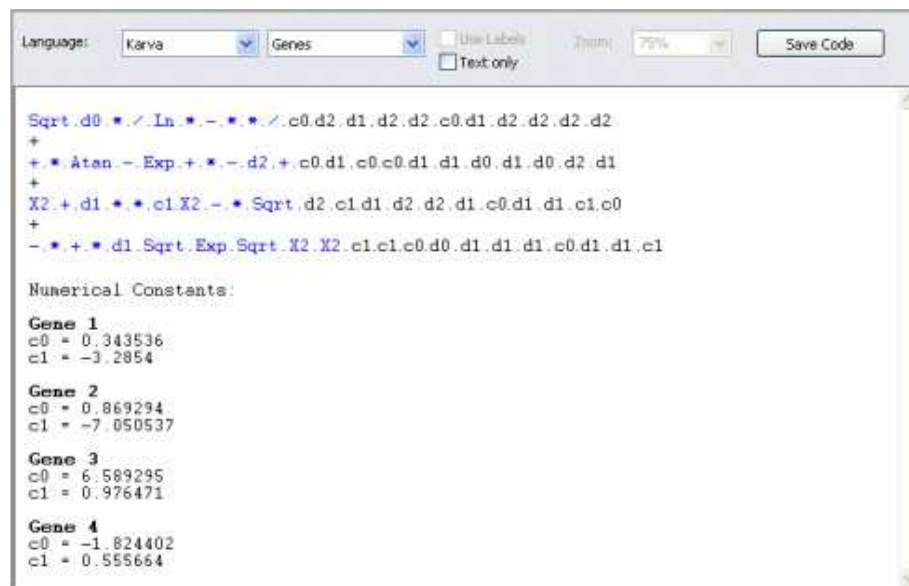
        blTemp = (d[5] || (!((d[1] && d[0]))));
        blTemp = blTemp && ((d[3] || d[0]) || (!d[1]));
        blTemp = blTemp && ((d[1] || d[0]) || d[2]);
        blTemp = blTemp && (d[5] || (!d[5]));
        blTemp = blTemp && ((d[4] || (!d[0])) || d[1]);

        return blTemp;
    }
}
```

Generating Code Automatically

Java Script

For mathematical models, GeneXproTools 4.0 uses the **Java Script Math grammar** to generate **Java Script code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Java Script grammar**.



```

Sqrt.d0 * / In * - * * / c0.d2 d1.d2.d2.c0.d1.d2.d2.d2.d2
+
+ * Aten - Exp + * - .d2 + c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + d1 * * c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

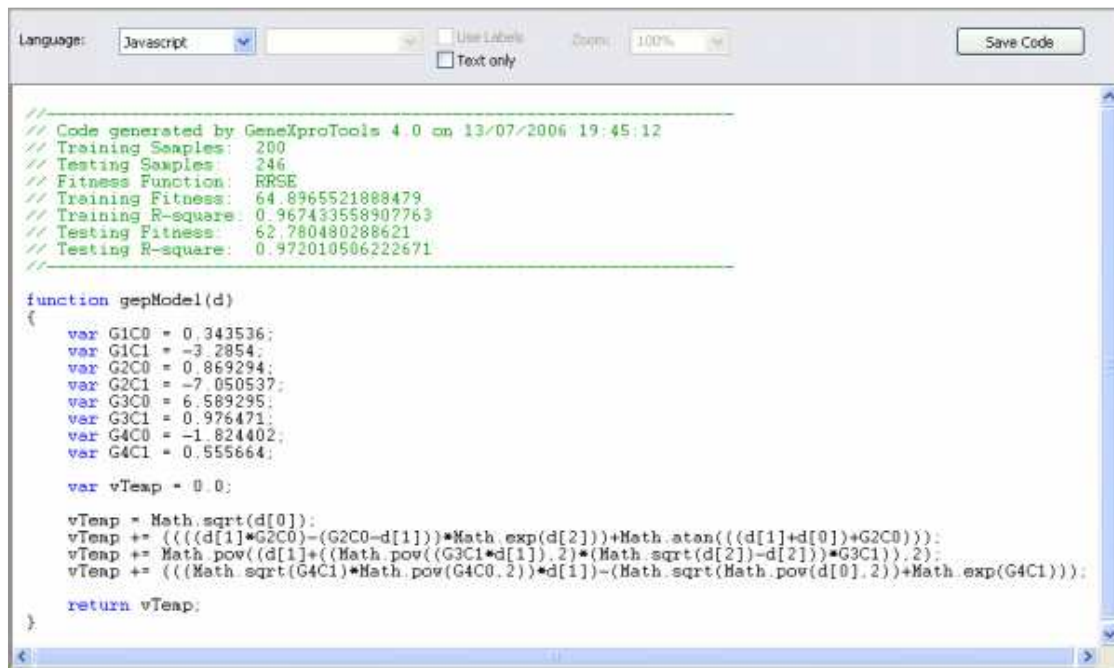
Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664
  
```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Java Script. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **Java Script**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Java Script.



Language: Javascript ☐ Use Labels Zoom: 100% ☐ Text only

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 19:45:12
// Training Samples: 200
// Testing Samples: 246
// Fitness Function: RRSE
// Training Fitness: 64.8965521888479
// Training R-square: 0.967433558907763
// Testing Fitness: 62.780480288621
// Testing R-square: 0.972010506222671

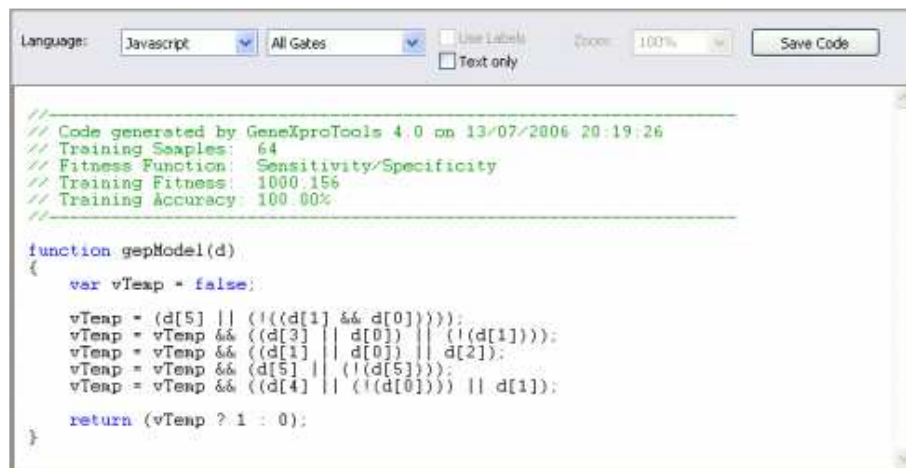
function gepModel(d)
{
    var G1C0 = 0.343536;
    var G1C1 = -3.2854;
    var G2C0 = 0.869294;
    var G2C1 = -7.050537;
    var G3C0 = 6.589295;
    var G3C1 = 0.976471;
    var G4C0 = -1.824402;
    var G4C1 = 0.555664;

    var vTemp = 0.0;

    vTemp = Math.sqrt(d[0]);
    vTemp += (((d[1]*G2C0)-(G2C0-d[1]))*Math.exp(d[2]))+Math.atan(((d[1]+d[0])*G2C0));
    vTemp += Math.pow((d[1]+((Math.pow((G3C1*d[1]),2)*(Math.sqrt(d[2])-d[2]))*G3C1)),2);
    vTemp += (((Math.sqrt(G4C1)*Math.pow(G4C0,2))*d[1])-(Math.sqrt(Math.pow(d[0],2))+Math.exp(G4C1)));

    return vTemp;
}
```

And to translate your logic circuits into **Java Script**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Java Script and All Gates on the Language list boxes.



Language: Javascript ☐ Use Labels Zoom: 100% ☐ Text only All Gates

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:19:26
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%

function gepModel(d)
{
    var vTemp = false;

    vTemp = (d[5] || (!((d[1] && d[0]))));
    vTemp = vTemp && ((d[3] || d[0]) || (!d[1]));
    vTemp = vTemp && ((d[1] || d[0]) || d[2]);
    vTemp = vTemp && (d[5] || (!d[5]));
    vTemp = vTemp && ((d[4] || (!d[0])) || d[1]);

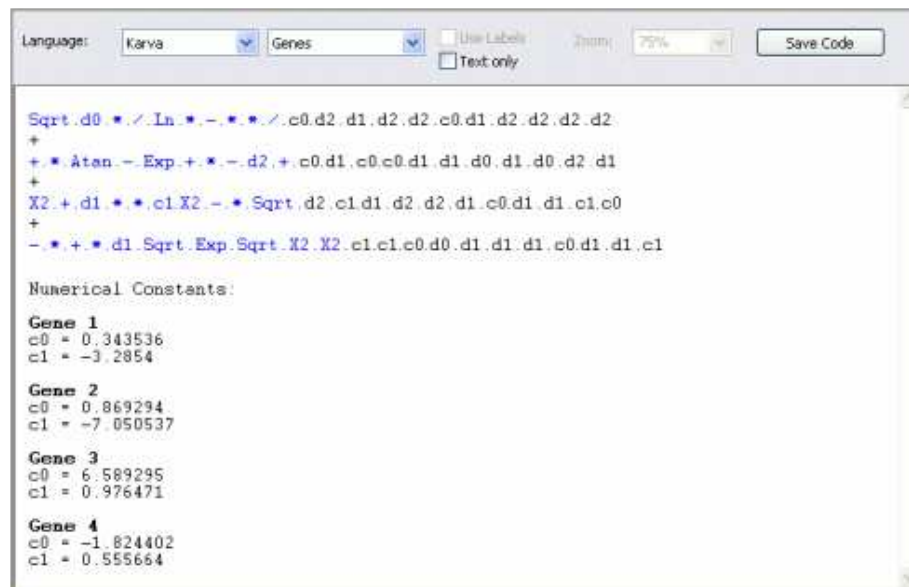
    return (vTemp ? 1 : 0);
}
```

◀ | ▶

Generating Code Automatically

Matlab

For mathematical models, GeneXproTools 4.0 uses the **Matlab Math grammar** to generate **Matlab code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Matlab grammar**.



Language: Karva Genes ☐ Use Labels Zoom: 75% ☐ Text only

```

Sqrt.d0 * / In * - * * / .c0.d2 d1.d2.d2.c0.d1.d2.d2.d2
+
+ * Atan - Exp + * - .d2 + .c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + .d1 * * .c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * .d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

```

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

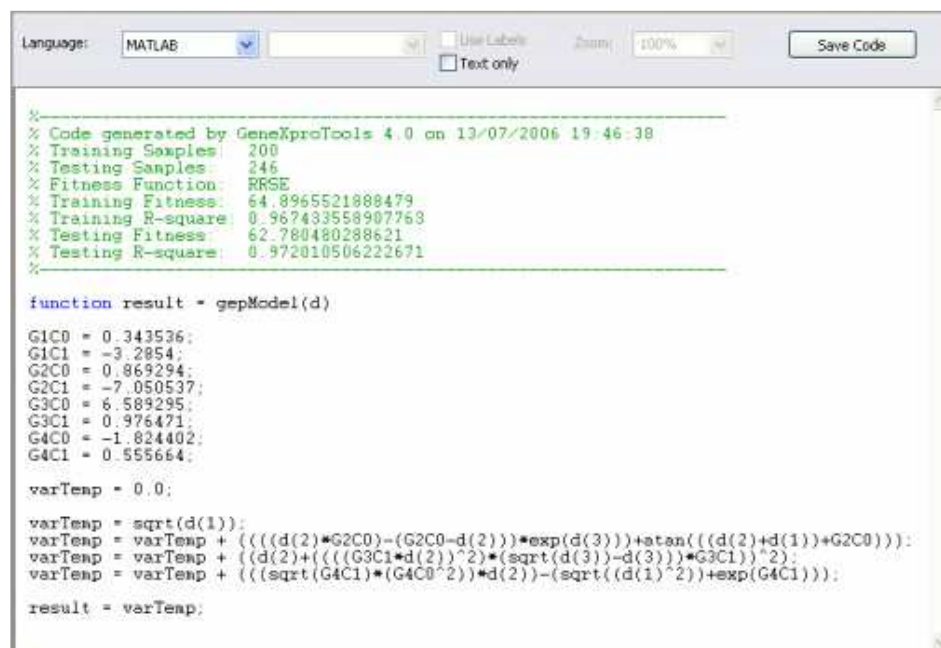
Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Matlab. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **Matlab**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Matlab.



Language: MATLAB ☐ Use Labels Zoom: 100% ☐ Text only

```

%-----
% Code generated by GeneXproTools 4.0 on 13/07/2006 19:46:38
% Training Samples: 200
% Testing Samples: 246
% Fitness Function: RRSE
% Training Fitness: 64.8965521888479
% Training R-square: 0.967433558907763
% Testing Fitness: 62.780480288621
% Testing R-square: 0.972010506222671
%-----

function result = gepModel(d)

G1C0 = 0.343536;
G1C1 = -3.2854;
G2C0 = 0.869294;
G2C1 = -7.050537;
G3C0 = 6.589295;
G3C1 = 0.976471;
G4C0 = -1.824402;
G4C1 = 0.555664;

varTemp = 0.0;

varTemp = sqrt(d(1));
varTemp = varTemp + (((d(2)*G2C0)-(G2C0-d(2)))*exp(d(3)))+atan(((d(2)+d(1))+G2C0));
varTemp = varTemp + ((d(2)+(((G3C1*d(2))^2)*(sqrt(d(3))-d(3)))*G3C1))^2);
varTemp = varTemp + (((sqrt(G4C1)*(G4C0^2))*d(2))-(sqrt((d(1)^2))+exp(G4C1)));

result = varTemp;

```

And to translate your logic circuits into **Matlab**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Matlab and All Gates on the Language list boxes.



```

Language: MATLAB All Gates ☐ Use Labels Zoom: 100% Save Code
☐ Text only

%-----
% Code generated by GeneXproTools 4.0 on 13/07/2006 20:20:27
% Training Samples: 64
% Fitness Function: Sensitivity/Specificity
% Training Fitness: 1000.156
% Training Accuracy: 100.00%
%-----

function result = gepModel(d)
TRUE = 1;
FALSE = 0;

varTemp = FALSE;

varTemp = (d(6) | (~((d(2) & d(1)))));
varTemp = varTemp & ((d(4) | d(1)) | (~d(2)));
varTemp = varTemp & ((d(2) | d(1)) | d(3));
varTemp = varTemp & (d(6) | (~d(6)));
varTemp = varTemp & ((d(5) | (~d(1))) | d(2));

result = varTemp;

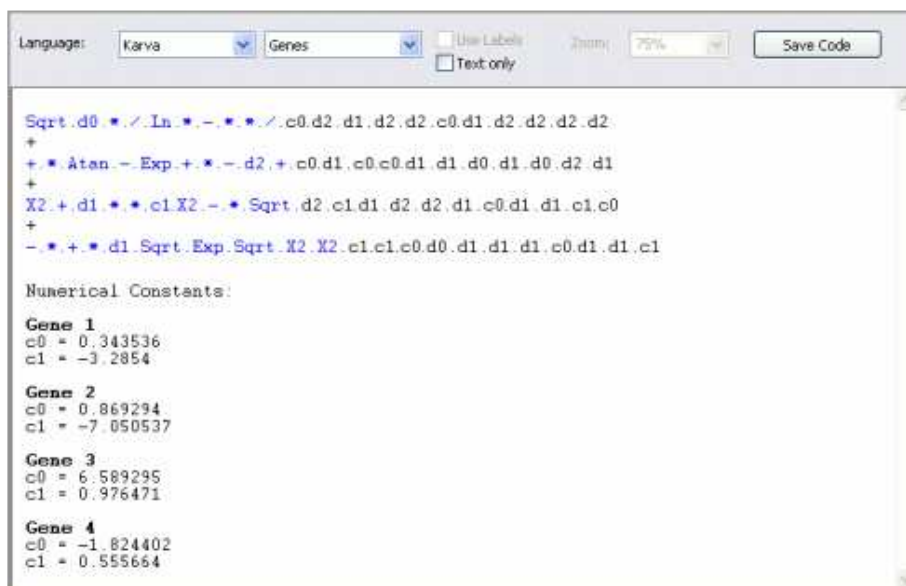
```



Generating Code Automatically

Pascal

For mathematical models, GeneXproTools 4.0 uses the **Pascal Math grammar** to generate **Pascal code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Pascal grammar**.



```

Language: Karva Genes ☐ Use Labels Zoom: 75% Save Code
☐ Text only

Sqrt.d0 = ln * - * * / c0.d2.d1.d2.d2.c0.d1.d2.d2.d2
+
+ * Atan - Exp + * - d2 + c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + d1 * * c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

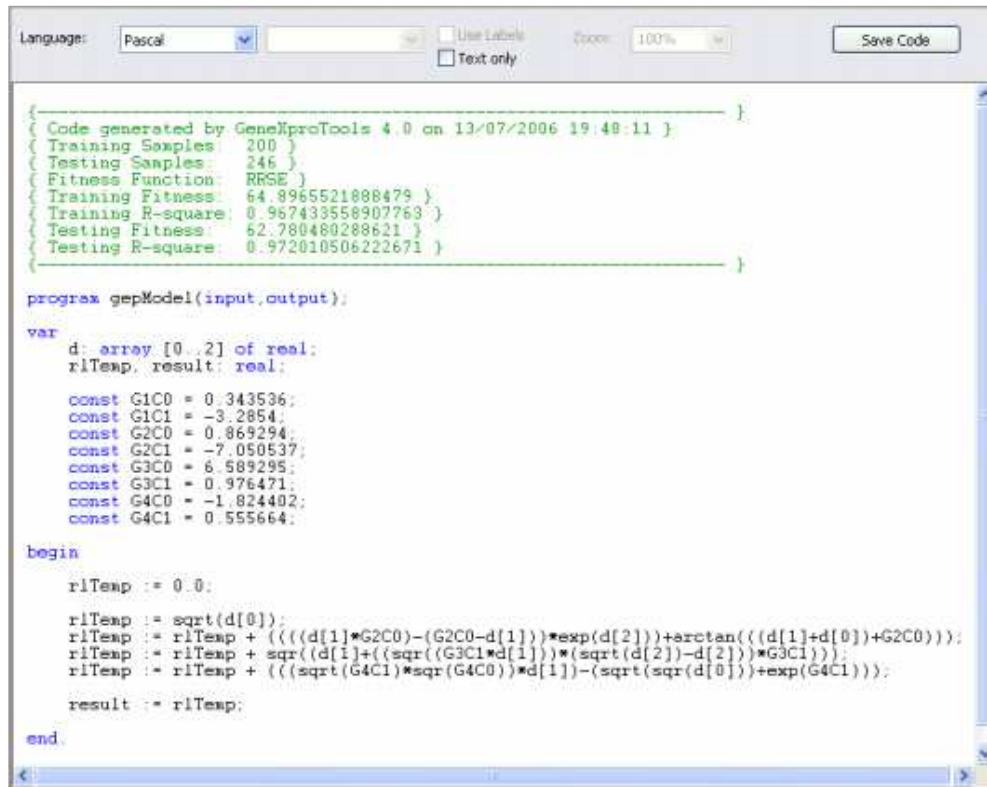
Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555654

```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Pascal. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **Pascal**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Pascal.



```

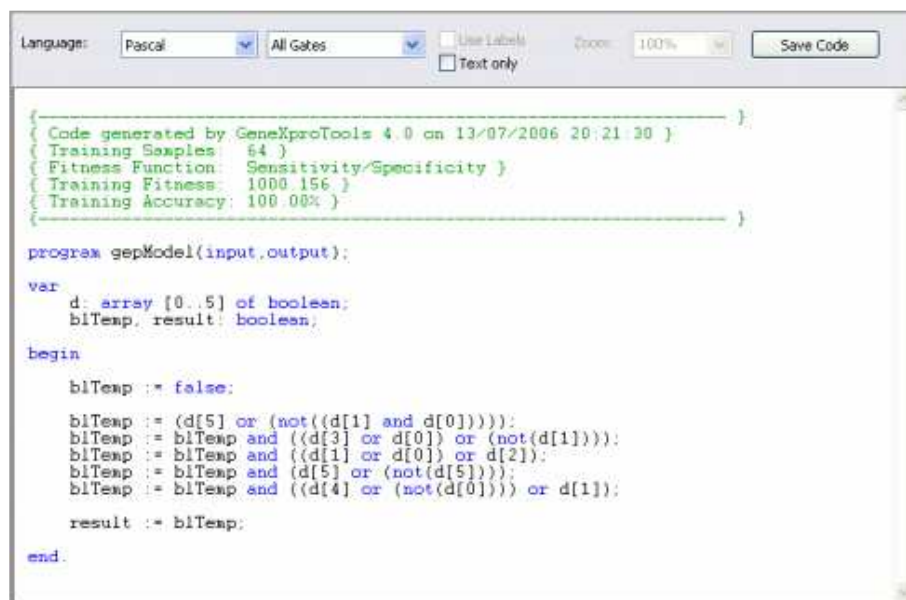
Language: Pascal
Use Labels
Zoom: 100%
Save Code

{
  Code generated by GeneXproTools 4.0 on 13/07/2006 19:48:11
  Training Samples: 200
  Testing Samples: 246
  Fitness Function: RMSE
  Training Fitness: 64.8965521888479
  Training R-square: 0.967433558907763
  Testing Fitness: 62.780480288621
  Testing R-square: 0.972010506222671
}

program gepModel(input,output);
var
  d: array [0..2] of real;
  riTemp, result: real;
const
  G1C0 = 0.343536;
  G1C1 = -3.2854;
  G2C0 = 0.869294;
  G2C1 = -7.050537;
  G3C0 = 6.589295;
  G3C1 = 0.976471;
  G4C0 = -1.824402;
  G4C1 = 0.555664;
begin
  riTemp := 0.0;
  riTemp := sqrt(d[0]);
  riTemp := riTemp + (((d[1]*G2C0)-(G2C0-d[1]))*exp(d[2]))+arctan(((d[1]+d[0])+G2C0));
  riTemp := riTemp + sqrt((d[1]+(sqrt(G3C1*d[1]))*(sqrt(d[2])-d[2]))*G3C1));
  riTemp := riTemp + (((sqrt(G4C1)*sqrt(G4C0))*d[1])-(sqrt(sqrt(d[0]))+exp(G4C1)));
  result := riTemp;
end.

```

And to translate your logic circuits into **Pascal**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Pascal and All Gates on the Language list boxes.



```

Language: Pascal
All Gates
Use Labels
Zoom: 100%
Save Code

{
  Code generated by GeneXproTools 4.0 on 13/07/2006 20:21:30
  Training Samples: 64
  Fitness Function: Sensitivity/Specificity
  Training Fitness: 1000.156
  Training Accuracy: 100.00%
}

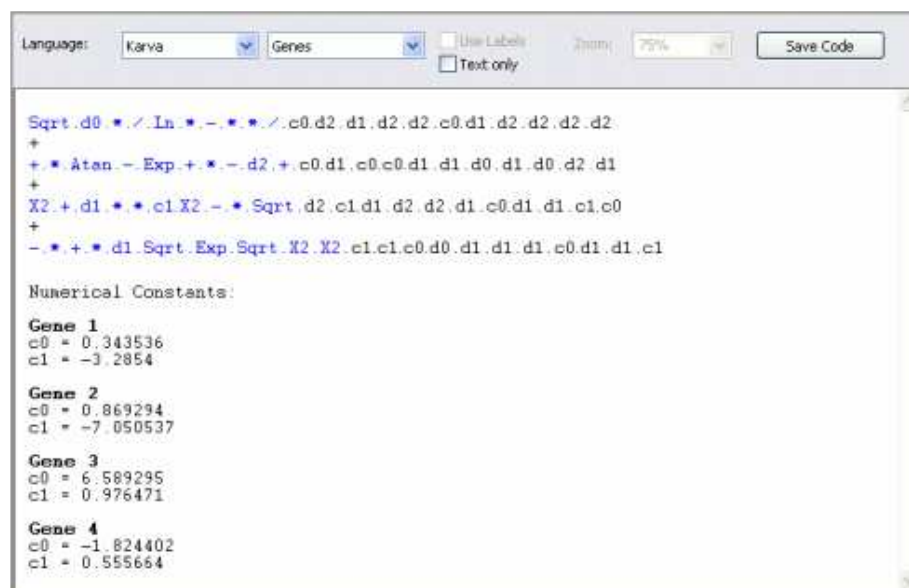
program gepModel(input,output);
var
  d: array [0..5] of boolean;
  biTemp, result: boolean;
begin
  biTemp := false;
  biTemp := (d[5] or (not((d[1] and d[0]))));
  biTemp := biTemp and ((d[3] or d[0]) or (not(d[1])));
  biTemp := biTemp and ((d[1] or d[0]) or d[2]);
  biTemp := biTemp and (d[5] or (not(d[5])));
  biTemp := biTemp and ((d[4] or (not(d[0])) or d[1]);
  result := biTemp;
end.

```

Generating Code Automatically

Perl

For mathematical models, GeneXproTools 4.0 uses the **Perl Math grammar** to generate **Perl code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Perl grammar**.



```

Sqrt.d0 * / In * - * * / .c0.d2 d1.d2.d2.c0.d1.d2.d2.d2.d2
+
+ * .Aten - Exp.+ * - .d2.+c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2.+d1.*.*.c1.X2.-.*.Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
-.*.+.*.d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

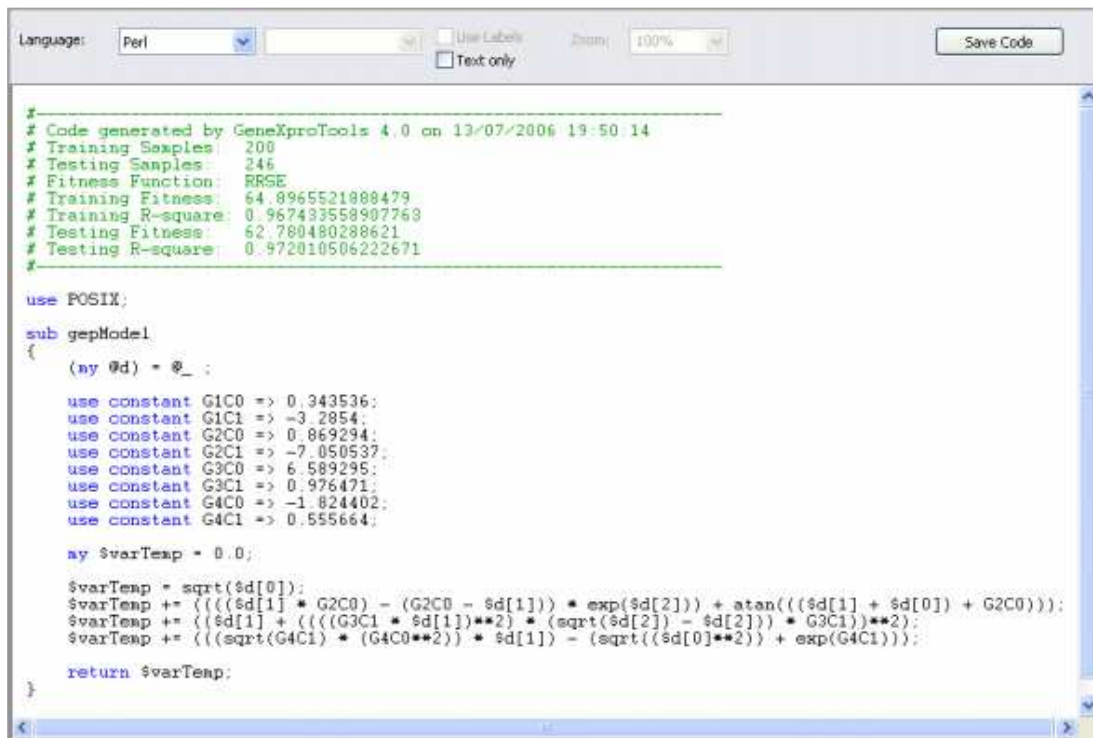
Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Perl. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **Perl**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Perl.



Language: Perl ☐ Use Labels Zoom: 100%

☐ Text only

```
#
# Code generated by GeneXproTools 4.0 on 13/07/2006 19:50:14
# Training Samples: 200
# Testing Samples: 246
# Fitness Function: RRGE
# Training Fitness: 64.8965521888479
# Training R-square: 0.967433558907769
# Testing Fitness: 62.780480288621
# Testing R-square: 0.972010506222671
#

use POSIX;
sub gepModel
{
    (my @d) = @_ ;

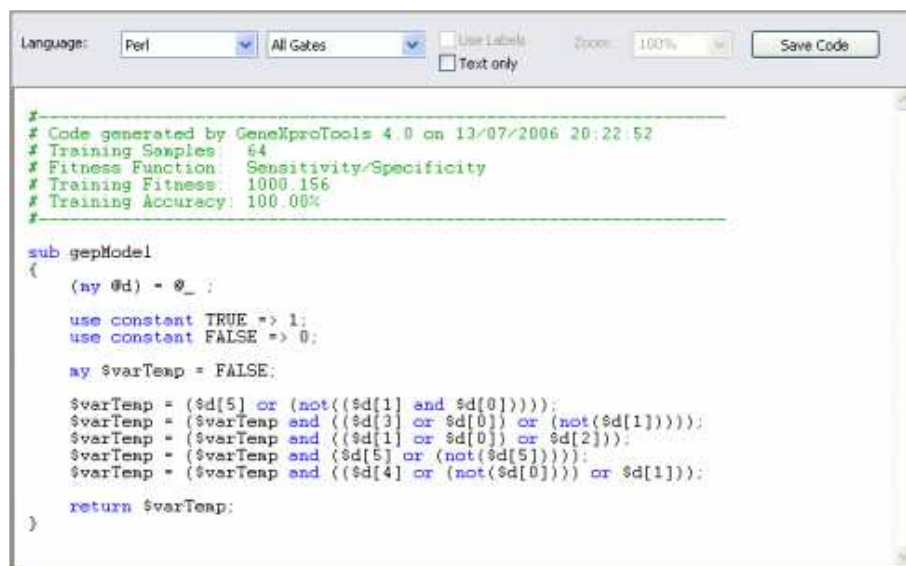
    use constant G1C0 => 0.343536;
    use constant G1C1 => -3.2854;
    use constant G2C0 => 0.869294;
    use constant G2C1 => -7.050537;
    use constant G3C0 => 6.589295;
    use constant G3C1 => 0.976471;
    use constant G4C0 => -1.824402;
    use constant G4C1 => 0.555664;

    my $varTemp = 0.0;

    $varTemp = sqrt($d[0]);
    $varTemp += (((($d[1] * G2C0) - (G2C0 - $d[1])) * exp($d[2])) + atan(($d[1] + $d[0]) + G2C0));
    $varTemp += (($d[1] + (((G3C1 * $d[1])**2) * (sqrt($d[2]) - $d[2])) * G3C1)**2);
    $varTemp += (((sqrt(G4C1) * (G4C0**2)) * $d[1]) - (sqrt(($d[0]**2) + exp(G4C1))));

    return $varTemp;
}
```

And to translate your logic circuits into **Perl**, on the GeneXproTools modeling environment, select the **Model Panel**, and then select both Perl and All Gates on the Language list boxes.



Language: Perl All Gates ☐ Use Labels Zoom: 100%

☐ Text only

```
#
# Code generated by GeneXproTools 4.0 on 13/07/2006 20:22:52
# Training Samples: 64
# Fitness Function: Sensitivity/Specificity
# Training Fitness: 1000.156
# Training Accuracy: 100.00%
#

sub gepModel
{
    (my @d) = @_ ;

    use constant TRUE => 1;
    use constant FALSE => 0;

    my $varTemp = FALSE;

    $varTemp = ($d[5] or (not(($d[1] and $d[0]))));
    $varTemp = ($varTemp and (($d[3] or $d[0]) or (not($d[1]))));
    $varTemp = ($varTemp and (($d[1] or $d[0]) or $d[2]));
    $varTemp = ($varTemp and ($d[5] or (not($d[5]))));
    $varTemp = ($varTemp and (($d[4] or (not($d[0])) or $d[1]));

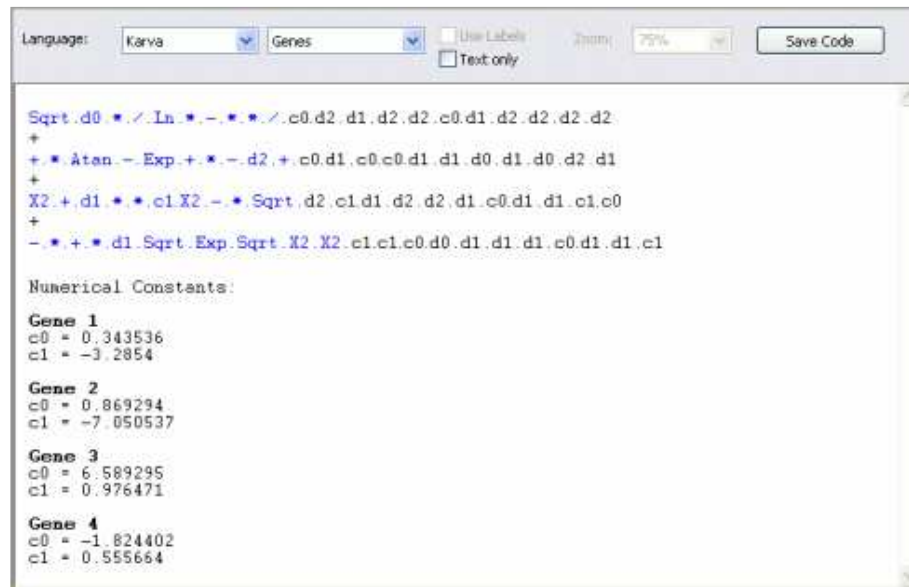
    return $varTemp;
}
```

◀ | ▶

Generating Code Automatically

PHP

For mathematical models, GeneXproTools 4.0 uses the **PHP Math grammar** to generate **PHP code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates PHP grammar**.



The screenshot shows the GeneXproTools 4.0 interface with the 'Language' dropdown set to 'Karva'. The main text area contains Karva code for a mathematical model. Below the code, the 'Numerical Constants' section lists values for four genes.

```

Sqrt.d0 * / In * - * * / .c0.d2 .d1.d2.d2.c0.d1.d2.d2.d2
+
+ * .Atan - Exp + * - .d2 + .c0.d1.c0.c0.d1 .d1.d0.d1.d0.d2.d1
+
X2 + .d1 * * .c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * .d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

```

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

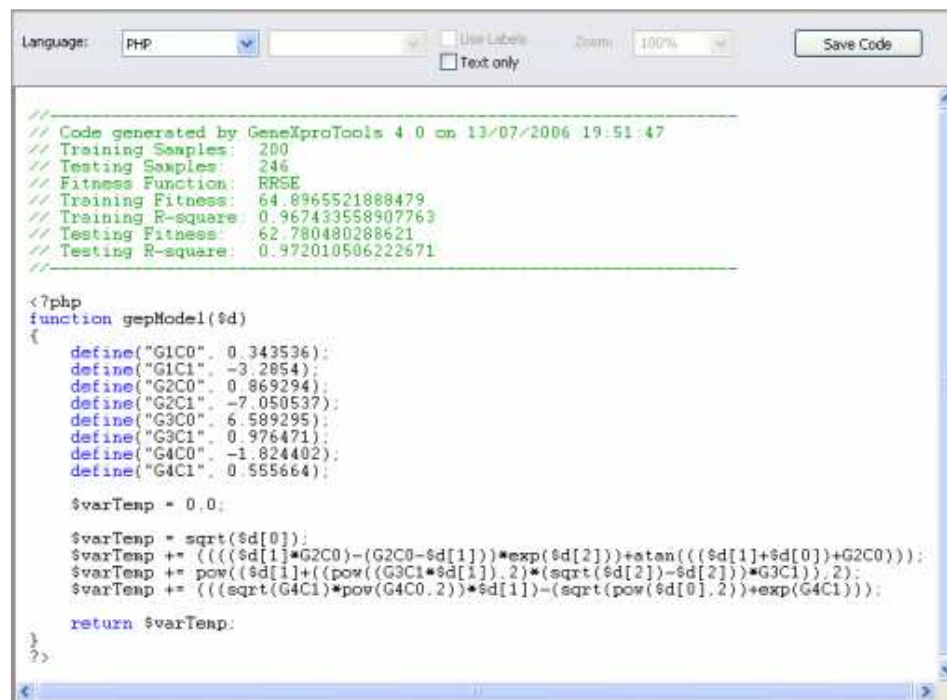
Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into PHP. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **PHP**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select PHP.



The screenshot shows the GeneXproTools 4.0 interface with the 'Language' dropdown set to 'PHP'. The main text area displays PHP code generated from a Karva model. The code includes training and testing statistics, and a function definition for 'gepModel'.

```

// Code generated by GeneXproTools 4.0 on 13/07/2006 19:51:47
// Training Samples: 200
// Testing Samples: 246
// Fitness Function: RRSE
// Training Fitness: 64.8965521888479
// Training R-square: 0.967433558907763
// Testing Fitness: 62.780480288621
// Testing R-square: 0.972010506222671

<?php
function gepModel($d)
{
    define("G1C0", 0.343536);
    define("G1C1", -3.2854);
    define("G2C0", 0.869294);
    define("G2C1", -7.050537);
    define("G3C0", 6.589295);
    define("G3C1", 0.976471);
    define("G4C0", -1.824402);
    define("G4C1", 0.555664);

    $varTemp = 0.0;

    $varTemp = sqrt($d[0]);
    $varTemp += (((($d[1]*G2C0)-(G2C0-$d[1]))*exp($d[2]))+atan(($d[1]+$d[0])+G2C0));
    $varTemp += pow(($d[1]+(pow((G3C1*$d[1]).2)*(sqrt($d[2])-$d[2]))*G3C1)).2);
    $varTemp += (((sqrt(G4C1)*pow(G4C0.2))*$d[1])-(sqrt(pow($d[0].2)+exp(G4C1))));

    return $varTemp;
}

```

And to translate your logic circuits into **PHP**, on the GeneXproTools modeling environment, select the

[Model Panel](#), and then select both PHP and All Gates on the Language list boxes.



```
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:24:02
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.15%
// Training Accuracy: 100.00%

<?php
function gepModel($d)
{
    $varTemp = false;

    $varTemp = ($d[5] or (((($d[1] and $d[0]))));
    $varTemp = ($varTemp and (($d[3] or $d[0]) or (($d[1]))));
    $varTemp = ($varTemp and (($d[1] or $d[0]) or $d[2]));
    $varTemp = ($varTemp and ($d[5] or (($d[5]))));
    $varTemp = ($varTemp and (($d[4] or (($d[0])) or $d[1])));

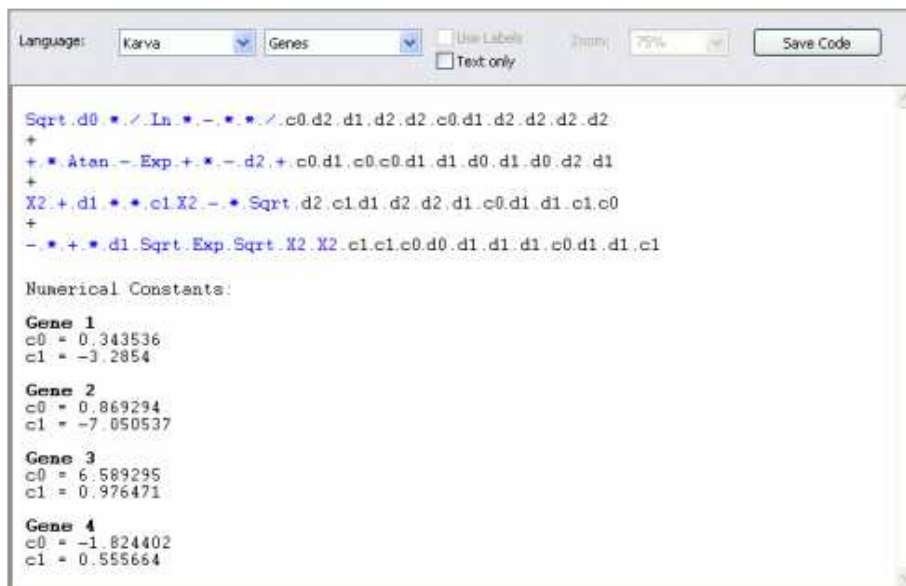
    return $varTemp;
}
?>
```

◀ | ▶

Generating Code Automatically

Python

For mathematical models, GeneXproTools 4.0 uses the **Python Math grammar** to generate **Python code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Python grammar**.



```
Sqrt.d0 * ./Ln.*.-.*./c0.d2.d1.d2.d2.c0.d1.d2.d2.d2.d2
+
+ * Atan - Exp.+*.-.d2.+c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2.+d1.*.c1.X2.-.*.Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
-.*.+.*.d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

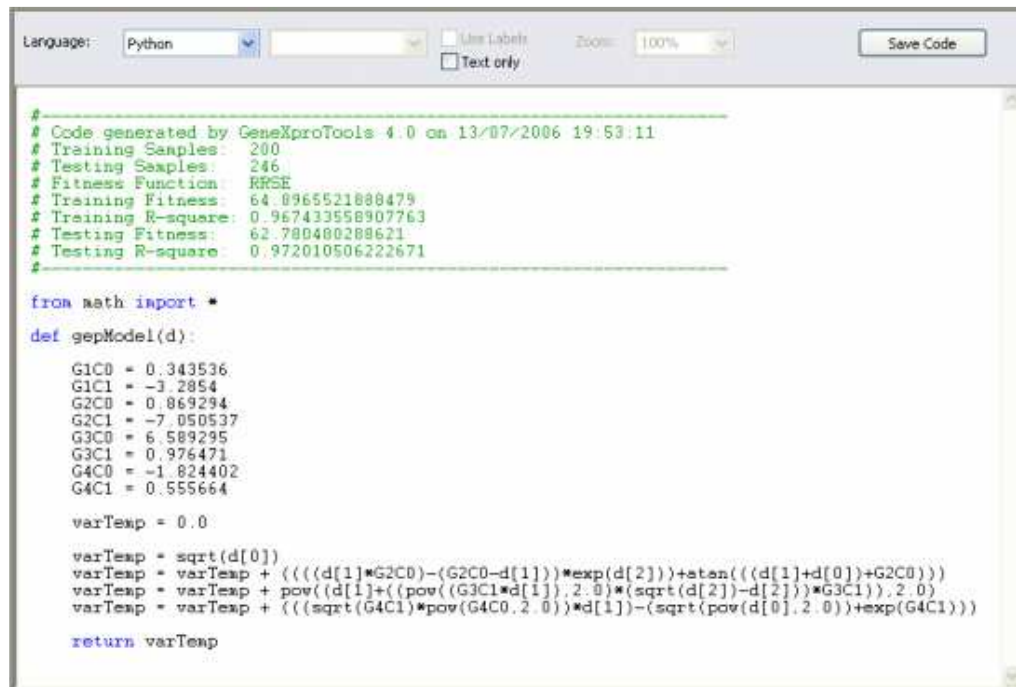
Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664
```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Python. The code can then be copied to the clipboard or saved to disk. This

code can be freely distributed in source or binary form.

To translate your mathematical models into **Python**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Python.



The screenshot shows the GeneXproTools modeling environment interface. The 'Language' dropdown is set to 'Python'. The code area contains the following text:

```
#
# Code generated by GeneXproTools 4.0 on 13/07/2006 19:53:11
# Training Samples: 200
# Testing Samples: 246
# Fitness Function: RRSE
# Training Fitness: 64.0965521888479
# Training R-square: 0.967433558907763
# Testing Fitness: 62.780480288621
# Testing R-square: 0.972010506222671
#

from math import *
def gepModel(d):
    G1C0 = 0.343536
    G1C1 = -3.2854
    G2C0 = 0.869294
    G2C1 = -7.050537
    G3C0 = 6.589295
    G3C1 = 0.976471
    G4C0 = -1.824402
    G4C1 = 0.555664

    varTemp = 0.0
    varTemp = sqrt(d[0])
    varTemp = varTemp + (((d[1]*G2C0)-(G2C0-d[1]))*exp(d[2]))+atan(((d[1]+d[0])+G2C0))
    varTemp = varTemp + pow((d[1]+(pow((G3C1*d[1]),2.0)*(sqrt(d[2])-d[2]))*G3C1)),2.0)
    varTemp = varTemp + (((sqrt(G4C1)*pow(G4C0,2.0))*d[1])-(sqrt(pow(d[0],2.0))+exp(G4C1)))

    return varTemp
```

And to translate your logic circuits into **Python**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Python and All Gates on the Language list boxes.



The screenshot shows the GeneXproTools modeling environment interface. The 'Language' dropdown is set to 'Python' and the 'All Gates' dropdown is also selected. The code area contains the following text:

```
#
# Code generated by GeneXproTools 4.0 on 13/07/2006 20:25:16
# Training Samples: 64
# Fitness Function: Sensitivity/Specificity
# Training Fitness: 1000.156
# Training Accuracy: 100.00%
#

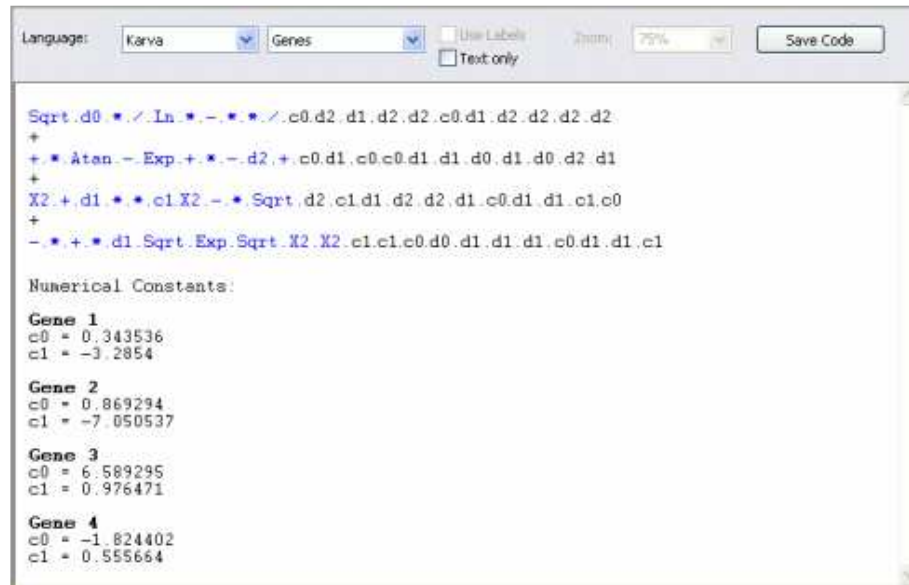
def gepModel(d):
    biTemp = False
    biTemp = (d[5] or (not((d[1] and d[0]))))
    biTemp = biTemp and ((d[3] or d[0]) or (not(d[1])))
    biTemp = biTemp and ((d[1] or d[0]) or d[2])
    biTemp = biTemp and (d[5] or (not(d[5])))
    biTemp = biTemp and ((d[4] or (not(d[0])) or d[1])

    return biTemp
```

Generating Code Automatically

Visual Basic

For mathematical models, GeneXproTools 4.0 uses the **Visual Basic Math grammar** to generate **Visual Basic code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates Visual Basic grammar**.



```

Language: Karva Genes Use Labels Zoom: 75% Save Code
Text only

Sqrt.d0 * ./ In * - * ./ c0.d2 d1.d2.d2.c0.d1.d2.d2.d2.d2
+
+ * Atan - Exp + * - d2 + c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + d1 * * c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

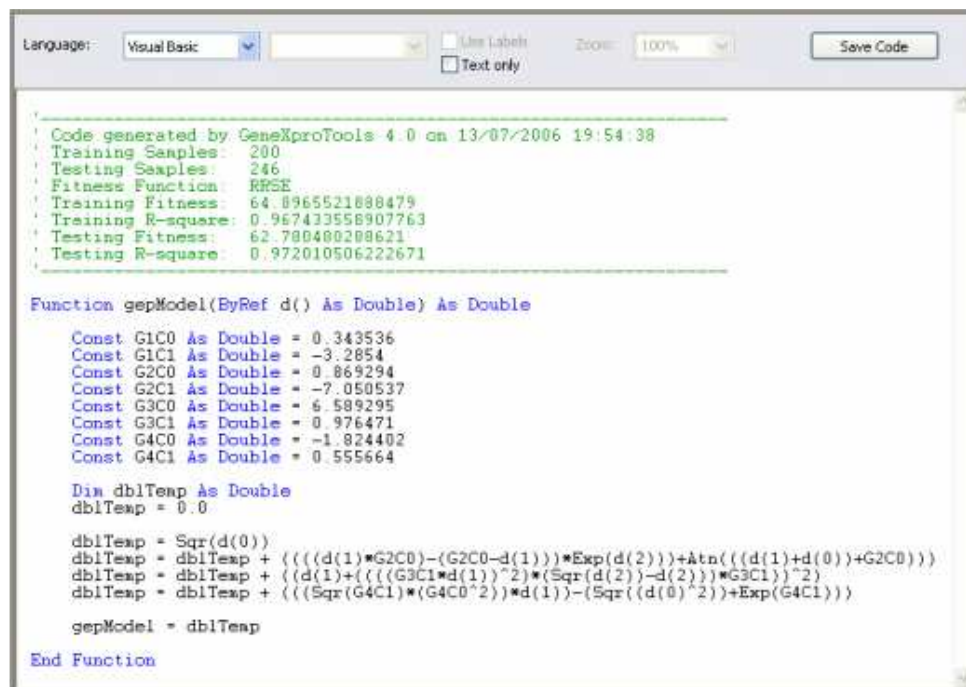
Gene 3
c0 = 6.589295
c1 = 0.976471

Gene 4
c0 = -1.824402
c1 = 0.555664

```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Visual Basic. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **Visual Basic**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select Visual Basic.



```

Language: Visual Basic Use Labels Zoom: 100% Save Code
Text only

Code generated by GeneXproTools 4.0 on 13/07/2006 19:54:38
Training Samples: 200
Testing Samples: 246
Fitness Function: RMSE
Training Fitness: 64.8965521888479
Training R-square: 0.967433558907763
Testing Fitness: 62.780480288621
Testing R-square: 0.972010506222671

Function gepModel(ByRef d() As Double) As Double

Const G1C0 As Double = 0.343536
Const G1C1 As Double = -3.2854
Const G2C0 As Double = 0.869294
Const G2C1 As Double = -7.050537
Const G3C0 As Double = 6.589295
Const G3C1 As Double = 0.976471
Const G4C0 As Double = -1.824402
Const G4C1 As Double = 0.555664

Dim dblTemp As Double
dblTemp = 0.0

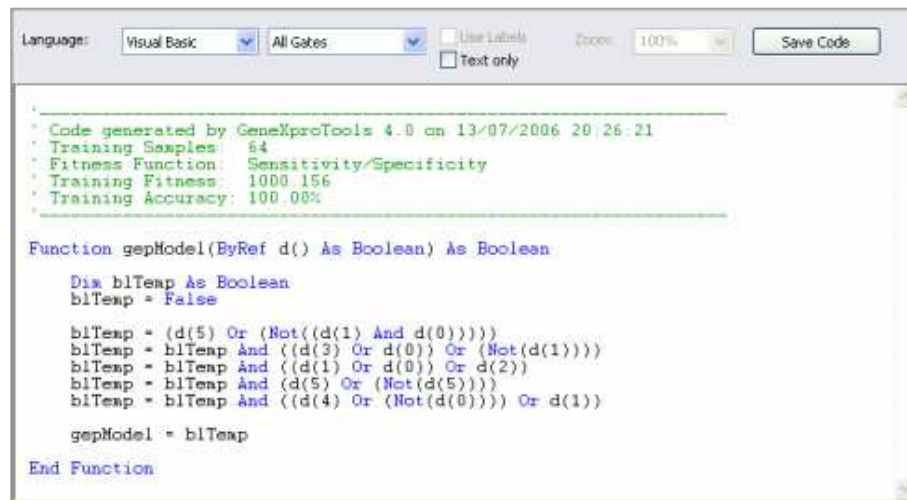
dblTemp = Sqr(d(0))
dblTemp = dblTemp + (((d(1)*G2C0)-(G2C0-d(1)))*Exp(d(2)))+Atn((((d(1)+d(0))+G2C0)))
dblTemp = dblTemp + ((d(1)+(((G3C1*d(1))^2)*(Sqr(d(2))-d(2)))*G3C1))^2
dblTemp = dblTemp + (((Sqr(G4C1)*(G4C0^2))*d(1))-(Sqr((d(0)^2))+Exp(G4C1)))

gepModel = dblTemp

End Function

```

And to translate your logic circuits into **Visual Basic**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Visual Basic and All Gates on the Language list boxes.



```

Language: Visual Basic All Gates Use Labels Zoom: 100% Save Code
Test only

Code generated by GeneXproTools 4.0 on 13/07/2006 20:26:21
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

Function gepModel(ByRef d() As Boolean) As Boolean
    Dim biTemp As Boolean
    biTemp = False

    biTemp = (d(5) Or (Not((d(1) And d(0))))))
    biTemp = biTemp And ((d(3) Or d(0)) Or (Not(d(1))))
    biTemp = biTemp And ((d(1) Or d(0)) Or d(2))
    biTemp = biTemp And (d(5) Or (Not(d(5))))
    biTemp = biTemp And ((d(4) Or (Not(d(0)))) Or d(1))

    gepModel = biTemp
End Function

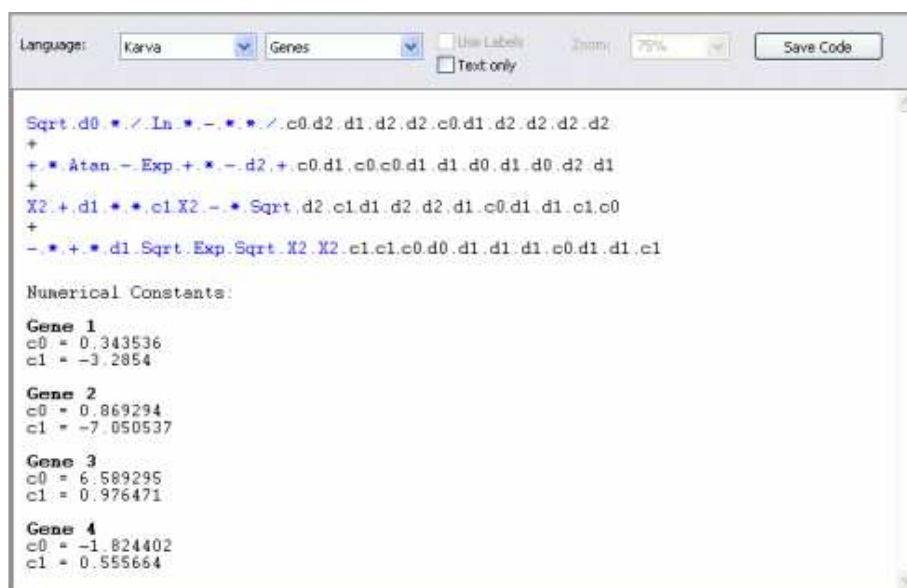
```



Generating Code Automatically

VB.Net

For mathematical models, GeneXproTools 4.0 uses the **VB.Net Math grammar** to generate **VB.Net code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates VB.Net grammar**.



```

Language: Karva Genes Use Labels Zoom: 75% Save Code
Test only

Sqrt.d0 * ./Ln * - * ./c0.d2.d1.d2.d2.c0.d1.d2.d2.d2
+
+ * Atan - Exp + * - .d2 + c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2 + d1 * .c1.X2 - * Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * + * d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:

Gene 1
c0 = 0.343536
c1 = -3.2854

Gene 2
c0 = 0.869294
c1 = -7.050537

Gene 3
c0 = 6.589295
c1 = 0.976471

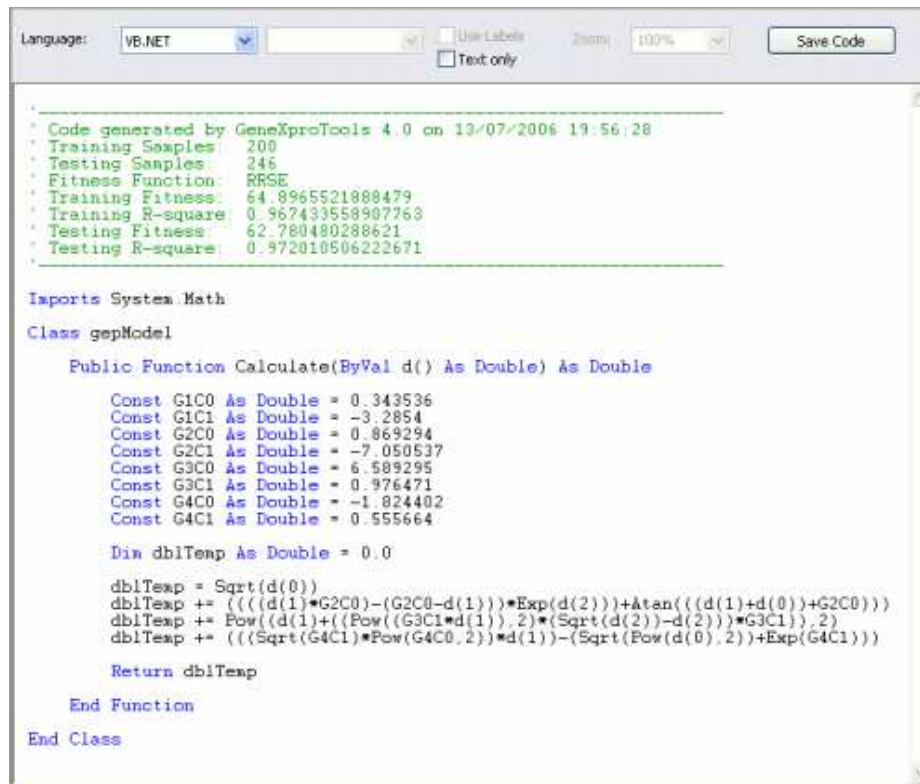
Gene 4
c0 = -1.824402
c1 = 0.555664

```

You can translate not only all the evolved models but also exogenous models introduced through the

[Change Seed window](#) into VB.Net. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **VB.Net**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select VB.Net.



The screenshot shows the GeneXproTools interface with the Language dropdown set to VB.NET. The code area displays the following content:

```

Code generated by GeneXproTools 4.0 on 13/07/2006 19:56:28
Training Samples: 200
Testing Samples: 246
Fitness Function: RRSE
Training Fitness: 64.8965521888479
Training R-square: 0.967433558907763
Testing Fitness: 62.780480288621
Testing R-square: 0.972010506222671

Imports System.Math
Class gepModel
    Public Function Calculate(ByVal d() As Double) As Double
        Const G1C0 As Double = 0.343536
        Const G1C1 As Double = -3.2854
        Const G2C0 As Double = 0.869294
        Const G2C1 As Double = -7.050537
        Const G3C0 As Double = 6.589295
        Const G3C1 As Double = 0.976471
        Const G4C0 As Double = -1.824402
        Const G4C1 As Double = 0.555664

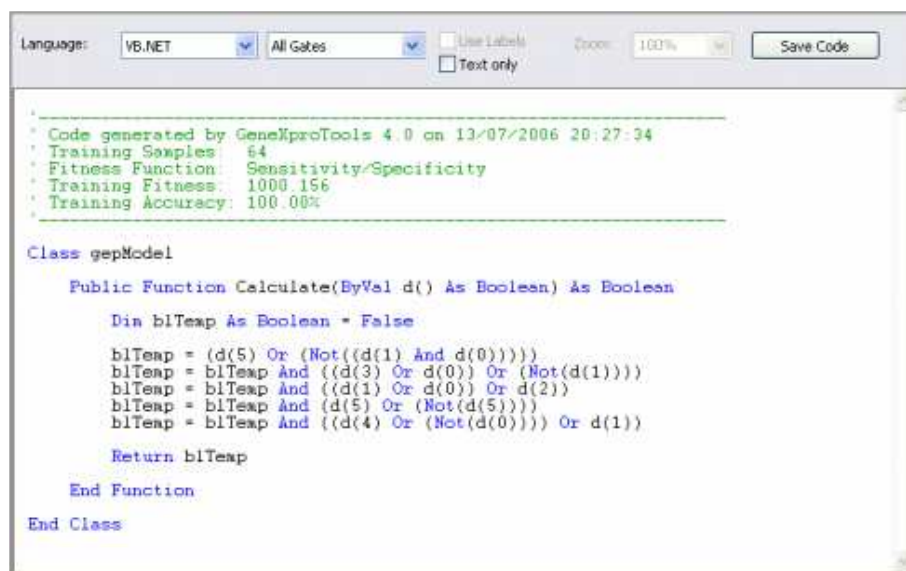
        Dim dblTemp As Double = 0.0

        dblTemp = Sqrt(d(0))
        dblTemp += (((d(1)*G2C0)-(G2C0-d(1)))*Exp(d(2)))+Atan(((d(1)+d(0))+G2C0)))
        dblTemp += Pow((d(1)+(Pow((G3C1*d(1)), 2)*(Sqrt(d(2))-d(2))*G3C1), 2)
        dblTemp += (((Sqrt(G4C1)*Pow(G4C0, 2))*d(1))-(Sqrt(Pow(d(0), 2))+Exp(G4C1)))

        Return dblTemp
    End Function
End Class

```

And to translate your logic circuits into **VB.Net**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both VB.Net and All Gates on the Language list boxes.



The screenshot shows the GeneXproTools interface with the Language dropdown set to VB.NET and the All Gates dropdown selected. The code area displays the following content:

```

Code generated by GeneXproTools 4.0 on 13/07/2006 20:27:34
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

Class gepModel
    Public Function Calculate(ByVal d() As Boolean) As Boolean
        Dim blTemp As Boolean = False

        blTemp = (d(5) Or (Not((d(1) And d(0))))))
        blTemp = blTemp And ((d(3) Or d(0)) Or (Not(d(1))))
        blTemp = blTemp And ((d(1) Or d(0)) Or d(2))
        blTemp = blTemp And (d(5) Or (Not(d(5))))
        blTemp = blTemp And ((d(4) Or (Not(d(0)))) Or d(1))

        Return blTemp
    End Function
End Class

```


Generating Code Automatically

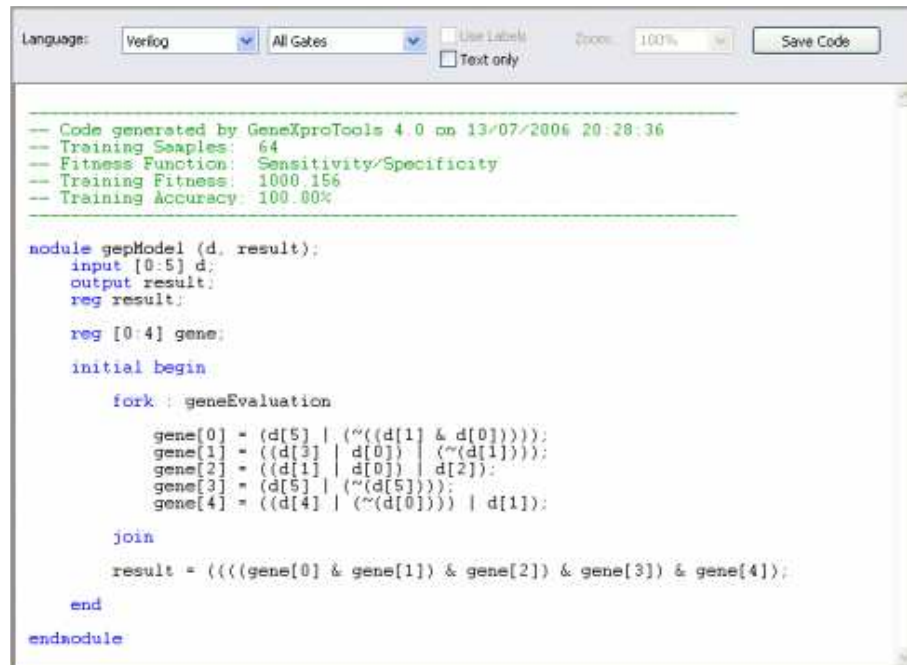
Verilog

The **Verilog grammar** is only available for Logic Synthesis problems and GeneXproTools 4.0 uses the basic **Boolean All Gates Verilog grammar** to generate **Verilog code** from its native [Karva code](#).



You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into Verilog. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

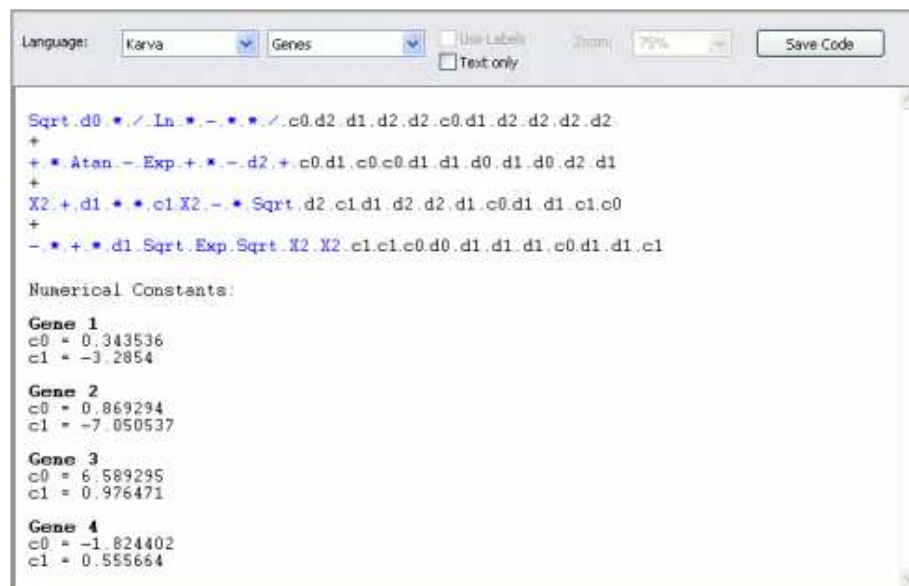
To translate your logic circuits into **Verilog**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Verilog and All Gates on the Language list boxes.



Generating Code Automatically

VHDL

For mathematical models, GeneXproTools 4.0 uses the **VHDL Math grammar** to generate **VHDL code** from its native [Karva code](#), whereas for Logic Synthesis problems, GeneXproTools 4.0 uses the basic **Boolean All Gates VHDL grammar**.



```

Language: Karva Genes Use Labels Zoom: 75% Save Code
Text only

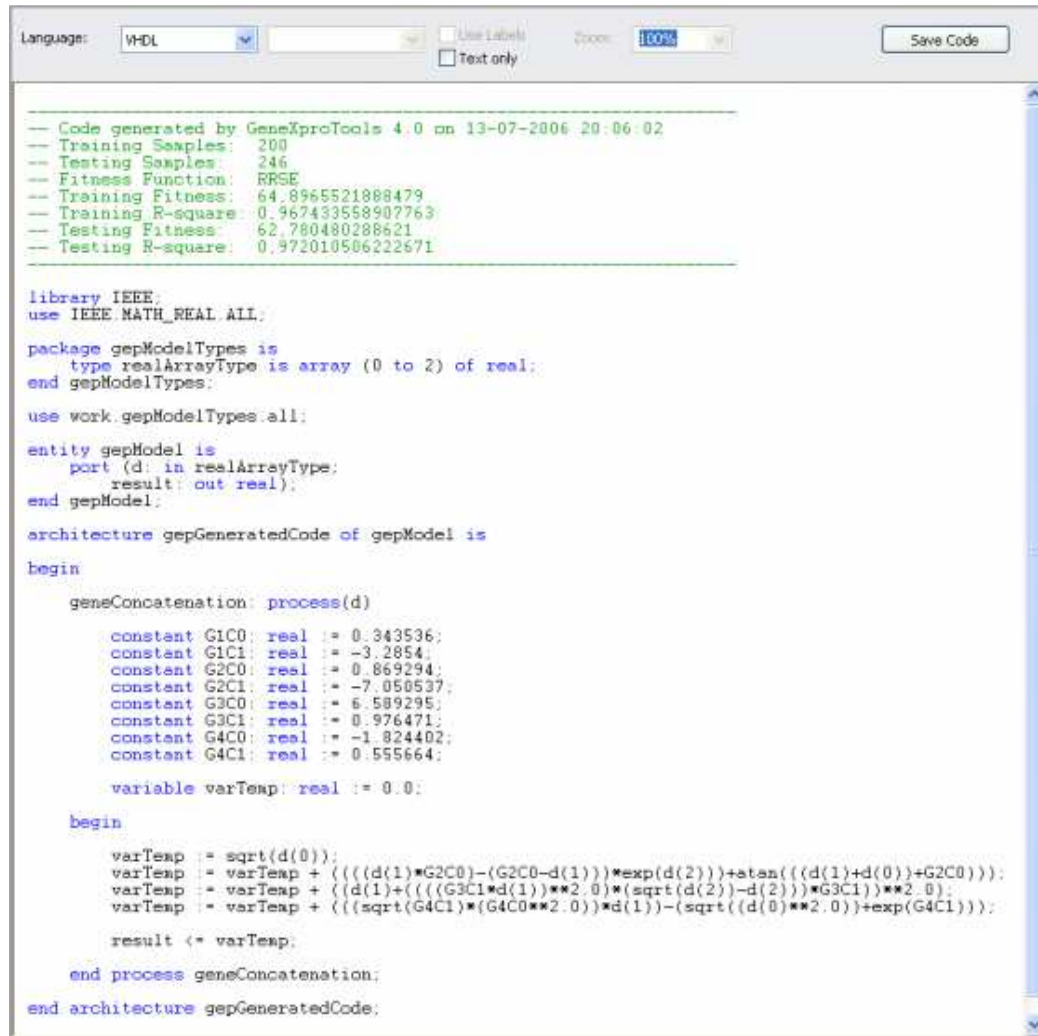
Sqrt.d0 * / In * - * * / .c0.d2 .d1.d2.d2.c0.d1.d2.d2.d2.d2
+
+ * * .Aten - Exp. + * - .d2. + .c0.d1.c0.c0.d1.d1.d0.d1.d0.d2.d1
+
X2. + .d1 * * .c1.X2 - * .Sqrt.d2.c1.d1.d2.d2.d1.c0.d1.d1.c1.c0
+
- * * + .d1.Sqrt.Exp.Sqrt.X2.X2.c1.c1.c0.d0.d1.d1.d1.c0.d1.d1.c1

Numerical Constants:
Gene 1
c0 = 0.343536
c1 = -3.2854
Gene 2
c0 = 0.869294
c1 = -7.050537
Gene 3
c0 = 6.589295
c1 = 0.976471
Gene 4
c0 = -1.824402
c1 = 0.555664

```

You can translate not only all the evolved models but also exogenous models introduced through the [Change Seed window](#) into VHDL. The code can then be copied to the clipboard or saved to disk. This code can be freely distributed in source or binary form.

To translate your mathematical models into **VHDL**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then, on the Language list box, select VHDL.



```

-- Code generated by GeneXproTools 4.0 on 13-07-2006 20:06:02
-- Training Samples: 200
-- Testing Samples: 246
-- Fitness Function: RRSE
-- Training Fitness: 64.8965521888479
-- Training R-square: 0.967433558907763
-- Testing Fitness: 62.780480288621
-- Testing R-square: 0.972010506222671

library IEEE;
use IEEE.MATH_REAL.ALL;

package gepModelTypes is
  type realArrayType is array (0 to 2) of real;
end gepModelTypes;

use work.gepModelTypes.all;

entity gepModel is
  port (d: in realArrayType;
        result: out real);
end gepModel;

architecture gepGeneratedCode of gepModel is
begin
  geneConcatenation: process(d)
    constant G1C0: real := 0.343536;
    constant G1C1: real := -3.2854;
    constant G2C0: real := 0.869294;
    constant G2C1: real := -7.050537;
    constant G3C0: real := 6.589295;
    constant G3C1: real := 0.976471;
    constant G4C0: real := -1.824402;
    constant G4C1: real := 0.555664;

    variable varTemp: real := 0.0;

  begin
    varTemp := sqrt(d(0));
    varTemp := varTemp + (((d(1)*G2C0)-(G2C0-d(1)))*exp(d(2))+atan(((d(1)+d(0))+G2C0)));
    varTemp := varTemp + ((d(1)+(((G3C1*d(1))**2.0)*(sqrt(d(2))-d(2))*G3C1))**2.0);
    varTemp := varTemp + (((sqrt(G4C1)*(G4C0**2.0))*d(1))-(sqrt((d(0)**2.0)+exp(G4C1))));

    result <= varTemp;
  end process geneConcatenation;
end architecture gepGeneratedCode;

```

And to translate your logic circuits into **VHDL**, on the GeneXproTools modeling environment, select the **Model Panel**, and then select both VHDL and All Gates on the Language list boxes.

```

-- Code generated by GeneXproTools 4.0 on 13/07/2006 20:29:59
-- Training Samples: 64
-- Fitness Function: Sensitivity/Specificity
-- Training Fitness: 1000.15%
-- Training Accuracy: 100.00%

library IEEE;
use IEEE.std_logic_1164.all;

entity gepModel is
  port (d: in bit_vector (0 to 5);
        result: out bit);
end gepModel;

architecture gepGeneratedCode of gepModel is
  signal gene: bit_vector (0 to 4);
begin

  gene(0) <= (d(5) or (not(d(1) and d(0)))));
  gene(1) <= ((d(3) or d(0)) or (not(d(1)))));
  gene(2) <= ((d(1) or d(0)) or d(2));
  gene(3) <= (d(5) or (not(d(5)))));
  gene(4) <= ((d(4) or (not(d(0)))) or d(1));

  result <= (((gene(0) and gene(1)) and gene(2)) and gene(3)) and gene(4));
end architecture gepGeneratedCode;

```



Generating Code Automatically

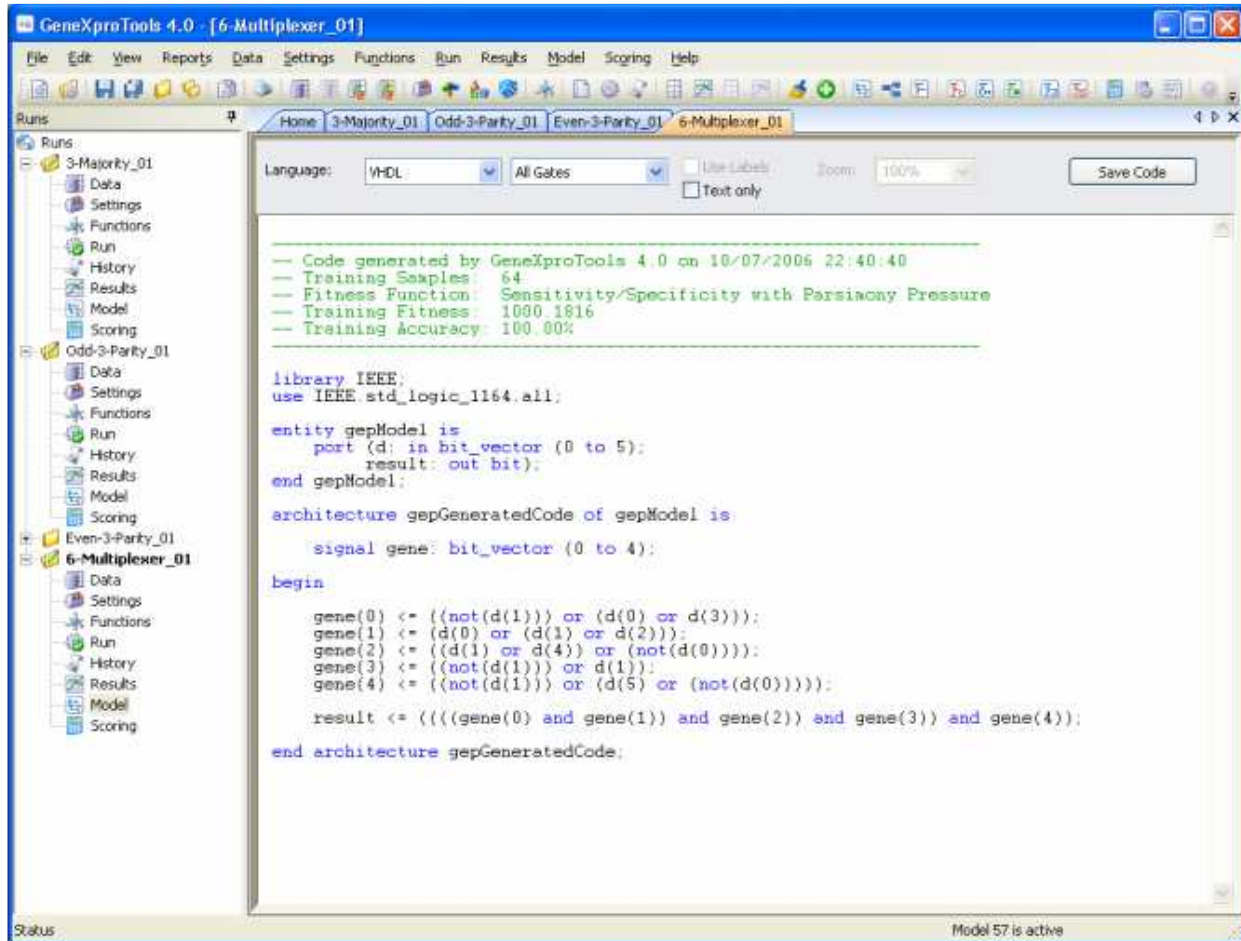
Built-in Logical Systems for Automatic Code Generation

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 [built-in logical functions](#), which are the logical functions at the disposal of GeneXproTools learning algorithms and also part of the basic [All Gates Boolean grammar](#) of all [built-in grammars](#) (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL). But GeneXproTools also allows you to redesign immediately the generated circuits using five additional **logical systems** (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System). Therefore, for all of them, a corresponding built-in grammar exists in all the 16 programming languages GeneXproTools 4.0 supports:

- [Ada](#)
- [C](#)
- [C++](#)
- [C#](#)
- [Fortran](#)
- [Java](#)
- [Java Script](#)
- [Matlab](#)
- [Pascal](#)
- [Perl](#)
- [PHP](#)
- [Python](#)
- [Visual Basic](#)
- [VB.Net](#)

- Verilog
- VHDL

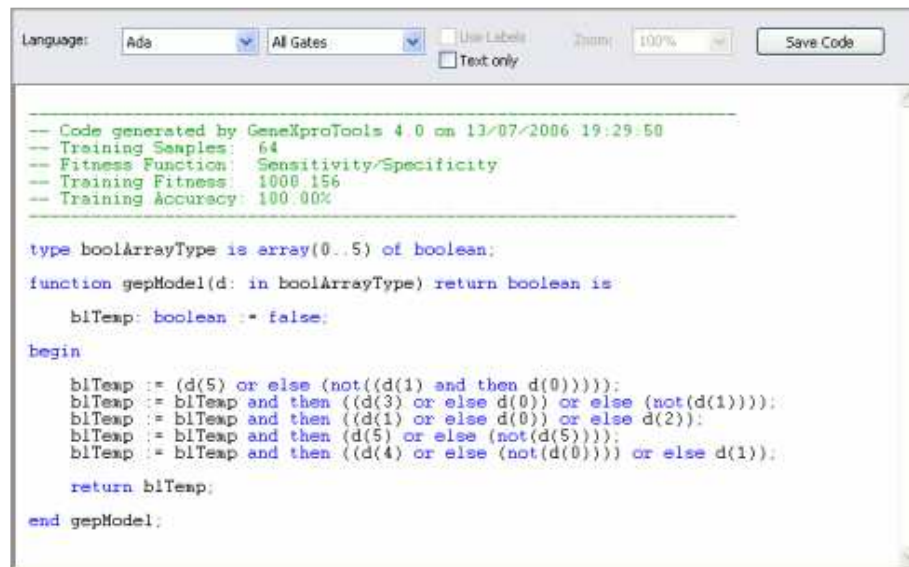
Other grammars can be easily designed using this wide set of [built-in grammars](#) as reference, as all of them can be accessed and modified by the user. As an example, the Reed-Muller System VHDL grammar of GeneXproTools is shown [here](#).



Generating Code Automatically

Ada

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Ada grammar](#) you can generate immediately **Ada code** using all the 258 built-in gates.



Language: Ada All Gates Use Labels Zoom: 100% Save Code

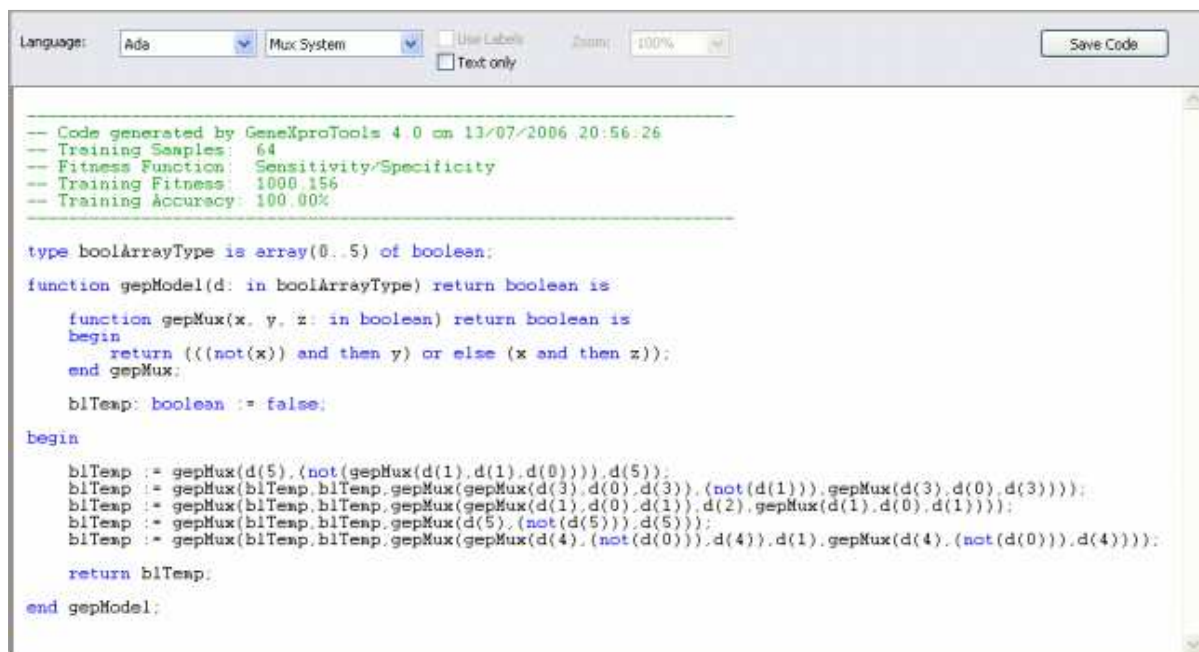
Code generated by GeneXproTools 4.0 on 13/07/2006 19:29:50
 Training Samples: 64
 Fitness Function: Sensitivity/Specificity
 Training Fitness: 1000.156
 Training Accuracy: 100.00%

```

type boolArrayType is array(0..5) of boolean;
function gepModel(d: in boolArrayType) return boolean is
  biTemp: boolean := false;
begin
  biTemp := (d(5) or else (not((d(1) and then d(0)))));
  biTemp := biTemp and then ((d(3) or else d(0)) or else (not(d(1))));
  biTemp := biTemp and then ((d(1) or else d(0)) or else d(2));
  biTemp := biTemp and then (d(5) or else (not(d(5))));
  biTemp := biTemp and then ((d(4) or else (not(d(0)))) or else d(1));
  return biTemp;
end gepModel;
  
```

But you can also redesign immediately the generated **Ada** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Ada**, on the GeneXproTools modeling environment, go to the [Model Panel](#), and then select both Ada and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



Language: Ada Mux System Use Labels Zoom: 100% Save Code

Code generated by GeneXproTools 4.0 on 13/07/2006 20:56:26
 Training Samples: 64
 Fitness Function: Sensitivity/Specificity
 Training Fitness: 1000.156
 Training Accuracy: 100.00%

```

type boolArrayType is array(0..5) of boolean;
function gepModel(d: in boolArrayType) return boolean is
  function gepMux(x, y, z: in boolean) return boolean is
    begin
      return (((not(x)) and then y) or else (x and then z));
    end gepMux;
  biTemp: boolean := false;
begin
  biTemp := gepMux(d(5), (not(gepMux(d(1), d(1), d(0))))), d(5));
  biTemp := gepMux(biTemp, biTemp, gepMux(gepMux(d(3), d(0), d(3)), (not(d(1))), gepMux(d(3), d(0), d(3))));
  biTemp := gepMux(biTemp, biTemp, gepMux(gepMux(d(1), d(0), d(1)), d(2), gepMux(d(1), d(0), d(1))));
  biTemp := gepMux(biTemp, biTemp, gepMux(d(5), (not(d(5))), d(5)));
  biTemp := gepMux(biTemp, biTemp, gepMux(gepMux(d(4), (not(d(0))), d(4)), d(1), gepMux(d(4), (not(d(0))), d(4))));
  return biTemp;
end gepModel;
  
```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Ada grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.

Generating Code Automatically

C

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates C grammar](#) you can generate immediately **C code** using all the 258 built-in gates.

```
//-----
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:11:13
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%
//-----

int gepModel(int d[]);
int gepModel(int d[])
{
    #define TRUE 1
    #define FALSE 0

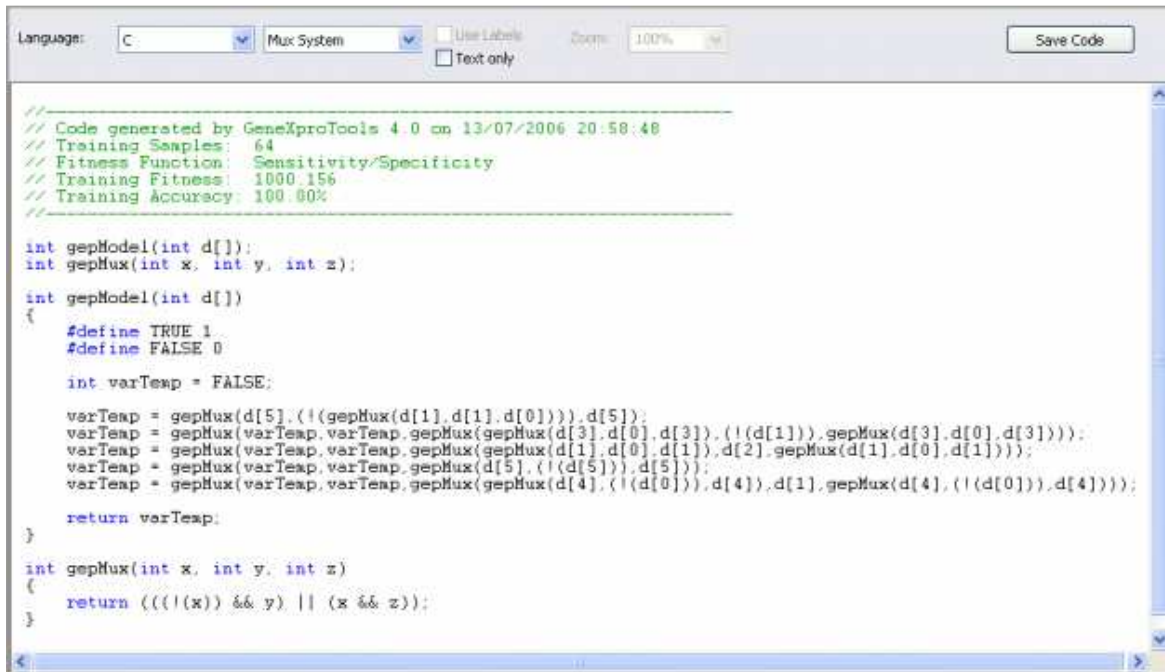
    int varTemp = FALSE;

    varTemp = (d[5] || (!((d[1] && d[0]))));
    varTemp = varTemp && ((d[3] || d[0]) || (!d[1]));
    varTemp = varTemp && ((d[1] || d[0]) || d[2]);
    varTemp = varTemp && (d[5] | (!d[5]));
    varTemp = varTemp && ((d[4] || (!d[0])) || d[1]);

    return varTemp;
}
```

But you can also redesign immediately the generated **C** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **C**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both C and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: C Mux System ☐ Use Labels ☐ Text only Score: 100% Save Code

// Code generated by GeneXproTools 4.0 on 13/07/2006 20:58:48
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%

int gepModel(int d[]);
int gepMux(int x, int y, int z);

int gepModel(int d[])
{
    #define TRUE 1
    #define FALSE 0

    int varTemp = FALSE;

    varTemp = gepMux(d[5], (!gepMux(d[1], d[1], d[0])) , d[5]);
    varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d[3], d[0], d[3]), (!d[1]), gepMux(d[3], d[0], d[3])));
    varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d[1], d[0], d[1]), d[2], gepMux(d[1], d[0], d[1])));
    varTemp = gepMux(varTemp, varTemp, gepMux(d[5], (!d[5]), d[5]));
    varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d[4], (!d[0]), d[4]), d[1], gepMux(d[4], (!d[0]), d[4])));

    return varTemp;
}

int gepMux(int x, int y, int z)
{
    return (((!(x)) && y) || (x && z));
}

```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates C grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.

◀ | ▶

Generating Code Automatically

C++

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates C++ grammar](#) you can generate immediately **C++ code** using all the 258 built-in gates.



```

Language: C++ All Gates Use Labels Zoom: 100% Save Code
Text only

// Code generated by GeneXproTools 4.0 on 13/07/2006 20:14:30
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.15%
// Training Accuracy: 100.00%

bool gepModel(bool d[]);
bool gepModel(bool d[])
{
    bool biTemp = false;

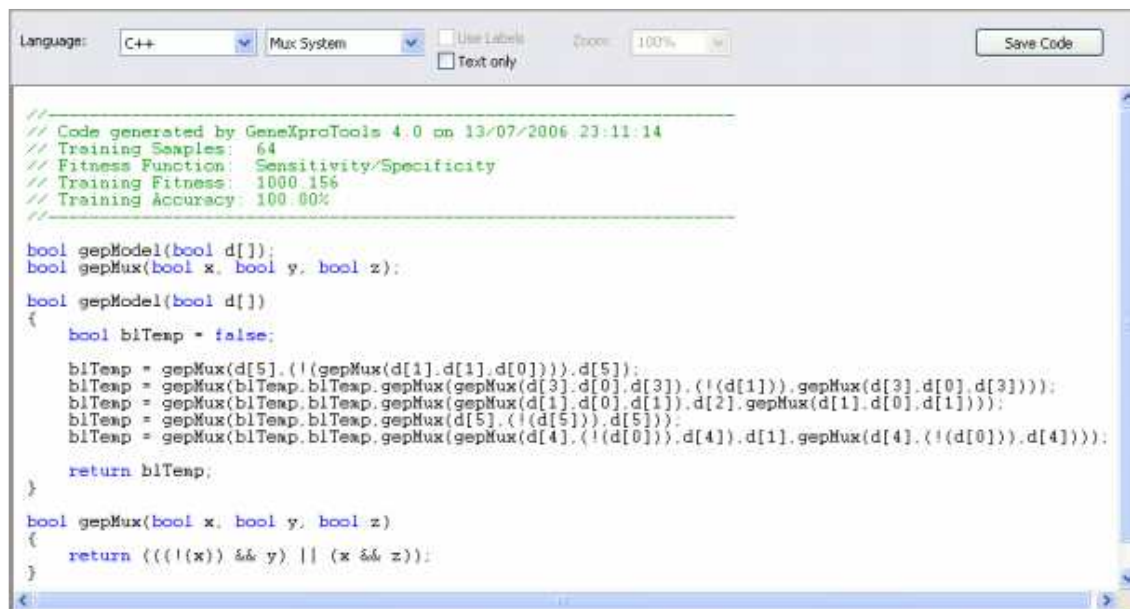
    biTemp = (d[5] || ((d[1] && d[0])));
    biTemp = biTemp && ((d[3] || d[0]) || ((d[1])));
    biTemp = biTemp && ((d[1] || d[0]) || d[2]);
    biTemp = biTemp && (d[5] || ((d[5])));
    biTemp = biTemp && ((d[4] || ((d[0])) || d[1]));

    return biTemp;
}

```

But you can also redesign immediately the generated **C++** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **C++**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both C++ and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: C++ Mux System Use Labels Zoom: 100% Save Code
Text only

// Code generated by GeneXproTools 4.0 on 13/07/2006 23:11:14
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.15%
// Training Accuracy: 100.00%

bool gepModel(bool d[]);
bool gepMux(bool x, bool y, bool z);
bool gepModel(bool d[])
{
    bool biTemp = false;

    biTemp = gepMux(d[5], ((gepMux(d[1], d[1], d[0]))), d[5]);
    biTemp = gepMux(biTemp, biTemp, gepMux(gepMux(d[3], d[0], d[3]), ((d[1])), gepMux(d[3], d[0], d[3])));
    biTemp = gepMux(biTemp, biTemp, gepMux(gepMux(d[1], d[0], d[1]), d[2], gepMux(d[1], d[0], d[1])));
    biTemp = gepMux(biTemp, biTemp, gepMux(d[5], ((d[5])), d[5]));
    biTemp = gepMux(biTemp, biTemp, gepMux(gepMux(d[4], ((d[0])), d[4]), d[1], gepMux(d[4], ((d[0])), d[4])));

    return biTemp;
}

bool gepMux(bool x, bool y, bool z)
{
    return (((!(x)) && y) || (x && z));
}

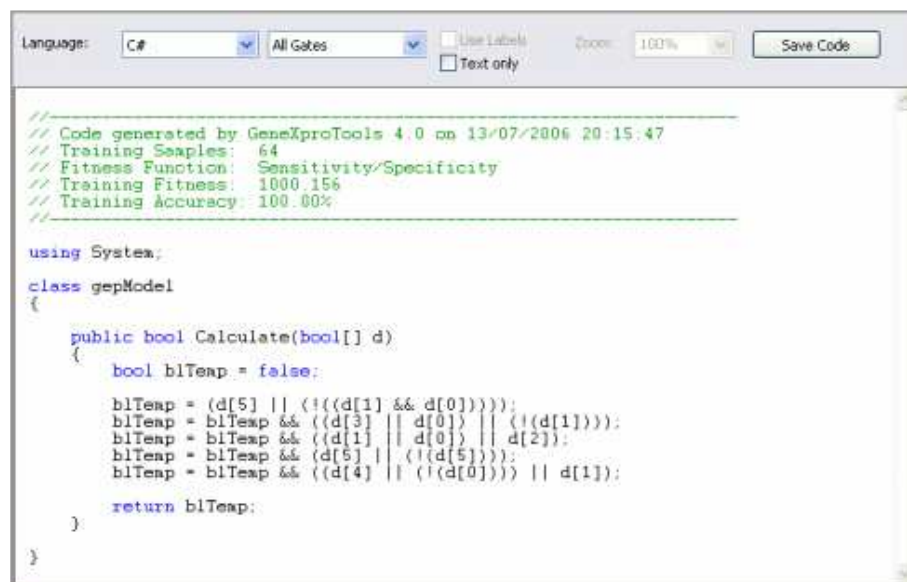
```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates C++ grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.

Generating Code Automatically

C#

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates C# grammar](#) you can generate immediately **C# code** using all the 258 built-in gates.



```
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:15:47
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.15%
// Training Accuracy: 100.00%

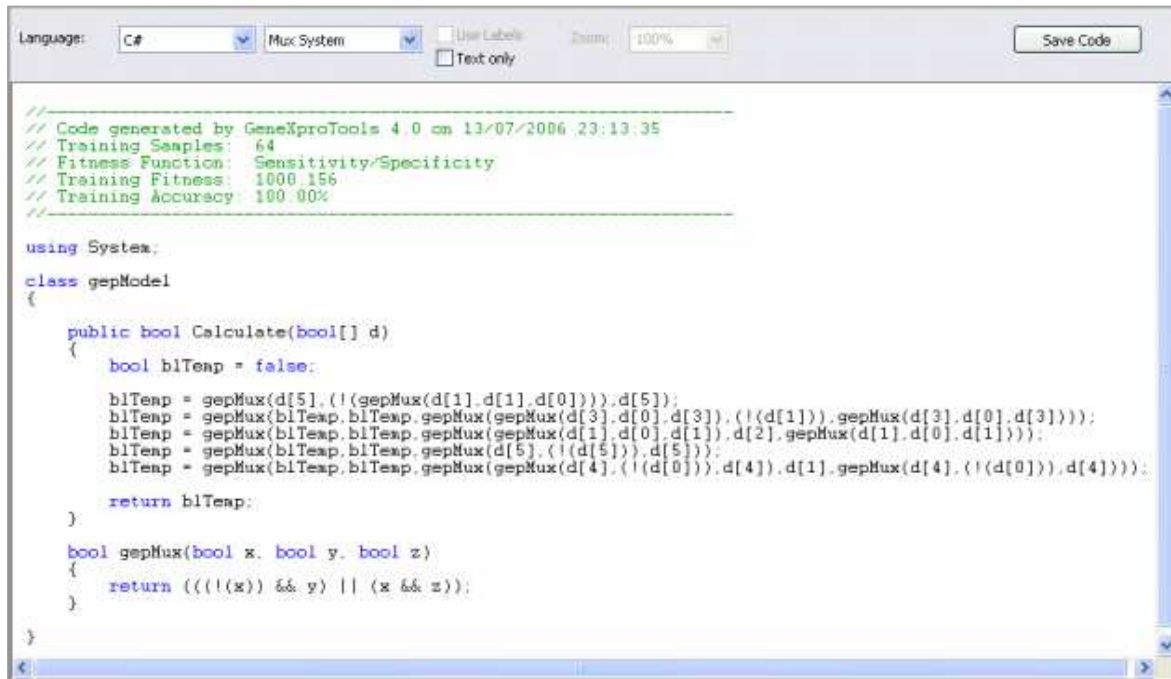
using System;
class gepModel
{
    public bool Calculate(bool[] d)
    {
        bool blTemp = false;

        blTemp = (d[5] || (!((d[1] && d[0]))));
        blTemp = blTemp && ((d[3] || d[0]) || (!d[1]));
        blTemp = blTemp && ((d[1] || d[0]) || d[2]);
        blTemp = blTemp && (d[5] || (!d[5]));
        blTemp = blTemp && ((d[4] || (!d[0])) || d[1]);

        return blTemp;
    }
}
```

But you can also redesign immediately the generated **C#** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **C#**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both C# and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: C# Mux System ☐ Use Labels Zoom: 100% 
☐ Text only

// Code generated by GeneXproTools 4.0 on 13/07/2006 23:13:35
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%
//

using System;
class gepModel
{
    public bool Calculate(bool[] d)
    {
        bool blTemp = false;

        blTemp = gepMux(d[5], (!gepMux(d[1], d[1], d[0])) , d[5]);
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[3], d[0], d[3]), (!d[1]), gepMux(d[3], d[0], d[3])));
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[1], d[0], d[1]), d[0], d[1]), d[2], gepMux(d[1], d[0], d[1])));
        blTemp = gepMux(blTemp, blTemp, gepMux(d[5], (!d[5]), d[5]));
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[4], (!d[0]), d[4]), d[1], gepMux(d[4], (!d[0]), d[4])));

        return blTemp;
    }

    bool gepMux(bool x, bool y, bool z)
    {
        return (((!(x)) && y) || (x && z));
    }
}

```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates C# grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

Fortran

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Fortran grammar](#) you can generate immediately **Fortran code** using all the 258 built-in gates.

```

Language: Fortran All Gates Use Labels Zoom: 100% Save Code
Text only

Code generated by GeneXproTools 4.0 on 13/07/2006 20:17:18
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

logical function gepModel(d)
    logical, intent(in) :: d(0:5)
    logical :: varTemp
    varTemp = .false.

    varTemp = (d(5) .or. (.not. ((d(1) .and. d(0)))))
    varTemp = varTemp .and. ((d(3) .or. d(0)) .or. (.not. (d(1))))
    varTemp = varTemp .and. ((d(1) .or. d(0)) .or. d(2))
    varTemp = varTemp .and. (d(5) .or. (.not. (d(5))))
    varTemp = varTemp .and. ((d(4) .or. (.not. (d(0)))) .or. d(1))

    gepModel = varTemp
end function gepModel

```

But you can also redesign immediately the generated **Fortran** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Fortran**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Fortran and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.

```

Language: Fortran Mux System Use Labels Zoom: 100% Save Code
Text only

Code generated by GeneXproTools 4.0 on 13/07/2006 23:15:23
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

logical function gepModel(d)
    logical, intent(in) :: d(0:5)
    logical, external :: gepMux
    logical :: varTemp
    varTemp = .false.

    varTemp = gepMux(d(5), (.not. (gepMux(d(1), d(1), d(0))))) , d(5))
    varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d(3), d(0), d(3)), (.not. (d(1))), gepMux(d(3), d(0), d(3))))
    varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d(1), d(0), d(1)), d(2), gepMux(d(1), d(0), d(1))))
    varTemp = gepMux(varTemp, varTemp, gepMux(d(5), (.not. (d(5))), d(5)))
    varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d(4), (.not. (d(0))), d(4)), d(1), gepMux(d(4), (.not. (d(0))), d(4))))

    gepModel = varTemp
end function gepModel

logical function gepMux(x, y, z)
    logical, intent(in) :: x, y, z
    gepMux = (((.not. (x)) .and. y) .or. (x .and. z))
end function gepMux

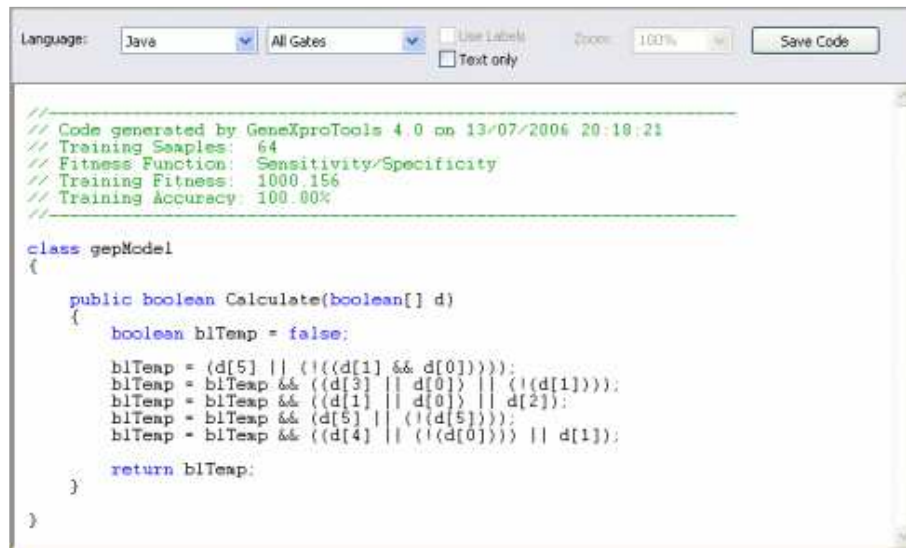
```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Fortran grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.

Generating Code Automatically

Java

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Java grammar](#) you can generate immediately **Java code** using all the 258 built-in gates.



```
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:18:21
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%
//

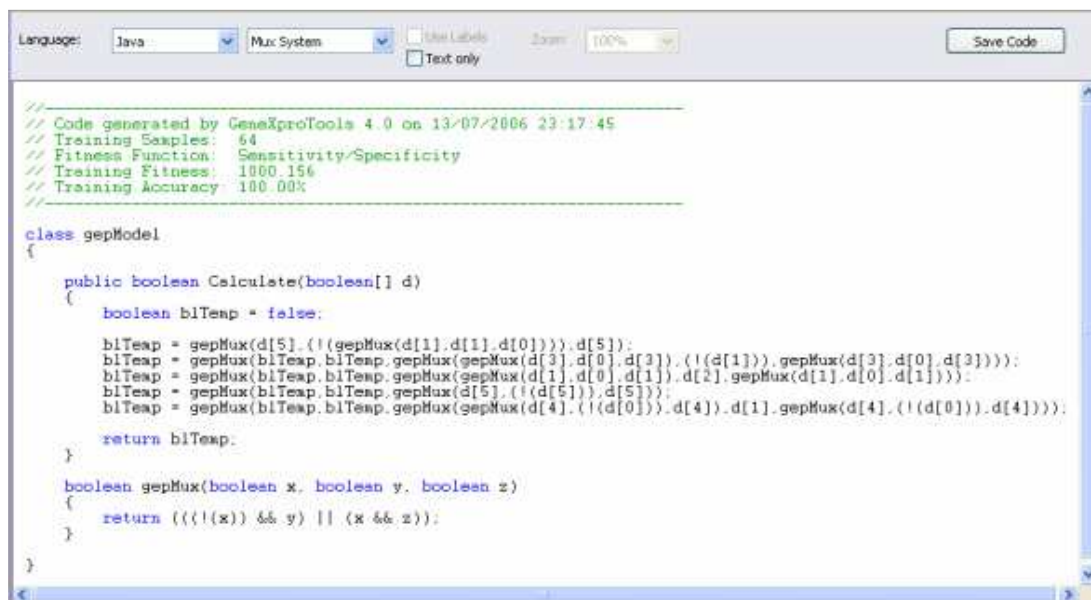
class gepModel
{
    public boolean Calculate(boolean[] d)
    {
        boolean blTemp = false;

        blTemp = (d[5] || ((!(d[1] && d[0]))));
        blTemp = blTemp && ((d[3] || d[0]) || (!(d[1])));
        blTemp = blTemp && ((d[1] || d[0]) || d[2]);
        blTemp = blTemp && (d[5] || (!(d[5])));
        blTemp = blTemp && ((d[4] || (!(d[0])) || d[1]);

        return blTemp;
    }
}
```

But you can also redesign immediately the generated **Java** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Java**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Java and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```
// Code generated by GeneXproTools 4.0 on 13/07/2006 23:17:45
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%
//

class gepModel
{
    public boolean Calculate(boolean[] d)
    {
        boolean blTemp = false;

        blTemp = gepMux(d[5], ((gepMux(d[1], d[1], d[0]))), d[5]);
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[3], d[0], d[3]), (!(d[1])), gepMux(d[3], d[0], d[3])));
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[1], d[0], d[1]), d[2], gepMux(d[1], d[0], d[1])));
        blTemp = gepMux(blTemp, blTemp, gepMux(d[5], (!(d[5])), d[5]));
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[4], (!(d[0])), d[4]), d[1], gepMux(d[4], (!(d[0])), d[4])));

        return blTemp;
    }

    boolean gepMux(boolean x, boolean y, boolean z)
    {
        return (((!(x)) && y) || (x && z));
    }
}
```


Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Java grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

Java Script

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Java Script grammar](#) you can generate immediately **Java Script code** using all the 258 built-in gates.

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:19:26
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%

function gepModel(d)
{
    var vTemp = false;
    vTemp = (d[5] || ((d[1] && d[0])));
    vTemp = vTemp && ((d[3] || d[0]) || (d[1]));
    vTemp = vTemp && ((d[1] || d[0]) || d[2]);
    vTemp = vTemp && (d[5] || (d[5]));
    vTemp = vTemp && ((d[4] || ((d[0])) || d[1]));
    return (vTemp ? 1 : 0);
}
```

But you can also redesign immediately the generated **Java Script** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Java Script**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Java Script and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.

```
// Code generated by GeneXproTools 4.0 on 13/07/2006 23:19:04
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%

function gepModel(d)
{
    var vTemp = false;

    vTemp = gepMux(d[5], (!gepMux(d[1], d[1], d[0])));
    vTemp = gepMux(vTemp, gepMux(gepMux(d[3], d[0], d[3]), (!d[1]), gepMux(d[3], d[0], d[3])));
    vTemp = gepMux(vTemp, gepMux(gepMux(d[1], d[0], d[1]), d[2], gepMux(d[1], d[0], d[1])));
    vTemp = gepMux(vTemp, gepMux(d[5], (!d[5]), d[5]));
    vTemp = gepMux(vTemp, gepMux(gepMux(d[4], (!d[0]), d[4]), d[1], gepMux(d[4], (!d[0]), d[4])));

    return (vTemp ? 1 : 0);
}

function gepMux(x, y, z)
{
    return (((!(x) && y) || (x && z));
}

```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Java Script grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

Matlab

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Matlab grammar](#) you can generate immediately **Matlab code** using all the 258 built-in gates.

```
% Code generated by GeneXproTools 4.0 on 13/07/2006 20:20:27
% Training Samples: 64
% Fitness Function: Sensitivity/Specificity
% Training Fitness: 1000.156
% Training Accuracy: 100.00%

function result = gepModel(d)
TRUE = 1;
FALSE = 0;

varTemp = FALSE;

varTemp = (d(6) | ~(d(2) & d(1)));
varTemp = varTemp & ((d(4) | d(1)) | ~(d(2)));
varTemp = varTemp & ((d(2) | d(1)) | d(3));
varTemp = varTemp & (d(6) | ~(d(6)));
varTemp = varTemp & ((d(5) | ~(d(1))) | d(2));

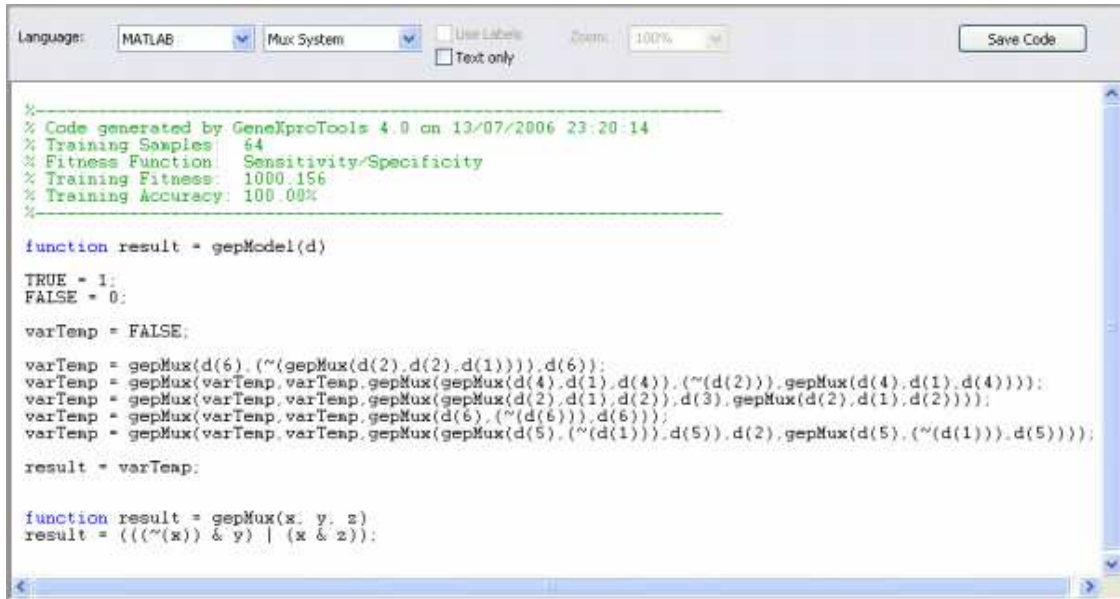
result = varTemp;

```

But you can also redesign immediately the generated **Matlab** circuits using five additional **logical**

grammars: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Matlab**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Matlab and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```
% Code generated by GeneXproTools 4.0 on 13/07/2006 23:20:14
% Training Samples: 64
% Fitness Function: Sensitivity/Specificity
% Training Fitness: 1000.156
% Training Accuracy: 100.00%
%

function result = gepModel(d)

TRUE = 1;
FALSE = 0;

varTemp = FALSE;

varTemp = gepMux(d(6), (~gepMux(d(2), d(2), d(1))) , d(6));
varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d(4), d(1), d(4)), (~d(2)), gepMux(d(4), d(1), d(4))));
varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d(2), d(1), d(2)), d(3), gepMux(d(2), d(1), d(2))));
varTemp = gepMux(varTemp, varTemp, gepMux(d(6), (~d(6)), d(6)));
varTemp = gepMux(varTemp, varTemp, gepMux(gepMux(d(5), (~d(1))), d(5), d(2), gepMux(d(5), (~d(1))), d(5))));

result = varTemp;

function result = gepMux(x, y, z)
result = (((~(x)) & y) | (x & z));
```

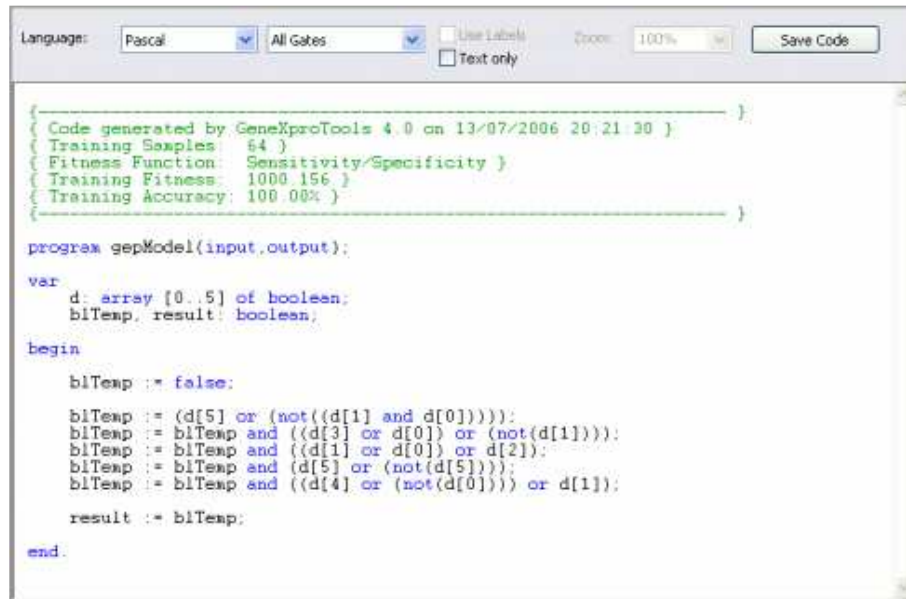
Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Matlab grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

Pascal

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Pascal grammar](#) you can generate immediately **Pascal code** using all the 258 built-in gates.



```

Language: Pascal All Gates Use Labels Zoom: 100% Save Code
Text only

{ Code generated by GeneXproTools 4.0 on 13/07/2006 20:21:30 }
{ Training Samples: 64 }
{ Fitness Function: Sensitivity/Specificity }
{ Training Fitness: 1000.156 }
{ Training Accuracy: 100.00% }

program gepModel(input,output);
var
  d: array [0..5] of boolean;
  blTemp, result: boolean;
begin
  blTemp := false;

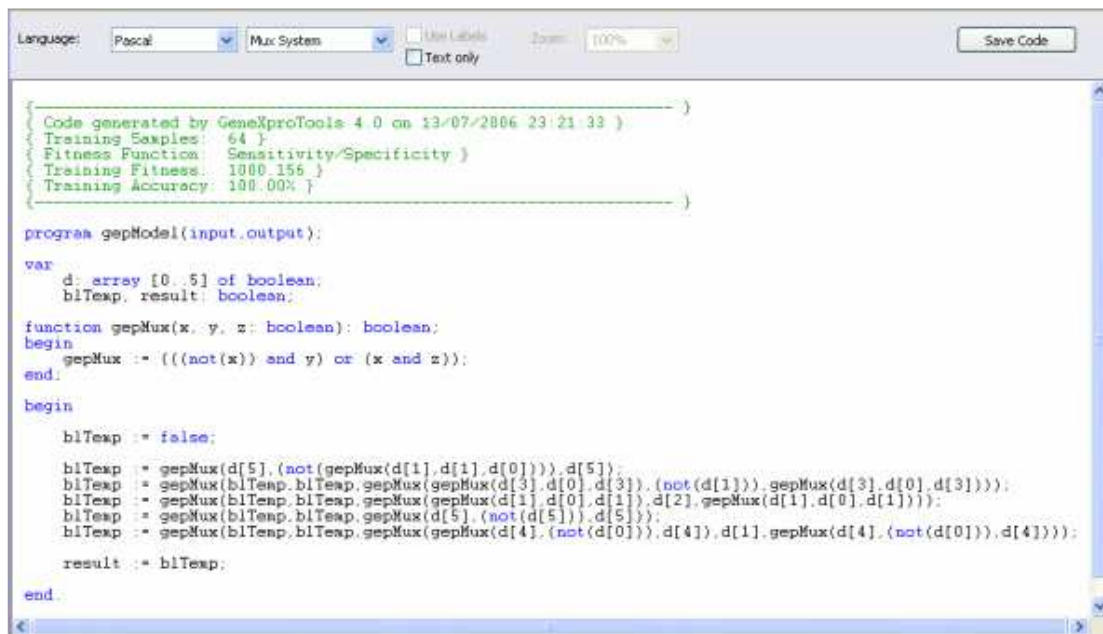
  blTemp := (d[5] or (not((d[1] and d[0]))));
  blTemp := blTemp and ((d[3] or d[0]) or (not(d[1])));
  blTemp := blTemp and ((d[1] or d[0]) or d[2]);
  blTemp := blTemp and (d[5] or (not(d[5])));
  blTemp := blTemp and ((d[4] or (not(d[0])) or d[1]));

  result := blTemp;
end.

```

But you can also redesign immediately the generated **Pascal** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Pascal**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Pascal and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: Pascal Mux System Use Labels Zoom: 100% Save Code
Text only

{ Code generated by GeneXproTools 4.0 on 13/07/2006 23:21:33 }
{ Training Samples: 64 }
{ Fitness Function: Sensitivity/Specificity }
{ Training Fitness: 1000.156 }
{ Training Accuracy: 100.00% }

program gepModel(input,output);
var
  d: array [0..5] of boolean;
  blTemp, result: boolean;

function gepMux(x, y, z: boolean): boolean;
begin
  gepMux := (((not(x)) and y) or (x and z));
end;

begin
  blTemp := false;

  blTemp := gepMux(d[5], (not(gepMux(d[1], d[1], d[0]))), d[5]);
  blTemp := gepMux(blTemp, blTemp, gepMux(gepMux(d[3], d[0], d[3]), (not(d[1])), gepMux(d[3], d[0], d[3])));
  blTemp := gepMux(blTemp, blTemp, gepMux(gepMux(d[1], d[0], d[1]), d[2], gepMux(d[1], d[0], d[1])));
  blTemp := gepMux(blTemp, blTemp, gepMux(d[5], (not(d[5])), d[5]));
  blTemp := gepMux(blTemp, blTemp, gepMux(gepMux(d[4], (not(d[0])), d[4]), d[1], gepMux(d[4], (not(d[0])), d[4])));

  result := blTemp;
end.

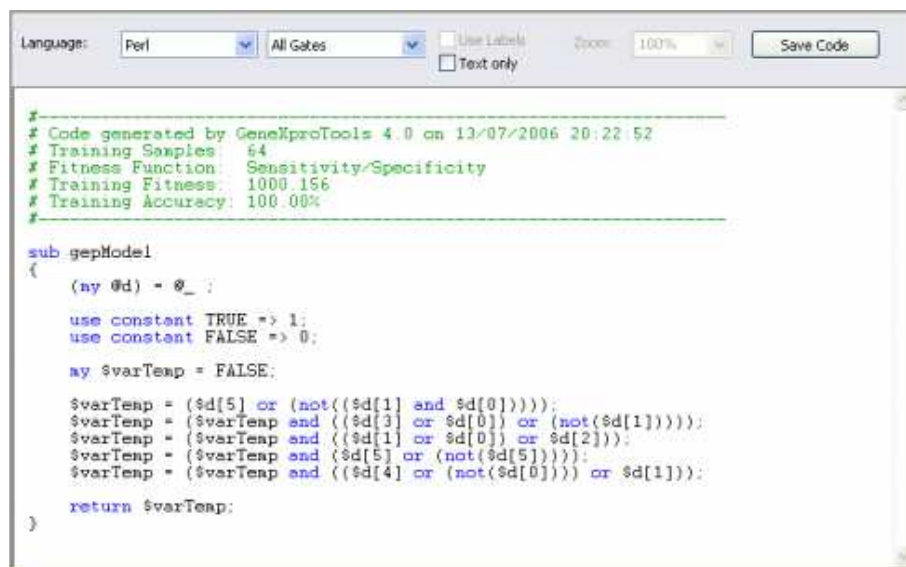
```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Pascal grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.

Generating Code Automatically

Perl

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Perl grammar](#) you can generate immediately **Perl code** using all the 258 built-in gates.



```
#
# Code generated by GeneXproTools 4.0 on 13/07/2006 20:22:52
# Training Samples: 64
# Fitness Function: Sensitivity/Specificity
# Training Fitness: 1000.156
# Training Accuracy: 100.00%
#

sub gepModel
{
    (my @d) = @_ ;
    use constant TRUE => 1;
    use constant FALSE => 0;

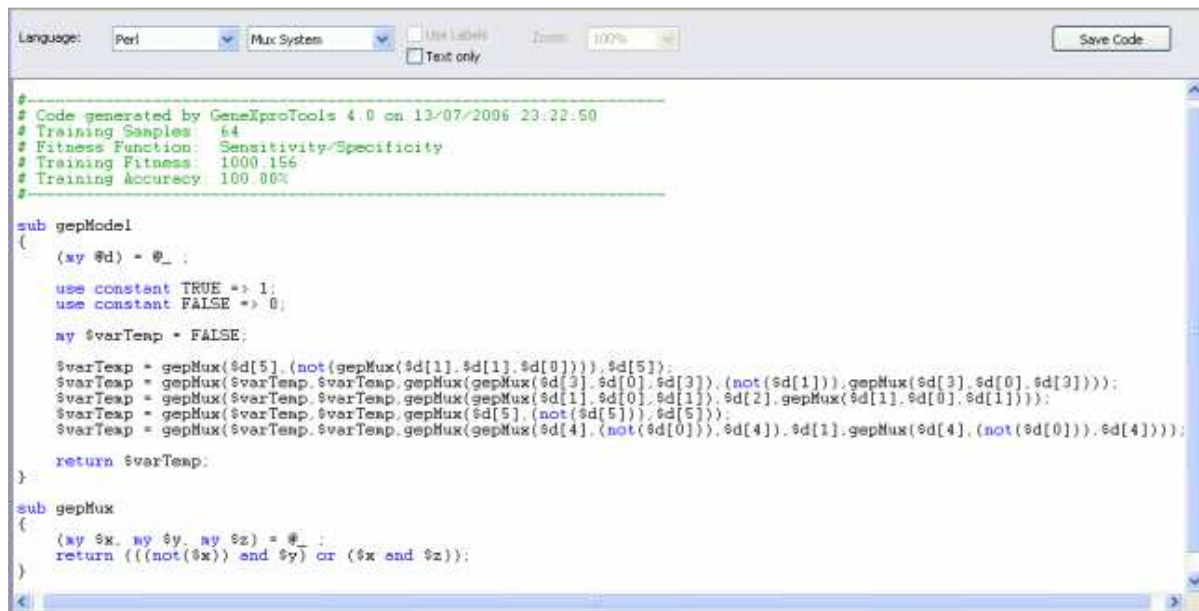
    my $varTemp = FALSE;

    $varTemp = ($d[5] or (not(($d[1] and $d[0]))));
    $varTemp = ($varTemp and (($d[3] or $d[0]) or (not($d[1]))));
    $varTemp = ($varTemp and (($d[1] or $d[0]) or $d[2]));
    $varTemp = ($varTemp and ($d[5] or (not($d[5]))));
    $varTemp = ($varTemp and (($d[4] or (not($d[0])) or $d[1]));

    return $varTemp;
}
```

But you can also redesign immediately the generated **Perl** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Perl**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Perl and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: Perl Mux System ☐ Use Labels Zoom: 100% 
☐ Text only

#
# Code generated by GeneXproTools 4.0 on 13/07/2006 23:22:50
# Training Samples: 64
# Fitness Function: Sensitivity/Specificity
# Training Fitness: 1000.156
# Training Accuracy: 100.00%
#

sub gepModel
{
    (my @d) = @_ ;

    use constant TRUE => 1;
    use constant FALSE => 0;

    my $varTemp = FALSE;

    $varTemp = gepMux($d[5], (not(gepMux($d[1], $d[1], $d[0]))), $d[5]);
    $varTemp = gepMux($varTemp, $varTemp, gepMux(gepMux($d[3], $d[0], $d[3]), (not($d[1])), gepMux($d[3], $d[0], $d[3])));
    $varTemp = gepMux($varTemp, $varTemp, gepMux(gepMux($d[1], $d[0], $d[1]), $d[2], gepMux($d[1], $d[0], $d[1])));
    $varTemp = gepMux($varTemp, $varTemp, gepMux($d[5], (not($d[5])), $d[5]));
    $varTemp = gepMux($varTemp, $varTemp, gepMux(gepMux($d[4], (not($d[0])), $d[4]), $d[1], gepMux($d[4], (not($d[0])), $d[4])));

    return $varTemp;
}

sub gepMux
{
    (my $x, my $y, my $z) = @_ ;
    return (((not($x)) and $y) or ($x and $z));
}

```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Perl grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

PHP

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates PHP grammar](#) you can generate immediately **PHP code** using all the 258 built-in gates.



```

Language: PHP All Gates ☐ Use Labels Zoom: 100% 
☐ Text only

//
// Code generated by GeneXproTools 4.0 on 13/07/2006 20:24:02
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000.156
// Training Accuracy: 100.00%
//

<?php
function gepModel($d)
{
    $varTemp = false;

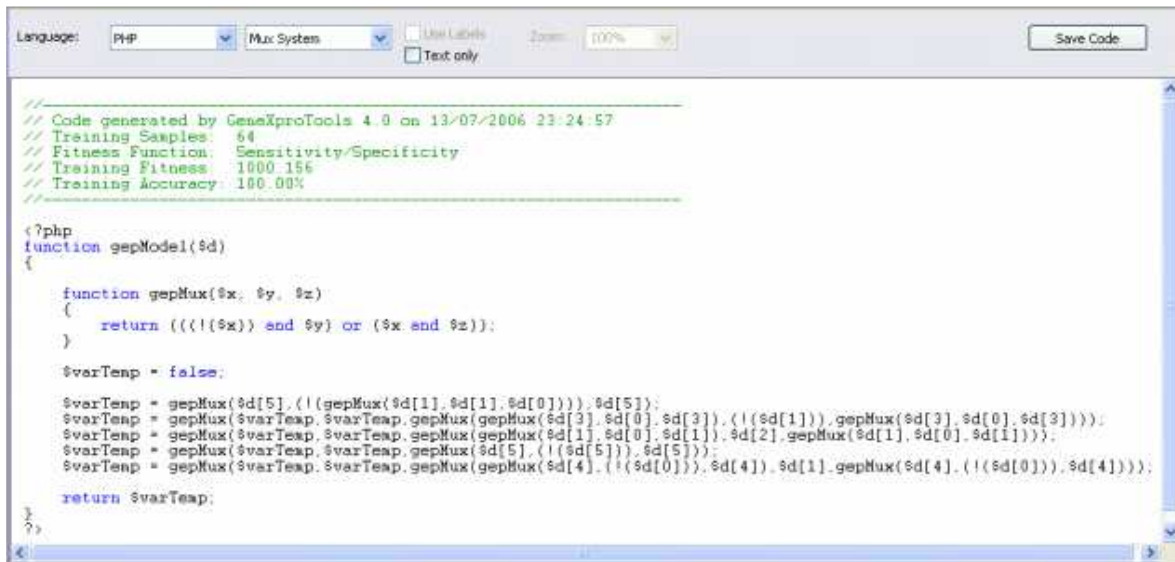
    $varTemp = ($d[5] or (((($d[1] and $d[0])))));
    $varTemp = ($varTemp and (($d[3] or $d[0]) or !($d[1])));
    $varTemp = ($varTemp and (($d[1] or $d[0]) or $d[2]));
    $varTemp = ($varTemp and ($d[5] or !($d[5])));
    $varTemp = ($varTemp and (($d[4] or !($d[0])) or $d[1]));

    return $varTemp;
}
?>

```


But you can also redesign immediately the generated **PHP** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **PHP**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both PHP and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```
// Code generated by GeneXproTools 4.0 on 13/07/2006 23:24:57
// Training Samples: 64
// Fitness Function: Sensitivity/Specificity
// Training Fitness: 1000 156
// Training Accuracy: 100.00%

<?php
function gepModel($d)
{
    function gepMux($x, $y, $z)
    {
        return (((($x)) and $y) or ($x and $z));
    }

    $varTemp = false;

    $varTemp = gepMux($d[5], ((gepMux($d[1], $d[1], $d[0]))), $d[5]);
    $varTemp = gepMux($varTemp, $varTemp, gepMux(gepMux($d[3], $d[0], $d[3]), ((($d[1])), gepMux($d[3], $d[0], $d[3]))));
    $varTemp = gepMux($varTemp, $varTemp, gepMux(gepMux($d[1], $d[0], $d[1]), $d[2], gepMux($d[1], $d[0], $d[1])));
    $varTemp = gepMux($varTemp, $varTemp, gepMux($d[5], ((($d[5])), $d[5])));
    $varTemp = gepMux($varTemp, $varTemp, gepMux(gepMux($d[4], ((($d[0])), $d[4]), $d[1], gepMux($d[4], ((($d[0])), $d[4]))));

    return $varTemp;
}
?>
```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates PHP grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

Python

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Python grammar](#) you can generate immediately **Python code** using all the 258 built-in gates.



```

Language: Python All Gates ☐ Use Labels ☐ Text only Zoom: 100% Save Code

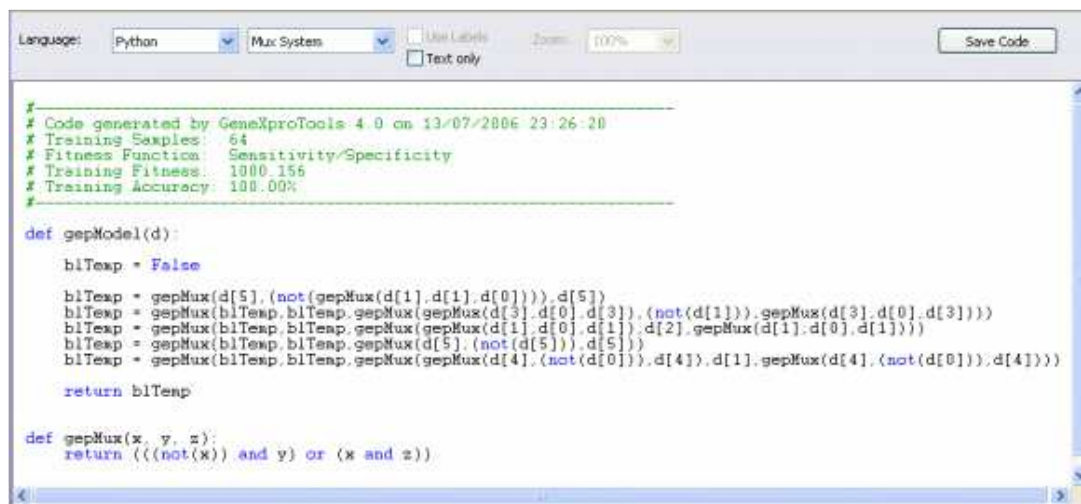
# Code generated by GeneXproTools 4.0 on 13/07/2006 20:25:16
# Training Samples: 64
# Fitness Function: Sensitivity/Specificity
# Training Fitness: 1000.156
# Training Accuracy: 100.00%

def gepModel(d):
    blTemp = False
    blTemp = (d[5] or (not((d[1] and d[0]))))
    blTemp = blTemp and ((d[3] or d[0]) or (not(d[1])))
    blTemp = blTemp and ((d[1] or d[0]) or d[2])
    blTemp = blTemp and (d[5] or (not(d[5])))
    blTemp = blTemp and ((d[4] or (not(d[0])) or d[1])
    return blTemp

```

But you can also redesign immediately the generated **Python** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Python**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Python and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: Python Mux System ☐ Use Labels ☐ Text only Zoom: 100% Save Code

# Code generated by GeneXproTools 4.0 on 13/07/2006 23:26:20
# Training Samples: 64
# Fitness Function: Sensitivity/Specificity
# Training Fitness: 1000.156
# Training Accuracy: 100.00%

def gepModel(d):
    blTemp = False
    blTemp = gepMux(d[5], (not(gepMux(d[1], d[1], d[0]))), d[5])
    blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[3], d[0], d[3]), (not(d[1])), gepMux(d[3], d[0], d[3]))))
    blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[1], d[0], d[1]), d[2], gepMux(d[1], d[0], d[1]))))
    blTemp = gepMux(blTemp, blTemp, gepMux(d[5], (not(d[5])), d[5]))
    blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d[4], (not(d[0])), d[4]), d[1], gepMux(d[4], (not(d[0])), d[4]))))
    return blTemp

def gepMux(x, y, z):
    return ((not(x)) and y) or (x and z)

```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Python grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

Visual Basic

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Visual Basic grammar](#) you can generate immediately **Visual Basic code** using all the 258 built-in gates.



```

Language: Visual Basic All Gates Use Labels Zoom: 100% Save Code
☐ Text only

Code generated by GeneXproTools 4.0 on 13/07/2006 20:26:21
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

Function gepModel(ByRef d() As Boolean) As Boolean
    Dim biTemp As Boolean
    biTemp = False

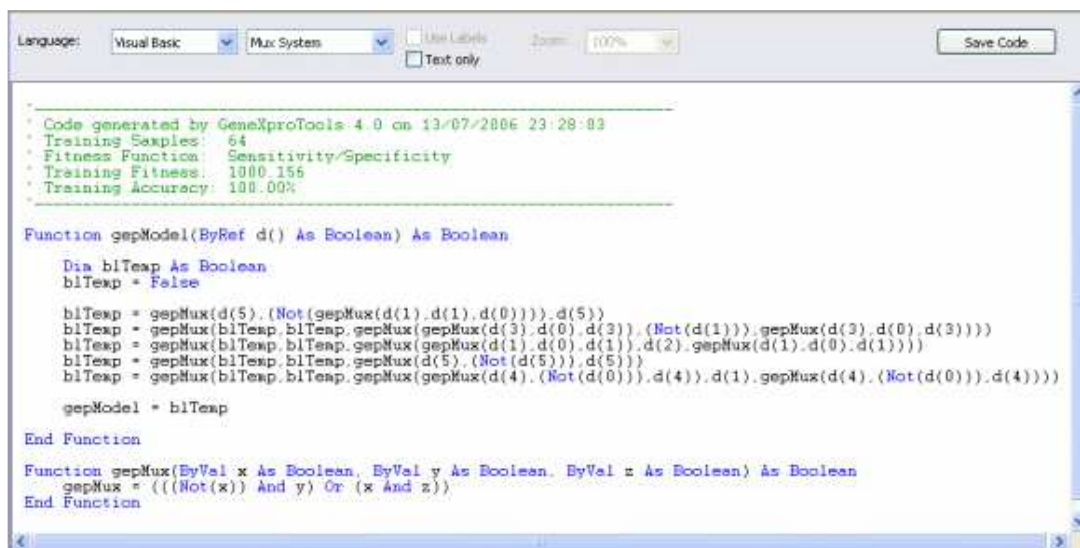
    biTemp = (d(5) Or (Not((d(1) And d(0))))))
    biTemp = biTemp And ((d(3) Or d(0)) Or (Not(d(1))))
    biTemp = biTemp And ((d(1) Or d(0)) Or d(2))
    biTemp = biTemp And (d(5) Or (Not(d(5))))
    biTemp = biTemp And ((d(4) Or (Not(d(0)))) Or d(1))

    gepModel = biTemp
End Function

```

But you can also redesign immediately the generated **Visual Basic** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Visual Basic**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both Visual Basic and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: Visual Basic Mux System Use Labels Zoom: 100% Save Code
☐ Text only

Code generated by GeneXproTools 4.0 on 13/07/2006 23:28:03
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

Function gepModel(ByRef d() As Boolean) As Boolean
    Dim biTemp As Boolean
    biTemp = False

    biTemp = gepMux(d(5), (Not(gepMux(d(1), d(1), d(0))))), d(5))
    biTemp = gepMux(biTemp, biTemp, gepMux(gepMux(d(3), d(0), d(3)), (Not(d(1))), gepMux(d(3), d(0), d(3)))))
    biTemp = gepMux(biTemp, biTemp, gepMux(gepMux(d(1), d(0), d(1)), d(2), gepMux(d(1), d(0), d(1)))))
    biTemp = gepMux(biTemp, biTemp, gepMux(d(5), (Not(d(5))), d(5)))
    biTemp = gepMux(biTemp, biTemp, gepMux(gepMux(d(4), (Not(d(0))), d(4)), d(1), gepMux(d(4), (Not(d(0))), d(4))))

    gepModel = biTemp
End Function

Function gepMux(ByVal x As Boolean, ByVal y As Boolean, ByVal z As Boolean) As Boolean
    gepMux = (((Not(x)) And y) Or (x And z))
End Function

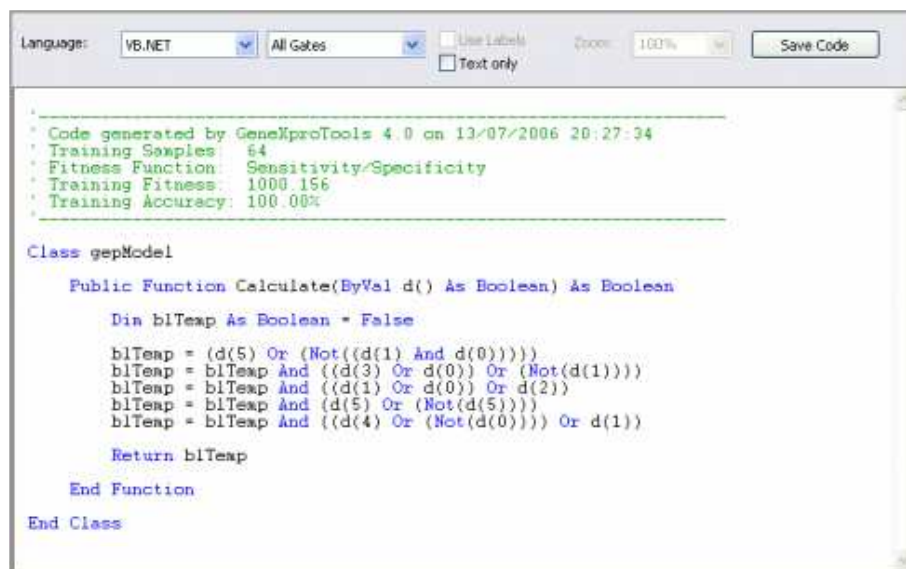
```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Visual Basic grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.

Generating Code Automatically

VB.Net

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates VB.Net grammar](#) you can generate immediately **VB.Net code** using all the 258 built-in gates.



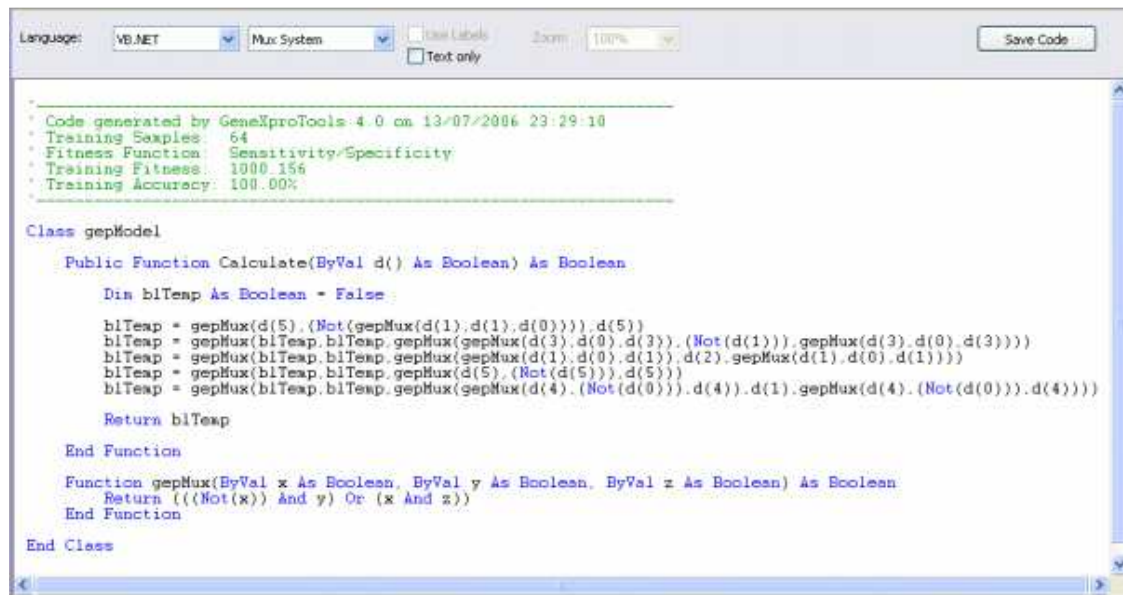
```
Code generated by GeneXproTools 4.0 on 13/07/2006 20:27:34
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

Class gepModel
    Public Function Calculate(ByVal d() As Boolean) As Boolean
        Dim blTemp As Boolean = False
        blTemp = (d(5) Or (Not((d(1) And d(0)))))
        blTemp = blTemp And ((d(3) Or d(0)) Or (Not(d(1))))
        blTemp = blTemp And ((d(1) Or d(0)) Or d(2))
        blTemp = blTemp And (d(5) Or (Not(d(5))))
        blTemp = blTemp And ((d(4) Or (Not(d(0)))) Or d(1))

        Return blTemp
    End Function
End Class
```

But you can also redesign immediately the generated **VB.Net** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **VB.Net**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both VB.Net and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



The screenshot shows the GeneXproTools 4.0 interface. At the top, there are dropdown menus for 'Language' (set to VB.NET) and 'Mux System'. To the right are checkboxes for 'Use Labels' and 'Text only', and a 'Zoom' dropdown set to 100%. A 'Save Code' button is in the top right corner. The main text area displays the following code:

```

Code generated by GeneXproTools 4.0 on 13/07/2006 23:29:10
Training Samples: 64
Fitness Function: Sensitivity/Specificity
Training Fitness: 1000.156
Training Accuracy: 100.00%

Class gepModel
    Public Function Calculate(ByVal d() As Boolean) As Boolean
        Dim blTemp As Boolean = False

        blTemp = gepMux(d(5), (Not(gepMux(d(1), d(1), d(0))))), d(5))
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d(3), d(0), d(3)), (Not(d(1))), gepMux(d(3), d(0), d(3)))))
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d(1), d(0), d(1)), d(2), gepMux(d(1), d(0), d(1)))))
        blTemp = gepMux(blTemp, blTemp, gepMux(d(5), (Not(d(5))), d(5)))
        blTemp = gepMux(blTemp, blTemp, gepMux(gepMux(d(4), (Not(d(0))), d(4)), d(1), gepMux(d(4), (Not(d(0))), d(4)))))

        Return blTemp
    End Function

    Function gepMux(ByVal x As Boolean, ByVal y As Boolean, ByVal z As Boolean) As Boolean
        Return (((Not(x)) And y) Or (x And z))
    End Function
End Class

```

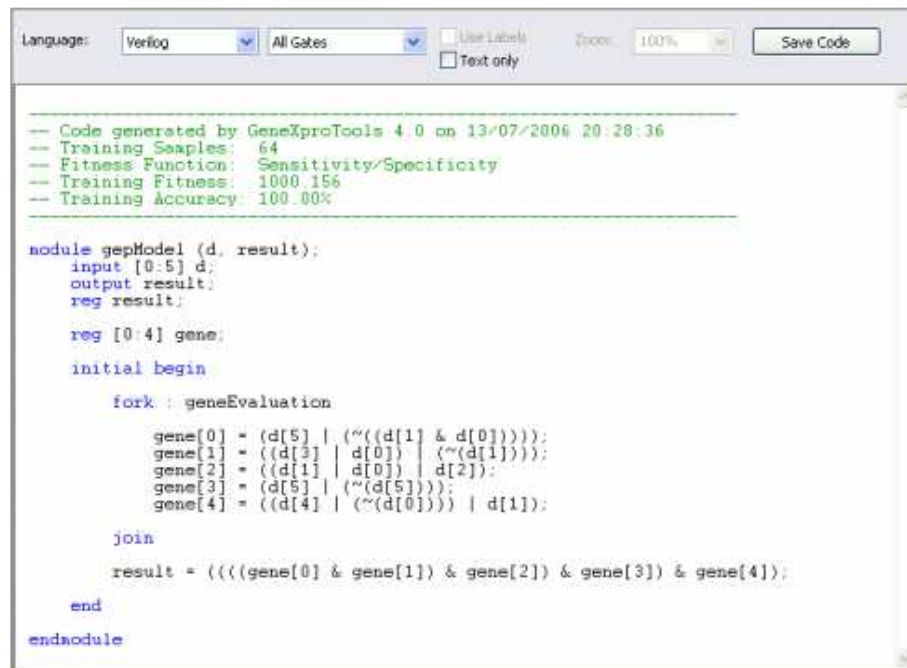
Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates VB.Net grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.

◀ | ▶

Generating Code Automatically

Verilog

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates Verilog grammar](#) you can generate immediately **Verilog code** using all the 258 built-in gates.



```

Language: Verilog All Gates Use Labels Zoom: 100% Save Code
Text only

-- Code generated by GeneXproTools 4.0 on 13/07/2006 20:28:36
-- Training Samples: 64
-- Fitness Function: Sensitivity/Specificity
-- Training Fitness: 1000.15%
-- Training Accuracy: 100.00%

module gepModel (d, result);
  input [0:5] d;
  output result;
  reg result;

  reg [0:4] gene;

  initial begin
    fork : geneEvaluation
      gene[0] = (d[5] | (~(d[1] & d[0])));
      gene[1] = ((d[3] | d[0]) | (~(d[1])));
      gene[2] = ((d[1] | d[0]) | d[2]);
      gene[3] = (d[5] | (~(d[5])));
      gene[4] = ((d[4] | (~(d[0])) | d[1]);

    join

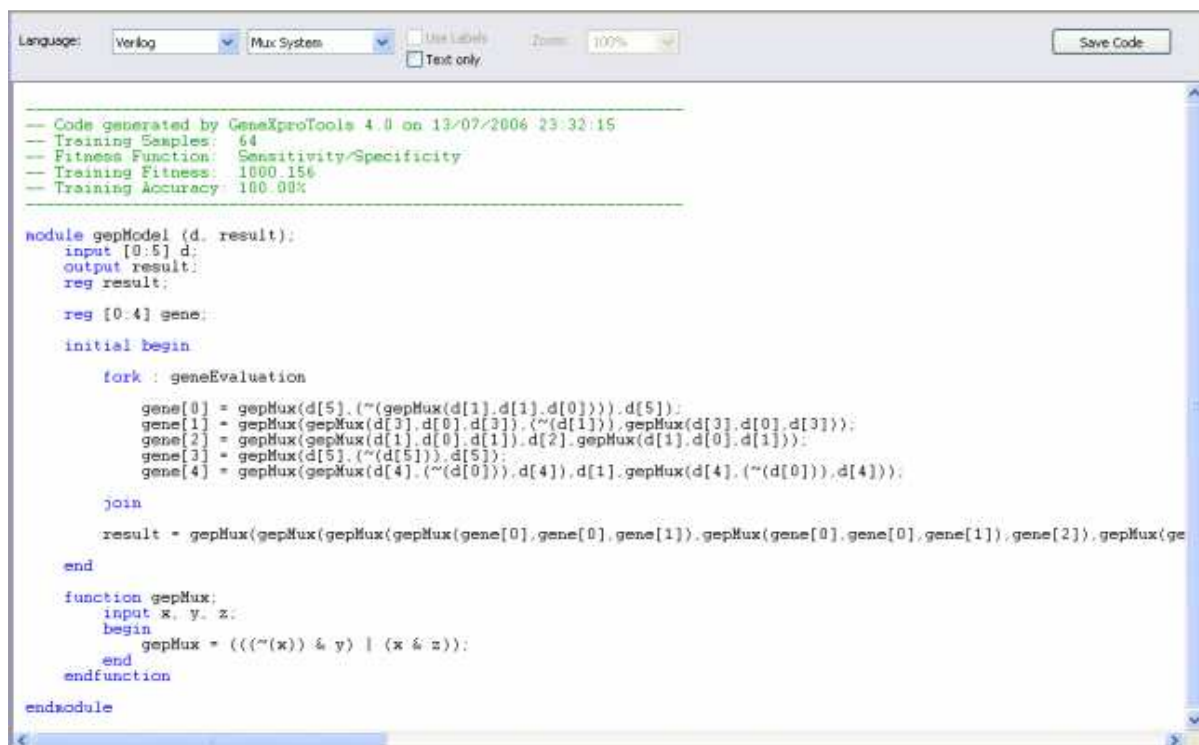
    result = (((gene[0] & gene[1]) & gene[2]) & gene[3]) & gene[4];

  end
endmodule

```

But you can also redesign immediately the generated **Verilog** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **Verilog**, on the GeneXproTools modeling environment, select the **Model Panel**, and then select both Verilog and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: Verilog Mux System Use Labels Zoom: 100% Save Code
Text only

-- Code generated by GeneXproTools 4.0 on 13/07/2006 23:32:15
-- Training Samples: 64
-- Fitness Function: Sensitivity/Specificity
-- Training Fitness: 1000.15%
-- Training Accuracy: 100.00%

module gepModel (d, result);
  input [0:5] d;
  output result;
  reg result;

  reg [0:4] gene;

  initial begin
    fork : geneEvaluation
      gene[0] = gepMux(d[5], (~(gepMux(d[1], d[1], d[0]))), d[5]);
      gene[1] = gepMux(gepMux(d[3], d[0], d[3]), (~(d[1])), gepMux(d[3], d[0], d[3]));
      gene[2] = gepMux(gepMux(d[1], d[0], d[1]), d[2], gepMux(d[1], d[0], d[1]));
      gene[3] = gepMux(d[5], (~(d[5])), d[5]);
      gene[4] = gepMux(gepMux(d[4], (~(d[0])), d[4]), d[1], gepMux(d[4], (~(d[0])), d[4]));

    join

    result = gepMux(gepMux(gepMux(gepMux(gene[0], gene[0], gene[1]), gepMux(gene[0], gene[0], gene[1]), gene[2]), gepMux(ge
  end

  function gepMux;
    input x, y, z;
    begin
      gepMux = (((~(x)) & y) | (x & z));
    end
  endfunction
endmodule

```


Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates Verilog grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

VHDL

GeneXproTools 4.0 allows you to design logic circuits using a total of 258 different [built-in logical functions](#). And with the [All Gates VHDL grammar](#) you can generate immediately **VHDL code** using all the 258 built-in gates.

```
-- Code generated by GeneXproTools 4.0 on 13/07/2006 20:29:59
-- Training Samples: 64
-- Fitness Function: Sensitivity/Specificity
-- Training Fitness: 1000.156
-- Training Accuracy: 100.00%

library IEEE;
use IEEE.std_logic_1164.all;

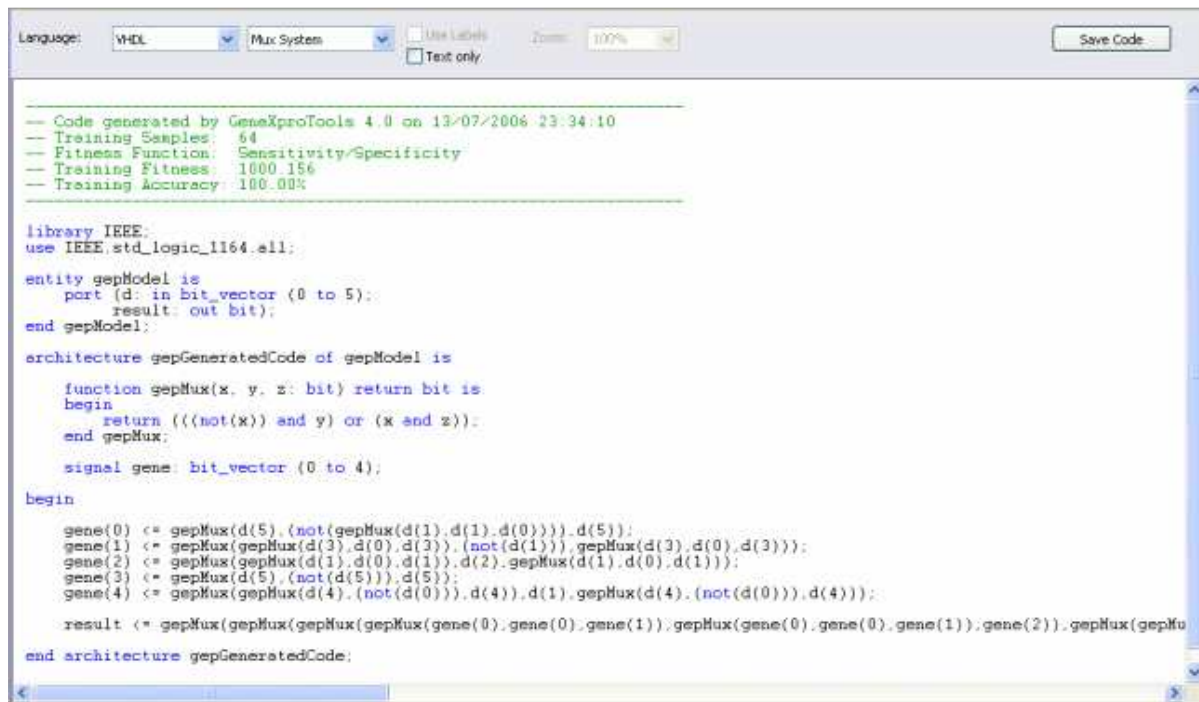
entity gepModel is
  port (d: in bit_vector (0 to 5);
        result: out bit);
end gepModel;

architecture gepGeneratedCode of gepModel is
  signal gene: bit_vector (0 to 4);
begin
  gene(0) <= (d(5) or (not((d(1) and d(0)))));
  gene(1) <= ((d(3) or d(0)) or (not(d(1))));
  gene(2) <= ((d(1) or d(0)) or d(2));
  gene(3) <= (d(5) or (not(d(5))));
  gene(4) <= ((d(4) or (not(d(0))) or d(1));

  result <= (((gene(0) and gene(1)) and gene(2)) and gene(3)) and gene(4);
end architecture gepGeneratedCode;
```

But you can also redesign immediately the generated **VHDL** circuits using five additional **logical grammars**: Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System. These grammars can be used to automatically generate Not-And-Or Only, Nand Only, Nor Only, Mux Only, and Not-And-Xor circuits.

To translate your logical circuits into **VHDL**, on the GeneXproTools modeling environment, select the [Model Panel](#), and then select both VHDL and the Logical System you are interested in (Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System) on the Language list boxes.



```

Language: VHDL Mux System ☐ Use Labels Zoom: 100% ☐ Text only Save Code

-- Code generated by GeneXproTools 4.0 on 13/07/2006 23:34:10
-- Training Samples: 64
-- Fitness Function: Sensitivity/Specificity
-- Training Fitness: 1000.156
-- Training Accuracy: 100.00%

library IEEE;
use IEEE.std_logic_1164.all;

entity gepModel is
  port (d: in bit_vector (0 to 5);
        result: out bit);
end gepModel;

architecture gepGeneratedCode of gepModel is
  function gepMux(x, y, z: bit) return bit is
  begin
    return ((not(x)) and y) or (x and z);
  end gepMux;

  signal gene: bit_vector (0 to 4);
begin
  gene(0) <= gepMux(d(5), (not(gepMux(d(1), d(1), d(0)))) d(5));
  gene(1) <= gepMux(gepMux(d(3), d(0), d(3)), (not(d(1))), gepMux(d(3), d(0), d(3)));
  gene(2) <= gepMux(gepMux(d(1), d(0), d(1)), d(2), gepMux(d(1), d(0), d(1)));
  gene(3) <= gepMux(d(5), (not(d(5))), d(5));
  gene(4) <= gepMux(gepMux(d(4), (not(d(0))), d(4)), d(1), gepMux(d(4), (not(d(0))), d(4)));

  result <= gepMux(gepMux(gepMux(gepMux(gene(0), gene(0), gene(1)), gepMux(gene(0), gene(0), gene(1)), gene(2)), gepMux(gepMux
end architecture gepGeneratedCode;

```

Obviously, GeneXproTools allows you to use exclusively the gates you are interested in as your building blocks and then use the [All Gates VHDL grammar](#) to generate your logic circuits. The advantage of this approach is that you will almost certainly end up with much more parsimonious circuits.



Generating Code Automatically

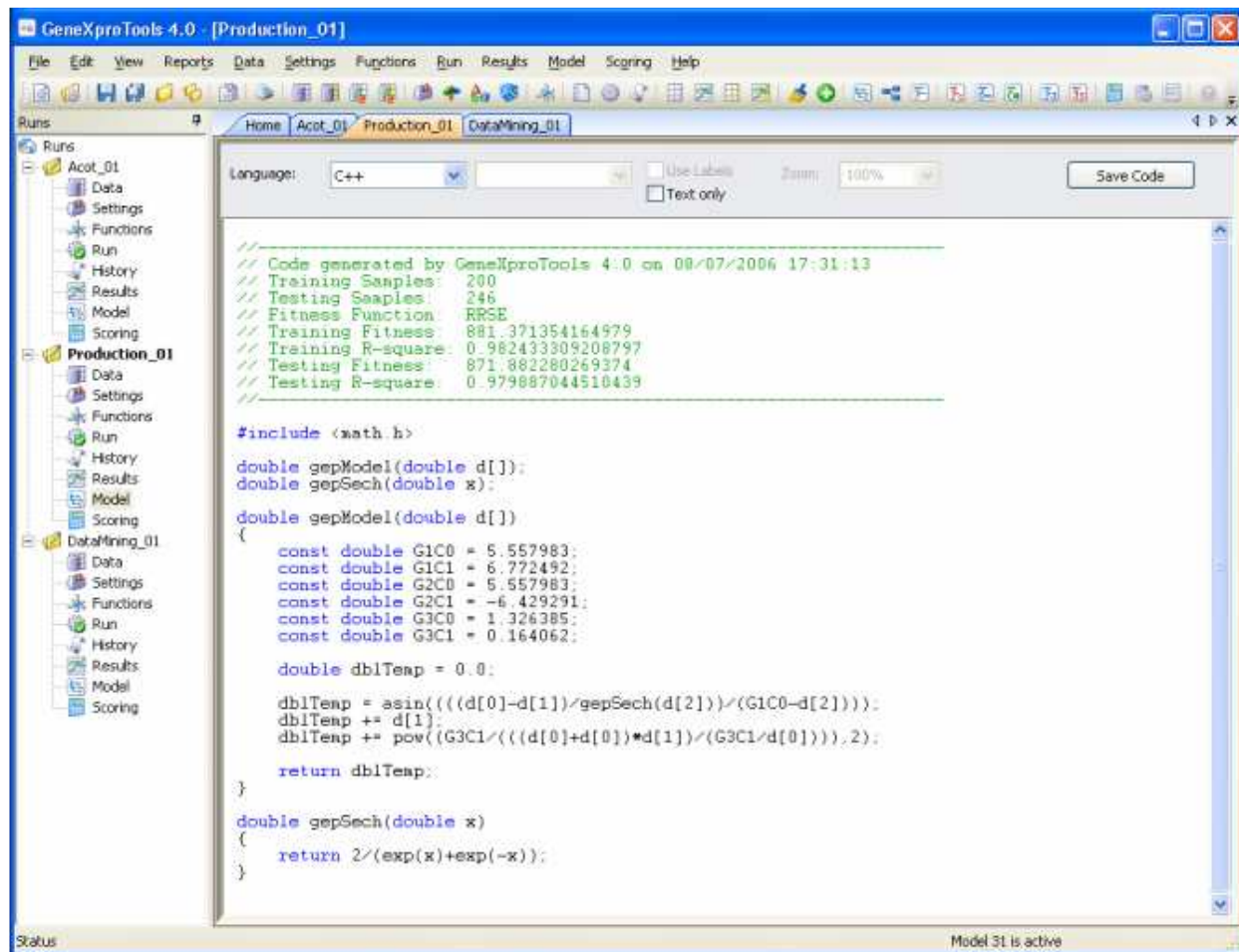
User Defined Grammars

GeneXproTools 4.0 allows you to design **User Defined Grammars** so that the models evolved by GeneXproTools in its native [Karva language](#) can be automatically translated into the programming language of your choice if you happen to prefer a programming language not already covered by the 16 [built-in grammars](#) of GeneXproTools 4.0 (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL).

GeneXproTools 4.0 allows you to define as many grammars as you wish. The design of an extra grammar in GeneXproTools 4.0 is very easy, since it can be done by using one of the 16 [built-in grammars](#) of GeneXproTools 4.0 as a starting point. As an illustration, you can take a look both at the [C++ Math grammar](#) and the [Boolean VHDL All Gates grammar](#) of GeneXproTools 4.0 by following the links, but all the 16 built-in grammars of GeneXproTools are accessible as xml files in the folder Grammars (usually C:\Program Files\GeneXproTools 4\Grammars). Then you just have to define correctly in this new programming language all the [mathematical](#) / [logical](#) functions used by GeneXproTools 4.0.

After its creation, the new grammar must be saved into an xml file. Then this file must be copied into the directory Grammars of GeneXproTools 4.0. After that you will be able to translate all the models evolved by GeneXproTools 4.0 into this new language.

To translate your model into this new language, on the GeneXproTools modeling environment, select the **Model Panel**, and then, on the Language list box, select the name you gave to your new language.



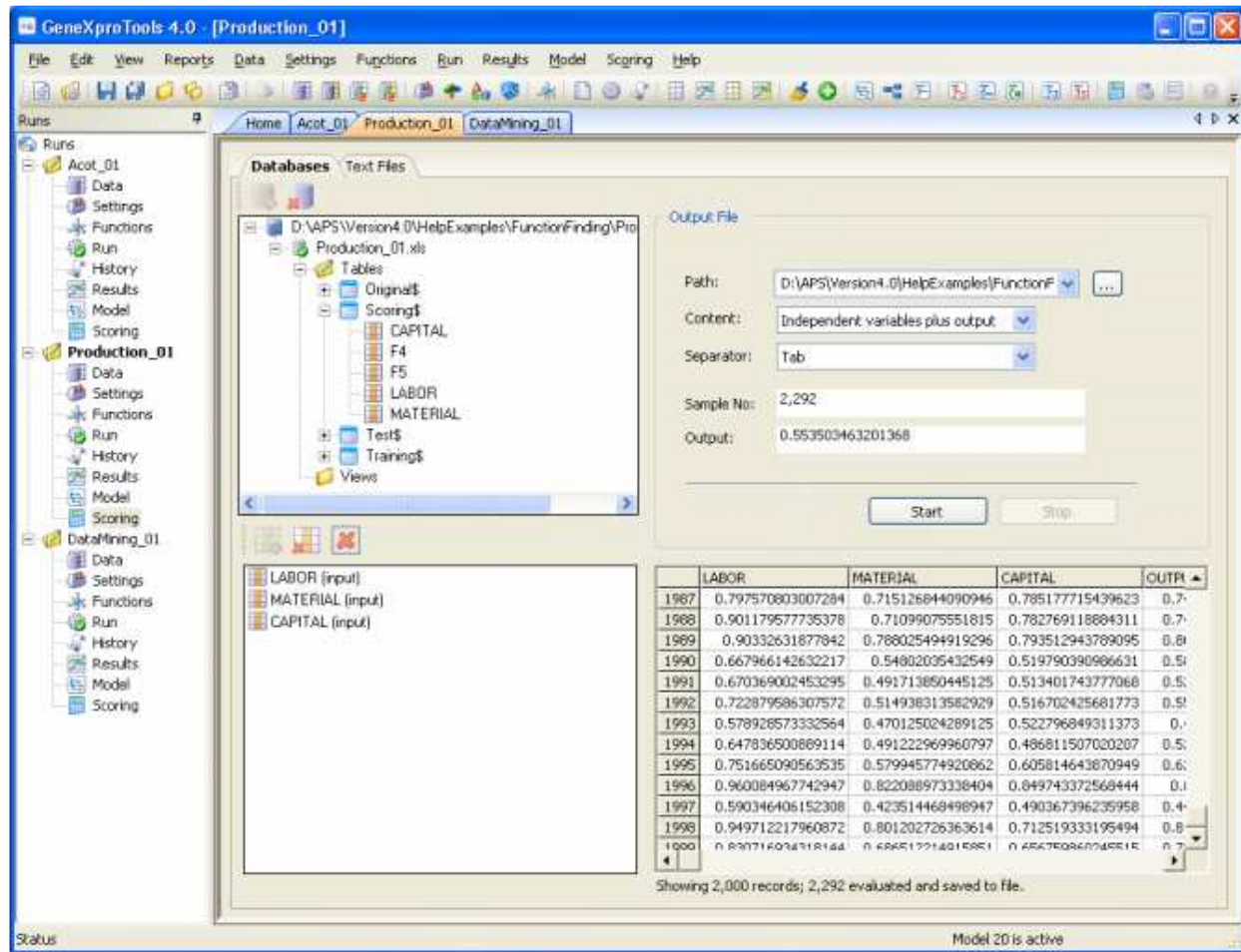
The code created with this new language can then be copied to the clipboard or saved to disk.



Making Predictions with a Model

In this chapter you'll learn how to apply immediately the evolved models to:

- [Score Excel files and databases.](#)
- [Score text files.](#)
- [Make predictions with time series models.](#)

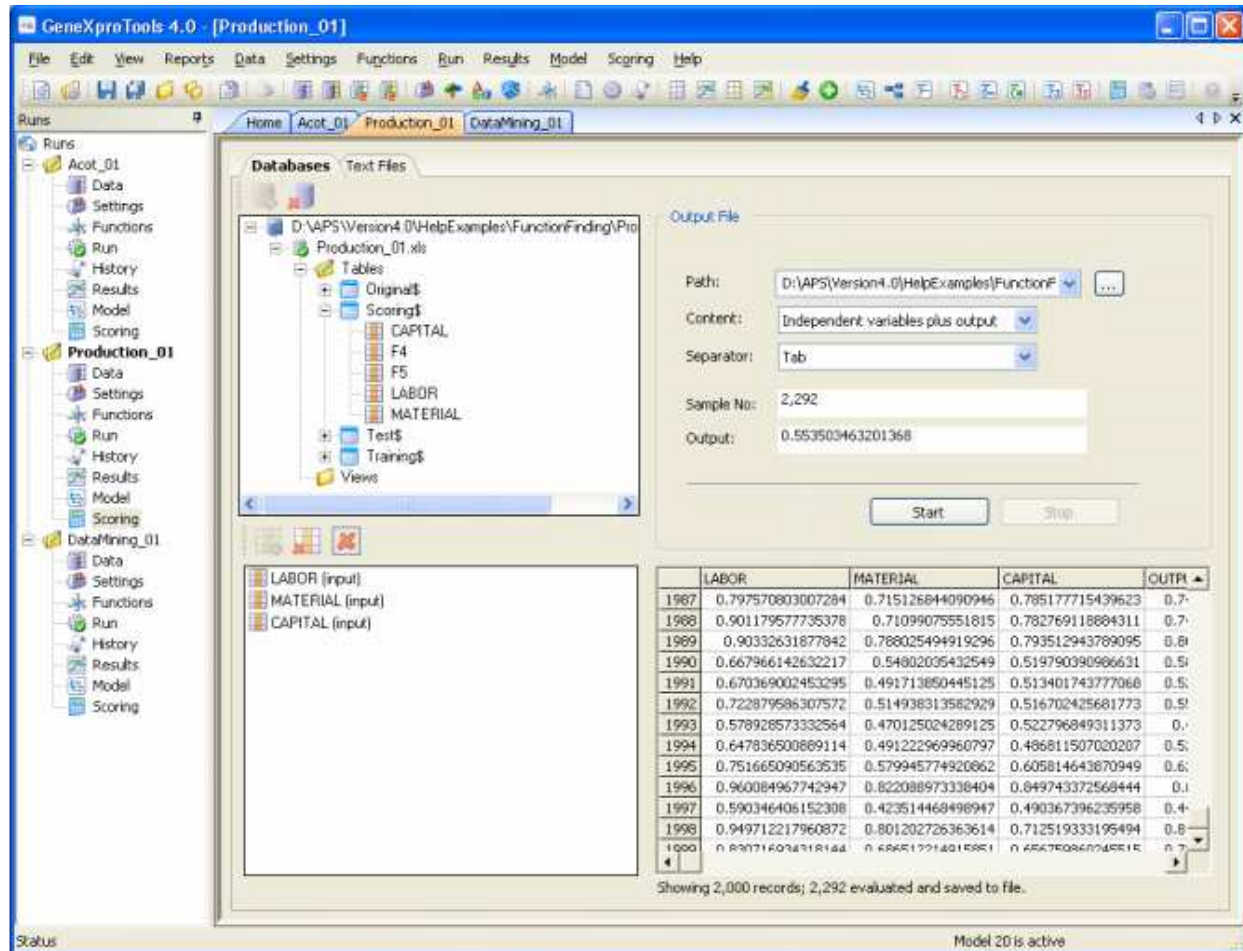


Making Predictions with a Model

Scoring from Excel and Databases

Once a model has been created it can be used hundreds or thousands of times to predict unknown behavior. The **scoring engine** of GeneXproTools allows you to do that without leaving the GeneXproTools modeling environment.

To score a database or Excel file, on the Scoring Menu select Databases or go to the Scoring Panel and select the Databases Tab.



The scoring engine of GeneXproTools 4.0 uses the [Java Script code](#) of your model to perform the computations since it already contains the code for the [UDFs](#) and [DDFs](#).

Language: Javascript ☐ Use Labels ☐ Text only Zoom: 100% Save Code

```
// Code generated by GeneXproTools 4.0 on 11/07/2006 23:57:39
// Training Samples: 300
// Testing Samples: 146
// Fitness Function: RRSE
// Training Fitness: 864.940137743324
// Training R-square: 0.977861370576141
// Testing Fitness: 854.354952286697
// Testing R-square: 0.974782317023882

function gepModel(d)
{
    var G1C0 = -5.373443;
    var G1C1 = 5.186767;
    var G2C0 = 4.433807;
    var G2C1 = -7.248138;
    var G3C0 = 3.755371;
    var G3C1 = 5.186767;

    var vTemp = 0.0;

    vTemp = ((G1C0+G1C1)/G1C0);
    vTemp += ((d[0]-UDF0())/DDF0(d[2],DDF0((G2C1/G2C1),d[1])));
    vTemp += d[1];

    return vTemp;
}

function DDF0(x0,x1)
{
    return (x0+x1)*(x0+x1);
}

function UDF0()
{
    var myValue = (terminals[0] + terminals[1]) / 2.0;
    return myValue;
}
```

The output is saved to a file which contains the predictions of your model for all the scores in the database. For small datasets (up to 16 variables and 2000 samples) GeneXproTools also shows the scoring results in the Scoring Panel.

Databases Text Files

D:\AP5\Version4.0\Help\examples\FunctionFinding\Pro

Production_01.xls

Tables

- Originals
- Scoring
- CAPITAL
- F4
- F5
- LABOR
- MATERIAL
- Test
- Training
- Views

LABOR (input)

MATERIAL (input)

CAPITAL (input)

Output File

Path: D:\AP5\Version4.0\Help\examples\FunctionF...

Content: Independent variables plus output

Separator: Tab

Sample No: 2,292

Output: 0.553503463201368

Start Stop

	LABOR	MATERIAL	CAPITAL	OUTPUT
1987	0.797570803007284	0.715126844090946	0.785177715439623	0.74586
1988	0.901179577735378	0.71099075551815	0.782769118684311	0.74027
1989	0.90332631877842	0.788025494919296	0.793512943789095	0.80818
1990	0.667966142632217	0.54802035432549	0.519790390986631	0.58371
1991	0.670369002453295	0.491713850445125	0.513401743777068	0.52542
1992	0.722879586307572	0.514938313582929	0.516702425681773	0.55350
1993	0.578928573332564	0.470125024289125	0.522796849311373	0.4945
1994	0.647836500889114	0.491222969960797	0.486811507020207	0.52315
1995	0.751665090563535	0.579945774920862	0.605814643870949	0.62033
1996	0.96084967742947	0.822088973338404	0.849743372568444	0.8344
1997	0.590346406152306	0.423514468498947	0.490367396235958	0.44172
1998	0.949712217960872	0.801202726363614	0.712519333195494	0.81666
1999	0.89714074718144	0.5861771401851	0.55768847045115	0.7763

Showing 2,000 records; 2,292 evaluated and saved to file.

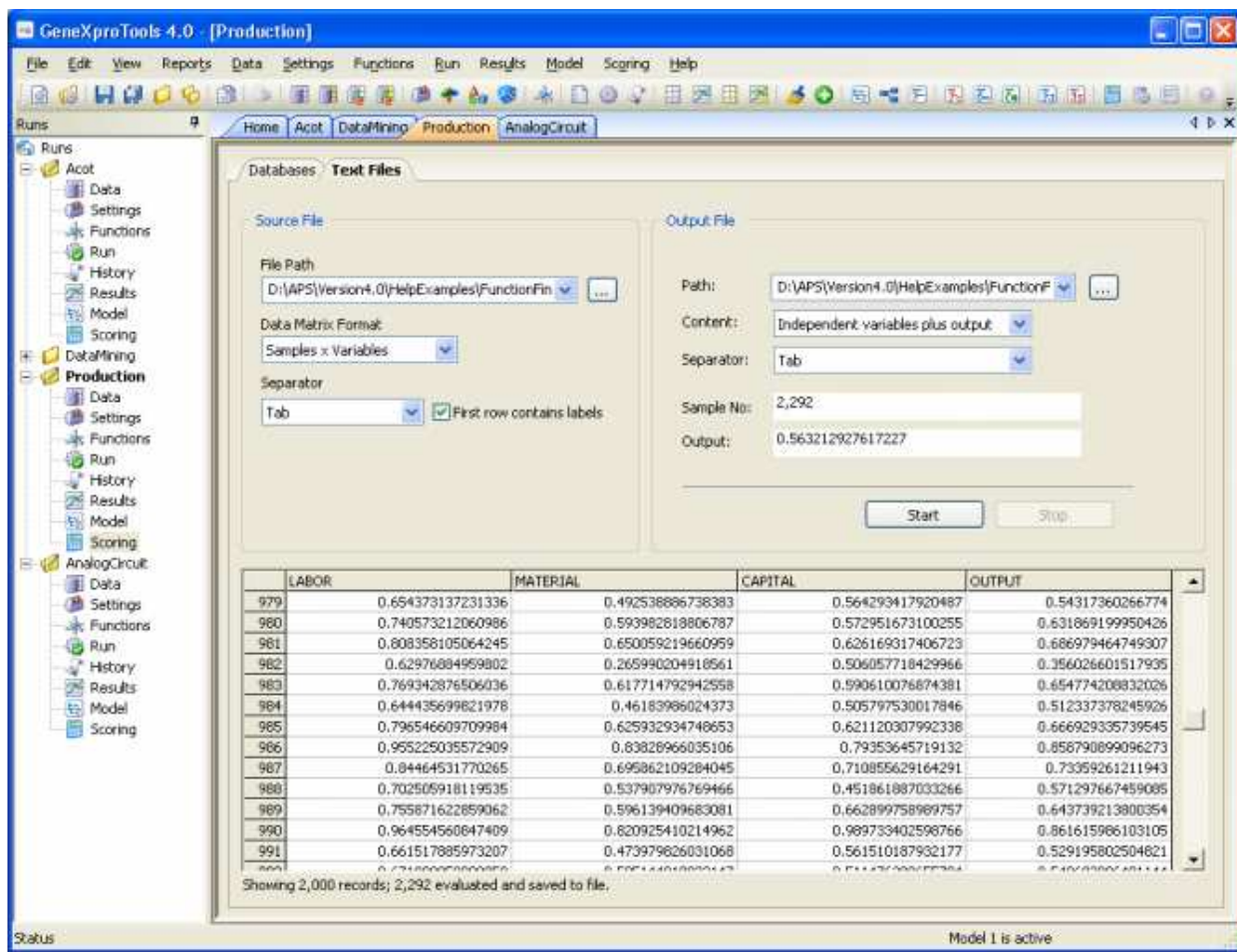


Making Predictions with a Model

Scoring from Text Files

Once a model has been created it can be used hundreds or thousands of times to predict unknown behavior. The **scoring engine** of GeneXproTools allows you to do that without leaving the GeneXproTools modeling environment.

For scoring data kept in text files, on the Scoring Menu select Text Files or go to the Scoring Panel and select the Text Files Tab.



The scoring engine of GeneXproTools 4.0 uses the [Java Script](#) code of your model to perform the computations since it already contains the code for the [UDFs](#) and [DDFs](#).

Language: Javascript ☐ Use Labels ☐ Text only Zoom: 100% Save Code

```
// Code generated by GeneXproTools 4.0 on 11/07/2006 23:57:39
// Training Samples: 300
// Testing Samples: 146
// Fitness Function: RRSE
// Training Fitness: 864.940137743324
// Training R-square: 0.977861370576141
// Testing Fitness: 854.354952286687
// Testing R-square: 0.974782317023882

function gepModel(d)
{
    var G1C0 = -5.373443;
    var G1C1 = 5.186767;
    var G2C0 = 4.433807;
    var G2C1 = -7.248138;
    var G3C0 = 3.755371;
    var G3C1 = 5.186767;

    var vTemp = 0.0;

    vTemp = ((G1C0+G1C1)/G1C0);
    vTemp += ((d[0]-UDF0())/DDF0(d[2],DDF0((G2C1/G2C1),d[1])));
    vTemp += d[1];

    return vTemp;
}

function DDF0(x0,x1)
{
    return (x0+x1)*(x0+x1);
}

function UDF0()
{
    var myValue = (terminals[0] + terminals[1]) / 2.0;
    return myValue;
}
```

The output is saved to a file which contains the predictions of your model for all the scores in the text file. For small datasets (up to 16 variables and 2000 samples) GeneXproTools also shows the scoring results in the Scoring Panel.

Databases Text Files

Source File

File Path: D:\APS\Version4.0\Help\examples\FunctionF ...

Data Matrix Format: Samples x Variables

Separator: Tab ☒ First row contains labels

Output File

Path: D:\APS\Version4.0\Help\examples\FunctionF ...

Content: Independent variables plus output

Separator: Tab

Sample No: 2,292

Output: 0.553503463201368

Start Stop

	LABOR	MATERIAL	CAPITAL	OUTPUT
1987	0.797570803007284	0.715126844090946	0.785177715439623	0.745860124335517
1988	0.901179577735378	0.71099075551815	0.782769118884311	0.740272671233412
1989	0.90332631877842	0.788025494919296	0.793512943789095	0.808187245837281
1990	0.667966142632217	0.54802035432549	0.519790390986631	0.583714519916069
1991	0.670369002453295	0.491713850445125	0.513401743777068	0.525425513025935
1992	0.722879986307572	0.514938313582929	0.516702425681773	0.553503463201368
1993	0.578928573332564	0.470125024289125	0.522796849311373	0.49455977578346
1994	0.647836500889114	0.491222969960797	0.486811507020207	0.523158403098033
1995	0.751665090563535	0.579945774920862	0.605814643870949	0.620336343118778
1996	0.960084967742947	0.822088973338404	0.849743372568444	0.83446603477471
1997	0.590346406152308	0.423514468498947	0.490367396235958	0.441721196745279
1998	0.949712217960872	0.801202726363614	0.712519333195494	0.816668616029998
1999	0.830716934318144	0.688512214915851	0.656759860245515	0.720585240805507
mean	0.85486473370545	0.474110708827944	0.478989827754438	0.708889245511767

Showing 2,000 records; 2,292 evaluated and saved to file.



Making Predictions with a Model

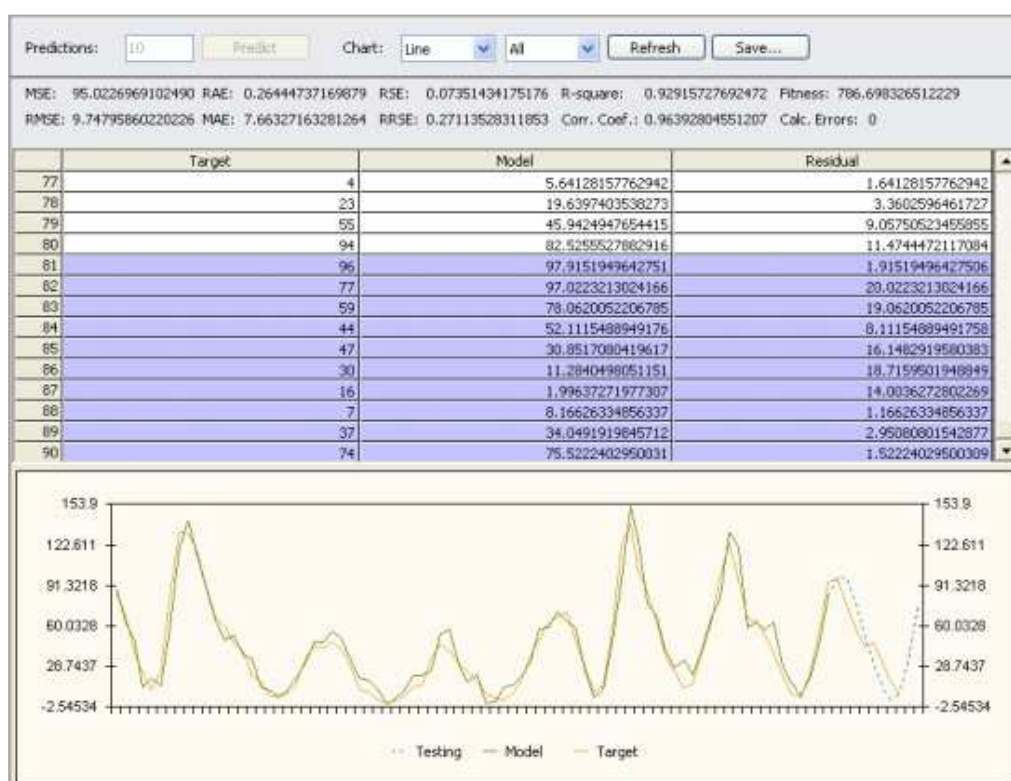
Making Predictions with Time Series Models

The algorithms GeneXproTools 4.0 uses for Time Series Prediction allow not only the creation of models but also the use of these models to make predictions.

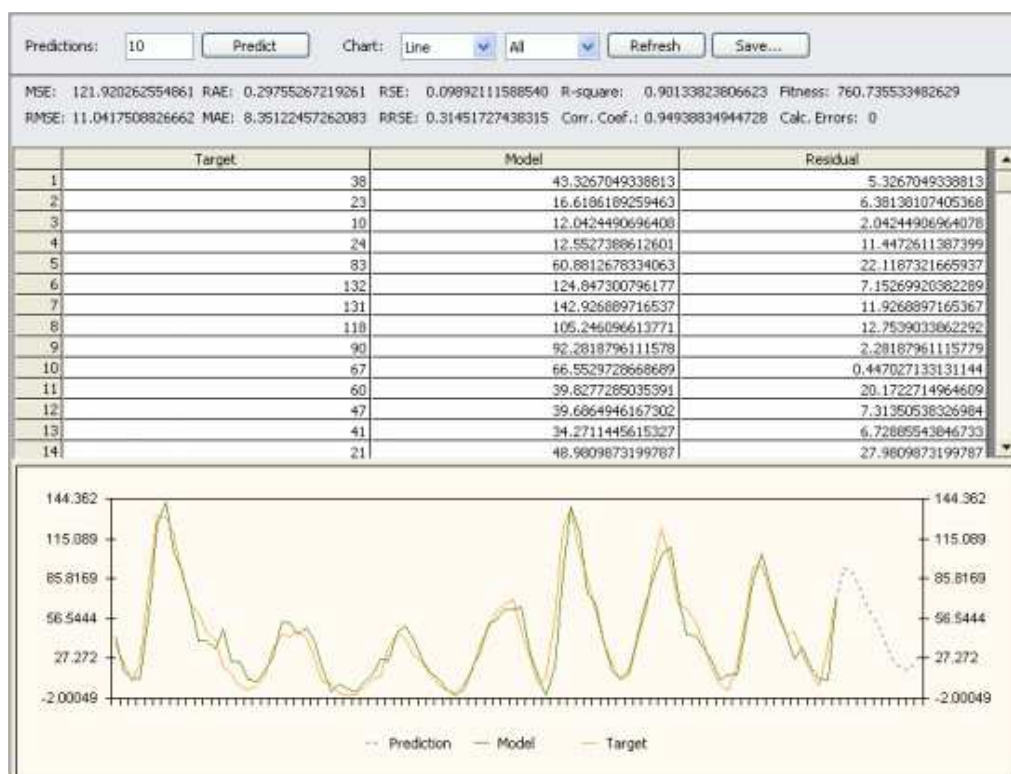
GeneXproTools 4.0 allows you to make **two kinds of predictions**: one for testing past known events and another for making predictions about the future or about unknown behavior. In both cases, though, predictions are made **recursively**, by evaluating the forecast at $t+1$, then using it to forecast $t+2$, and so on.

You must choose either one of these methods while loading your time series data, as this imposes some constraints on the restructuring of the time series for training. Namely, n testing records (the n last ones) are saved for testing and sometimes a small number of records from the top must be deleted. However, you can also change these parameters later in the Settings Panel -> General Settings Tab.

Thus, the first type of predictions (**Testing**) can be used for research or pre-evaluation purposes, as it allows you to test the forecasting capabilities of your model on a set of test observations that are shown in blue in the [Predictions Panel](#).



The second type of predictions (**Prediction**) is used obviously to predict unknown behavior, and GeneXproTools 4.0 allows you to venture into the future as far as you see fit, by setting the number of predictions you want to make and then click the Predict button on the [Predictions Panel](#).

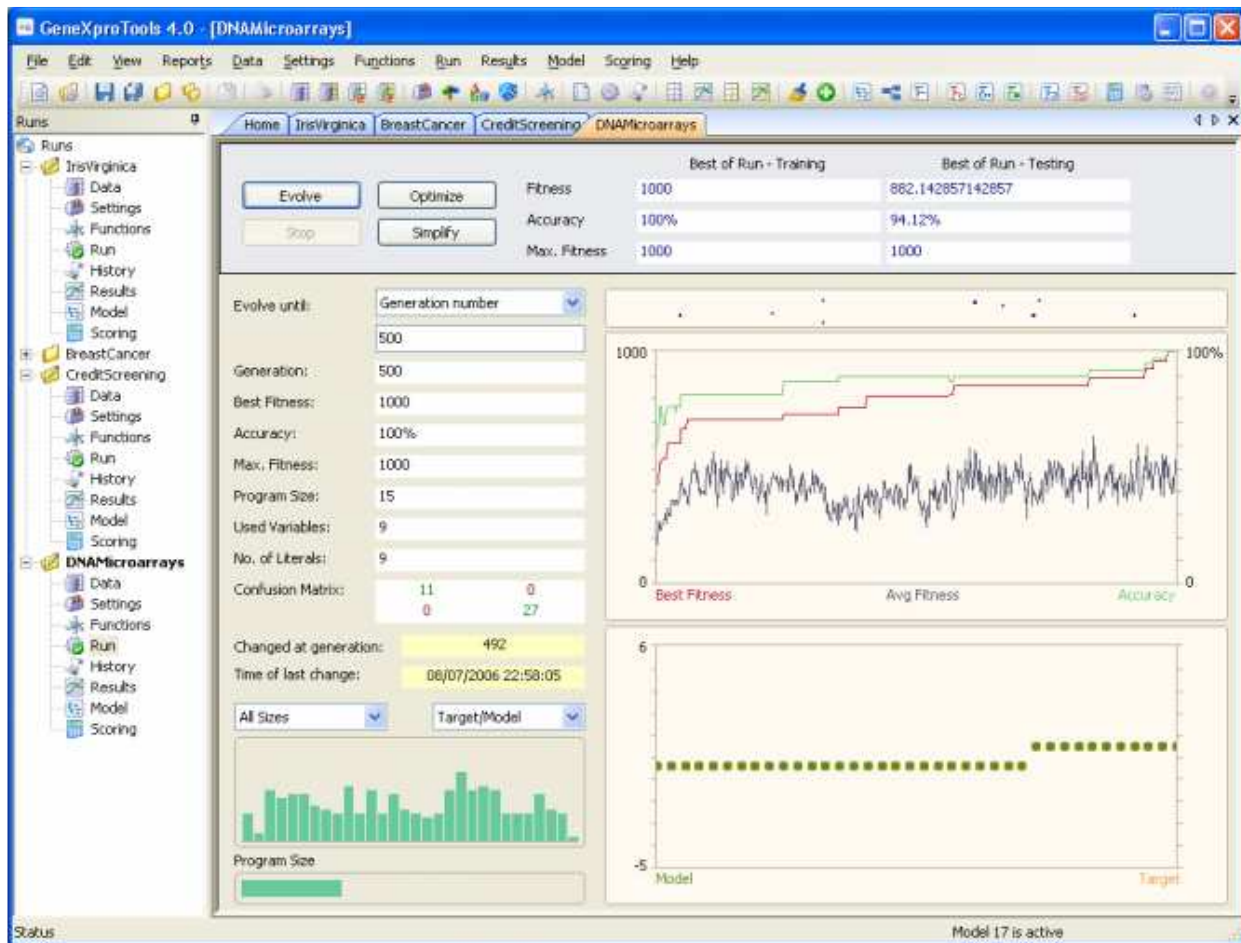




Settings and Features

This chapter summarizes all the settings and features of GeneXproTools 4.0 available for:

- [Function finding problems.](#)
- [Classification problems.](#)
- [Time series prediction problems.](#)
- [Logic synthesis problems.](#)

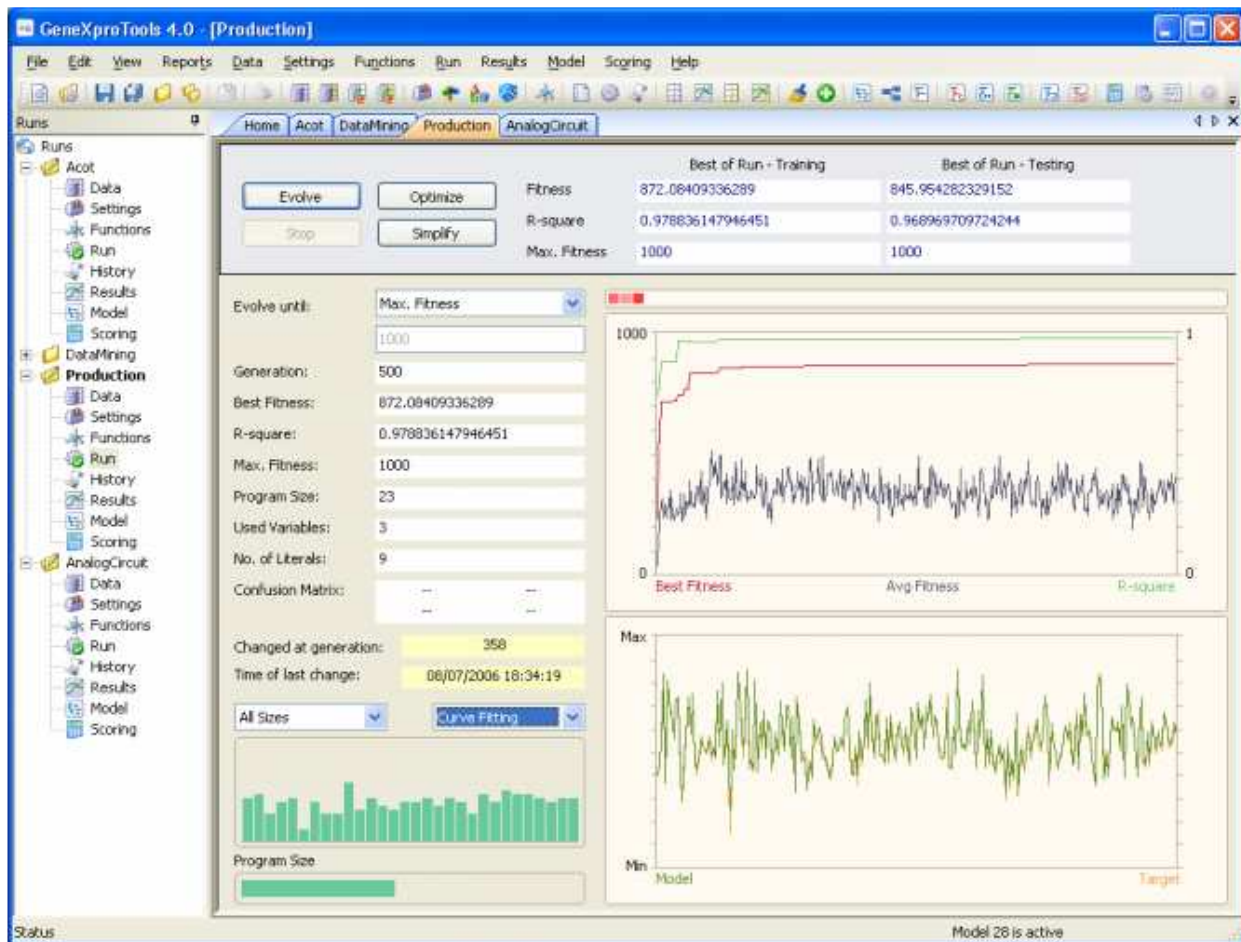


Settings and Features

Settings and Features for Function Finding Problems

The settings and features of GeneXproTools 4.0 available for Function Finding problems are organized as follows:

- [New Run Wizard](#)
- [Report Panel](#)
- [Data Panel](#)
- [Change/Add Dataset Engine](#)
- [Settings Panel](#)
- [Functions Panel](#)
- [Run Panel](#)
- [History Panel](#)
- [Results Panel](#)
- [Model Panel](#)
- [Change Seed Window](#)
- [Scoring Panel](#)
- [Menus](#)
- [Run Templates](#)
- [Sample Runs](#)

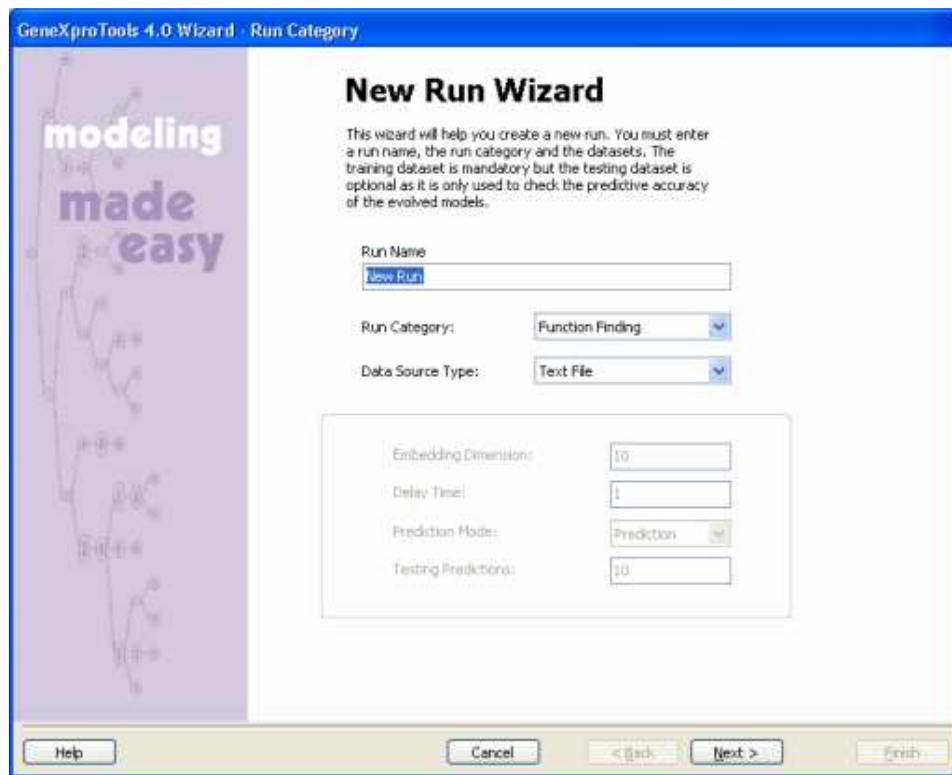


Settings and Features

New Run Wizard

The **New Run Wizard** of GeneXproTools 4.0 includes the following windows:

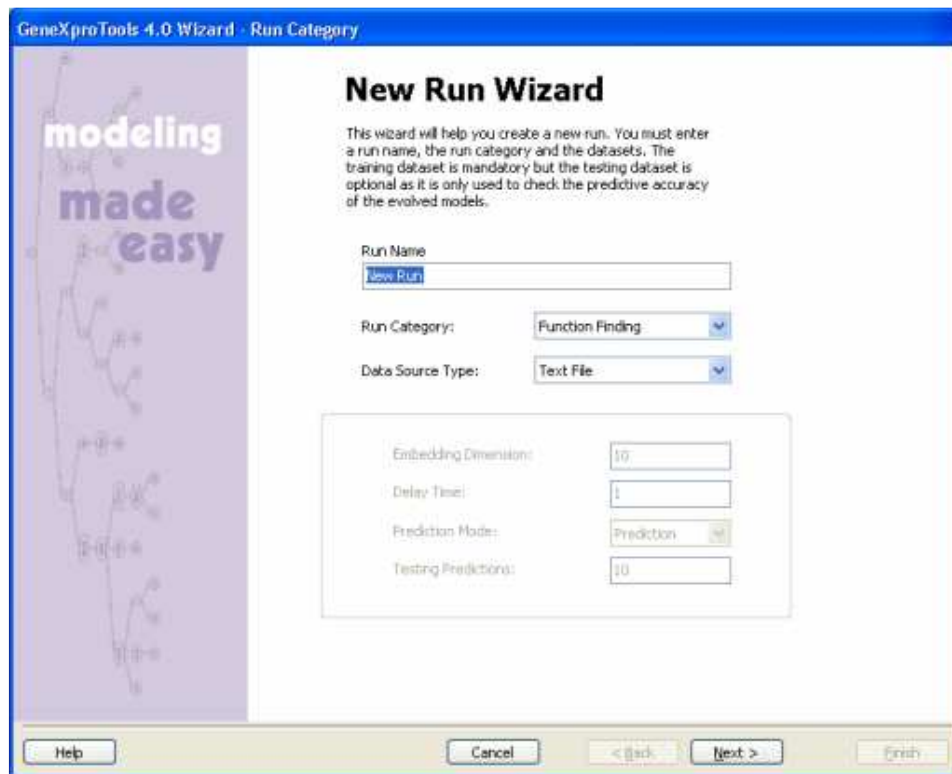
- [Run Category Window](#)
- [Training Data Window - Text Files](#)
- [Testing Data Window - Text Files](#)
- [Training Data Window - Excel & Databases](#)
- [Testing Data Window - Excel & Databases](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Run Category Window

The **Run Category Window** gives access to the following settings and features:

**Run Name**

The name of the .gep file in which the run will be saved.

Run Category

For function finding problems, you must obviously choose Function Finding.

Data Source Type

The type of data source you are using for modeling: Text Files or Excel & Databases.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Next Button

Takes you to the next window of the New Run Wizard.



Settings and Features

Training Data Window - Text Files

The **Training Data Window** gives access to the following settings and features:

GeneXproTools 4.0 Wizard - Training Data

Select a data source for training

File Path: D:\AP5\Version4.0\HelpExamples\FunctionFin

Data Matrix Format: Samples x Variables

Separator: Tab ☐ First row contains labels

File Name: Production-train.txt

Columns: 4

Rows: 200

☒ Show loaded data

d1	d2	D.V.
0.516655165912555	0.557043467001425	0.559270018369077
0.526656071996779	0.580672365829537	0.576228993499991
0.600690812447234	0.470057308963182	0.625533272885326
0.866811660478692	0.786951360964306	0.880267610267729
0.518628463221194	0.619300423887075	0.58430525559642
0.572150934313329	0.571731633157213	0.614308166260421
0.570113222811581	0.603254095375931	0.622167341254458
0.587350756615800	0.609451024538595	0.631889837915772
0.492953705527863	0.423604218262322	0.546844855175264
0.528547754024542	0.581649487701763	0.551903451776029
0.549876806347279	0.624971328794823	0.612441825083443
0.337519500366186	0.405527109223744	0.400102626024947
0.70410328467573	0.768829371426235	0.764637675177947
0.552834446384063	0.605017872838062	0.61479533537168
0.650883000688709	0.62330230860331	0.682851676803336
0.621271395204945	0.63722445443929	0.682709488543848
0.462479273220669	0.509051084027995	0.520885582393818
0.613401686022201	0.606943910056324	0.64664366838185
0.515567323959423	0.583829167652391	0.584149862238093
0.619615417400466	0.658870794792417	0.677391781780581
0.582377472475921	0.610119766648041	0.632756075041409
0.647251073777606	0.655848780704837	0.702986176801319
0.420422193840617	0.4368485034028854	0.470274507472525
0.548467607607012	0.5564757357964	0.575649297577445

Showing 4 columns and 200 rows: Loaded 4 columns and 200 rows

File Path

Click the button on the right of the list box to load the training file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies.

Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the training dataset.

Rows

GeneXproTools shows in this box the number of rows (samples) in the training dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the training set.

Clear Button

Clears the training set.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

Opens the Save As dialog box to save the new run.



Settings and Features

Testing Data Window - Text Files

The **Testing Data Window** gives access to the following settings and features:

File Path: D:\AP5\Version4.0\HelpExamples\FunctionFin

Data Matrix Format: Samples x Variables

Separator: Tab ☐ First row contains labels

File Name: Production-test.txt

Columns: 4

Rows: 246

☒ Show loaded data

d0	d1	d2	d.v.
0.76400603213758	0.590894222679104	0.651302362048791	0.668
0.824260336424283	0.712115997452169	0.698261407264005	0.738
0.663716192308473	0.474526514877375	0.523573879507317	0.558
0.716991613646645	0.749775086766838	0.737346320698492	0.75
0.578928573332564	0.470125024289125	0.522796849311373	0.498
0.702457163654926	0.557685195092099	0.560164715444461	0.58
0.781563080879931	0.593607062237543	0.606067778808582	0.64
0.631501366716096	0.439551338173223	0.508351189564081	0.51
0.886186909985177	0.730563617162146	0.764854658543964	0.77
0.899752141161488	0.742449965006724	0.70842665002423	0.76
0.793077986069546	0.643722774648507	0.627735948379581	0.678
0.854296784467984	0.674113622696059	0.688556198859948	0.728
0.630380632898429	0.465777795176287	0.485511673500227	0.508
0.639364684498696	0.472980298395411	0.513983771633166	0.51
0.740630929159615	0.604410397053559	0.585978144930915	0.62
0.832449521644299	0.683495734433642	0.648291254340763	0.71
0.653769812996165	0.457548920121502	0.477163584988068	0.52
0.791461750640124	0.597766453831909	0.584060347954123	0.658
0.913275738166747	0.763579524741268	0.750758436387552	0.808
0.733193727707238	0.556625719583768	0.481886576772093	0.588
0.673942740854614	0.53914808867877	0.463616448448966	0.56
0.677068260455278	0.44766550611768	0.496164130157592	0.52
0.648830375504696	0.476106768952288	0.527087459849945	0.53
0.541700000777594	0.570020071554004	0.530010060071008	0.50

Showing 4 columns and 246 rows; Loaded 4 columns and 246 rows.

File Path

Click the button on the right of the list box to load the testing file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies.

Separator

GeneXproTools identifies automatically the separator in the testing file and shows it in the combo box under this label. And if the variables in your testing data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the testing dataset.

Rows

GeneXproTools shows in this box the number of rows (samples) in the testing dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the testing set.

Clear Button

Clears the testing set.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

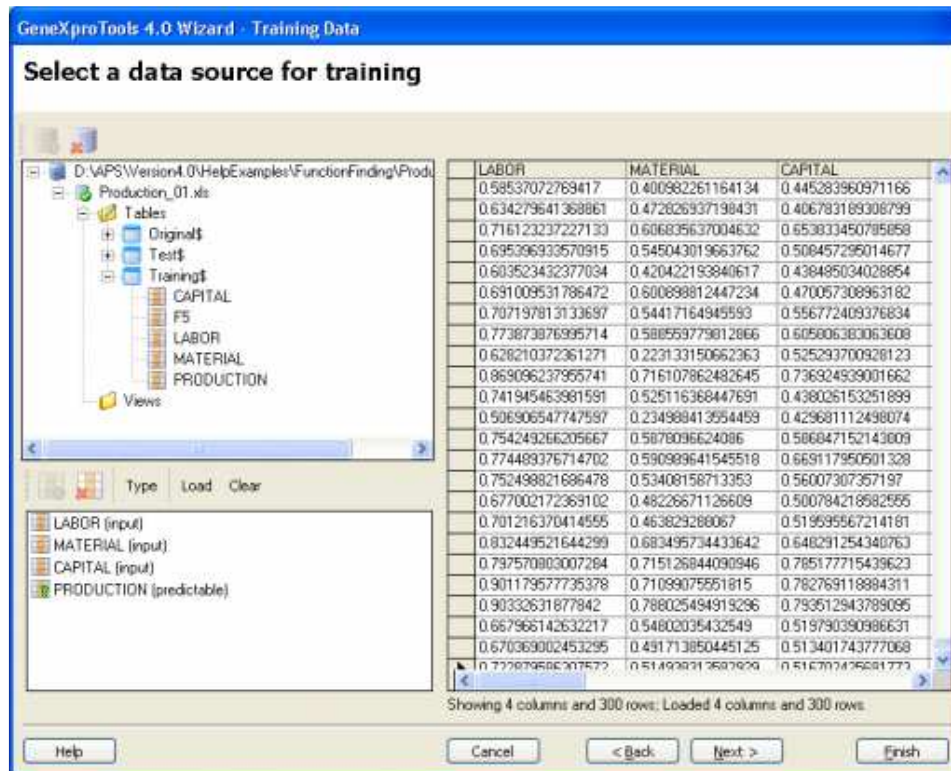
Opens the Save As dialog box to save the new run.



Settings and Features

Training Data Window - Excel & Databases

The **Training Data Window** gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Training Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Training Data Window.

Training Data Window

You prepare your training data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or

down with the mouse.

Training Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

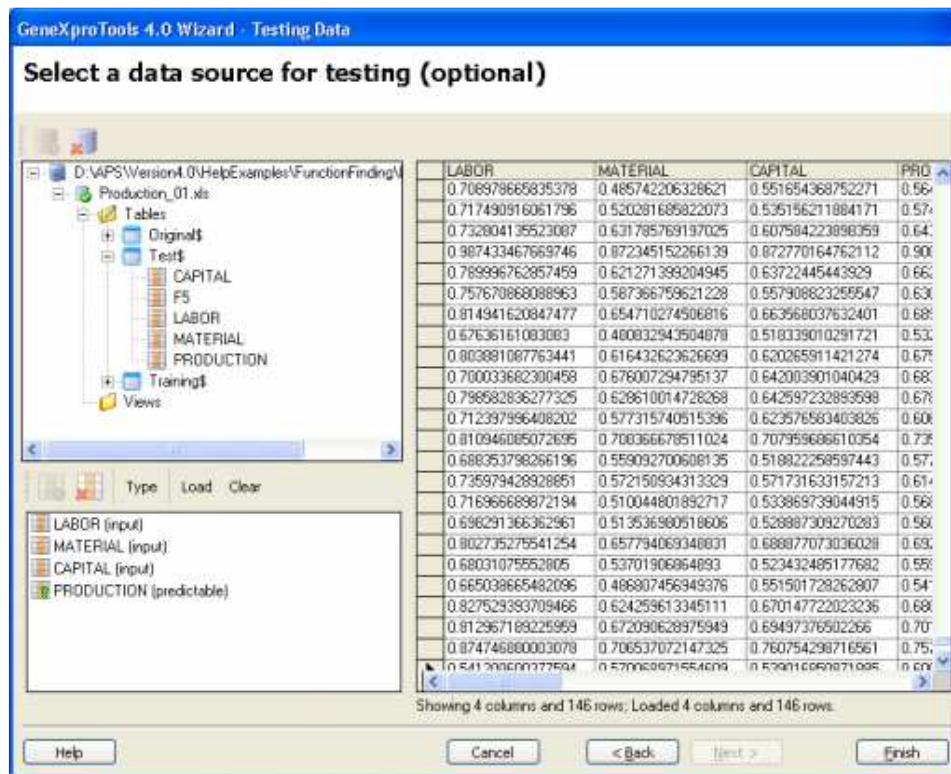
Opens the Save As dialog box to save the new run.



Settings and Features

Testing Data Window - Excel & Databases

The **Testing Data Window** gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Testing Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Testing Data Window.

Testing Data Window

You prepare your testing data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse.

Testing Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

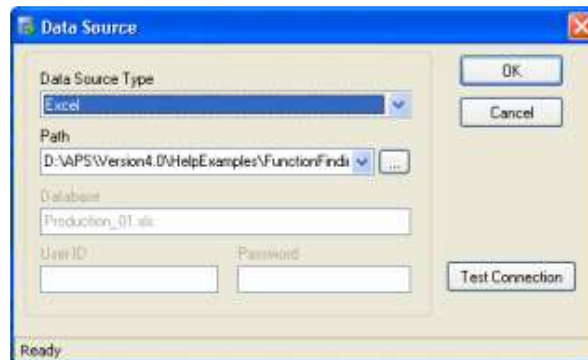
Opens the Save As dialog box to save the new run.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:

**Driver**

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.

Microsoft Access: path to the database file.

Microsoft SQL Server: server name or IP address.

Oracle: server name or IP address.

MySQL: server name or IP address.

DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.

Oracle - Name and database instance (HOST/INSTANCE).

MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

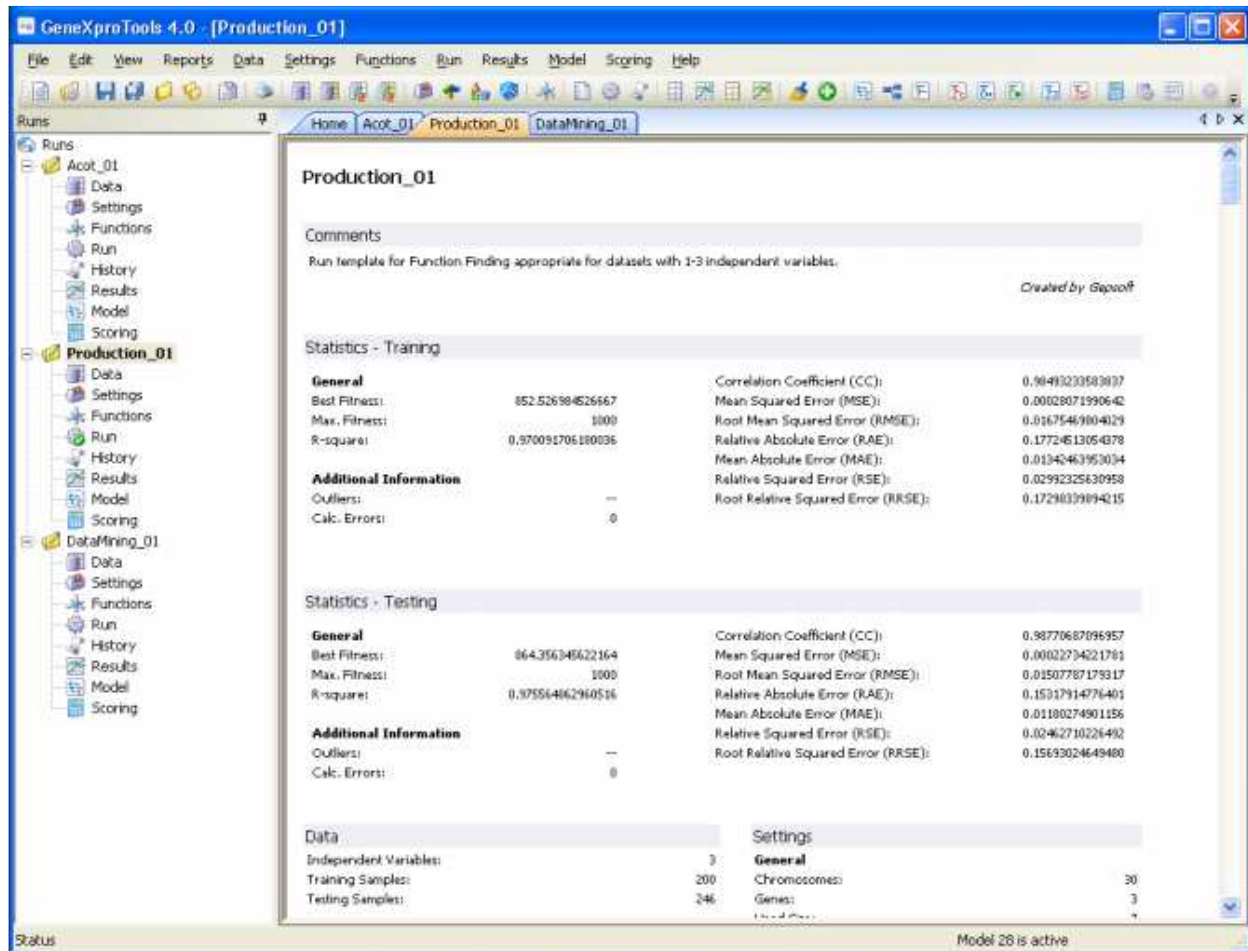
Cancels all the action on the data source connection.



Settings and Features

Report Panel

The **Report Panel** of GeneXproTools 4.0 keeps a summary of all the relevant information of a run, and is organized into the following headings:



Comments

This heading is usually used to give a small description of the problem, the objectives, the kind of data used, their source, etc. Both the comments and the author's name are introduced by choosing Properties on the File Menu.

Statistics – Training

Under this heading are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the training set: [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), and [R-square](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, number of calculation errors (they should always be zero, unless the seed was modified or the training set was changed after the model had been created), and number of outliers (when the fitness function is one of the following: [Relative with SR](#), [Relative/Hits](#), [Absolute with SR](#), and [Absolute/Hits](#)).

A note of warning, though: All the parameters listed here refer to the last evaluated model in the [Results Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	24	9	d0(5), d1(3), d2(1)	847.262339126746	859.152192900566	0.960006472679246	0.937587454532183
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Atan } X3 \cdot d0 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d0 \cdot d2 \cdot d0} \\ & + \\ & \sqrt{X2} / \sqrt{\text{Cos } d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d0 \cdot d0} \\ & + \\ & X3 \cdot \text{Atan } d0 \cdot d0 \cdot \text{Cos } \text{Ln } \sqrt{d0 \cdot d2 \cdot d2 \cdot d0 \cdot d1 \cdot d1 \cdot d0 \cdot d1} \end{aligned}$							
12	25	9	d0(4), d1(4), d2(1)	852.285418790807	864.046489990001	0.970670596225273	0.976046016260914
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Atan } X3 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0} \\ & + \\ & \sqrt{X2} / \sqrt{\text{Cos } d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d2 \cdot d0} \\ & + \\ & \text{Atan } X3 \cdot \text{Atan } d0 \cdot d0 \cdot \text{Cos } \text{Exp } d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d1 \cdot d2 \cdot d1 \end{aligned}$							
23	25	9	d0(3), d1(4), d2(2)	864.701706874743	864.588575207565	0.97254807136764	0.976472602618102
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Atan } X3 \cdot d1 \cdot d2 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d2 \cdot d0} \\ & + \\ & \sqrt{X2} / \sqrt{\text{Cos } d1 \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d0 \cdot d0 \cdot d0} \\ & + \\ & \text{Atan } X3 \cdot \text{Atan } d0 \cdot d0 \cdot \text{Cos } \text{Exp } d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d2 \cdot d0 \cdot d0 \end{aligned}$							

Statistics – Testing

Under this heading are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the testing set, that is, the efficiency of your model at predicting unknown behavior: [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), and [R-square](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, number of calculation errors, and number of outliers (when the fitness function is one of the following: [Relative with SR](#), [Relative/Hits](#), [Absolute with SR](#), and [Absolute/Hits](#)).

Note again that all the parameters listed here refer to the last evaluated model in the [Results Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	11	4	d0(1), d1(2), d2(1)	850.429030313036	870.616009249946	0.976590014908023	0.982091633615242
$\begin{aligned} & d1 \cdot \sqrt{\text{Exp} - \text{Atan } X3 \cdot d2 \cdot c1 \cdot c1 \cdot c1 \cdot d0 \cdot d2 \cdot d0 \cdot d0 \cdot d0 \cdot c1} \\ & + \\ & \text{Atan } X3 / \text{Atan } \text{Exp} \cdot 3 \cdot \text{Rt } d1 \cdot * d2 \cdot d0 \cdot d2 \cdot d2 \cdot c1 \cdot d0 \cdot d0 \cdot c1 \cdot c0 \end{aligned}$							
Numerical Constants: Gene 1 c0 = 7.749878 c1 = 8.115112 Gene 2 c0 = -9.974549 c1 = -6.347351							
2	10	3	d1(2), d2(1)	859.910797156278	872.16215754529	0.975751560710861	0.980563947942955
$\begin{aligned} & d1 \cdot \sqrt{\text{Exp} - d2 \cdot d2 \cdot X3 \cdot d2 \cdot c1 \cdot c1 \cdot c1 \cdot d0 \cdot d2 \cdot d0 \cdot d0 \cdot d0 \cdot c1} \\ & + \\ & \text{Atan } X3 / \text{Atan } \text{Exp} \cdot 3 \cdot \text{Rt } d1 \cdot X2 \cdot d2 \cdot d0 \cdot c1 \cdot d2 \cdot c1 \cdot d0 \cdot d0 \cdot c0 \cdot c0 \end{aligned}$							
Numerical Constants: Gene 1 c0 = 7.749878 c1 = 8.115112 Gene 2 c0 = -9.974549 c1 = -6.347351							

Data

Under this heading are listed the number of samples in the training set actually used during training (remember that GeneXproTools 4.0 allows you to choose a smaller sub-set of the training set to use during training), the number of testing samples used to test the generalizing capabilities of your model (again, not all the samples on the testing set were necessarily used to test your model), and the number of terminals or independent variables in your data.

Program Structure

The program size, the number of literals, and a detailed list of all used variables and respective weights are shown here. This information refers to the active model highlighted in yellow in the History heading.

Program Structure	
Program Size:	11
Literals:	4
Used Variables:	d0(1), d1(2), d2(1)

Function Set

Under this heading are listed all the functions used to create your model, together with the symbols used to represent them in [Karva notation](#), their weights and arities.

Function Set			
Function	Symbol	Weight	Arity
Addition	+	1	2
Subtraction	-	1	2
Multiplication	*	1	2
Division	/	1	2
Square root	Sqrt	1	1
Exponential	Exp	1	1
Natural logarithm	Ln	1	1
x to the power of 2	X2	1	1
x to the power of 3	X3	1	1
Sine	Sin	1	1
Cosine	Cos	1	1
Cube root	3Rt	1	1
Arctangent	Atan	1	1

Note that the highest value of arity in your function set determines the structure of each gene, making the length of the [tail](#) t grow according to the equation:

$$t = h (n-1) + 1$$

where h is the length of the [head](#) and n is maximum arity.

Settings

Under this heading there are five entries: General Settings, Complexity Increase, Fitness Function, Genetic Operators, and Numerical Constants. Under General Settings are listed the number of chromosomes, number of genes, head size, tail size, Dc size, gene size, and linking function. Under Complexity Increase are listed the number of Generations Without Change, the Number of Tries, and Max. Complexity. The name of the fitness function and all the relevant parameters for its design are given below the entry Fitness Function. Under Genetic Operators are listed the rates of all the genetic operators used to create genetic variation. And, finally, all the parameters relevant to the manipulation of random numerical constants are listed under Numerical Constants.

Settings:	
General	
Chromosomes:	30
Genes:	3
Head Size:	8
Tail Size:	9
Dc Size:	9
Gene Size:	26
Linking Function:	Addition
Complexity Increase	
Generations without Change:	2000
Number of Tries:	3
Max. Complexity:	5
Fitness Function	
Error Type:	RRSE
Precision:	--
Selection Range:	--
Genetic Operators	
Mutation Rate:	0.044
Inversion Rate:	0.1
IS Transposition Rate:	0.1
RIS Transposition Rate:	0.1
One-Point Recombination Rate:	0.3
Two-Point Recombination Rate:	0.3
Gene Recombination Rate:	0.1
Gene Transposition Rate:	0.1
Numerical Constants	
Constants per Gene:	2
Data Type:	Floating-Point
Lower Bound:	-10
Upper Bound:	10
RNC Mutation:	0.01
Dc Mutation:	0.044
Dc Inversion:	0.1
Dc IS Transposition:	0.1

Data Sources

Under this heading the paths to the input files (training and testing files) used to create your model are given so that their origin could be easily identified.

File Path

Under this heading the path to the gep file that was used to save your model is given so that it can be easily traced for future reference.

History

Under this heading are listed all the best-of-generation models, including obviously the best-of-run model, discovered during the evolutionary process. Note, however, that the notation "best-of-run" and "best-of-generation" refers to the performance of these models on the training set, which could be slightly different on the testing set. This is why GeneXproTools 4.0 allows you to analyze the performance of all these models on the testing set and, if appropriate, choose the best model based on these results. GeneXproTools 4.0 gives you the possibility of highlighting in yellow one of these models (the active model) and we advise you to use this feature to highlight the best one. Note also that, for a quick reference, not only the program structure (program size, number of literals, and list of used variables and respective weights) but also the fitness and the R-square of all these models both on the training and testing sets are also given, although, for the testing set, they must be evaluated in the [History Panel](#).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	24	9	di(5), di(3), di(1)	847.282339126748	859.152192900566	0.960006472679246	0.973587484532183
/.X3/.Sqrt.+Atan.X3.d0.d0.d2.d1.d0.d0.d2.d0 + /.X2/.Sqrt.+Cos.d1.d0.d0.d1.d2.d2.d2.d0.d0 + X3.Atan.d0.d0.Cos.Ln.Sqrt.d0.d2.d2.d0.d1.d1.d0.d1							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
12	25	9	di(4), di(4), di(1)	852.285419790807	864.046489990801	0.970678596225273	0.976046816260914
/.X3/.Sqrt.+Atan.X3.d1.d0.d2.d1.d0.d2.d1.d0 + /.X2/.Sqrt.+Cos.d1.d0.d0.d1.d2.d2.d2.d2.d0 + Atan.X3.Atan.d0.d0.Cos.Exp.d1.d2.d2.d2.d1.d1.d2.d1							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
23	25	9	di(3), di(4), di(2)	854.701706874743	864.588575207565	0.97254807136764	0.976472602618102
/.X3/.Sqrt.+Atan.X3.d1.d2.d2.d1.d0.d2.d2.d0 + /.X2/.Sqrt.+Cos.d1.d0.d0.d1.d2.d2.d0.d0.d0 + Atan.X3.Atan.d0.d0.Cos.Exp.d1.d2.d2.d2.d1.d2.d0.d0							

For brevity sake, all the best-of-generation models are represented in the compact [Karva notation](#), but you can translate any one of them into a more canonical representation by activating any of these models in the [History Panel](#) and then translate it into the language of your choice in the [Model Panel](#).

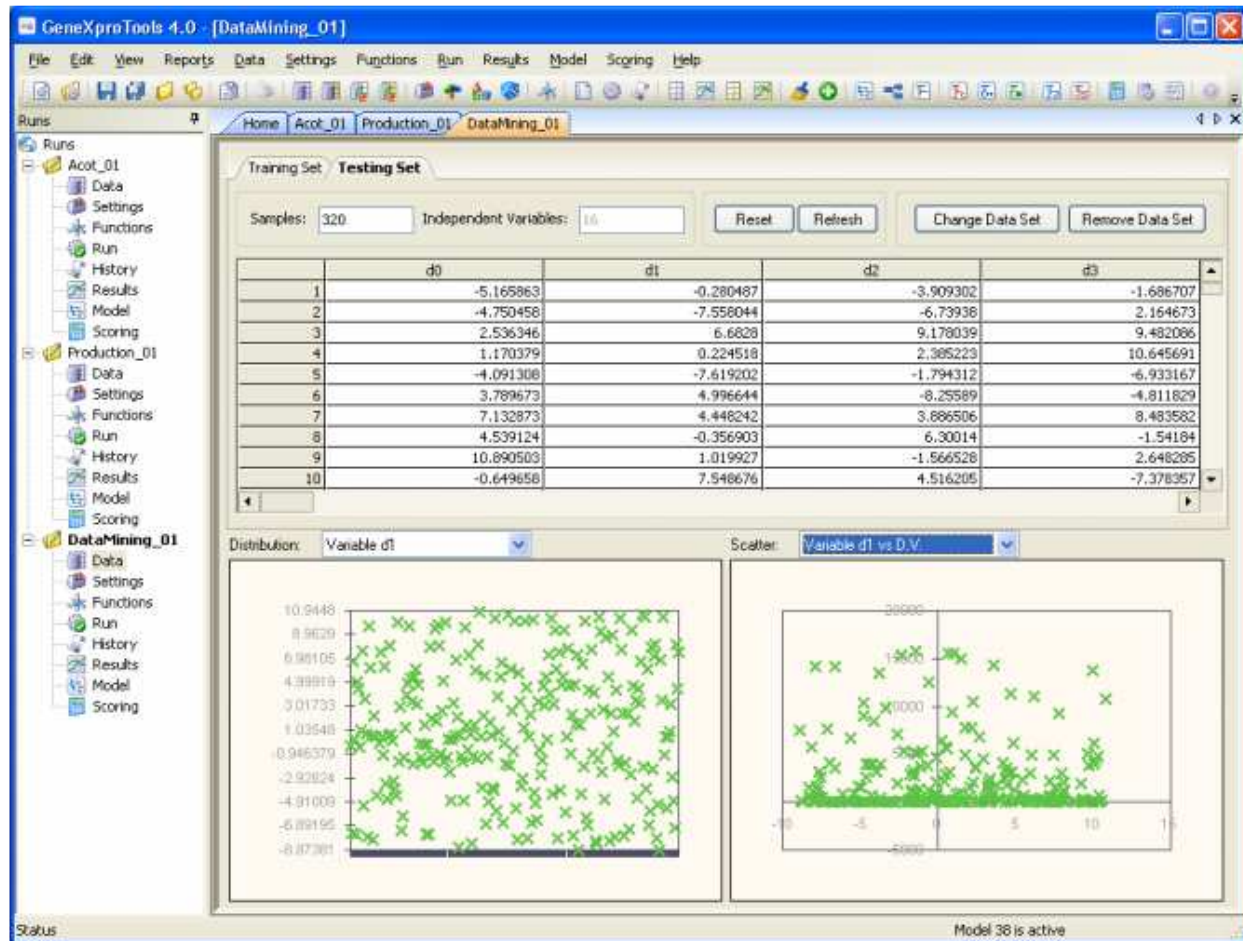


Settings and Features

Data Panel

The **Data Panel** of GeneXproTools 4.0 includes the following tabs:

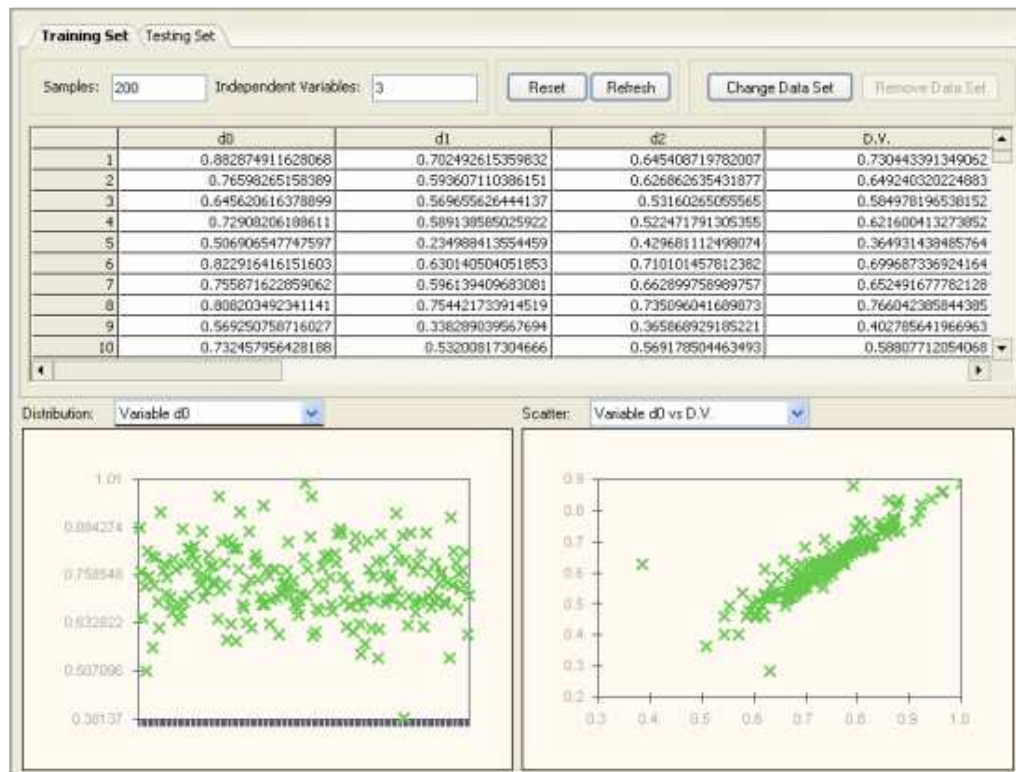
- [Training Set Tab](#)
- [Testing Set Tab](#)



Settings and Features

Training Set Tab

The **Training Set Tab** gives access to the following settings and features:



Samples

Allows you to select a subset of samples for training.

Independent Variables

This box is informative only and shows the number of independent variables in the training set.

Reset Button

Reverts to the total number of training samples in your data.

Refresh Button

Refreshes the panel after changing the training data set.

Change Data Set Button

Allows you to choose another training set for modeling. Changing a training set by another with the same structure, that is, the same number of independent variables, is very useful for model refreshing and we advise you to pre-test your models in the [History Panel](#) before proceeding with refreshing. In addition, GeneXproTools 4.0 also allows you to replace the training set by another one with a different structure. Obviously, in this case, all the models in the run history will be invalidated and the testing set, if there is one, will be automatically removed.

Table

Shows all the variables in the training set. The entire table can be copied to the clipboard by choosing Copy on the Edit Menu. You can also copy selected columns to the clipboard with the mouse right-click. This is particularly useful in huge datasets for selecting relevant variables identified in particularly good models.

Distribution

Allows you to visualize the distribution of values for each variable in the training set. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to

draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.

Scatter

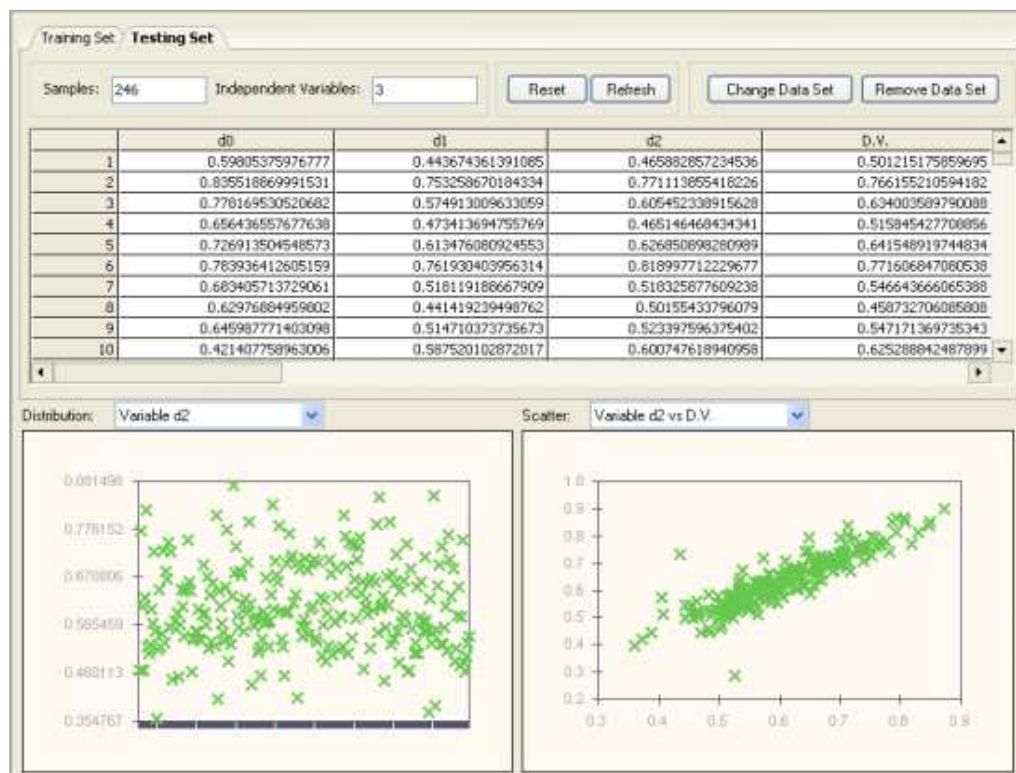
Allows you to visualize the scatter plot of each independent variable in the training set against the dependent variable. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.



Settings and Features

Testing Set Tab

The **Testing Set Tab** gives access to the following settings and features:



Samples

Allows you to select a subset of samples for testing.

Independent Variables

This box is informative only and shows the number of independent variables in the testing set.

Reset Button

Reverts to the total number of testing samples in your data.

Refresh Button

Refreshes the panel after changing the testing data set.

Change Data Set Button

Allows you to choose another testing set for evaluating the generalizing capabilities of your models. The new testing set must obviously have the same structure of the training set, that is, the same number of independent variables.

Remove Data Set Button

Allows you to remove the testing set (you can always add it later by selecting Add Testing Data on the Data Menu).

Table

Shows all the variables in the testing set. The entire table can be copied to the clipboard by choosing Copy on the Edit Menu. You can also copy selected columns to the clipboard with the mouse right-click. This is particularly useful in huge datasets for selecting relevant variables identified in particularly good models.

Distribution

Allows you to visualize the distribution of values for each variable in the testing set. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.

Scatter

Allows you to visualize the scatter plot of each independent variable in the testing set against the dependent variable. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.



Settings and Features

Change/Add Dataset Engine

The **Change/Add Dataset Engine** of GeneXproTools 4.0 includes the following windows:

- [Change Training Data Window - Text Files](#)
- [Change/Add Testing Data Window - Text Files](#)
- [Change Training Data Window - Excel & Databases](#)
- [Change/Add Testing Data Window - Excel & Databases](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Change Training Data Window - Text Files

The **Change Training Data Window** allows you to change the training set and gives access to the following settings and features:

d0	d1	d2
0.672587425376084	0.516695165912555	0.557049467001425
0.715402374770146	0.526656071996779	0.560672365829537
0.691009531786472	0.600899812447234	0.470057308963182
0.791992949256483	0.866811660478692	0.786951360964306
0.698467138913523	0.518628463221194	0.619300423887075
0.735979428928851	0.572150934313329	0.571731633157213
0.760697261308288	0.570113222811591	0.603254095375931
0.76809664251092	0.587360756615803	0.609451024538595
0.698635322368081	0.492953705527863	0.423604218262322
0.681218693376227	0.528547754024542	0.581649487701763
0.740887325341902	0.549876806947279	0.624971328794823
0.542228376455329	0.337519500366186	0.405527109223744
0.911397251575951	0.70410328467573	0.768829371426235
0.734231917365668	0.552834446384063	0.605017872838062
0.820884956989202	0.650883000688709	0.62330230860331
0.789996762857458	0.621271395204945	0.63722445443829
0.666441482099157	0.462479273220669	0.509051084027995
0.762868137562726	0.613401686022201	0.606943910056324
0.722793510261045	0.515567323959423	0.583829167652391
0.770735160895059	0.619615417400466	0.658870794792417
0.73293090107082	0.582377472475921	0.610119766648041
0.818597092481414	0.647251073777606	0.655848780704837
0.603523432377034	0.420422193840617	0.438495034028854
0.7051645770277074	0.548462507507012	0.5504757267964

File Path

Click the button on the right of the list box to load the training file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies.

Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the training dataset. You can replace the old training set by another one with a different number of variables, but obviously, in this case, all the models in the run history will be invalidated and the testing set (if there is one) will be automatically removed by GeneXproTools.

Rows

GeneXproTools shows in this box the number of rows (samples) in the training dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns on the table of the Change Training Data Window, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the training set.

Clear Button

Clears the training set.

Cancel Button

Cancels all the action on the replacement of the training set.

Finish Button

Saves the new training set.



Settings and Features

Change/Add Testing Data Window - Text Files

The **Change/Add Testing Data Window** allows you to change the testing set and gives access to the following settings and features:

File Path

Click the button on the right of the list box to load the testing file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies.

Separator

GeneXproTools identifies automatically the separator in the testing file and shows it in the combo box under this label. And if the variables in your testing data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the testing dataset. Your new testing set must have the same number of variables as the training set. If you want to change the number of variables, you must first [change the training set](#).

Rows

GeneXproTools shows in this box the number of rows (samples) in the testing dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns on the table of the Change Testing Data Window, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the testing set.

Clear Button

Clears the testing set.

Cancel Button

Cancels all the action on the replacement of the testing set.

Finish Button

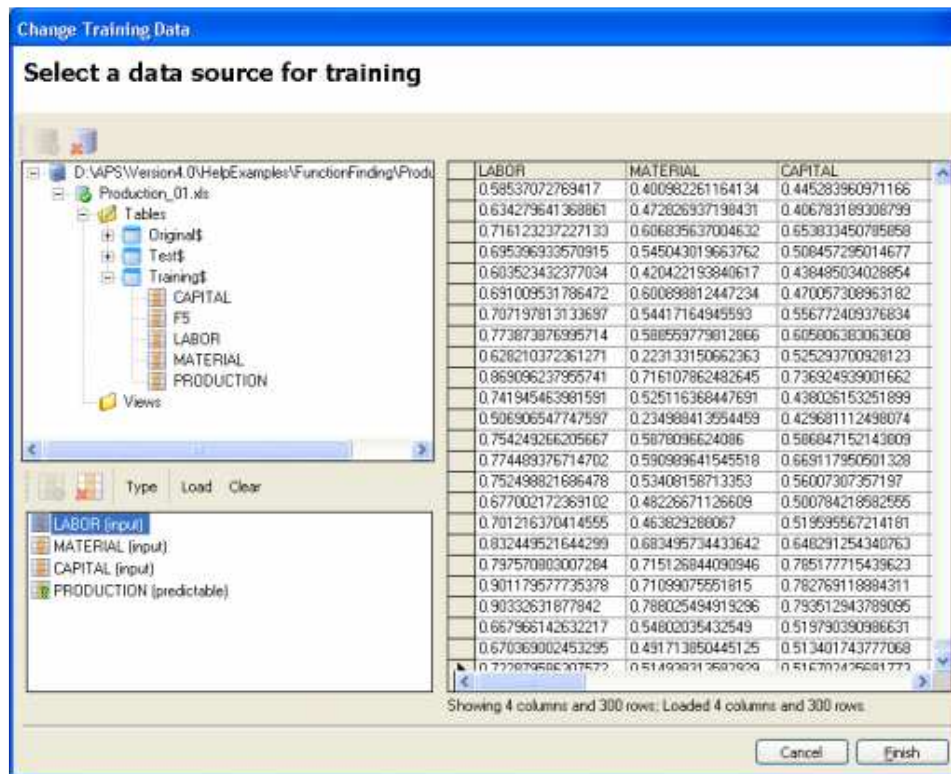
Saves the new testing set.



Settings and Features

Change Training Data Window - Excel & Databases

The **Change Training Data Window** allows you to change the training set and gives access to the following settings and features:

**Add Data Source Icon**

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Training Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Training Data Window.

Training Data Window

You prepare your training data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. GeneXproTools allows you to replace the old training set by another one with a different number of variables, but obviously, in this case, all the models in the run history will be

invalidated and the testing set (if there is one) will be automatically removed by GeneXproTools.

Training Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns on this table, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Cancel Button

Cancels all the action on the replacement of the training set.

Finish Button

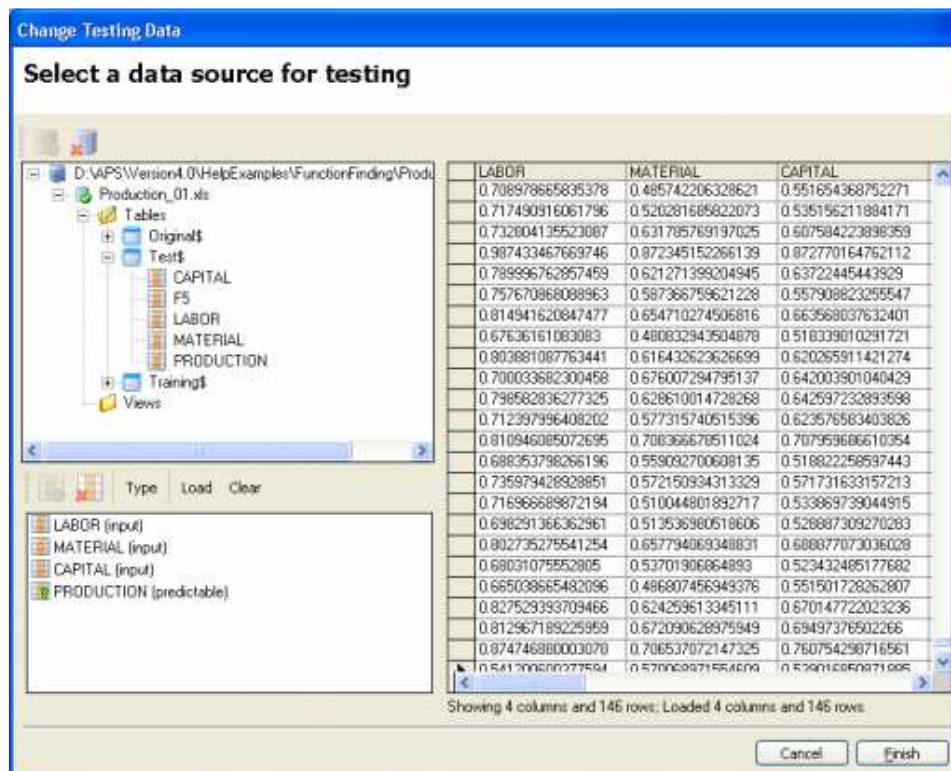
Saves the new training set.



Settings and Features

Change/Add Testing Data Window - Excel & Databases

The **Change/Add Testing Data Window** allows you to change the testing set (or add a new one) and gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Testing Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Testing Data Window.

Testing Data Window

You prepare your testing data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Your new testing set must have the same number of variables as the training set. If you want to change the number of variables, you must first [change the training set](#).

Testing Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns on this table, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Cancel Button

Cancels all the action on the replacement of the testing set.

Finish Button

Saves the new testing set.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:



Driver

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.

Microsoft Access: path to the database file.

Microsoft SQL Server: server name or IP address.

Oracle: server name or IP address.

MySQL: server name or IP address.

DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.

Oracle - Name and database instance (HOST/INSTANCE).

MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

Cancels all the action on the data source connection.



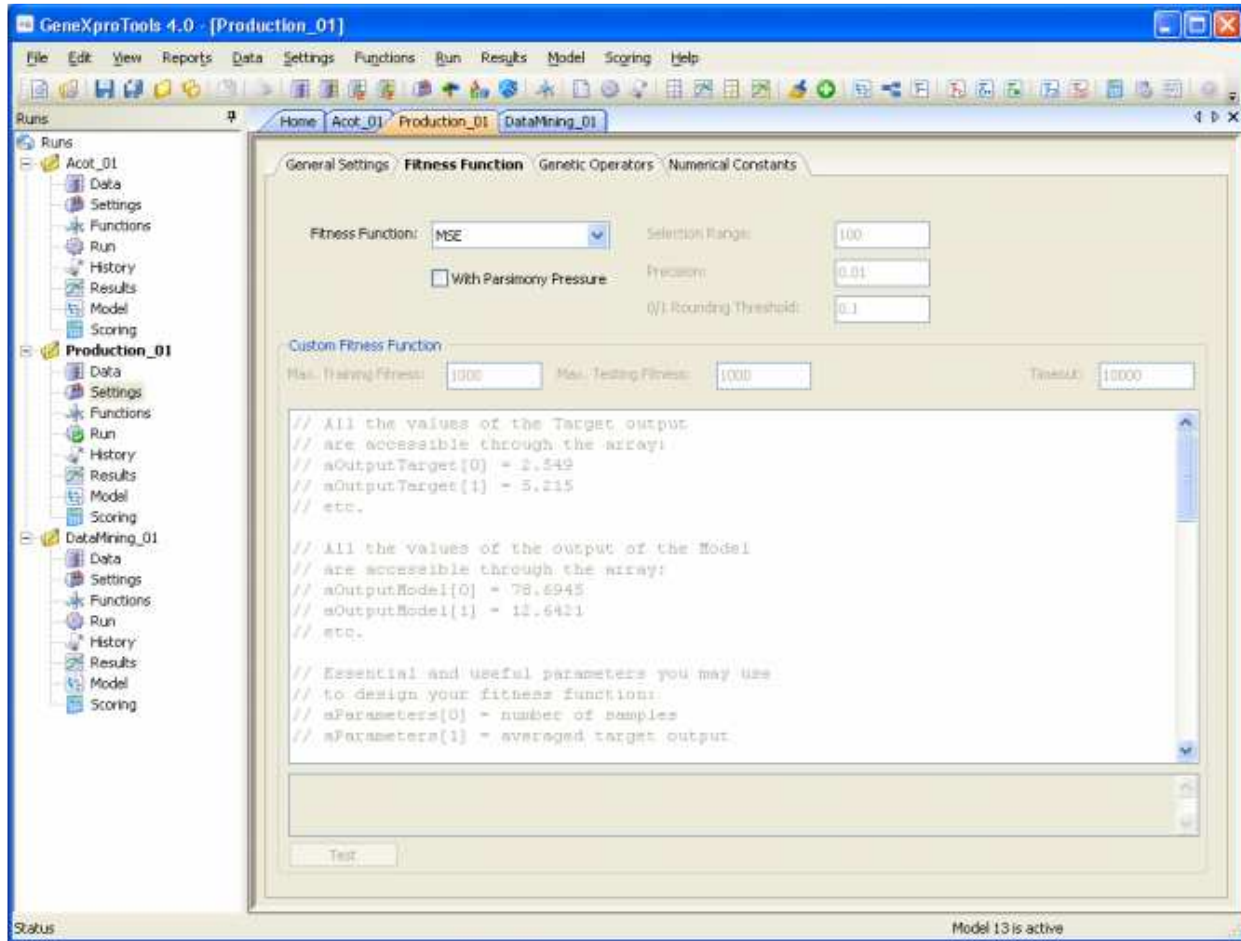
Settings and Features

Settings Panel

The **Settings Panel** of GeneXproTools 4.0 includes the following tabs:

- [General Settings Tab](#)

- Fitness Function Tab
- Genetic Operators Tab
- Numerical Constants Tab



Settings and Features

General Settings Tab

The **General Settings Tab** gives access to the following settings and features:

General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 200 [Reset]

Testing Samples: 246 [Reset]

Number of Chromosomes: 30

Head Size: 7

Number of Genes: 3

Linking Function: Addition

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 5

☒ Save Run History

Training Samples

Number of samples used for training.

Reset

Reverts to the total number of training samples in your data.

Testing Samples

Number of samples used for evaluating the generalizing capabilities of the evolved models.

Reset

Reverts to the total number of testing samples in your data.

Number of Chromosomes

Size of the population of models created each generation for breeding new ones.

Head Size

Size of the [head](#) of each gene. This parameter determines the upper limit for the size of the programs encoded in the [gene](#).

Number of Genes

Number of genes per chromosome or, in other words, the number of complex terms in the evolving models.

Linking Function

Function used to link the different terms encoded in the genes of a model. You can choose either addition, subtraction, multiplication, or division to link the [sub-ETs](#) or terms of your models.

Enable Complexity Increase

Check this box if you wish to allow GeneXproTools to automatically increase the number of genes in the evolving models.

Generations without Change

The period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction is created or a neutral gene (an extra term) is automatically added to all evolving models.

Number of Tries

The number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models.

Max. Complexity

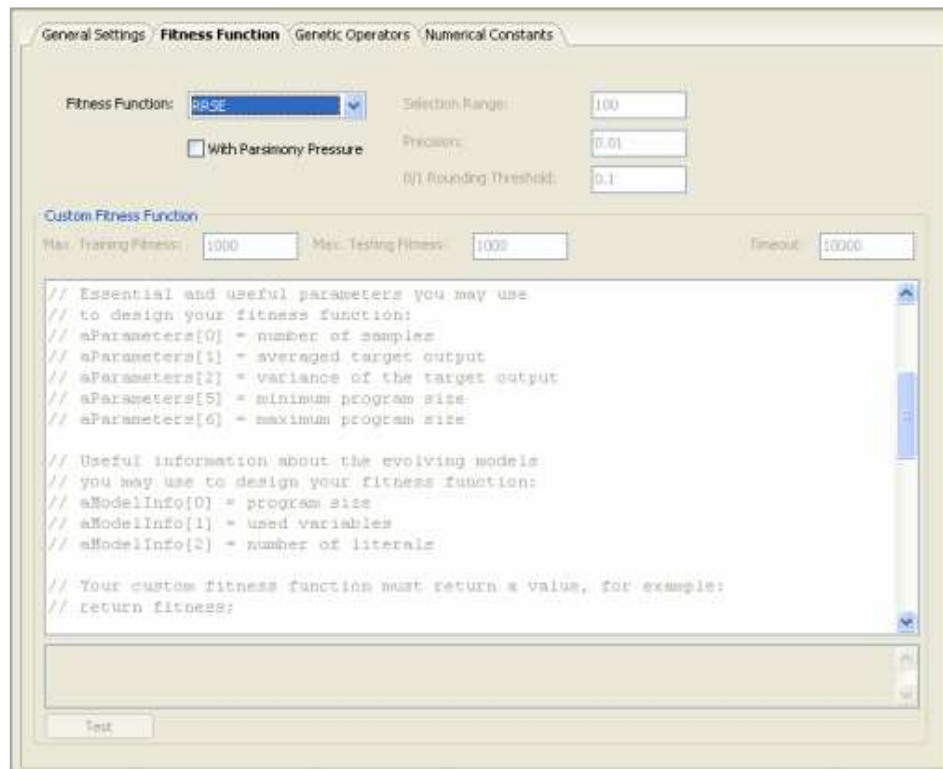
The maximum number of terms (genes) you think reasonable for modeling your data. Beyond this threshold no other terms are introduced automatically by the Complexity Increase Engine and GeneXproTools stops evolving new models.



Settings and Features

Fitness Function Tab

The **Fitness Function Tab** gives access to the following settings and features:



Fitness Function

Here you select either one of the 36 basic [built-in fitness functions](#) of GeneXproTools 4.0 for function finding or a [custom fitness function](#).

With Parsimony Pressure Check Box

By checking this box, you select a fitness function with [parsimony pressure](#).

Selection Range

The Selection Range (SR) relates exclusively to the fitness functions [Relative with SR](#) and [Absolute with SR](#) and is used as a limit for selection to operate, above which the performance of a program on a particular fitness case contributes nothing to its fitness.

Precision

The Precision in GeneXproTools 4.0 is associated with four different fitness functions. The Precision related to the fitness functions [Relative with SR](#) and [Absolute with SR](#) corresponds to the limit for improvement, for the closer solutions are to this limit the better they fit the data. The Precision related to the fitness functions [Relative/Hits](#) and [Absolute/Hits](#) corresponds to the error (relative or absolute) below which the performance of a program on a particular fitness case counts as a hit.

0/1 Rounding Threshold

This box is specific of Classification problems, and therefore is disabled in Function Finding.

Custom Fitness Function Frame

Select Custom in the Fitness Function combo box in order to activate this frame. You can now design or edit your own fitness function.

Custom Fitness Function Code Window

You design or edit the code of your own fitness function in the code window. The code must be in Java Script and can be tested for syntax errors before evolving models with it.

Test

Tests the syntax of the Java Script code of your fitness function for syntax errors. The results of the test are written in the gray window below the code window.

Max. Training Fitness

The maximum fitness corresponding to the custom fitness function on the training set. If maximum fitness depends on the number of training samples, don't forget to update this value every time you change the number of training samples, otherwise you might be creating some inaccuracies or errors. For instance, the plot of the Evolutionary Dynamics on the [Run Panel](#) won't show correctly if your maximum fitness is not correctly set.

Max. Testing Fitness

The maximum fitness corresponding to the custom fitness function on the testing set. If maximum fitness depends on the number of testing samples, don't forget to update this value every time you change the number of testing samples, otherwise you might be creating some inaccuracies.

Timeout

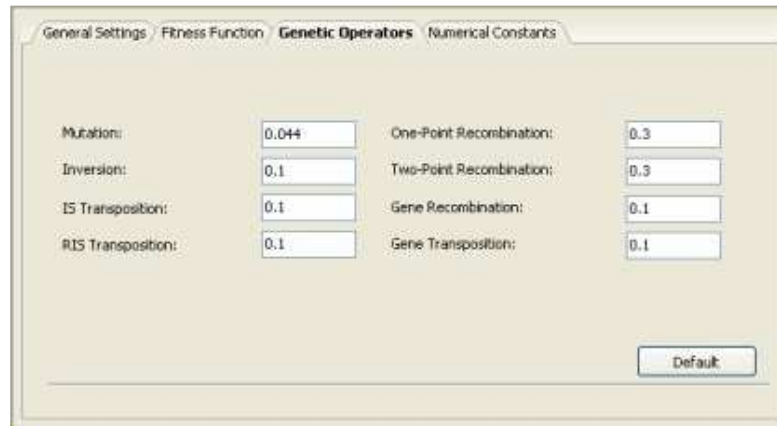
Despite one's best efforts, the code one sometimes writes contains errors that keep computers in infinite loops. This setting gives you a gracious way out of such loops.



Settings and Features

Genetic Operators Tab

The **Genetic Operators Tab** gives access to the following settings and features:



Mutation

The [mutation](#) rate used to create genetic variation. The default is 0.044.

Inversion

The [inversion](#) rate used to create genetic variation. The default is 0.1.

IS Transposition

The [IS transposition](#) rate used to create genetic variation. The default is 0.1.

RIS Transposition

The [RIS transposition](#) rate used to create genetic variation. The default is 0.1.

One-Point Recombination

The [one-point recombination](#) rate used to create genetic variation. The default is 0.3.

Two-Point Recombination

The [two-point recombination](#) rate used to create genetic variation. The default is 0.3.

Gene Recombination

The [gene recombination](#) rate used to create genetic variation. The default is 0.1.

Gene Transposition

The [gene transposition](#) rate used to create genetic variation. The default is 0.1.

Default

Resets the rates of all the above genetic operators to the default values.



Settings and Features

Numerical Constants Tab

The **Numerical Constants Tab** gives access to the following settings and features:

General Settings Fitness Function Genetic Operators **Numerical Constants**

☒ Use Random Numerical Constants

Constants per Gene: 2 Lower Bound: -10

Data Type: Floating-Point Upper Bound: 10

RNC Mutation: 0.01

Dc Mutation: 0.044

Dc Inversion: 0.1

Dc IS Transposition: 0.1

Default

Use Random Numerical Constants

Check this box if you want to use the [GEP-RNC learning algorithm](#) of GeneXproTools. This algorithm has a facility for handling random numerical constants and it evolves models with them.

Constants per Gene

The number of random numerical constants associated with each gene.

Data Type

The data type (integer or floating-point) you think appropriate for the random numerical constants that will be used in the creation of your models. For Integer constants only integer values in the interval $[-2147483648, 2147483647]$ are acceptable. For Floating-point constants a double-precision floating-point value with a range of $-1.79769313486231E308$ to $-4.94065645841247E-324$ for negative values, $4.94065645841247E-324$ to $1.79769313486231E308$ for positive values, and 0 are acceptable.

Lower Bound

Sets the lower bound of the interval from which random numerical constants are drawn.

Upper Bound

Sets the upper bound of the interval from which random numerical constants are drawn.

RNC Mutation

The [RNC mutation](#) rate used to create genetic variation in the arrays of random numerical constants. The default is 0.01.

Dc Mutation

The [Dc mutation](#) rate used to create genetic variation. The default is 0.044.

Dc Inversion

The [Dc inversion](#) rate used to create genetic variation. The default is 0.1.

Dc IS Transposition

The [Dc IS transposition](#) rate used to create genetic variation. The default is 0.1.

Default

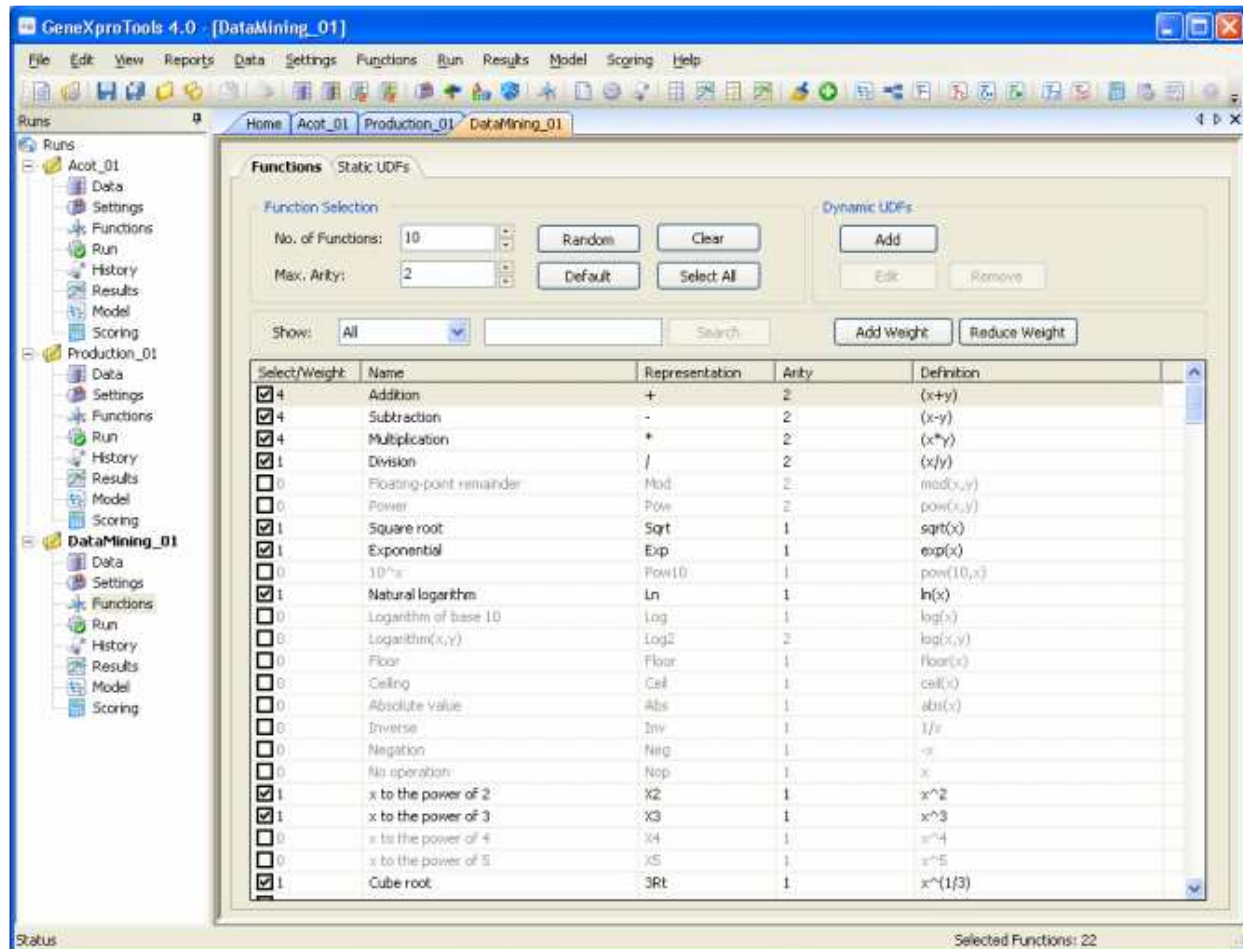
Resets the rates of all the above genetic operators to the default values.

Settings and Features

Functions Panel

The **Functions Panel** of GeneXproTools 4.0 includes the following tabs and windows:

- [Functions Tab](#)
- [Functions Tab – Edit DDF Window](#)
- [Static UDFs Tab](#)
- [Static UDFs Tab – Edit UDF Window](#)



Settings and Features

Functions Tab

The **Functions Tab** lists all the 279 [built-in mathematical functions](#) of GeneXproTools and gives access to the following settings and features:

Functions Static UDFs

Function Selection:

No. of Functions: 10 Random Clear

Max. Arity: 2 Default Select All

Dynamic UDFs:

Add Edit Remove

Show: All Search Add Weight Reduce Weight

Select/Weight	Name	Representation	Arity	Definition
☒ 4	Addition	+	2	(x+y)
☒ 4	Subtraction	-	2	(x-y)
☒ 4	Multiplication	*	2	(x*y)
☒ 1	Division	/	2	(x/y)
☐ 0	Floating-point remainder	Mod	2	mod(x,y)
☐ 0	Power	Pow	2	pow(x,y)
☒ 1	Square root	Sqrt	1	sqrt(x)
☒ 1	Exponential	Exp	1	exp(x)
☐ 0	10^x	Pow10	1	pow(10,x)
☒ 1	Natural logarithm	Ln	1	ln(x)
☐ 0	Logarithm of base 10	Log	1	log(x)
☐ 0	Logarithm(x,y)	Log2	2	log(x,y)
☐ 0	Floor	Floor	1	floor(x)
☐ 0	Ceiling	Ceil	1	ceil(x)
☐ 0	Absolute value	Abs	1	abs(x)
☐ 0	Inverse	Inv	1	1/x
☐ 0	Negation	Neg	1	-x
☐ 0	No operation	Nop	1	x
☒ 1	x to the power of 2	X2	1	x^2
☒ 1	x to the power of 3	X3	1	x^3
☐ 0	x to the power of 4	X4	1	x^4
☐ 0	x to the power of 5	X5	1	x^5
☒ 1	Cube root	3Rt	1	x^(1/3)

Function Selection Framework

Allows the selection of different function sets very quickly through the combination of the Show options with the Random/Default/Clear/Select All buttons plus the Increase/Reduce Weight buttons.

Number of Functions

The number of functions with a certain arity you wish to select randomly by using the Random button.

Maximum Arity

Determines the sub-set of functions from which the Random button will draw a specified number of functions. For instance, by setting maximum arity to 1, only functions of one argument will be drawn; by setting maximum arity to 2, only functions of 1 and 2 arguments will be randomly chosen, and so forth.

Random Button

Selects randomly a specified number of functions belonging to four different sub-sets of functions: functions of just one argument, functions of 1-2 arguments, functions of 1-3 arguments, and functions of 1-4 arguments.

Default Button

Adds the function set of the [run template](#) to the current function set. To revert exactly to the default function set you must press the Clear button before pressing Default.

Clear Button

Clears all the functions from the function set.

Select All Button

Selects all the functions showing currently on the table.

Show List Box

Shows a wide variety of function sub-sets (All functions, Searched functions, Selected Only, Arity 1 Only, Arity 2 Only, Arity 3 Only, Arity 4 Only, and DDFs Only). By selecting Search, the search box

becomes activated and you can enter specific words to find functions that match your criteria in the Name column.

Search Button

Searches, in the Name column, for the functions matching your search words.

Add Weight

Increments by one the weight of each selected function.

Reduce Weight

Reduces by one the weight of each selected function; functions with weight 1, though, remain as they are.

Select/Weight Column

To select a function, check the box on the left. By doing this, the function is included in your function set with the default weight of 1. To increase its weight, click the box below Select/Weight and then introduce its weight.

Name Column

Under this heading is shown the common name of all the [mathematical functions](#) used by GeneXproTools.

Representation Column

Under this heading is shown how all the functions are represented in [Karva notation](#).

Arity Column

Under this heading is shown the arity (number of arguments) of all the functions.

Definition Column

Under this heading is shown the mathematical definition of all GeneXproTools functions and the definition you gave to all the [DDFs](#).

Dynamic UDFs Framework

Allows the introduction of [dynamic UDFs](#) in your function set.

Add Button

Opens the [Edit DDF window](#) for designing a new [DDF](#).

Edit Button

Opens the [Edit DDF window](#) for editing an existing [DDF](#). To edit a DDF, you must first select the DDF you want to modify, and then click the Edit button.

Remove Button

Removes an existing [DDF](#). To remove a DDF, you must first select the DDF you want to remove, and then click the Remove button. A less radical way of removing a DDF, consists of unchecking this function or setting its weight to zero. This way you will be able to use it later if you wish.

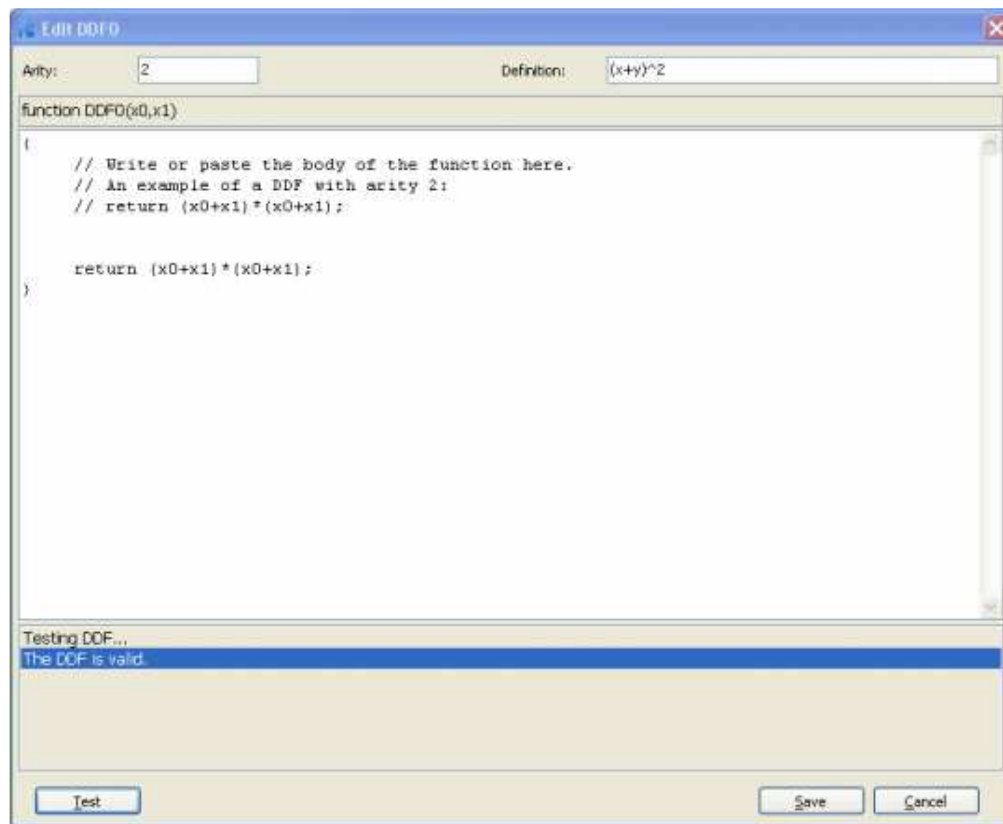


Settings and Features

Functions Tab – Edit DDF Window

The **Edit DDF Window** allows you to introduce your own functions and gives access to the following

settings and features:



Arity

The arity (number of arguments) of your [DDF](#). In GeneXproTools 4.0, the maximum allowed value is 4, and the minimum is obviously 1. After writing the arity, the function header appears in the gray box above the code window.

Definition

Use this box to write a brief description of the [DDF](#) for your future reference. The text you write here will appear in the Definition column on the [Functions Panel](#).

DDF Code Window

You design or edit the code of your own functions ([DDFs](#)) in the code window. The code must be in Java Script and can be tested for syntax errors before including it in the function set.

Test Button

Tests the syntax of the Java Script code of your [DDF](#). The results of the test are written in the gray window below the code window.

Save Button

Saves the [DDF](#) and includes it in your function set with the default weight of 1.

Cancel Button

Closes the Edit DDF window without saving the modifications made there.

Settings and Features

Static UDFs Tab

The **Static UDFs Tab** allows you to design your own [static UDFs](#) and gives access to the following settings and features:

Use UDFs

Check this box if you want to use or design [static UDFs](#).

Function Selection Framework

Allows the selection of different sub-sets of UDFs very quickly through the combination of the Show options with the Random/Clear/Select All buttons plus the Increase/Reduce Weight buttons.

Number of Functions

The number of functions you wish to select randomly by using the Random button.

Maximum Arity

This box is obviously disabled as all UDFs have arity zero.

Random Button

Selects randomly a specified number of functions.

Default Button

This button is obviously disabled as it does not apply to UDFs.

Clear Button

Clears all the functions from the function set.

Select All Button

Selects all the functions showing currently on the table.

Show List Box

Shows three different function sub-sets (All functions, Searched functions, and Selected Only). By selecting Search, the search box becomes activated and you can enter specific words to find functions that match your criteria in the Name column.

Search Button

Searches, in the Name column, for the functions matching your search words.

Add Weight

Increments by one the weight of each selected function.

Reduce Weight

Reduces by one the weight of each selected function; functions with weight 1, though, remain as they are.

Select/Weight Column

To select a [UDF](#), check the box on the left. By doing this, the UDF is included in your function set with the default weight of 1. To increase its weight, click the box below Select/Weight and then introduce its weight.

Name Column

Under this heading is shown the common name of all the [UDFs](#).

Representation Column

Under this heading is shown how all the [UDFs](#) are represented in [Karva notation](#).

Definition Column

Under this heading is shown the definition you gave to all the [UDFs](#).

Static UDFs Framework

Allows the introduction of [static UDFs](#) in your function set.

Add Button

Opens the [Edit UDF window](#) for designing a new [UDF](#).

Edit Button

Opens the [Edit UDF window](#) for editing an existing [UDF](#). To edit a UDF, you must first select the UDF you want to modify, and then click the Edit button.

Remove Button

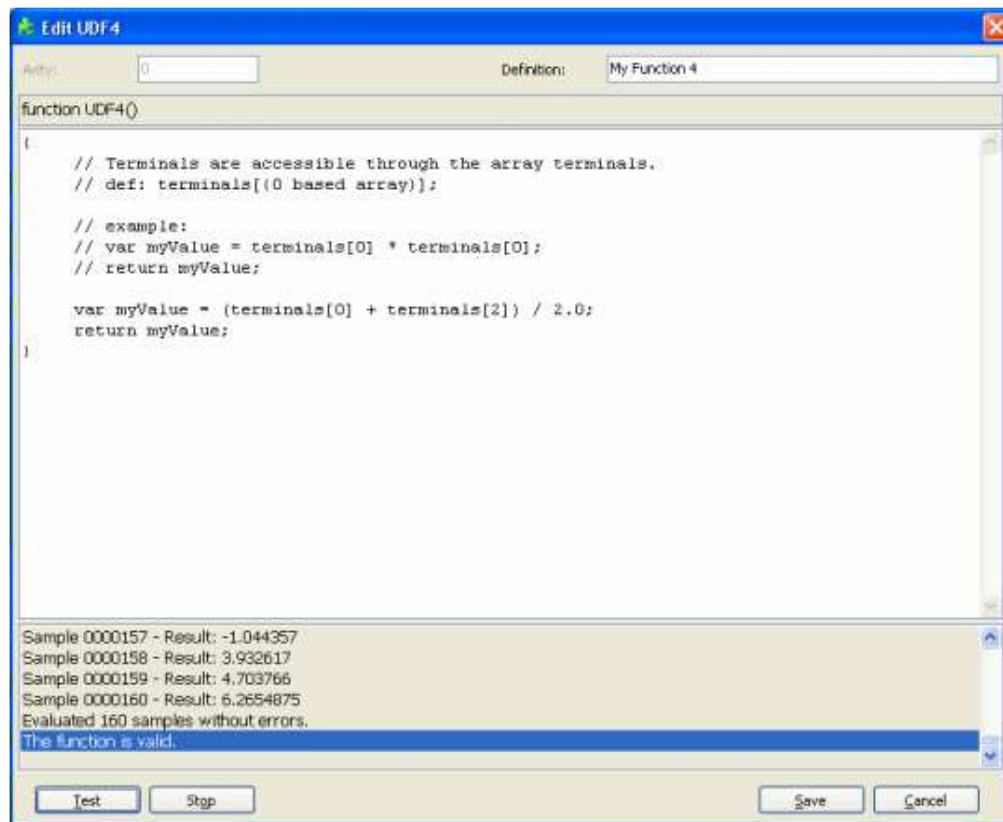
Removes an existing [UDF](#). To remove a UDF, you must first select the UDF you want to remove, and then click the Remove button. A less radical way of removing a UDF, consists of unchecking this function or setting its weight to zero. This way you will be able to use it later if you wish.



Settings and Features

Static UDFs Tab – Edit UDF Window

The **Edit UDF Window** allows you to introduce your own [static UDFs](#) and gives access to the following settings and features:



Arity

Reminds you that all the [static UDFs](#) take no parameters and, therefore, have zero arity (notice the function header that appears in the gray box above the code window).

Definition

Use this box to write a brief description of the [UDF](#) for your future reference. The text you write here will appear in the Definition column on the Functions Panel -> Static UDFs.

UDF Code Window

You design or edit the code of your [UDFs](#) in the code window. The code must be in Java Script and can be tested for syntax and calculation errors before including it in the function set.

Test Button

Tests the syntax of the Java Script code of your [UDF](#) and ensures that it is free of calculation errors. The results of the test are written in the gray window below the code window.

Stop Button

Allows you to stop the testing process.

Save Button

Saves the [UDF](#) and includes it in your function set with the default weight of 1.

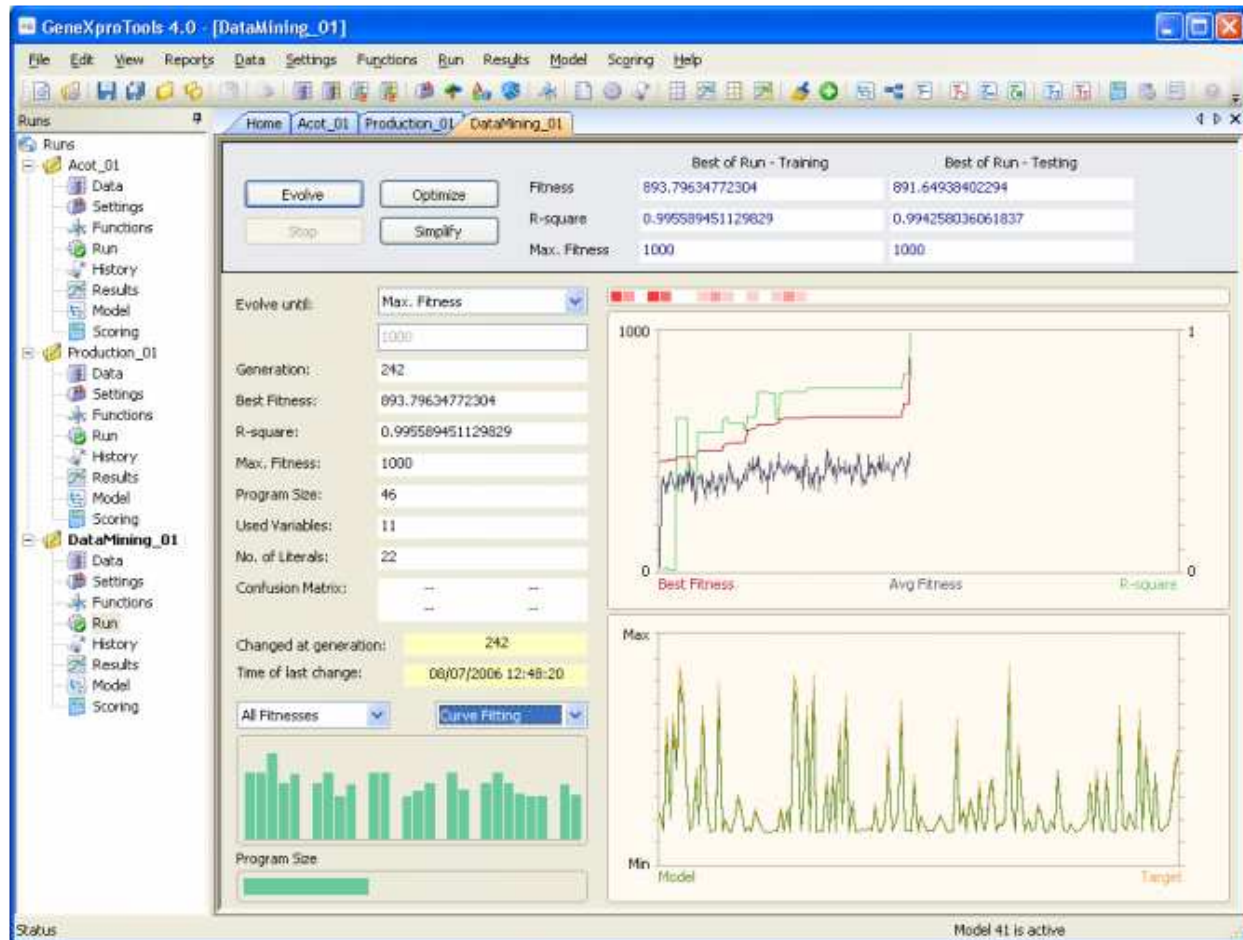
Cancel Button

Closes the Edit UDF window without saving the modifications made there.

Settings and Features

Run Panel

The **Run Panel** allows you to control and monitor the modeling process and gives access to the following settings and features:



Evolve Button

Starts the evolutionary process from scratch, that is, from a completely random initial population.

Stop Button

Stops the evolutionary process.

Optimize Button

Starts the evolutionary process with a [seed model](#), using it to breed better models.

Simplify Button

Also starts an evolutionary process with a seed model, but applies [parsimony pressure](#) to design more compact solutions.

Best of Run – Training

Under this heading are shown the values of [Fitness](#), [R-square](#), and Max. Fitness obtained on the training set for the best model of each run. These values are only updated at the end of each run or between Complexity Increase cycles. They are also updated if some relevant setting is changed, say, a different fitness function, a different training set, or a different model introduced through the Change

Seed window, and if the model is afterwards processed in the Results Panel.

Best of Run – Testing

Under this heading are shown the values of [Fitness](#), [R-square](#), and Max. Fitness obtained on the testing set for the best model of each run. These values are only updated at the end of each run or between Complexity Increase cycles. They are also updated if some relevant setting is changed, say, a different fitness function, a different testing set, or a different model introduced through the Change Seed window, and if the model is afterwards processed in the Results Panel. If no testing set is used, a dash is shown in all the boxes.

Evolve Until

Sets the stop condition. This stop condition is formulated using two different boxes. In the first you choose the stop condition and, in the second, you set the value for the chosen condition. There are four different stop conditions to choose from. The first allows you to evolve models for a certain number of generations. In this case, you must write in the box underneath the number of generations you want. The second allows you to evolve models until the best fitness is equal to or higher than a certain value. In this case, you must write in the box underneath this fitness value. The third allows you to evolve models until the [R-square](#) of the best model is equal to or higher than a certain value. In this case, you must write in the box underneath this R-square value. Obviously, only values higher than 0 and equal to or less than 1 are valid, and if an invalid value is used GeneXproTools uses the value of 1 by default. The fourth allows you to evolve models until the best fitness is equal to maximum fitness. In this case, the correct value of maximum fitness is automatically introduced by GeneXproTools in the box underneath.

Generation

Updates the generation number throughout a run.

Best Fitness

Updates the best fitness of each generation throughout a run.

R-square

Updates the [R-square](#) of the best-of-generation model throughout a run.

Max. Fitness

Shows the value of maximum fitness for the current run.

Program Size

Updates the program size of the best-of-generation model throughout a run.

Used Variables

Updates the number of variables used in the best-of-generation model throughout a run.

No. of Literals

Updates the number of literals used in the best-of-generation model throughout a run.

Confusion Matrix

This box is specific of Classification problems, and therefore only dashes are shown in Function Finding.

Changed at Generation

Updates, throughout a run, the number of the generation at which a change in best fitness occurred.

Time of Last Change

Updates, throughout a run, the time when a change in best fitness occurred.

Charts List Boxes

The one on the left allows you to select three different bar plots: Sub-Program Sizes, All Sizes, and All Fitnesses. And the one on the right allows you to select two different charts: Curve Fitting and Avg/Best Size.

Sub-Program Sizes Chart

This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be visualized any time during evolution by selecting Sub-Program Sizes in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

All Sizes Chart

This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be visualized any time during evolution by selecting All Sizes in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

All Fitnesses Chart

This chart shows the fitnesses of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be visualized any time during evolution by selecting All Fitnesses in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Program Size Chart

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Curve Fitting Chart

This chart plots the first 250 data points of the training set and shows how well the evolving models are fitting the target. The Curve Fitting chart can be visualized any time during evolution by selecting Curve Fitting in the rightmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click. And by placing the cursor over the line, you can access the value of that point. You can also access the values corresponding to the Max and Min labels that sometimes appear on the Y axis by placing the cursor over them.

Avg/Best Size Chart

This chart plots together the average program size of the population and the program size of the best-of-generation model. This chart is especially useful during simplification and can be visualized any time during evolution by selecting Avg/Best Size in the rightmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click. You can also access the values corresponding to the Max and Min labels that sometimes appear on the Y axis by placing the cursor over them.

Variables Usage Map

This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square gives you access to the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Evolutionary Dynamics Chart

This chart shows the average fitness of the population plus the fitness and [R-square](#) of the best-of-generation model. The Y axis on the left refers to best and average fitness, whereas the Y axis on the right refers to the [R-square](#). The X axis corresponds obviously to the number of generations, with a complete screen spanning a total of 500 generations. For a good adaptation, the plot for average fitness should never come near the plot for best fitness, otherwise the system is losing genetic diversity and becoming too uniform for an efficient evolution. Whenever this is happening in your system, you should obviously increase the [genetic diversity](#) of the evolving populations by choosing the

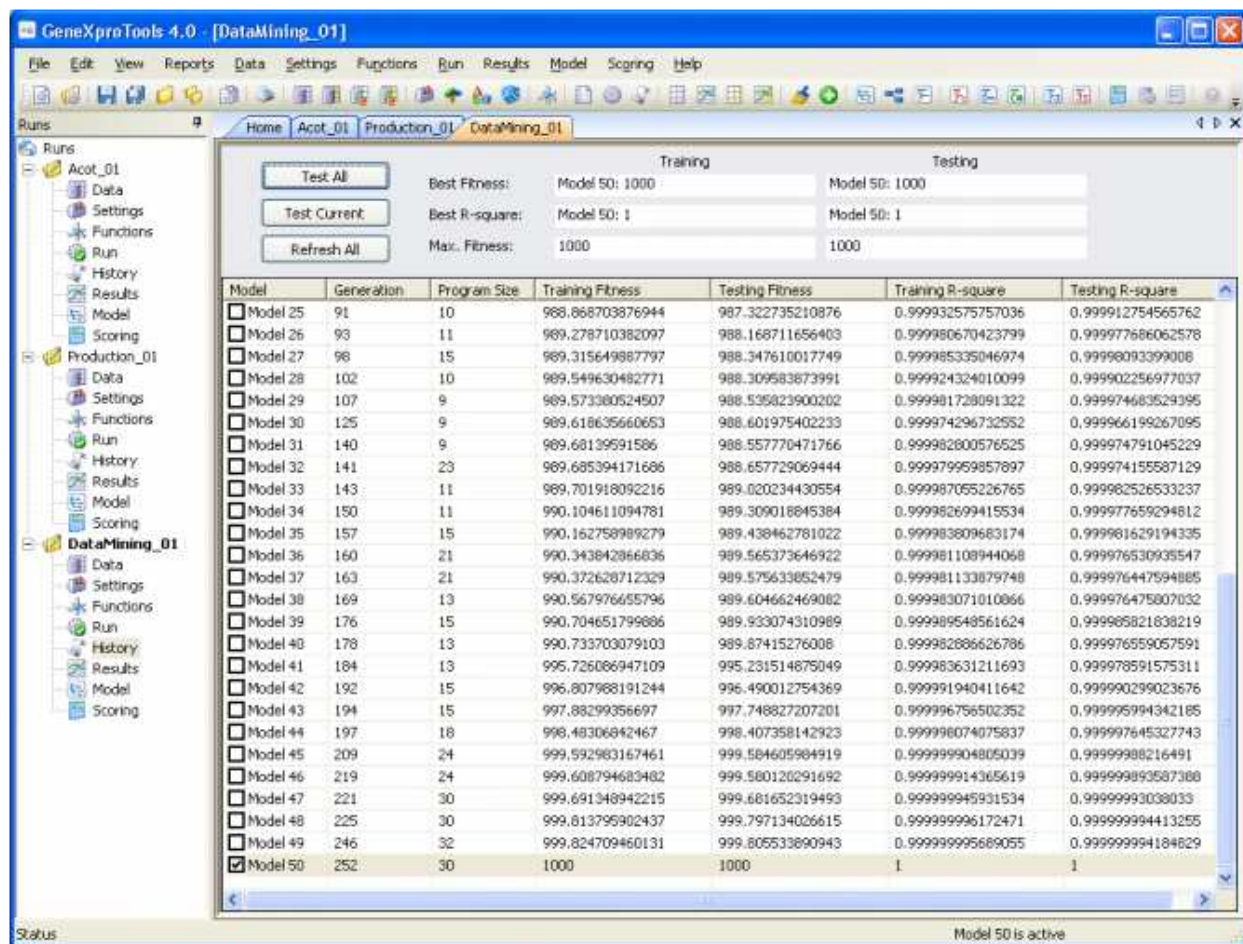
right set of genetic operators and adjusting their rates. GeneXproTools allows you to copy or print this chart through the mouse right-click.



Settings and Features

History Panel

The **History Panel** lists all the best-of-generation models created in a run and gives access to the following settings and features:



Test All Button

Evaluates the performance of all the models in the History Panel on the testing set, showing their respective [fitnesses](#) and [R-squares](#). If no testing set is used, then this button is disabled.

Test Current Button

Evaluates the performance of the selected model on the testing set, showing its [fitness](#) and [R-square](#). If no testing set is used, then this button is disabled.

Refresh All Button

Evaluates the performance of all the models in the History Panel both on the training and testing sets, and updates all the [fitnesses](#) and [R-squares](#). This button is particularly useful when you have changed the fitness function or the training set after a run and would want to evaluate the performance on the training set with this new fitness function or new training set. This new training set is usually introduced for model refreshing, and is always good practice to pre-test all the models in the History Panel with this new data set.

Best of Run – Training

This heading is used to identify the best models of the run on the training set in terms of [fitness](#) and [R-square](#), showing also the exact values of their fitness and R-square. For reference, the value of Max. Fitness on the training set is also shown.

Best of Run – Testing

This heading is used to identify the best models of the run on the testing set in terms of [fitness](#) and [R-square](#), showing also the exact values of their fitness and R-square. For reference, the value of Max. Fitness on the testing set is also shown. If no testing set is used, then a dash is shown in all the boxes under this heading.

Model Column

In this column are shown the names of all the best-of-generation models created in the run. By clicking a model, you are making it your active model. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order.

Generation Column

In this column is shown the number of the generation in which the model was created. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order.

Program Size Column

In this column is shown the program size of all the best-of-generation models. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their program sizes.

Training Fitness Column

In this column is shown the [fitness](#) obtained by each model on the training set. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their training fitnesses.

Testing Fitness Column

In this column is shown the [fitness](#) obtained by each model on the testing set. Note that, after a run, only the fitness of the best-of-run model is shown; all the others are set to zero. In order to update these values, you must either click on the Test All or Test Current buttons. If no testing set is used, though, a dash is shown in all the entries under this heading. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their testing fitnesses.

Training R-square Column

In this column is shown the [R-square](#) obtained by each model on the training set. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their training R-squares.

Testing R-square Column

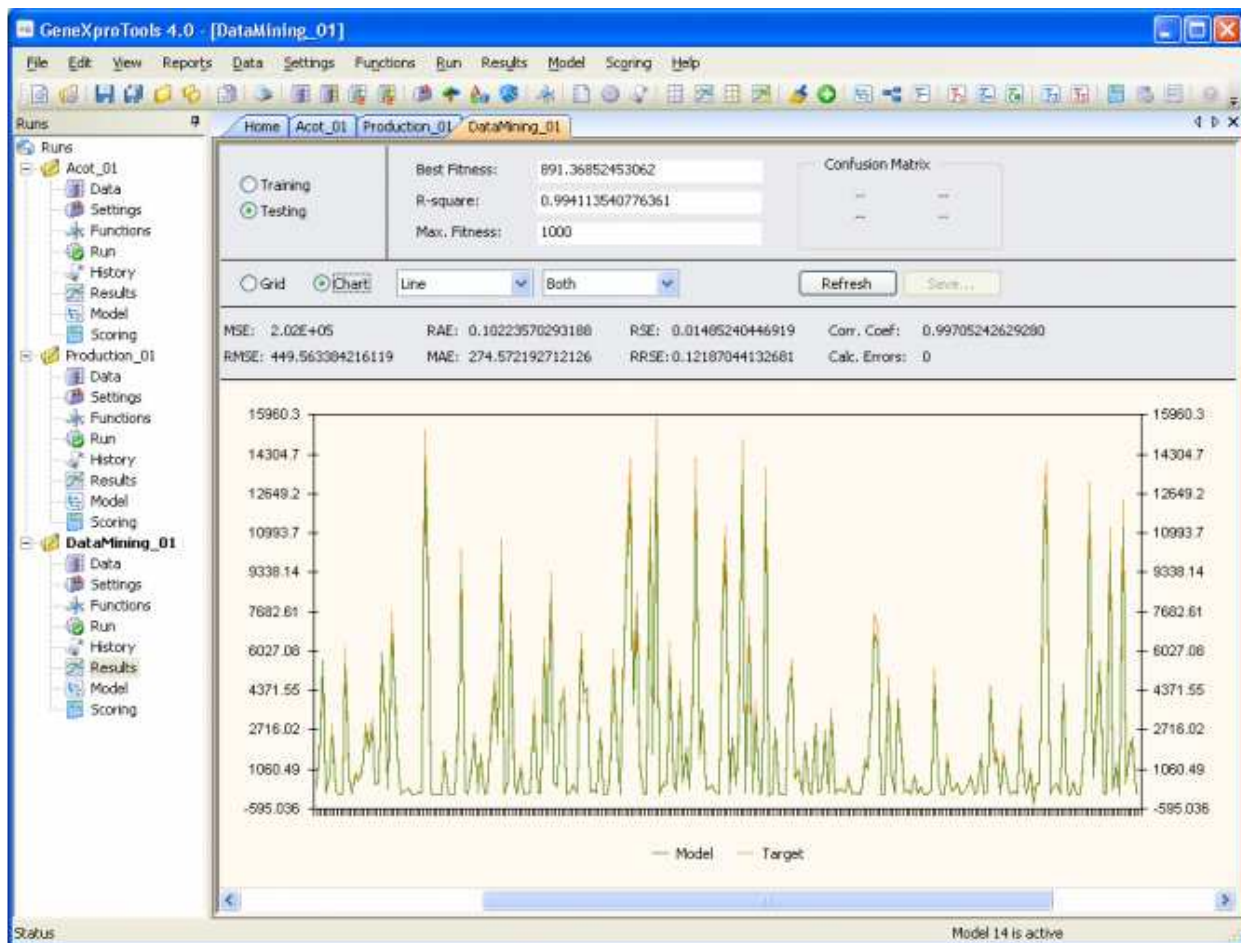
In this column is shown the [R-square](#) obtained by each model on the testing set. Note that, after a run, only the R-square of the best-of-run model is shown; all the others are set to zero. In order to update these values, you must either click on the Test All or Test Current buttons. If no testing set is used, though, a dash is shown in all the entries under this heading. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their testing R-squares.

Settings and Features

Results Panel

The **Results Panel** allows you to analyze and evaluate the performance of your models and includes the following options:

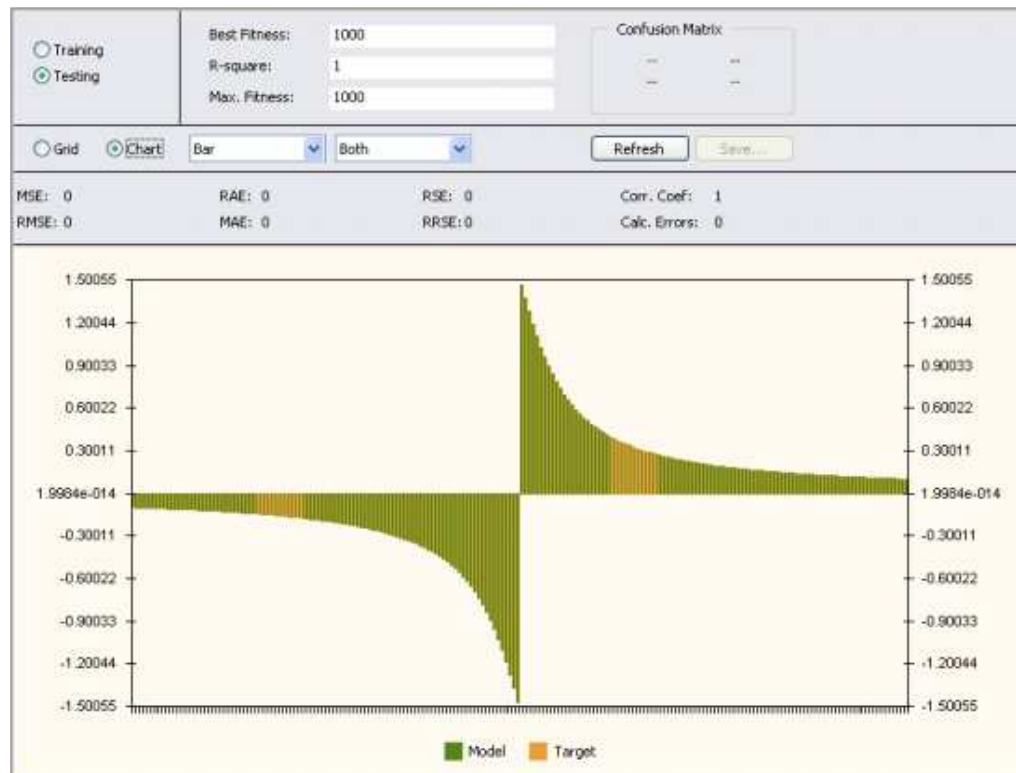
- [Training & Testing Results](#)
- [Grid Option](#)
- [Chart Option](#)



Settings and Features

Training & Testing Results

The **Training & Testing Results Windows** allow you to visualize and evaluate the accuracy of your models and gives access to the following settings and features:



Training Option

By selecting this option you have access to all the [statistical evaluations](#) of your model on the training set. You also can compare the output of your model with the target output on the training set using either a [table](#) or different [charts](#).

Testing Option

By selecting this option you have access to all the [statistical evaluations](#) of your model on the testing set. You also can compare the output of your model with the target output on the testing set using either a [table](#) or different [charts](#).

Best Fitness

Shows the [fitness](#) of the active model on the current set.

R-square

Shows the [R-square](#) of the active model on the current set.

Max. Fitness

Shows the value of [maximum fitness](#) on the current set.

Confusion Matrix

This box is specific of Classification problems, and therefore only dashes are shown in Function Finding.

Grid Option

By selecting the [Grid Option](#) you can compare the output of your model with the target output on the current set on a spreadsheet where the target and model output are placed side by side.

Chart Option

By selecting the [Chart Option](#) you can compare the output of your model with the target output on the

current set using different curve fitting charts. You choose the chart type with the aid of two boxes. In the first you select the chart type (Line, Area, Bar, Step, 3D Line, 3D Area, 3D Bar, or 3D Step). And in the second you select the data you want to plot (Model, Target, or Both). The plots show only a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar at the bottom of the chart to see all the results. Then, by placing the cursor over the chart, GeneXproTools shows you the exact range used to draw the plot.

Refresh Button

Refreshes the panel. This button is particularly useful when you change your model in the [Change Seed window](#) or change the current set or the fitness function.

Save Button

This button is only active when the [Grid Option](#) is selected as it saves all the results in the table to a file.

MSE

Shows the value of the [Mean Squared Error](#) on the current set.

RMSE

Shows the value of the [Root Mean Squared Error](#) on the current set.

RAE

Shows the value of the [Relative Absolute Error](#) on the current set.

MAE

Shows the value of the [Mean Absolute Error](#) on the current set.

RSE

Shows the value of the [Relative Squared Error](#) on the current set.

RRSE

Shows the value of the [Root Relative Squared Error](#) on the current set.

Corr. Coef.

Shows the value of the [Correlation Coefficient](#) on the current set. This parameter is not shown when the fitness function is [Relative With SR](#), [Relative/Hits](#), [Absolute With SR](#), or [Absolute/Hits](#). In this case the number of outliers is shown instead.

Outliers

Shows the number of outliers in the current set. This parameter refers only to the fitness functions [Relative With SR](#), [Relative/Hits](#), [Absolute With SR](#), and [Absolute/Hits](#). The rows with the outliers are easily identified by a yellow strip.

Calc. Errors

Shows the number of calculation errors in the current set. Note that, unless you've changed either the training set or the model after a run, no calculation errors should appear in the training set as models returning calculation errors during training have zero fitness and are, therefore, excluded. On the testing set, though, calculation errors might occur and you can see straightaway how many times in the Calculation Errors box. In addition, the rows with calculation errors are easily identified by a red strip and the identifier ERROR. In the charts, you can also easily spot the points returning calculation errors as they are not drawn, creating a discontinuity in the plots.



Settings and Features

Chart Option

The **Chart Option** allows you to compare visually the output of your model with the target output using different kinds of curve fitting charts:



Chart Type Box

In this list box you select the chart type: Line, Area, Bar, Step, 3D Line, 3D Area, 3D Bar, or 3D Step.

Chart Data Box

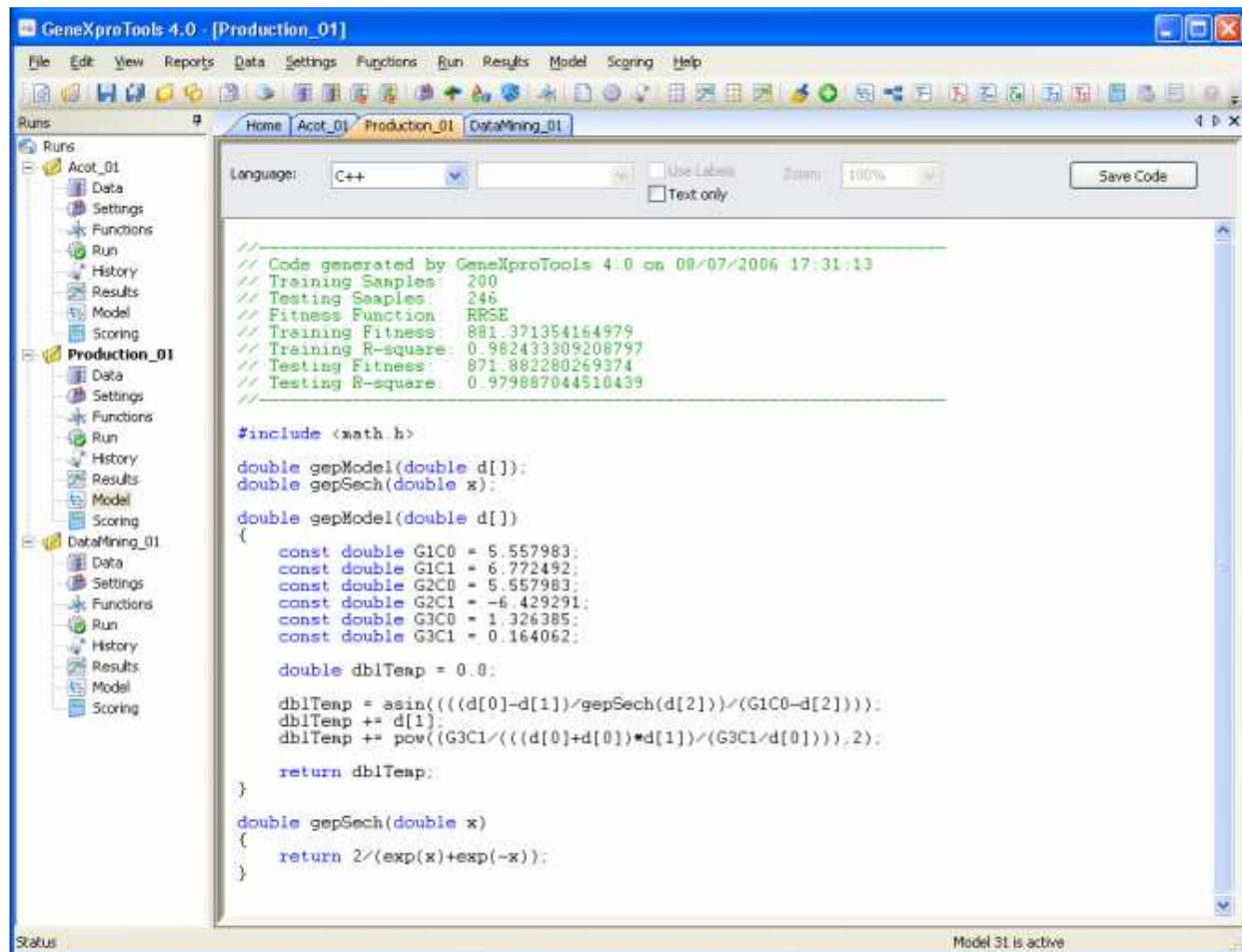
In this list box you select the data you want to plot: Model, Target, or Both. In case of calculation errors, the sample with the calculation error is skipped and a discontinuity is observed in the plot. The plots show only a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar at the bottom of the chart to see all the results. Then, by placing the cursor over the chart, GeneXproTools shows you the exact range used to draw the plot.



Settings and Features

Model Panel

The **Model Panel** allows you to see the structure of your models and translate them into a wide range of [programming languages](#). It gives access to the following settings and features:



Language

In the list box on the left, you can choose different ways of representing your models, including the native [Karva code](#), a universal diagram representation ([Expression Tree](#)), 15 different [programming languages](#) (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL) originally packaged with GeneXproTools 4.0, and any additional languages created by you through the use of [User Defined Grammars](#). The list box on the right is only activated when Karva is selected, and allows you to choose either [Genes](#) or [K-expressions](#). In both cases, though, the elements in the [heads](#) are shown in blue.

Use Labels

This option is only active when you are using data sets with labels, that is, when the columns with each variable are identified by a name. By checking this box, the variables in your code, which are usually represented by the generic d_0 , d_1 , ... are replaced by their respective names. And for the sake of clarity, this option is disabled when either Karva or Expression Tree are selected as, in both those cases, the generic representation is always used.

Text Only

By checking this box the default HTML is replaced by simple unformatted text.

Zoom

This option is only active when you are visualizing Expression Trees. It allows you to either choose pre-set values to resize the expression trees (200%, 150%, 100%, 85%, 75%, 50%, and 25%) or enter any other value up to 500%.

Save Code

Saves the selected code to an appropriate format: [Karva code](#) is saved as .txt, [Ada code](#) as .ada, [C code](#) as .c, [C++ code](#) as .hpp, [C# code](#) as .cs, [Fortran code](#) as .f, [Java code](#) as .java, [Java Script code](#) as .js, [Matlab code](#) as .m, [Pascal code](#) as .pas, [Perl code](#) as .pl, [PHP code](#) as .php, [Python code](#) as .py, [Visual Basic code](#) as .bas, [VB.Net code](#) as .vb, and [VHDL code](#) as .vhd.

Save Image

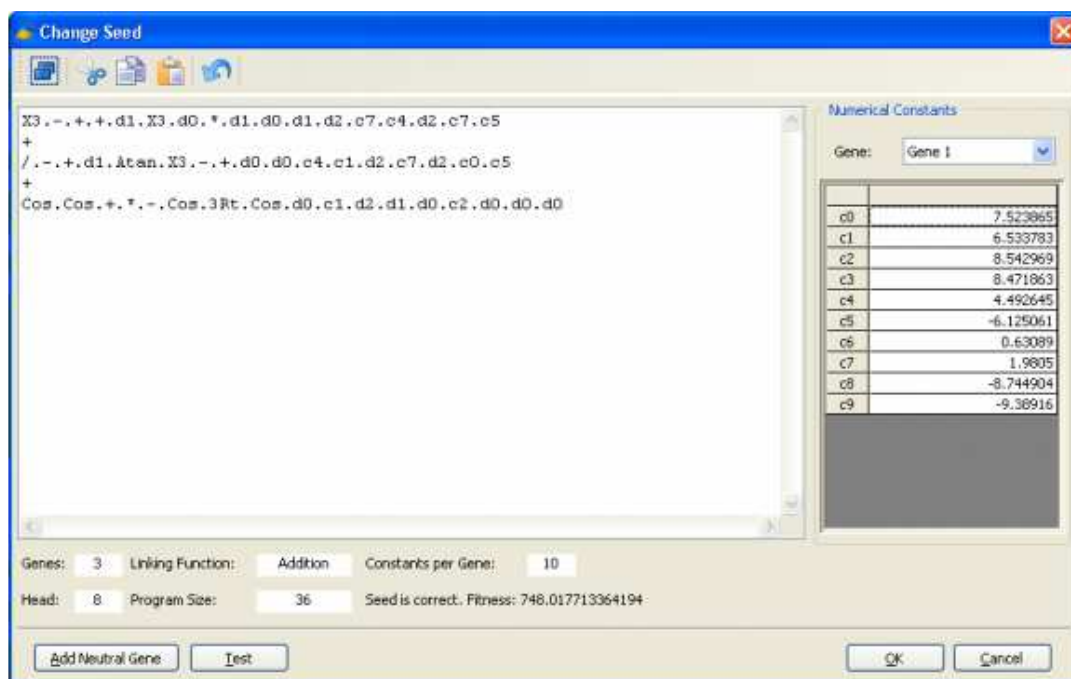
Saves the [Expression Trees](#) as .bmp files.



Settings and Features

Change Seed Window

The **Change Seed Window** allows you to change your model or introduce a completely new one. It gives access to the following settings and features:



Seed Code Window

You design or edit the code of your seed model in the code window. The code must be in [Karva notation](#) and can be tested for structural errors before saving it as seed.

Genes

Shows the current [number of genes](#) in your seed model.

Head

Shows the [head](#) size of your seed model.

Linking Function

Shows the current [linking function](#) in your seed model.

Program Size

Shows the actual size of your seed model, that is, the total number of nodes in all the [sub-ETs](#).

Constants per Gene

This box only appears if [random numerical constants](#) are being used. It shows the number of random numerical constants used per gene in your seed model.

Numerical Constants Framework

This frame only appears if [random numerical constants](#) are being used. Here you have access to all the numerical constants of your seed model by selecting the gene number in the Gene list box. The numerical constants appear in a table where you can edit them. Only values within the range of -1.79769313486231E308 to -4.94065645841247E-324 for negative values, 4.94065645841247E-324 to 1.79769313486231E308 for positive values, and 0 are acceptable.

Add Neutral Gene Button

Adds automatically a neutral gene to your seed model. Neutrality is important for an efficient evolution and, by introducing a neutral gene, you are giving the [learning algorithm](#) more room to play and, hopefully, a better, more complex program will evolve.

Test Button

Tests both the structural soundness and the [fitness](#) of your seed model. It also updates the program size when changes have been made on the seed.

OK Button

Saves the current seed as your seed model so that you can breed new and better models with it. GeneXproTools only saves viable seeds and warns you if there is a structural problem with your seed or if it has zero fitness.

Cancel Button

Cancels all the action on the change seed window.

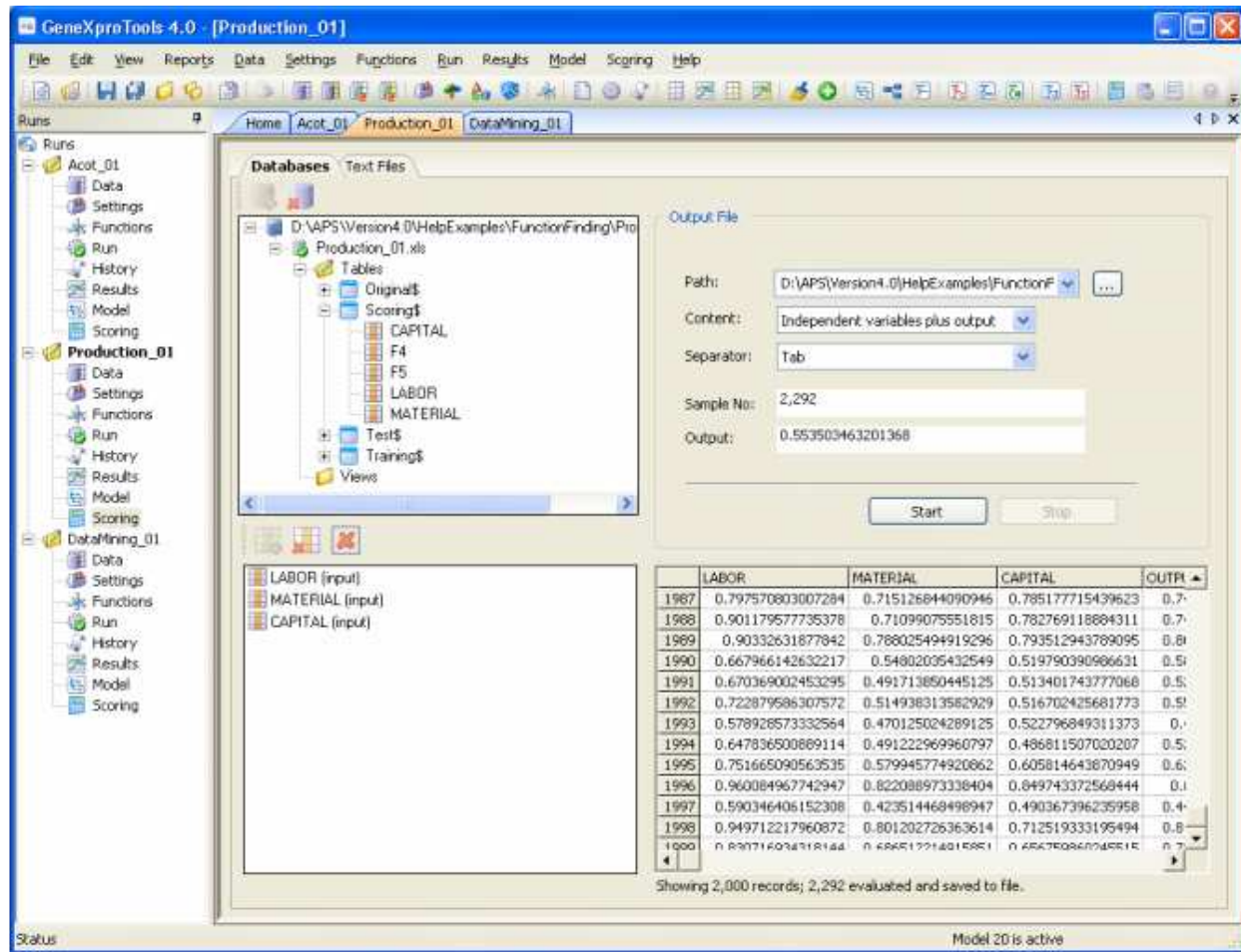


Settings and Features

Scoring Panel

The **Scoring Panel** allows you to score your models immediately without leaving the GeneXproTools modeling environment. It includes the following tabs and windows:

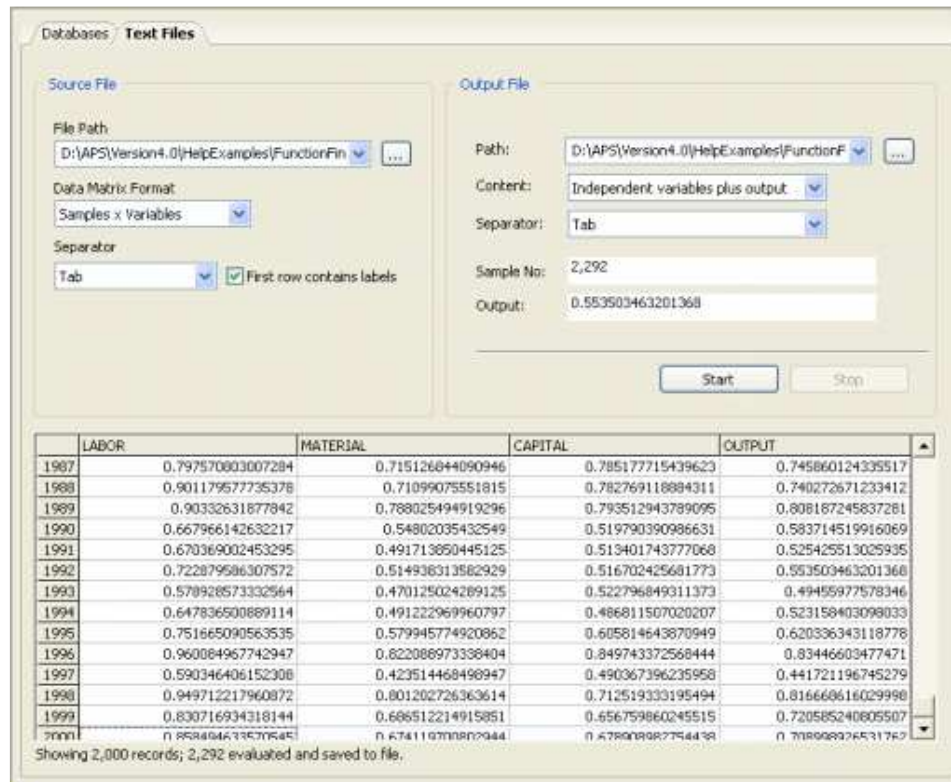
- [Text Files Tab](#)
- [Databases Tab](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Text Files Tab

The **Text Files Tab** allows you to score text files and gives access to the following settings and features:



Source File

File Path: D:\APS\Version4.0\HelpExamples\FunctionF...

Data Matrix Format: Samples x Variables

Separator: Tab ☒ First row contains labels

Output File

Path: D:\APS\Version4.0\HelpExamples\FunctionF...

Content: Independent variables plus output

Separator: Tab

Sample No: 2,292

Output: 0.553503463201368

	LABOR	MATERIAL	CAPITAL	OUTPUT
1987	0.797570003007284	0.715126844090946	0.785177715439623	0.745860124335517
1988	0.901179577735378	0.71099075551815	0.782769118894311	0.740272671233412
1989	0.90332631877842	0.788025494919296	0.793512943789095	0.808167245837281
1990	0.667966142632217	0.54802035432549	0.519790390986631	0.583714519916069
1991	0.670369002453295	0.491713850445125	0.513401743777068	0.525425513025935
1992	0.722879586307572	0.514936313582929	0.516702425681773	0.553503463201368
1993	0.578928573332564	0.470125024289125	0.522796849311373	0.49455977578346
1994	0.647836500889114	0.491222969960797	0.486811507020207	0.523158403098033
1995	0.751665090563535	0.579945774920862	0.605814643870949	0.620336343118778
1996	0.960064967742947	0.622088973338404	0.849743372568444	0.83446603477471
1997	0.590346406152308	0.423514468498947	0.490367396235958	0.441721196745279
1998	0.949712217960872	0.801202726363614	0.712519333195494	0.816668616029998
1999	0.830716934318144	0.686512214915851	0.656759860245515	0.720585240805507
2000	0.85486433570545	0.674110700829444	0.678989827754438	0.708988924531767

Showing 2,000 records; 2,292 evaluated and saved to file.

Source File Framework

In the Source File Framework you enter all the information concerning the source file.

Source File Path

Click the button on the right of the list box to load the scoring file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, you must select the Samples x Variables format. If your samples are in columns and variables in rows, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies.

Source File Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

Output File Framework

In the Output File Framework you enter all the information concerning the output file.

Output File Path

You can either enter the path for the output file or click the button on the right of the list box to browse the Path for the output file.

Output File Content

Here you decide whether you want to include in the output file the columns with the independent variables or just the output.

Output File Separator

Here you choose the separator you want to use in the output file (this is only relevant if you'd choose also to include in the output file the columns with the independent variables).

Sample Number

Shows the number of the sample being processed.

Output

Shows the output obtained for the current sample.

Start Button

Starts the scoring process.

Stop Button

Stops the scoring process.

Scoring Results Table

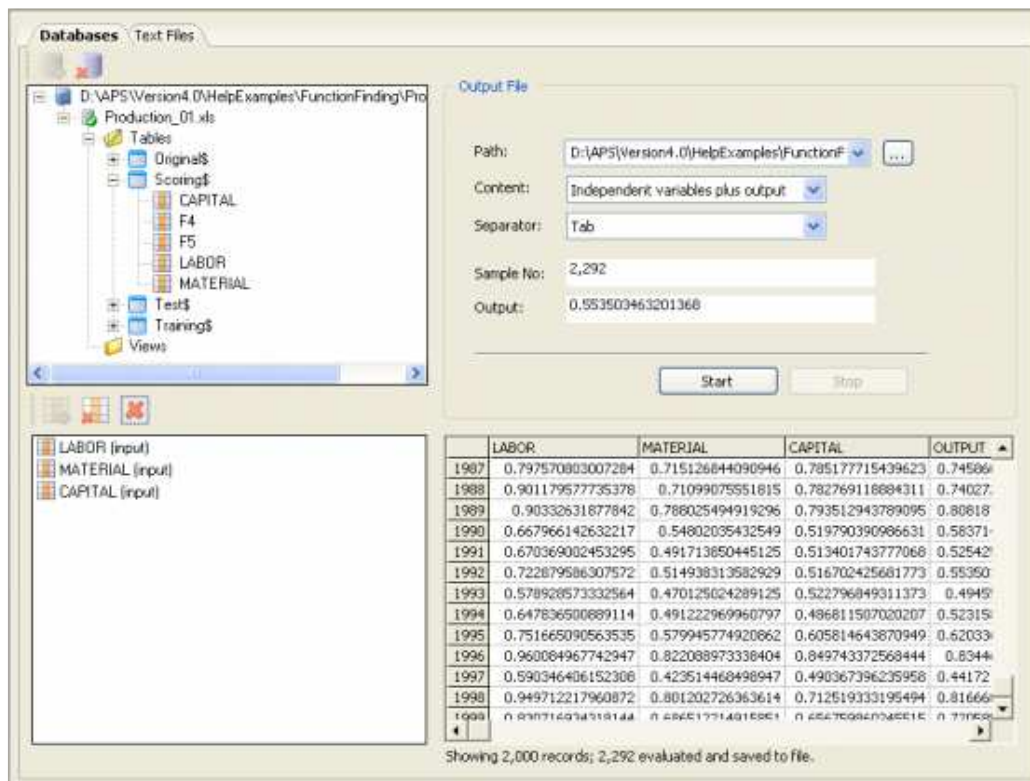
GeneXproTools saves the scoring results to a file, but for small datasets (up to 16 variables and 2000 samples), GeneXproTools also shows the scoring results in their entirety on the Scoring Panel for an immediate assessment. For datasets with more than 16 variables, though, only a single column with the output is shown.



Settings and Features

Databases Tab

The **Databases Tab** allows you to score databases and Excel files and gives access to the following settings and features:



Databases Tab

Output File

Path: D:\AP5\Version4.0\HelpExamples\FunctionF...

Content: Independent variables plus output

Separator: Tab

Sample No: 2,292

Output: 0.553503463201368

	LABOR	MATERIAL	CAPITAL	OUTPUT
1987	0.797570803007284	0.715126844090946	0.785177715439623	0.745866
1988	0.901179577735378	0.71099075551815	0.782769118684311	0.74027
1989	0.90332631877842	0.788025494919296	0.793512943789095	0.80818
1990	0.667966142632217	0.54802035432549	0.519790390986631	0.58371
1991	0.670369002453295	0.491713850445125	0.513401743777068	0.52542
1992	0.722879586307572	0.514938313582929	0.516702425681773	0.55350
1993	0.578928573332564	0.470125024289125	0.522796849311373	0.4945
1994	0.647836500889114	0.491222969960797	0.486811507020207	0.52315
1995	0.751665090563535	0.579945774920862	0.605814643870949	0.62033
1996	0.960084967742947	0.822088973338404	0.849743372568444	0.8344
1997	0.590346406152306	0.423514468498947	0.490367396235958	0.44172
1998	0.949712217960872	0.801202726363614	0.712519333195494	0.81666
1999	0.89714274718144	0.58451771491851	0.55768847045111	0.77953

Showing 2,000 records; 2,292 evaluated and saved to file.

Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Scoring Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Remove All the Columns Icon

Removes all the columns from the Scoring Data Window.

Scoring Data Window

You prepare your scoring data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Remember, however, that your scoring dataset must have exactly the same number of inputs as there are independent variables in your training data.

Output File Framework

In the Output File Framework you enter all the information concerning the output file.

Output File Path

You can either enter the path for the output file or click the button on the right of the list box to browse the Path for the output file.

Output File Content

Here you decide whether you want to include in the output file the columns with the independent variables or just the output.

Output File Separator

Here you choose the separator you want to use in the output file (this is only relevant if you'd choose also to include in the output file the columns with the independent variables).

Sample Number

Shows the number of the sample being processed.

Output

Shows the output obtained for the current sample.

Start Button

Starts the scoring process.

Stop Button

Stops the scoring process.

Scoring Results Table

GeneXproTools saves the scoring results to a file, but for small datasets (up to 16 variables and 2000 samples), GeneXproTools also shows the scoring results in their entirety on the Scoring Panel for an immediate assessment. For datasets with more than 16 variables, though, only a single column with the output is shown.

Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:



Driver

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.
Microsoft Access: path to the database file.
Microsoft SQL Server: server name or IP address.
Oracle: server name or IP address.
MySQL: server name or IP address.
DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.
Oracle - Name and database instance (HOST/INSTANCE).
MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

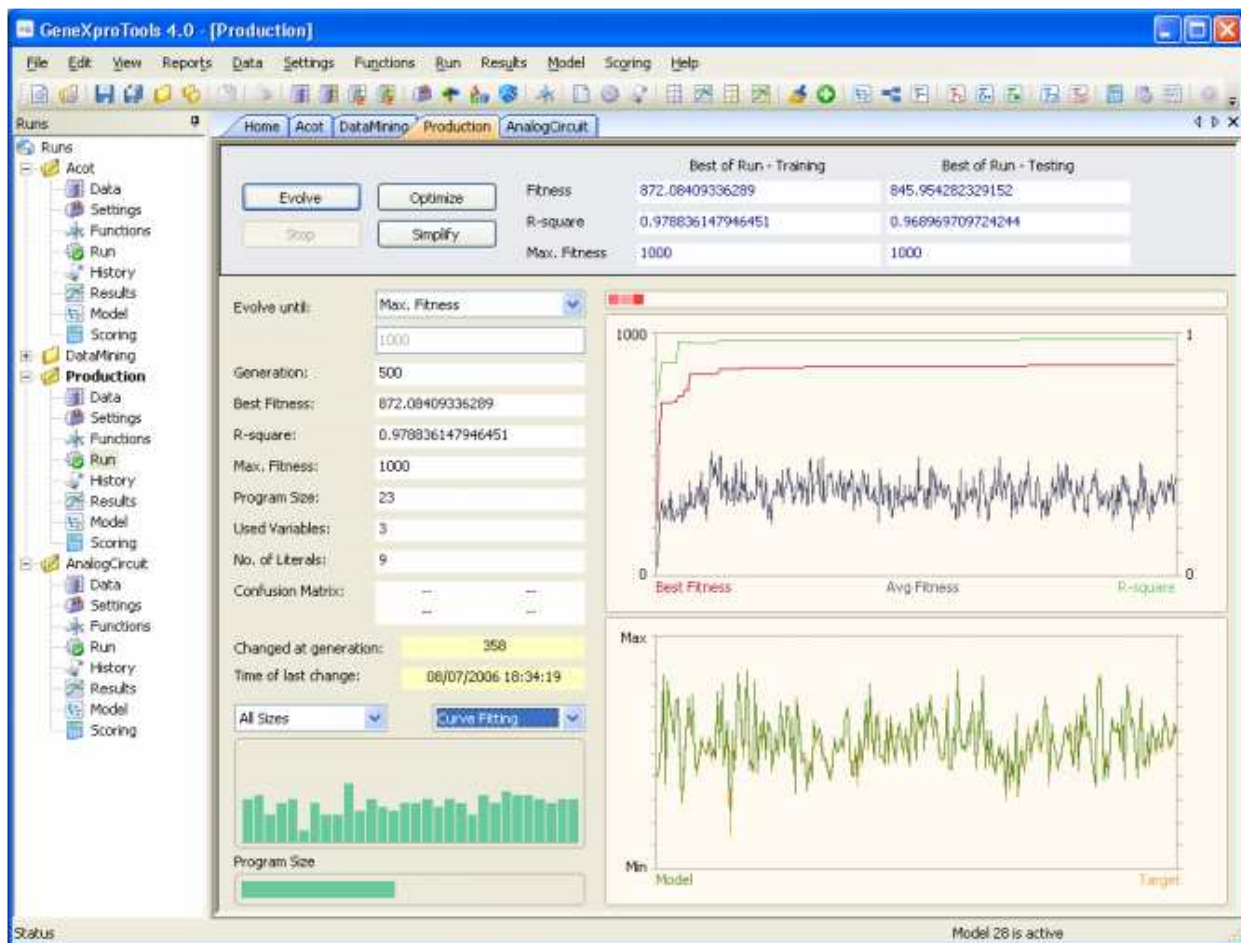
Cancels all the action on the data source connection.

Settings and Features

Menus

For Function Finding problems, GeneXproTools has the following menus:

- [File](#)
- [Edit](#)
- [View](#)
- [Reports](#)
- [Data](#)
- [Settings](#)
- [Functions](#)
- [Run](#)
- [Results](#)
- [Model](#)
- [Scoring](#)
- [Help](#)



Settings and Features

File

The **File Menu** gives access to the following sub-menus:

New

Creates a new gep file.

Open

Opens a gep file.

Save

Saves the selected run.

Save As

Saves a gep file with a different name.

Save All

Saves all open runs.

Close

Closes the selected run.

Close All

Closes all open runs.

Print

Prints the report, history, code, tree, or charts of the run.

Properties

Loads the identification form for input (Author Name and Comments).

Exit

Closes GeneXproTools.



Settings and Features

Edit

The **Edit Menu** gives access to the following sub-menus:

Copy

Copies the report, tables, code, tree, or charts of the run for pasting elsewhere.

Change Seed

Opens the [Change Seed window](#) for changing exogenous or GeneXproTools evolved models for further optimization.

Add Neutral Gene

Adds automatically a neutral gene to your seed model.

Add DDF

Opens the [Edit DDF window](#) for designing a new [DDF](#).

Add UDF

Opens the [Edit UDF window](#) for designing a new [UDF](#).

Custom Fitness Function

Goes to the [Custom Fitness Function Code window](#) for editing or designing a new [custom fitness function](#).



Settings and Features

View

The **View Menu** gives access to the following sub-menus:

Status Bar

Shows/Hides the Status Bar.

Warnings

Turns on/off warning messages.



Settings and Features

Reports

The **Reports Menu** gives access to the following sub-menus:

Home

Goes to the welcome screen of GeneXproTools.

Run Report

Goes to the [Report Panel](#) of the active run.



Settings and Features

Data

The **Data Menu** gives access to the following sub-menus:

Training Data

Goes to the [Training Set Tab](#) of the selected run.

Testing Data

Goes to the [Testing Set Tab](#) of the selected run.

Change Training Data

Opens the [Change Training Data Window](#) for changing the training set of the selected run.

Change Testing Data

Opens the [Change Testing Data Window](#) for changing the testing set of the selected run.

Add Testing Data

Opens the [Add Testing Data Window](#) for changing the testing set of the selected run.



Settings and Features

Settings

The **Settings Menu** gives access to the following sub-menus:

General Settings

Goes to the [General Settings Tab](#) of the selected run.

Fitness Function

Goes to the [Fitness Function Tab](#) of the selected run.

Genetic Operators

Goes to the [Genetic Operators Tab](#) of the selected run.

Numerical Constants

Goes to the [Numerical Constants Tab](#) of the selected run.



Settings and Features

Functions

The **Functions Menu** gives access to the following sub-menus:

Functions

Goes to the [Functions Tab](#) of the selected run.

Static UDFs

Goes to the [Static UDFs Tab](#) of the selected run.



Settings and Features

Run

The **Run Menu** gives access to the following sub-menus:

Run

Goes to the [Run Panel](#) of the active run.

Evolve

Starts a new run.

Optimize

Starts an optimization process.

Simplify

Starts a simplification process.

Variables Usage

Allows the selection of three different kinds of mapping variables usage during a run: Heat Map, Random Colors, or Monochromatic, and two different showing options: White for None and Grey for None.

Charts

Allows the selection of five different kinds of charts during a run: Curve Fitting, Avg/Best Size, Sub-Program Sizes, All Sizes, and All Fitnesses.



Settings and Features

Results

The **Results Menu** gives access to the following sub-menus:

History

Goes to the [History Panel](#) of the active run.

Training Results Table

Goes to the [Results Panel -> Training and Grid Options](#) of the active run.

Training Results Chart

Goes to the [Results Panel -> Training and Chart Options](#) of the active run.

Testing Results Table

Goes to the [Results Panel -> Training and Grid Options](#) of the active run.

Testing Results Chart

Goes to the [Results Panel -> Training and Chart Options](#) of the active run.



Settings and Features

Model

The **Model Menu** gives access to the following sub-menus:

Expression Tree

Shows the active model as an [Expression Tree](#).

Karva

Shows the active model in the native [Karva Notation](#).

Ada

Translates the active model into [Ada](#).

C

Translates the active model into [C code](#).

C++

Translates the active model into [C++](#).

C#

Translates the active model into [C#](#).

Fortran

Translates the active model into [Fortran](#).

Java

Translates the active model into [Java](#).

JavaScript

Translates the active model into [JavaScript](#).

Matlab

Translates the active model into [Matlab](#).

Pascal

Translates the active model into [Pascal](#).

Perl

Translates the active model into [Perl](#).

PHP

Translates the active model into [PHP](#).

Python

Translates the active model into [Python](#).

Visual Basic

Translates the active model into [Visual Basic](#).

VB.Net

Translates the active model into [VB.Net](#).

VHDL

Translates the active model into [VHDL](#).



Settings and Features

Scoring

The **Scoring Menu** gives access to the following sub-menus:

Databases

Goes to the [Scoring Panel -> Databases Tab](#) for scoring a database with the active model.

Text Files

Goes to the [Scoring Panel -> Text Files Tab](#) for scoring a database with the active model.



Settings and Features

Help

The **Help Menu** gives access to the following sub-menus:

Contents

Shows the contents of the Help File.

About GeneXproTools

Shows information about GeneXproTools.



Settings and Features

Run Templates

For Function Finding problems, GeneXproTools 4.0 offers five different **Run Templates** designed so that you can start the modeling process with just a mouse click and yet obtain excellent results. In addition, they also make the discovery of good models quite enjoyable and fast as small populations of just 30 models are used in all of them.

The run templates of GeneXproTools are essentially based on the number of independent variables in your data and are organized as follows (in the Function Set, weights greater than 1 are indicated in parentheses):

Template A

Independent Variables: 1-3
Fitness Function: RRSE
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, and arctangent.
Chromosomes: 30
Genes: 3
Head Size: 7
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 5
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template B

Independent Variables: 4-8
Fitness Function: RRSE
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, and arctangent.
Chromosomes: 30
Genes: 3
Head Size: 8
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 5
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1

Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template C

Independent Variables: 9-32
Fitness Function: RRSE
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, and arctangent.
Chromosomes: 30
Genes: 4
Head Size: 10
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 6
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template D

Independent Variables: 33-64
Fitness Function: RRSE
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, and arctangent.
Chromosomes: 30
Genes: 5
Head Size: 12
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 7
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3

Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template E

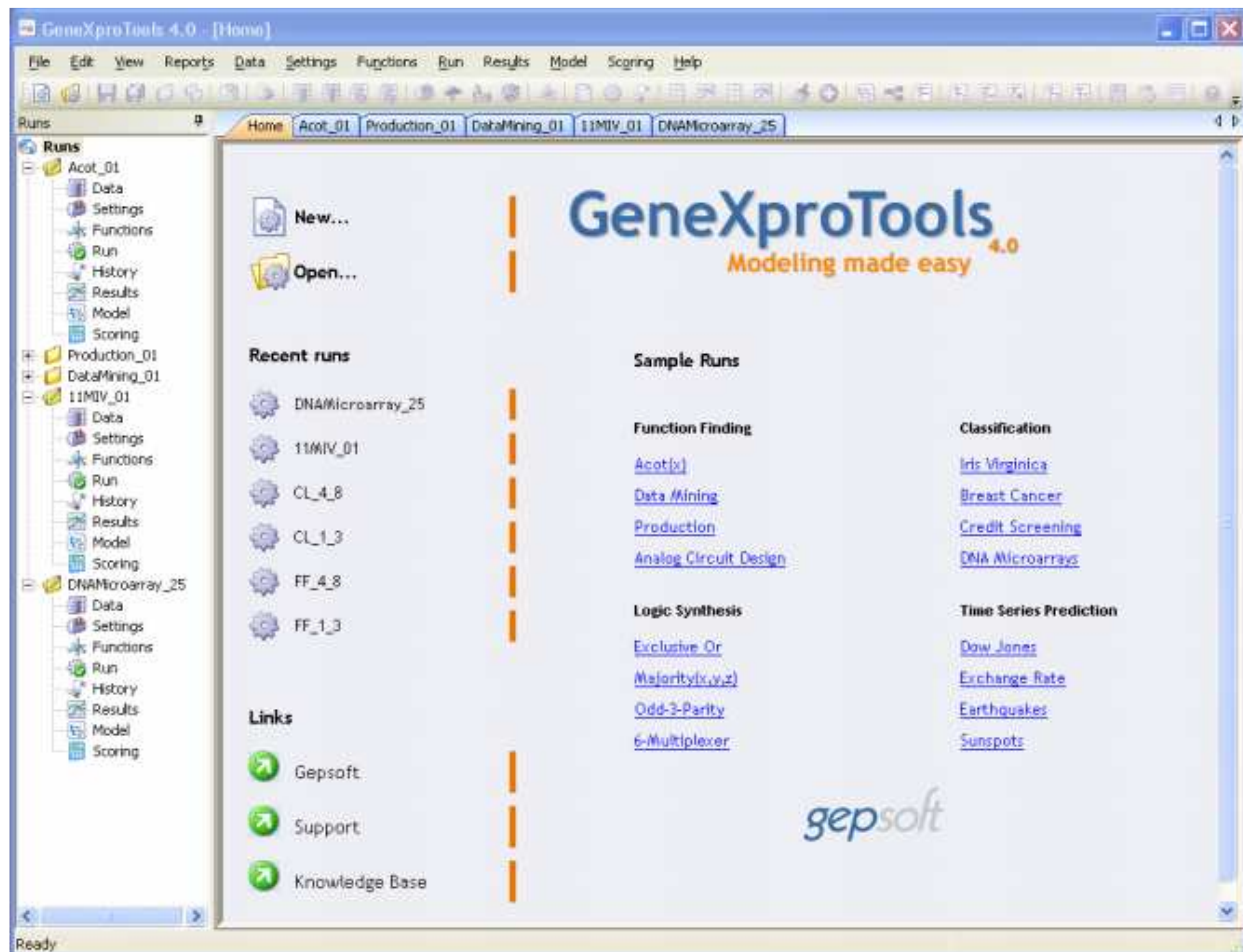
Independent Variables: 65-Unlimited
Fitness Function: RRSE
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, and arctangent.
Chromosomes: 30
Genes: 6
Head Size: 15
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 8
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1



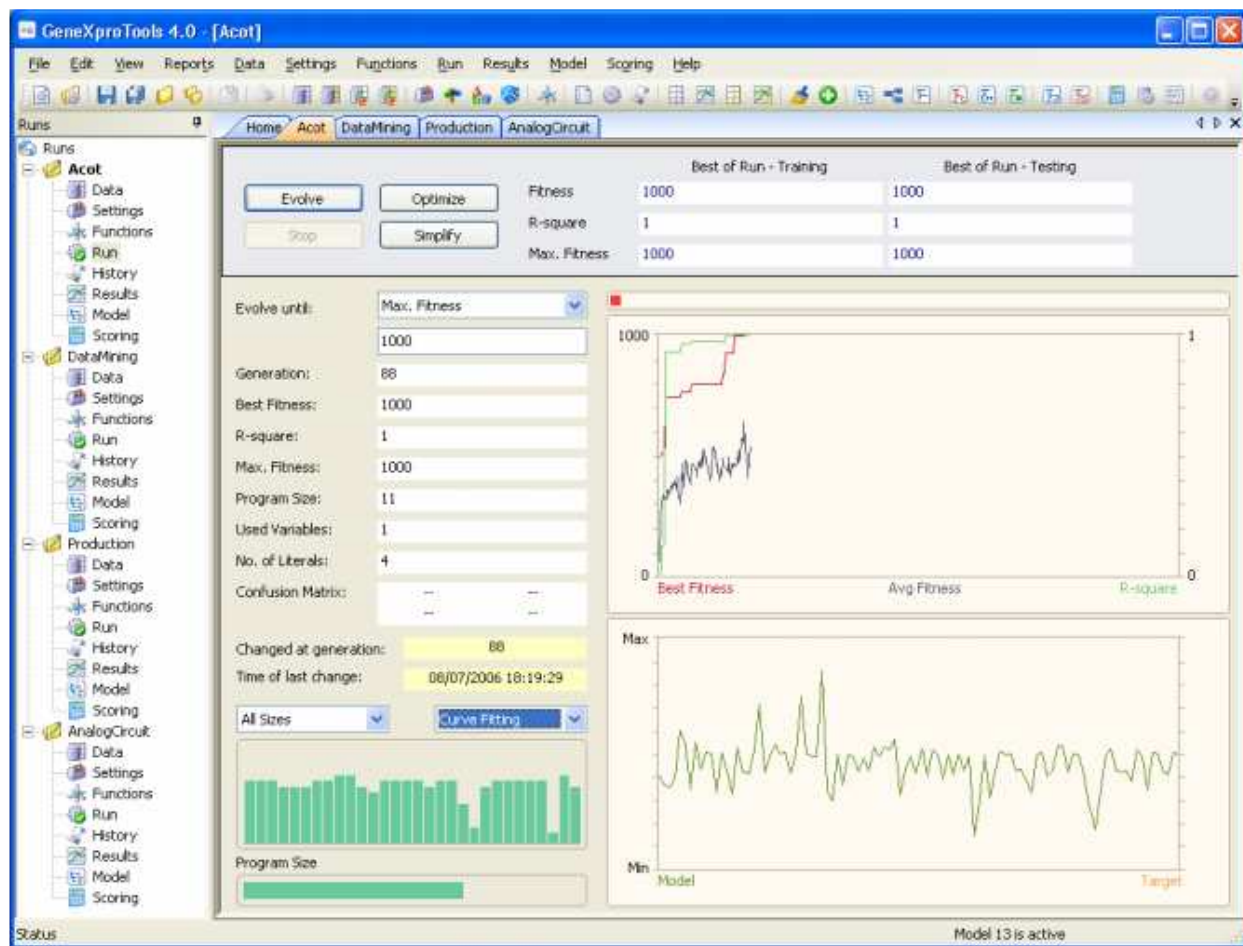
Settings and Features

Sample Runs

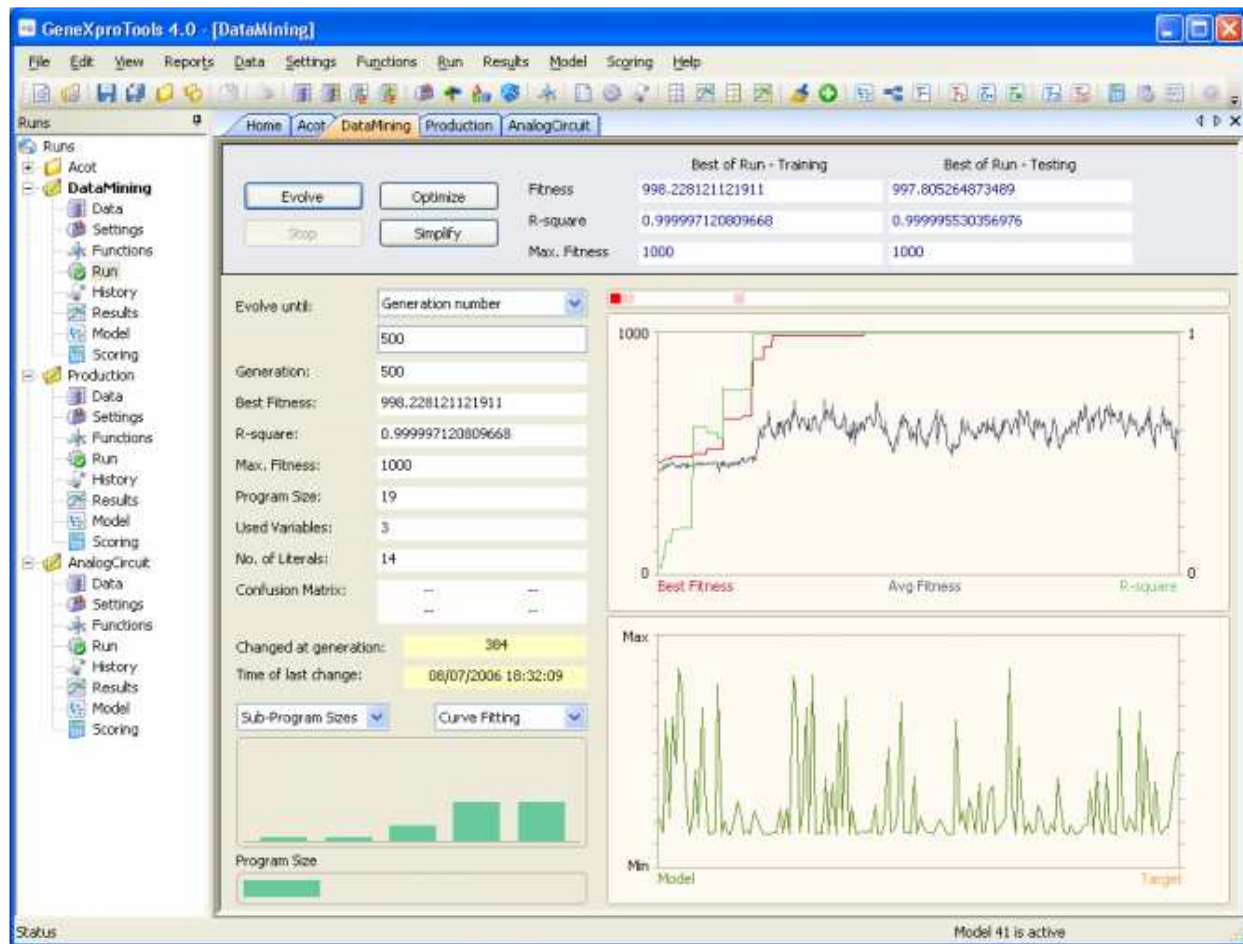
GeneXproTools 4.0 ships with four different **Sample Runs** for Function Finding: [Arccotangent](#), [Data Mining](#), [Production](#), and [Analog Circuit Design](#). To try any one of them, you just have to click its link on the Welcome Screen of GeneXproTools.



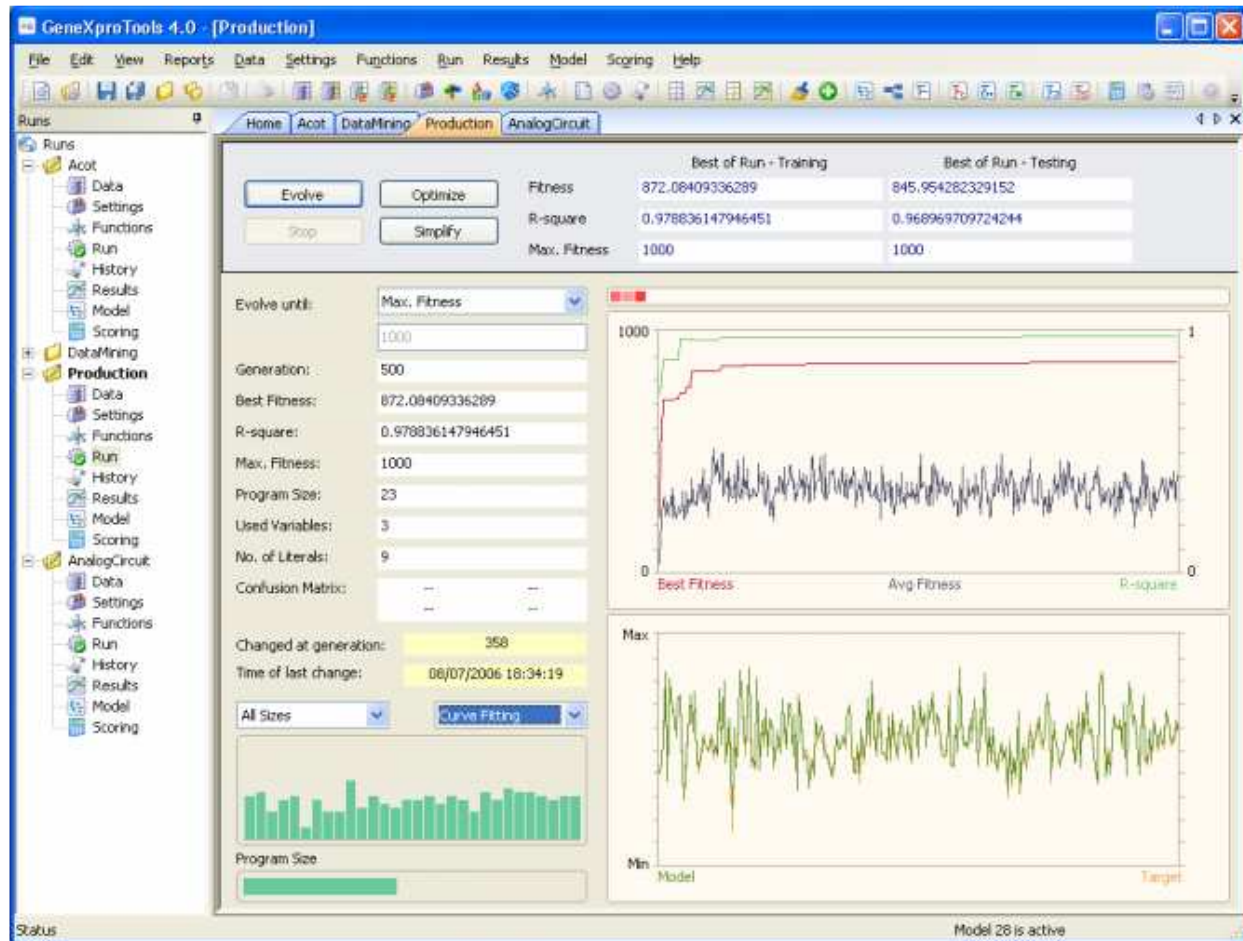
The **Acot(x)** is a well-known mathematical function that you can use to learn your way around GeneXproTools: try making a few runs, optimizing and simplifying a model, check the generated code, see how well the models generalize in the [Results Panel](#), try different fitness functions, different function sets, different architectures, and so on.



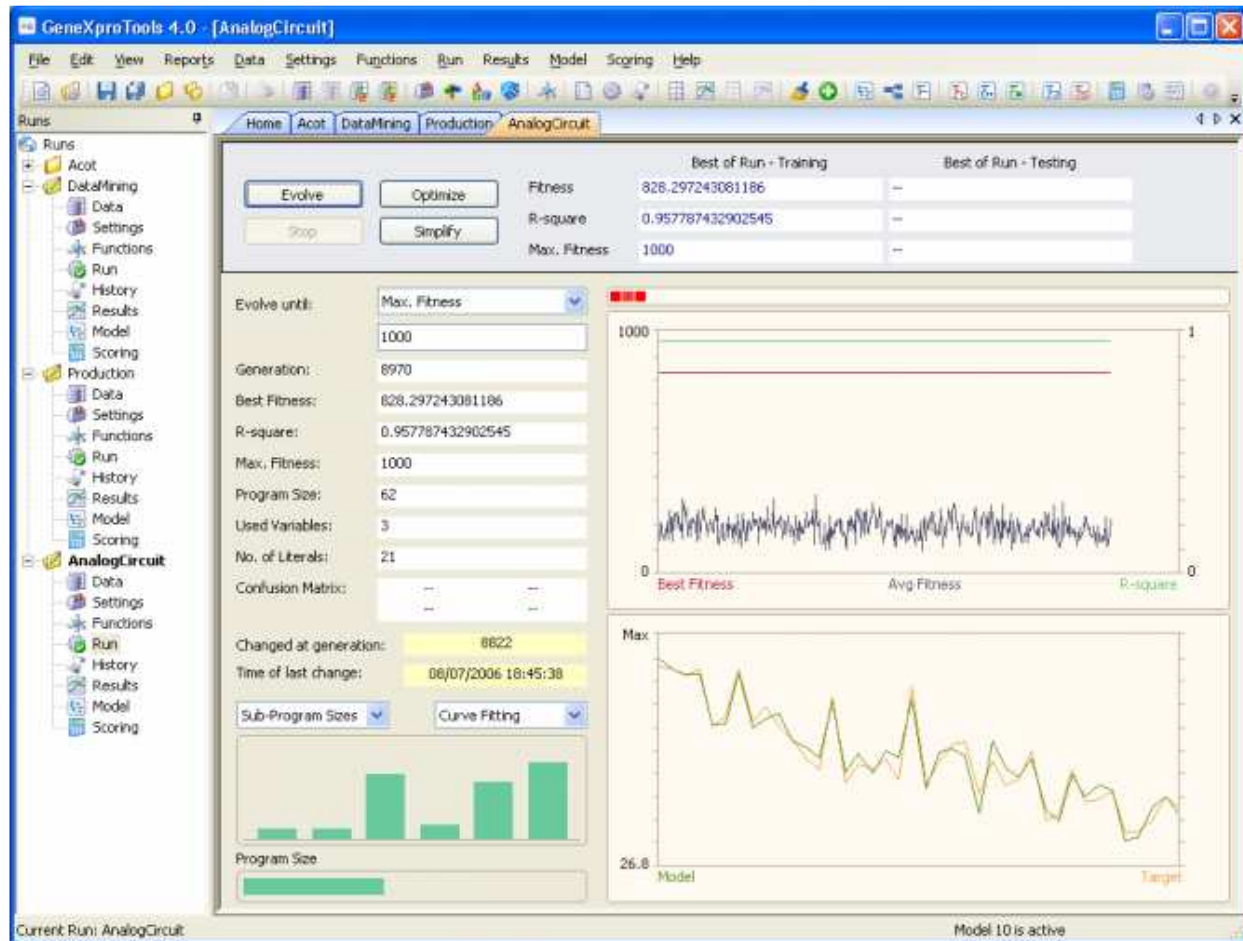
The **Data Mining** sample run is a high-dimensional test problem that illustrates clearly how GeneXproTools can deal quite outstandingly with noisy data. Even though 15 out of 16 variables in this dataset are noise, you will see that GeneXproTools has no problems finding a perfect solution to this problem.



The **Production** sample run is a real-world problem from Economics and the goal is to explain the Production as a function of Labor, Material, and Capital. You will see that this is an easy problem for GeneXproTools and exceptionally good models are obtained in virtually all runs.



The **Analog Circuit Design** sample run is a challenging problem that illustrates how well GeneXproTools approximates a function, fine-tuning it with greater and greater precision. Here, the goal is to find the transfer function expressing the yield of an analog circuit in terms of three parameter tolerances. The training set, consisting of $n = 40$ pairs of tolerances and their corresponding yields, was obtained from n runs of Monte Carlo simulations.



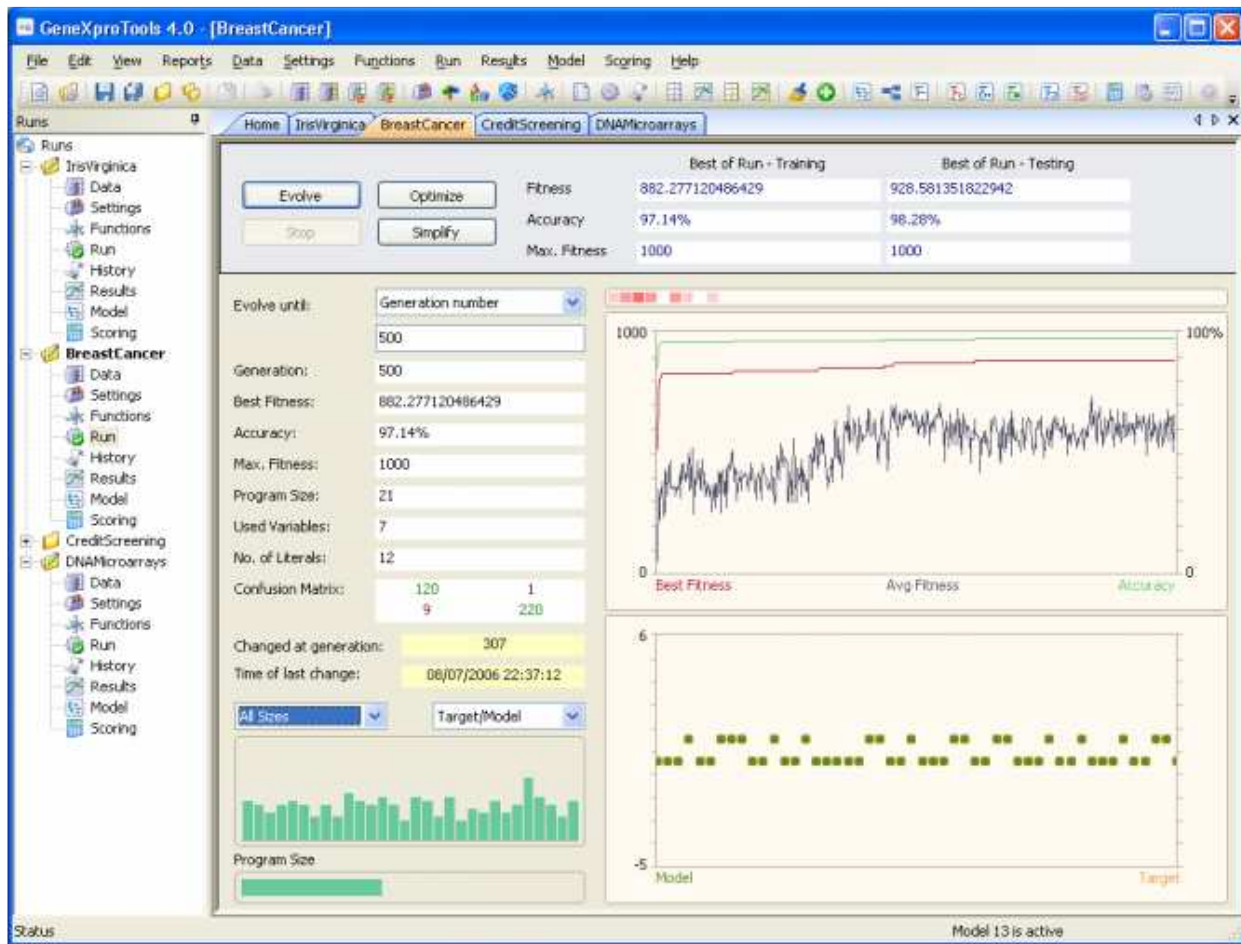
Settings and Features

Settings and Features for Classification Problems

The settings and features of GeneXproTools 4.0 available for Classification problems are organized as follows:

- [New Run Wizard](#)
- [Report Panel](#)
- [Data Panel](#)
- [Change/Add Dataset Engine](#)
- [Settings Panel](#)
- [Functions Panel](#)
- [Run Panel](#)
- [History Panel](#)
- [Results Panel](#)
- [Model Panel](#)
- [Change Seed Window](#)

- Scoring Panel
- Menus
- Run Templates
- Sample Runs

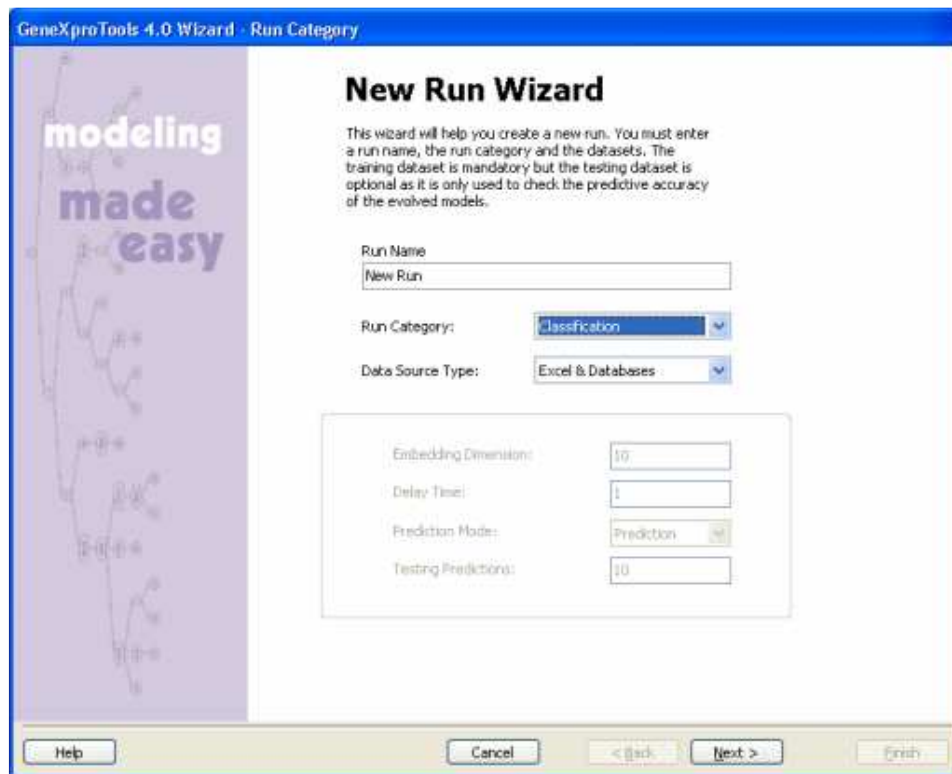


Settings and Features

New Run Wizard

The **New Run Wizard** of GeneXproTools 4.0 includes the following windows:

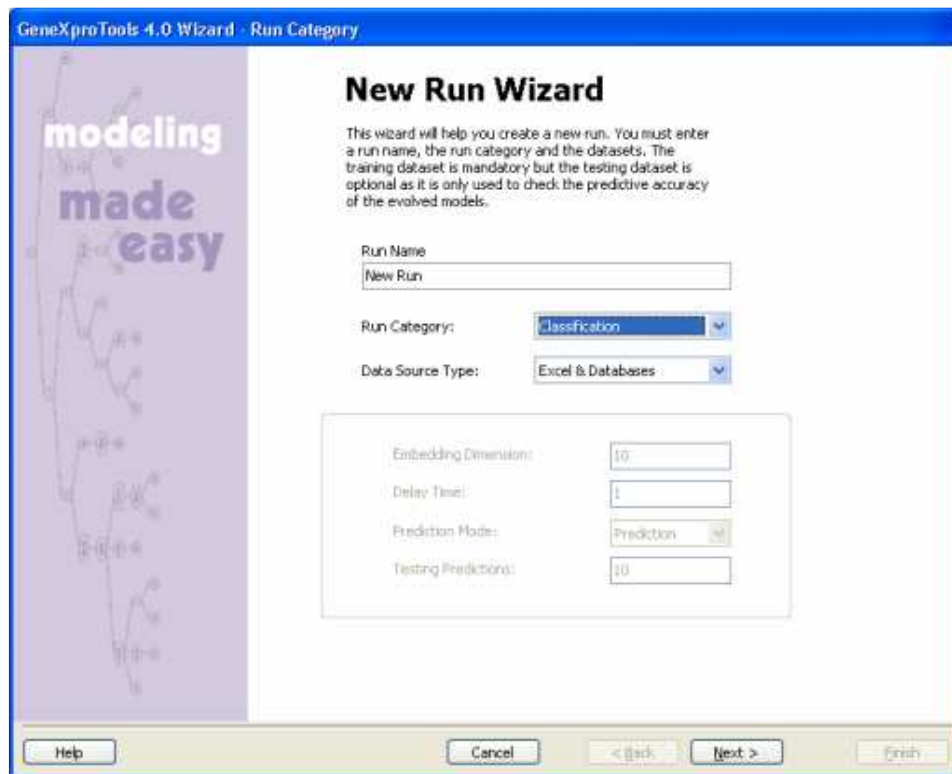
- Run Category Window
- Training Data Window - Text Files
- Testing Data Window - Text Files
- Training Data Window - Excel & Databases
- Testing Data Window - Excel & Databases
- Data Source Dialog Box - Excel & Databases



Settings and Features

Run Category Window

The **Run Category Window** gives access to the following settings and features:

**Run Name**

The name of the .gep file in which the run will be saved.

Run Category

For classification problems, you must obviously choose Classification.

Data Source Type

The type of data source you are using for modeling: Text Files or Excel & Databases.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Next Button

Takes you to the next window of the New Run Wizard.



Settings and Features

Training Data Window - Text Files

The **Training Data Window** gives access to the following settings and features:

GeneXproTools 4.0 Wizard - Training Data

Select a data source for training

File Path: D:\AP5\Version4.0\HelpExamples\Classificac...

Data Matrix Format: Samples x Variables

Separator: Tab ☐ First row contains labels

File Name: Iris-Virginica-train.txt

Columns: 5

Rows: 100

☒ Show loaded data

d2	d3	D.V.
3.3	1	0
1.9	0.4	0
1	0.2	0
1.4	0.3	0
5	2	1
5.9	2.3	1
1.3	0.3	0
5.5	2.1	1
3.3	1	0
1.4	0.2	0
1.5	0.4	0
5.7	2.5	1
4.6	1.3	0
1.6	0.2	0
1.3	0.2	0
1.4	0.1	0
1.3	0.2	0
4	1.3	0
5	1.7	0
5.1	1.9	1
1.6	0.2	0
5.6	2.4	1
1.4	0.2	0
1.2	0.2	0

Showing 5 columns and 100 rows: Loaded 5 columns and 100 rows.

File Path

Click the button on the right of the list box to load the training file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable or class occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable or class occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. And obviously, only 0's and 1's are acceptable for the dependent variable or class.

Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the training dataset.

Rows

GeneXproTools shows in this box the number of rows (samples) in the training dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the training set.

Clear Button

Clears the training set.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

Opens the Save As dialog box to save the new run.



Settings and Features

Testing Data Window - Text Files

The **Testing Data Window** gives access to the following settings and features:

GeneXproTools 4.0 Wizard - Testing Data

Select a data source for testing (optional)

File Path: D:\AP5\Version4.0\HelpExamples\Classificac...

Data Matrix Format: Samples x Variables

Separator: Tab ☐ First row contains labels

File Name: Iris-Virginica-test.txt

Columns: 5

Rows: 50

☒ Show loaded data

d1	d2	d3	D.V.
2.9	5.6	1.8	1
3.9	1.3	0.4	0
2.9	4.7	1.4	0
3	1.1	0.1	0
3.3	4.7	1.6	0
3.4	1.4	0.2	0
3.1	1.5	0.1	0
3	4.8	1.8	1
2.8	5.1	2.4	1
3	6.1	2.3	1
3	5.2	2	1
3.1	5.4	2.1	1
3	5.1	1.8	1
3.2	4.7	1.4	0
3.8	6.4	2	1
3.4	1.5	0.2	0
3.1	5.5	1.8	1
3.2	4.5	1.5	0
3.7	1.5	0.2	0
2.8	4.8	1.4	0
2.8	5.6	2.1	1
2.7	3.9	1.4	0
3	5.5	1.8	1
3	2.5	1	0

Showing 5 columns and 50 rows; Loaded 5 columns and 50 rows.

File Path

Click the button on the right of the list box to load the testing file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable or class occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable or class occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. And obviously, only 0's and 1's are acceptable for the dependent variable or class.

Separator

GeneXproTools identifies automatically the separator in the testing file and shows it in the combo box under this label. And if the variables in your testing data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the testing dataset.

Rows

GeneXproTools shows in this box the number of rows (samples) in the testing dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the testing set.

Clear Button

Clears the testing set.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

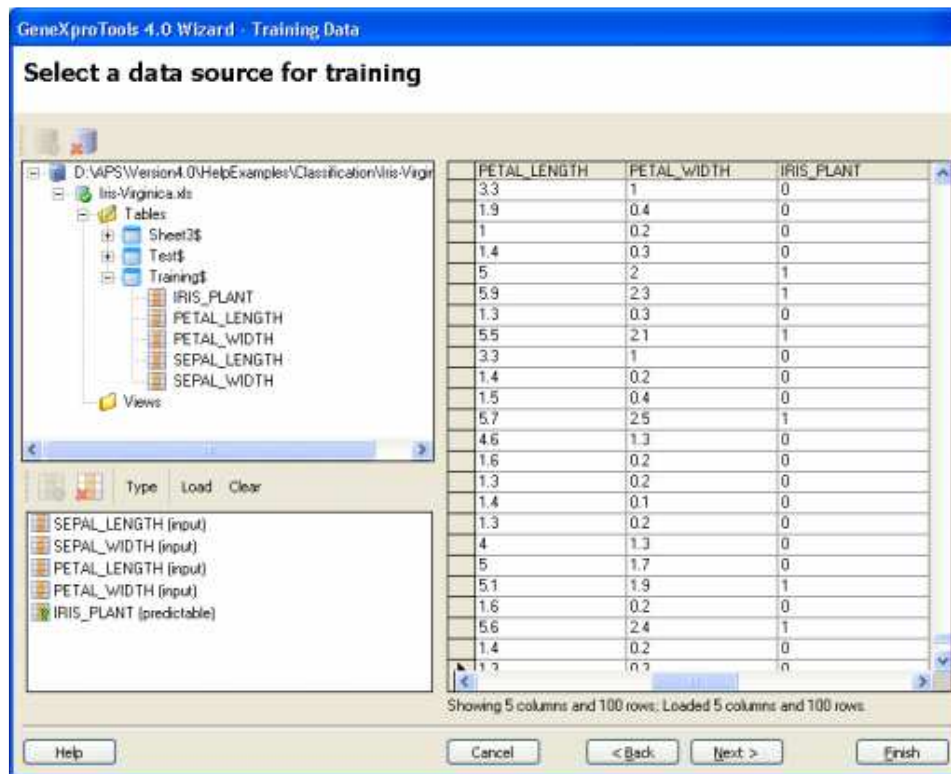
Opens the Save As dialog box to save the new run.



Settings and Features

Training Data Window - Excel & Databases

The **Training Data Window** gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Training Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Training Data Window.

Training Data Window

You prepare your training data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. And obviously, only 0's and 1's are acceptable for the dependent variable or class.

Training Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

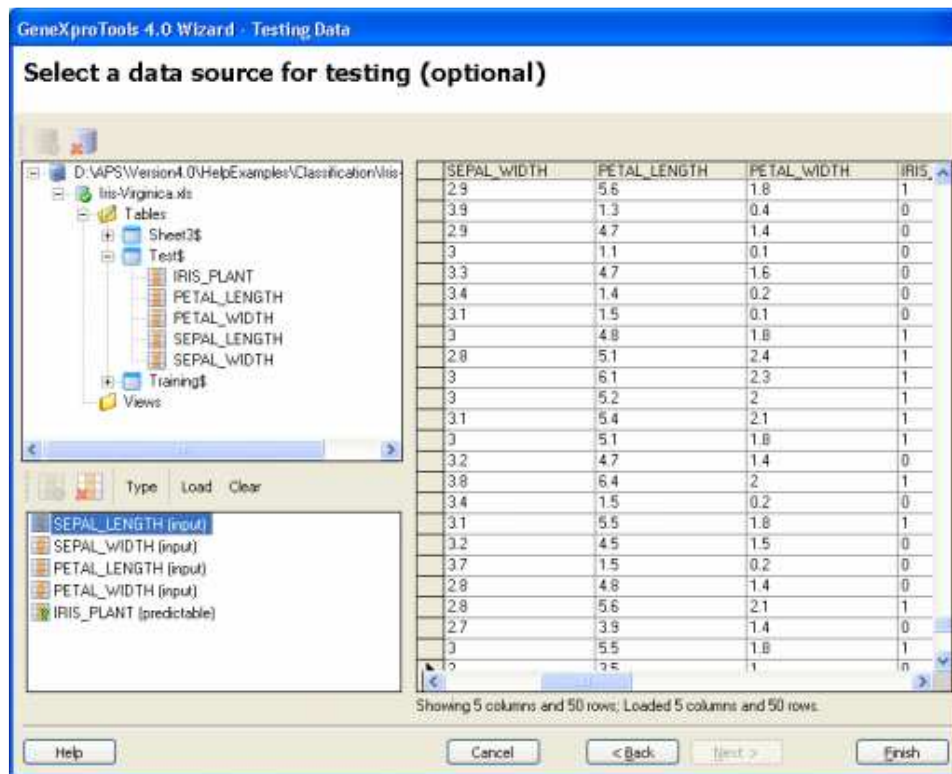
Opens the Save As dialog box to save the new run.



Settings and Features

Testing Data Window - Excel & Databases

The **Testing Data Window** gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Testing Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Testing Data Window.

Testing Data Window

You prepare your testing data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. And obviously, only 0's and 1's are acceptable for the dependent variable or class.

Testing Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

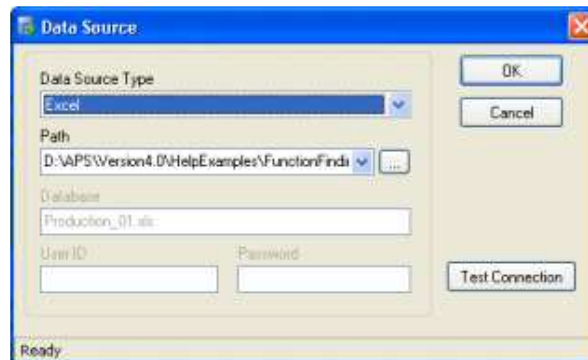
Opens the Save As dialog box to save the new run.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:

**Driver**

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.

Microsoft Access: path to the database file.

Microsoft SQL Server: server name or IP address.

Oracle: server name or IP address.

MySQL: server name or IP address.

DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.

Oracle - Name and database instance (HOST/INSTANCE).

MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

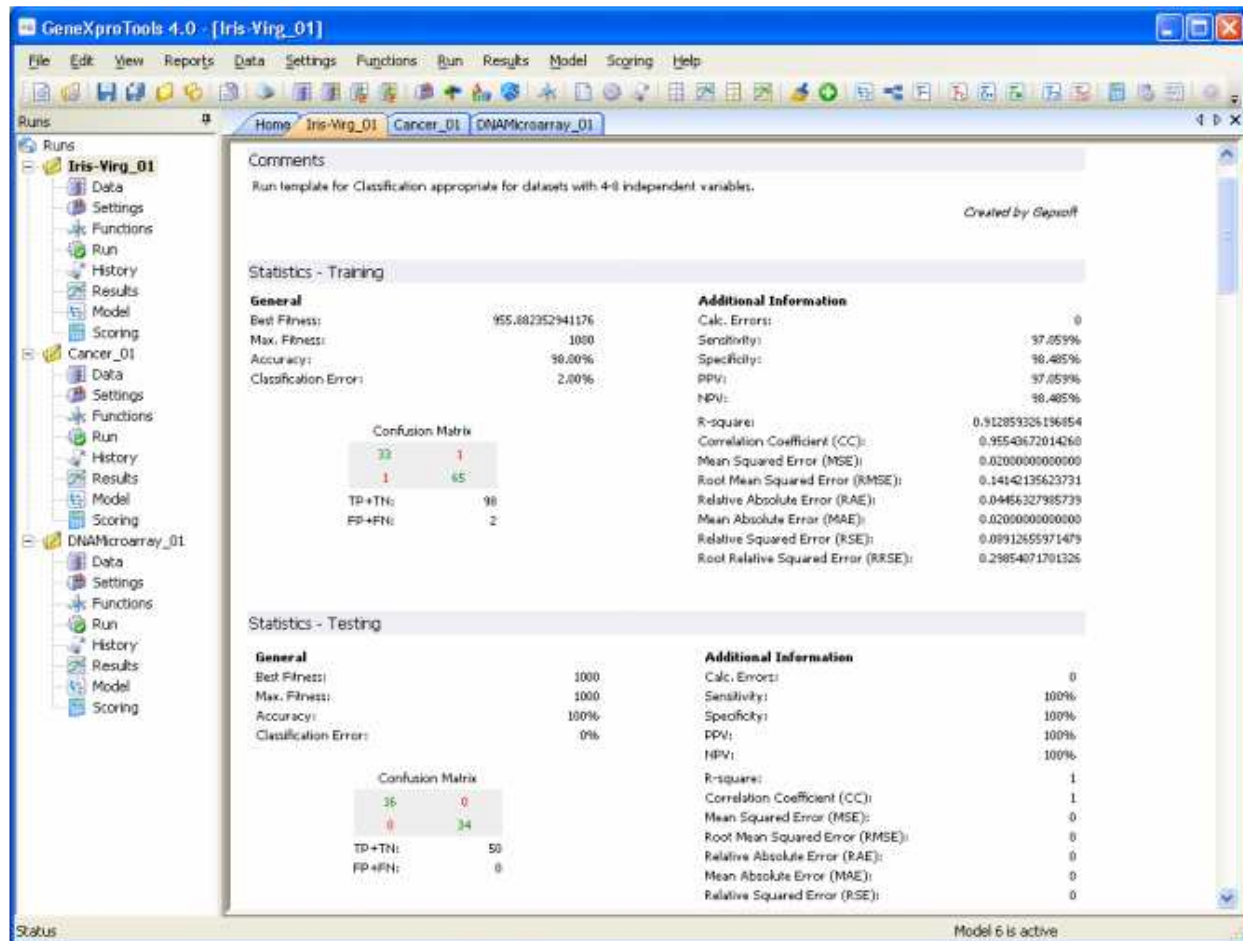
Cancels all the action on the data source connection.



Settings and Features

Report Panel

The **Report Panel** of GeneXproTools 4.0 keeps a summary of all the relevant information of a run, and is organized into the following headings:



Comments

This heading is usually used to give a small description of the problem, the objectives, the kind of data used, their source, etc. Both the comments and the author's name are introduced by choosing Properties on the File Menu.

Statistics – Training

Under this heading are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the training set: [classification error](#), [classification accuracy](#), [confusion matrix](#) (true positives, true negatives, false positives, and false negatives), [sensitivity](#), [specificity](#), [positive predictive value](#), [negative predictive value](#), [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), and [R-square](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, and number of calculation errors (they should always be zero, unless the seed was modified or the training set was changed after the model had been created).

A note of warning, though: All the parameters listed here refer to the last evaluated model in the [Results Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
0	41	20	d(4), d1(3), d3(7), d3(6)	964.546464646465	1000	97.00%	100%
LT2F X3 d3 LT2A Sin LT2F -- d0 d1 d2 d3 d1 d1 d2 d1 d1 + LT2F d2 OR1 LT2B GOE2C LT2G GOE2E GOE2A d2 d3 d2 d3 d0 d2 d3 d3 d3 + / X3 d0 LT2A X2 - GOE2G GOE2A d0 d1 d2 d3 d2 d2 d1 d0 d2							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
50	37	17	d(3), d1(3), d2(6), d3(5)	969.696969697	1000	98.00%	100%
/ X3 d0 LT2A X2 -- GOE2A d0 d3 d3 d1 d2 d2 d3 d2 d2 + LT2F X3 d3 LT2A Sin LT2E -- d2 d1 d2 d3 d0 d0 d2 d2 d1 + LT2G d3 LT2A Sin X3 d1 * - d2 d2 d2 d3 d0 d0 d2 d3 d2							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
608	41	20	d(4), d1(5), d2(3), d3(6)	969.696969697	1000	98.00%	100%
/ X3 - LT2A LT2C -- GOE2A d3 d0 d1 d3 d2 d0 d2 d0 d1 + LT2G d3 LT2E Sin GOE2E d1 LT2F AND1 d1 d3 d0 d3 d2 d2 d3 d0 d1 + LT2F X3 d3 LT2A Sin OR2 GOE2C - d3 d1 d2 d1 d3 d2 d0 d3 d0							

Statistics – Testing

Under this heading are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the testing set, that is, the efficiency of your model at predicting unknown behavior: [classification error](#), [classification accuracy](#), [confusion matrix](#) (true positives, true negatives, false positives, and false negatives), [sensitivity](#), [specificity](#), [positive predictive value](#), [negative predictive value](#), [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), and [R-square](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, and number of calculation errors.

Note again that all the parameters listed here refer to the last evaluated model in the [Results Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
0	40	21	d0(6), d1(7), d2(4), d3(4)	938.393939393939	1000	96.00%	100%
-- LT2B.GOE2F.d0.d0.Atan.d2.d3.d1.d3.d1.d1.d1.d3.d3.d1 + ++ LT2F.AND2.LT2C.LT2B.*--d3.d0.d1.d2.d1.d3.d2.d1.d1 + --d0.OR1.GOE2A.d1.GOE2F.GOE2B.d0.d3.d0.d1.d2.d3.d1.d3.d1							
21	46	23	d0(8), d1(8), d2(4), d3(3)	955.882352941176	875	98.00%	96.00%
GOE2F.Cos.LT2B./AND1.AND2.d0.Cos.d3.d1.d0.d2.d1.d0.d0.d3.d1 + +.GOE2A.LT2F.AND2.LT2C.--*.OR2.d1.d0.d1.d2.d1.d3.d2.d3.d1 + --d0.OR1.GOE2A.GOE2G.GOE2F.GOE2G.d0.d0.d0.d0.d1.d2.d1.d0.d1							
104	47	25	d0(3), d1(8), d2(10), d3(4)	970.588235294118	937.5	99.00%	98.00%
--d0.OR1.GOE2A+.GOE2F.GOE2G.d2.d2.d2.d2.d1.d3.d1.d0.d3 + AND2././GOE2F.d0.AND2././d1.d1.d2.d1.d1.d2.d2.d3.d3 + +.GOE2F.LT2F.LT2C.LT2C.--*--d3.d0.d1.d2.d1.d3.d2.d3.d2							
247	46	23	d0(3), d1(5), d2(8), d3(7)	970.588235294118	937.5	99.00%	98.00%
--d0.OR1.GOE2A.X3.Cos.Sqrt.d2.d2.d0.d2.d2.d1.d2.d2.d2 + +.GOE2F.LT2F.LT2C.LT2C.--*--d3.d0.d1.d2.d1.d3.d2.d3.d2 + AND2*.LT2C.GOE2F.GOE2F.AND2.GOE2F./d1.d2.d3.d1.d3.d1.d2.d3.d3							

Data

Under this heading are listed the number of samples in the training set actually used during training (remember that GeneXproTools 4.0 allows you to choose a smaller sub-set of the training set to use during training), the number of testing samples used to test the generalizing capabilities of your model (again, not all the samples on the testing set were necessarily used to test your model), and the number of attributes or independent variables in your data.

Program Structure

The program size, the number of literals, and a detailed list of all used variables and respective weights are shown here. This information refers to the active model highlighted in yellow in the History heading.

Program Structure	
Program Size:	11
Literals:	4
Used Variables:	d0(1), d1(2), d2(1)

Function Set

Under this heading are listed all the functions used to create your model, together with the symbols used to represent them in [Karva notation](#), their weights and arities.

Function Set		
Function	Symbol	Weight Arity
Addition	+	1 2
Subtraction	-	1 2
Multiplication	*	1 2
Division	/	1 2
Square root	Sqrt	1 1
Exponential	Exp	1 1
Natural logarithm	Ln	1 1
x to the power of 2	X2	1 1
x to the power of 3	X3	1 1
Sine	Sin	1 1
Cosine	Cos	1 1
Cube root	3Rt	1 1
Arctangent	Atan	1 1

Note that the highest value of arity in your function set determines the structure of each gene, making the length of the **tail** t grow according to the equation:

$$t = h (n-1) + 1$$

where h is the length of the **head** and n is maximum arity.

Settings

Under this heading there are five entries: General Settings, Complexity Increase, Fitness Function, Genetic Operators, and Numerical Constants. Under General Settings are listed the number of chromosomes, number of genes, head size, tail size, Dc size, gene size, and linking function. Under Complexity Increase are listed the number of Generations Without Change, the Number of Tries, and Max. Complexity. The name of the fitness function and the 0/1 rounding threshold are given below the entry Fitness Function. Under Genetic Operators are listed the rates of all the genetic operators used to create genetic variation. And, finally, all the parameters relevant to the manipulation of random numerical constants are listed under Numerical Constants.

Settings	
General	
Chromosomes:	30
Genes:	3
Head Size:	8
Tail Size:	9
Dc Size:	9
Gene Size:	26
Linking Function:	Addition
Complexity Increase	
Generations without Change:	2000
Number of Tries:	3
Max. Complexity:	5
Fitness Function	
Error Type:	Sensitivity/Specificity
0/1 Rounding Threshold:	0.5
Genetic Operators	
Mutation Rate:	0.044
Inversion Rate:	0.1
IS Transposition Rate:	0.1
RIS Transposition Rate:	0.1
One-Point Recombination Rate:	0.3
Two-Point Recombination Rate:	0.3
Gene Recombination Rate:	0.1
Gene Transposition Rate:	0.1
Numerical Constants	
Constants per Gene:	2
Data Type:	Floating-Point
Lower Bound:	-10
Upper Bound:	10
RNC Mutation:	0.01
Dc Mutation:	0.044
Dc Inversion:	0.1
Dc IS Transposition:	0.1

Data Sources

Under this heading the paths to the input files (training and testing files) used to create your model are given so that their origin could be easily identified.

File Path

Under this heading the path to the gep file that was used to save your model is given so that it can be easily traced for future reference.

History

Under this heading are listed all the best-of-generation models, including obviously the best-of-run model, discovered during the evolutionary process. Note, however, that the notation "best-of-run" and "best-of-generation" refers to the performance of these models on the training set, which could be slightly different on the testing set. This is why GeneXproTools 4.0 allows you to analyze the

performance of all these models on the testing set and, if appropriate, choose the best model based on these results. GeneXproTools 4.0 gives you the possibility of highlighting in yellow one of these models (the active model) and we advise you to use this feature to highlight the best one. Note also that, for a quick reference, not only the program structure (program size, number of literals, and list of used variables and respective weights) but also the fitness and the [accuracy](#) of all these models both on the training and testing sets are also given, although, for the testing set, they must be evaluated in the [History Panel](#).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
0	41	20	d0(4), d1(3), d2(7), d3(6)	96454646464645	1000	97.00%	100%
LT2F X3.d3.LT2A.Sin.LT2F -- d0.d1.d2.d3.d1.d1.d2.d1.d1 + LT2F.d2.OR1.LT2B.GOE2C.LT2G.GOE2E.GOE2A.d2.d3.d2.d3.d0.d2.d3.d3.d3 + X3.d0.LT2A.X2 - GOE2G.GOE2A.d0.d1.d2.d3.d2.d2.d1.d0.d2							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
50	37	17	d0(3), d1(3), d2(6), d3(5)	96969696969697	1000	98.00%	100%
X3.d0.LT2A.X2 -- GOE2A.d0.d3.d3.d1.d2.d2.d3.d2.d2 + LT2F.X3.d3.LT2A.Sin.LT2E -- d2.d1.d2.d3.d0.d0.d2.d2.d1 + LT2G.d3.LT2A.Sin.X3.d1 * - d2.d2.d2.d3.d0.d0.d2.d3.d2							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
608	41	20	d0(4), d1(5), d2(3), d3(8)	96969696969697	1000	98.00%	100%
X3 -- LT2A.LT2C -- GOE2A.d3.d0.d1.d3.d2.d0.d2.d0.d1 + LT2G.d3.LT2E.Sin.GOE2E.d1.LT2F.AND1.d1.d3.d0.d3.d2.d2.d3.d0.d1 + LT2F.X3.d3.LT2A.Sin.OR2.GOE2C - d3.d1.d2.d1.d3.d2.d0.d3.d0							

For brevity sake, all the best-of-generation models are represented in the compact [Karva notation](#), but you can translate any one of them into a more canonical representation by activating any of these models in the [History Panel](#) and then translate it into the language of your choice in the [Model Panel](#).

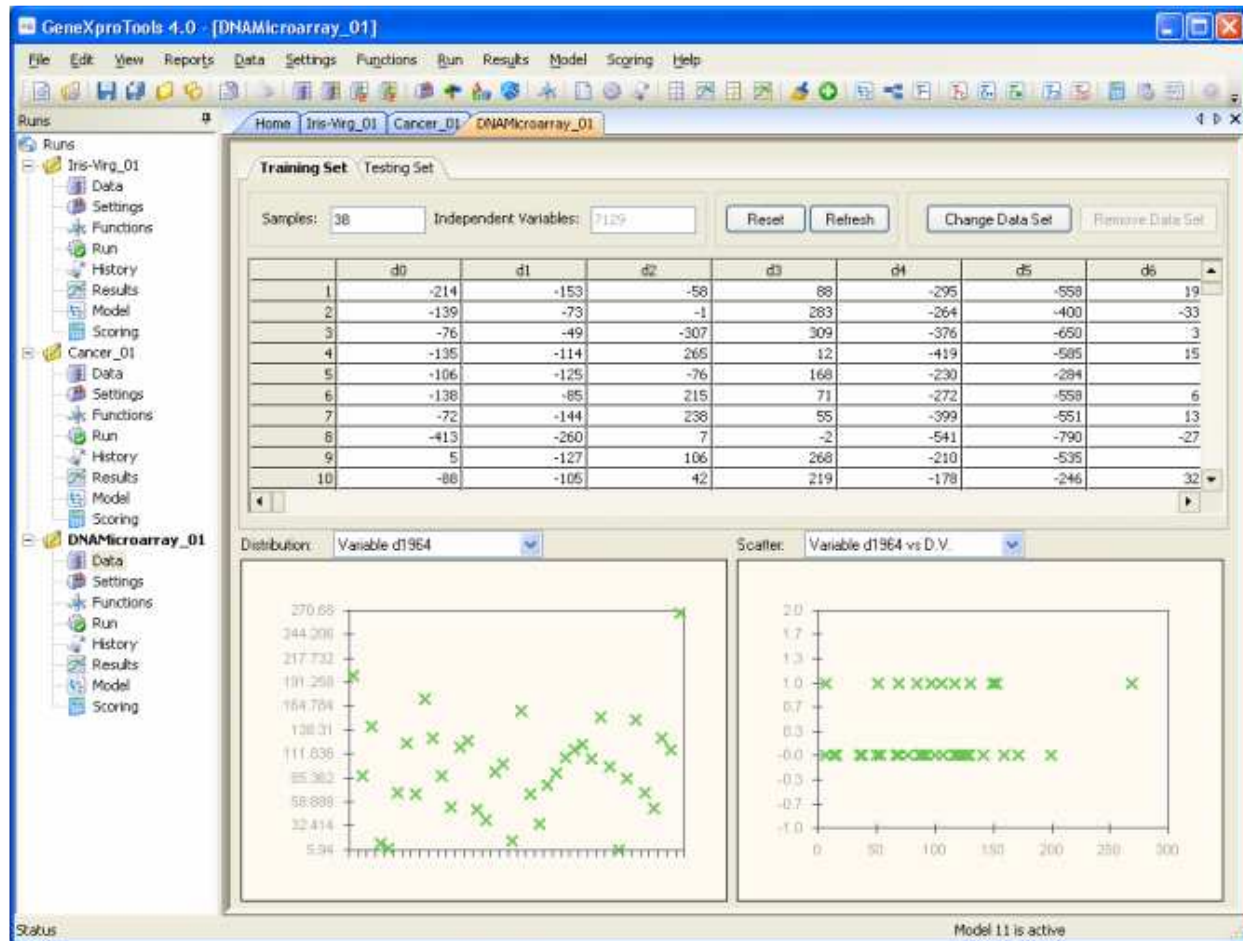


Settings and Features

Data Panel

The **Data Panel** of GeneXproTools 4.0 includes the following tabs:

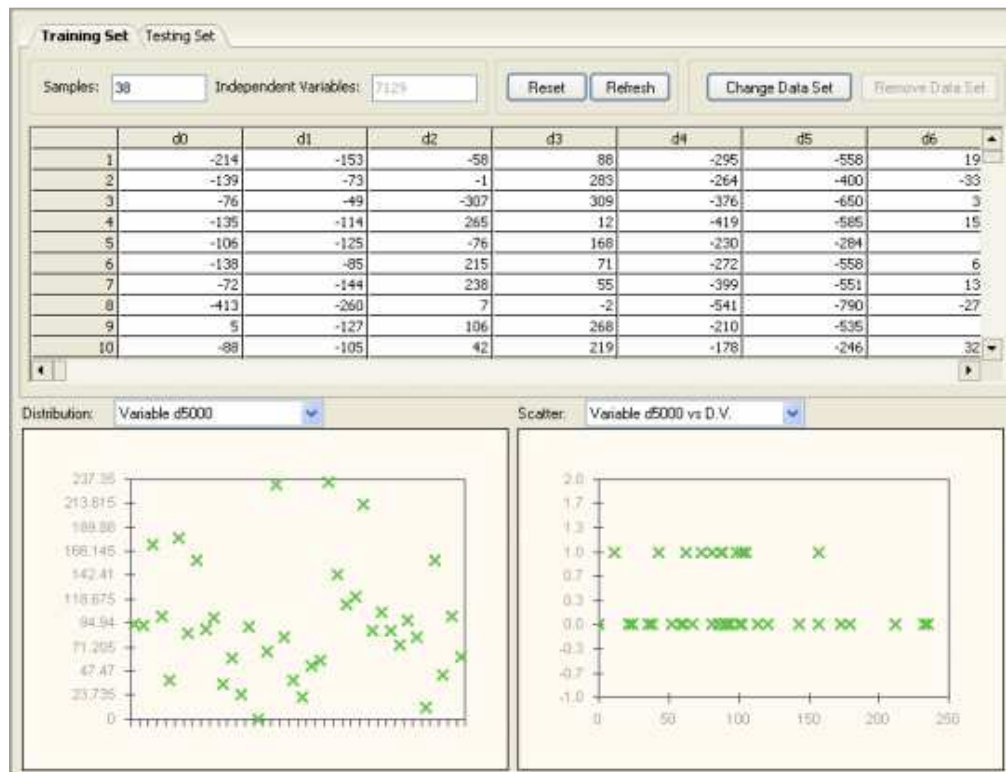
- [Training Set Tab](#)
- [Testing Set Tab](#)



Settings and Features

Training Set Tab

The **Training Set Tab** gives access to the following settings and features:



Samples

Allows you to select a subset of samples for training.

Independent Variables

This box is informative only and shows the number of independent variables or attributes in the training set.

Reset Button

Reverts to the total number of training samples in your data.

Refresh Button

Refreshes the panel after changing the training data set.

Change Data Set Button

Allows you to choose another training set for modeling. Changing a training set by another with the same structure, that is, the same number of independent variables, is very useful for model refreshing and we advise you to pre-test your models in the [History Panel](#) before proceeding with refreshing. In addition, GeneXproTools 4.0 also allows you to replace the training set by another one with a different structure. Obviously, in this case, all the models in the run history will be invalidated and the testing set, if there is one, will be automatically removed.

Table

Shows all the variables in the training set. The entire table can be copied to the clipboard by choosing Copy on the Edit Menu. You can also copy selected columns to the clipboard with the mouse right-click. This is particularly useful in huge datasets such as DNA microarrays for selecting relevant variables identified in particularly good models.

Distribution

Allows you to visualize the distribution of values for each variable in the training set. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first

visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.

Scatter

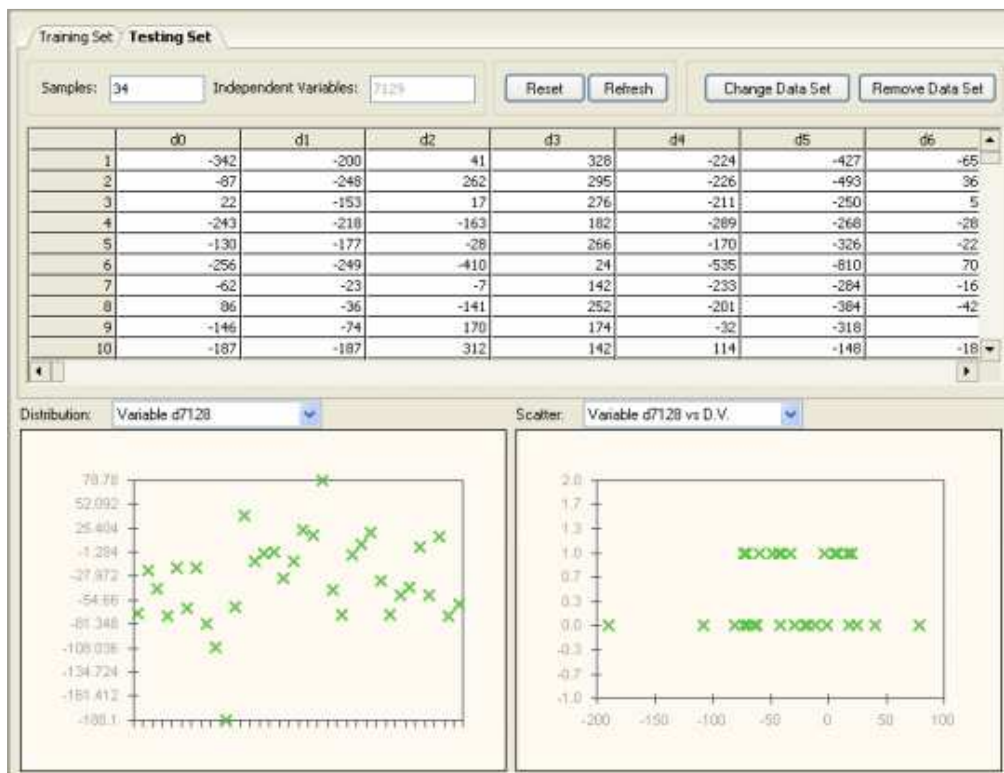
Allows you to visualize the scatter plot of each independent variable in the training set against the dependent variable. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.



Settings and Features

Testing Set Tab

The **Testing Set Tab** gives access to the following settings and features:



Samples

Allows you to select a subset of samples for testing.

Independent Variables

This box is informative only and shows the number of independent variables or attributes in the testing set.

Reset Button

Reverts to the total number of testing samples in your data.

Refresh Button

Refreshes the panel after changing the testing data set.

Change Data Set Button

Allows you to choose another testing set for evaluating the generalizing capabilities of your models. The new testing set must obviously have the same structure of the training set, that is, the same number of independent variables.

Remove Data Set Button

Allows you to remove the testing set (you can always add it later by selecting Add Testing Data on the Data Menu).

Table

Shows all the variables in the testing set. The entire table can be copied to the clipboard by choosing Copy on the Edit Menu. You can also copy selected columns to the clipboard with the mouse right-click. This is particularly useful in huge datasets such as DNA microarrays for selecting relevant variables identified in particularly good models.

Distribution

Allows you to visualize the distribution of values for each variable in the testing set. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.

Scatter

Allows you to visualize the scatter plot of each independent variable in the testing set against the dependent variable. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.



Settings and Features

Change/Add Dataset Engine

The **Change/Add Dataset Engine** of GeneXproTools 4.0 includes the following windows:

- [Change Training Data Window - Text Files](#)
- [Change/Add Testing Data Window - Text Files](#)
- [Change Training Data Window - Excel & Databases](#)
- [Change/Add Testing Data Window - Excel & Databases](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Change Training Data Window - Text Files

The **Change Training Data Window** allows you to change the training set and gives access to the following settings and features:

The screenshot shows the 'Change Training Data' window with the title 'Select a data source for training'. The settings are as follows:

- File Path:** D:\AP5\Version4.0\HelpExamples\Classificac...
- Data Matrix Format:** Samples x Variables
- Separator:** Tab
- First row contains labels:** ☐
- File Name:** Iris-Virginica-train.txt
- Columns:** 5
- Rows:** 100
- Show loaded data:** ☒
- Buttons:** Load, Clear, Cancel, Finish

The data preview table shows the following values:

d0	d1	d2
5	2.3	3.3
5.1	3.8	1.9
4.6	3.6	1
5.1	3.5	1.4
5.7	2.5	5
6.8	3.2	5.9
4.5	2.3	1.3
6.8	3	5.5
4.9	2.4	3.3
4.6	3.2	1.4
5.7	4.4	1.5
6.7	3.3	5.7
6.6	2.9	4.6
5	3	1.6
4.7	3.2	1.3
4.8	3	1.4
4.4	3	1.3
5.5	2.3	4
6.7	3	5
5.8	2.7	5.1
4.8	3.4	1.6
6.3	3.4	5.6
5.1	3.5	1.4
5	2.5	1.2

Showing 5 columns and 100 rows; Loaded 5 columns and 100 rows.

File Path

Click the button on the right of the list box to load the training file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable or class occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable or class occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. And obviously, only 0's and 1's are acceptable for the dependent variable or class.

Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the training dataset. You can replace the old training set by another one with a different number of variables, but obviously, in this case, all the models in the run history will be invalidated and the testing set (if there is one) will be automatically removed by GeneXproTools.

Rows

GeneXproTools shows in this box the number of rows (samples) in the training dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns on the table of the Change Training Data Window, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the training set.

Clear Button

Clears the training set.

Cancel Button

Cancels all the action on the replacement of the training set.

Finish Button

Saves the new training set.



Settings and Features

Change/Add Testing Data Window - Text Files

The **Change/Add Testing Data Window** allows you to change the testing set and gives access to the following settings and features:

Change Testing Data

Select a data source for testing

File Path: D:\AP5\Version4.0\HelpExamples\Classificac... ...

Data Matrix Format: Samples x Variables

Separator: Tab ☐ First row contains labels

File Name: Iris-Virginica-test.txt

Columns: 5

Rows: 50

☒ Show loaded data Load Clear

d0	d1	d2
6.3	2.9	5.6
5.4	3.9	1.3
6.1	2.9	4.7
4.3	3	1.1
6.3	3.3	4.7
5.2	3.4	1.4
4.9	3.1	1.5
6	3	4.8
5.8	2.8	5.1
7.7	3	6.1
6.5	3	5.2
6.9	3.1	5.4
5.9	3	5.1
7	3.2	4.7
7.9	3.8	6.4
5.1	3.4	1.5
6.4	3.1	5.5
6.4	3.2	4.5
5.4	3.7	1.5
6.8	2.8	4.8
6.4	2.8	5.6
5.2	2.7	3.9
6.5	3	5.5
6	3	1.5

Showing 5 columns and 50 rows; Loaded 5 columns and 50 rows.

Cancel Finish

File Path

Click the button on the right of the list box to load the testing file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable or class occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable or class occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. And obviously, only 0's and 1's are acceptable for the dependent variable or class.

Separator

GeneXproTools identifies automatically the separator in the testing file and shows it in the combo box under this label. And if the variables in your testing data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the testing dataset. Your new testing set must have the same number of variables as the training set. If you want to change the number of variables, you must first [change the training set](#).

Rows

GeneXproTools shows in this box the number of rows (samples) in the testing dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns on the table of the Change Testing Data Window, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the testing set.

Clear Button

Clears the testing set.

Cancel Button

Cancels all the action on the replacement of the testing set.

Finish Button

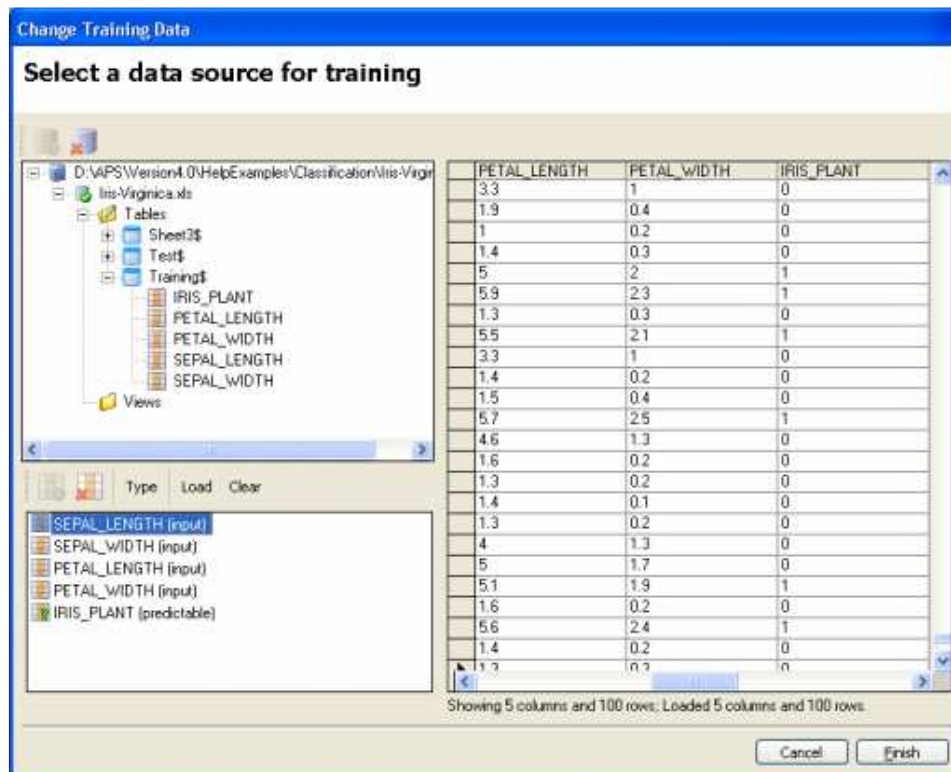
Saves the new testing set.



Settings and Features

Change Training Data Window - Excel & Databases

The **Change Training Data Window** allows you to change the training set and gives access to the following settings and features:

**Add Data Source Icon**

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Training Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Training Data Window.

Training Data Window

You prepare your training data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Recall that only 0's and 1's are acceptable for the dependent variable or class. GeneXproTools allows you to replace the old training set by another one with a different number of variables, but obviously, in this case, all the models in the run history will be invalidated and the testing set (if there is one) will be automatically removed by GeneXproTools.

Training Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns on this table, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Cancel Button

Cancels all the action on the replacement of the training set.

Finish Button

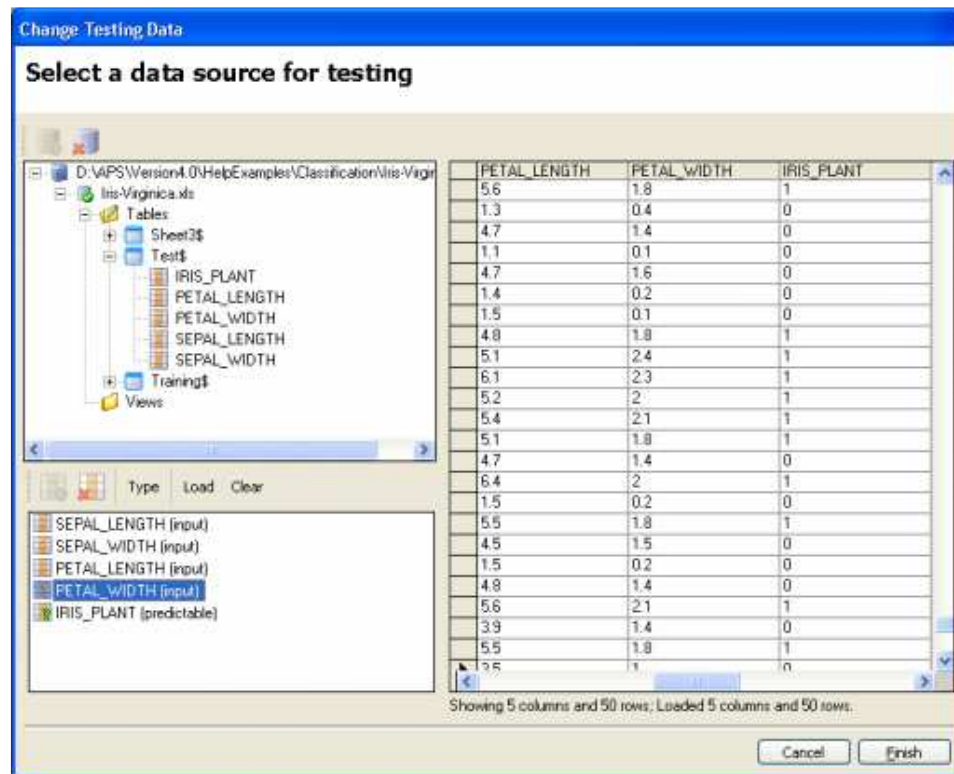
Saves the new training set.



Settings and Features

Change/Add Testing Data Window - Excel & Databases

The **Change/Add Testing Data Window** allows you to change the testing set (or add a new one) and gives access to the following settings and features:

**Add Data Source Icon**

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Testing Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Testing Data Window.

Testing Data Window

You prepare your testing data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Recall that only 0's and 1's are acceptable for the dependent variable or class. Your new testing set must have the same number of variables as the training set. If you want to change the

number of variables, you must first [change the training set](#).

Testing Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns on this table, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Cancel Button

Cancels all the action on the replacement of the testing set.

Finish Button

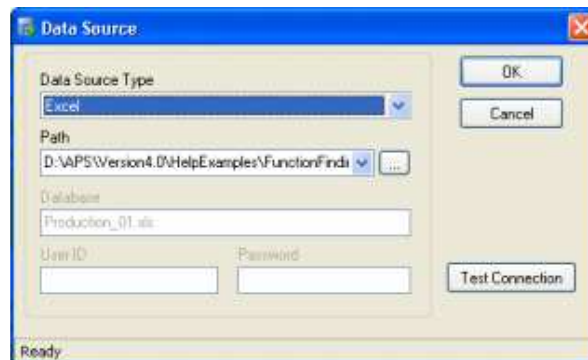
Saves the new testing set.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:



Driver

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.
 Microsoft Access: path to the database file.
 Microsoft SQL Server: server name or IP address.
 Oracle: server name or IP address.
 MySQL: server name or IP address.
 DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.

Oracle - Name and database instance (HOST/INSTANCE).

MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

Cancels all the action on the data source connection.

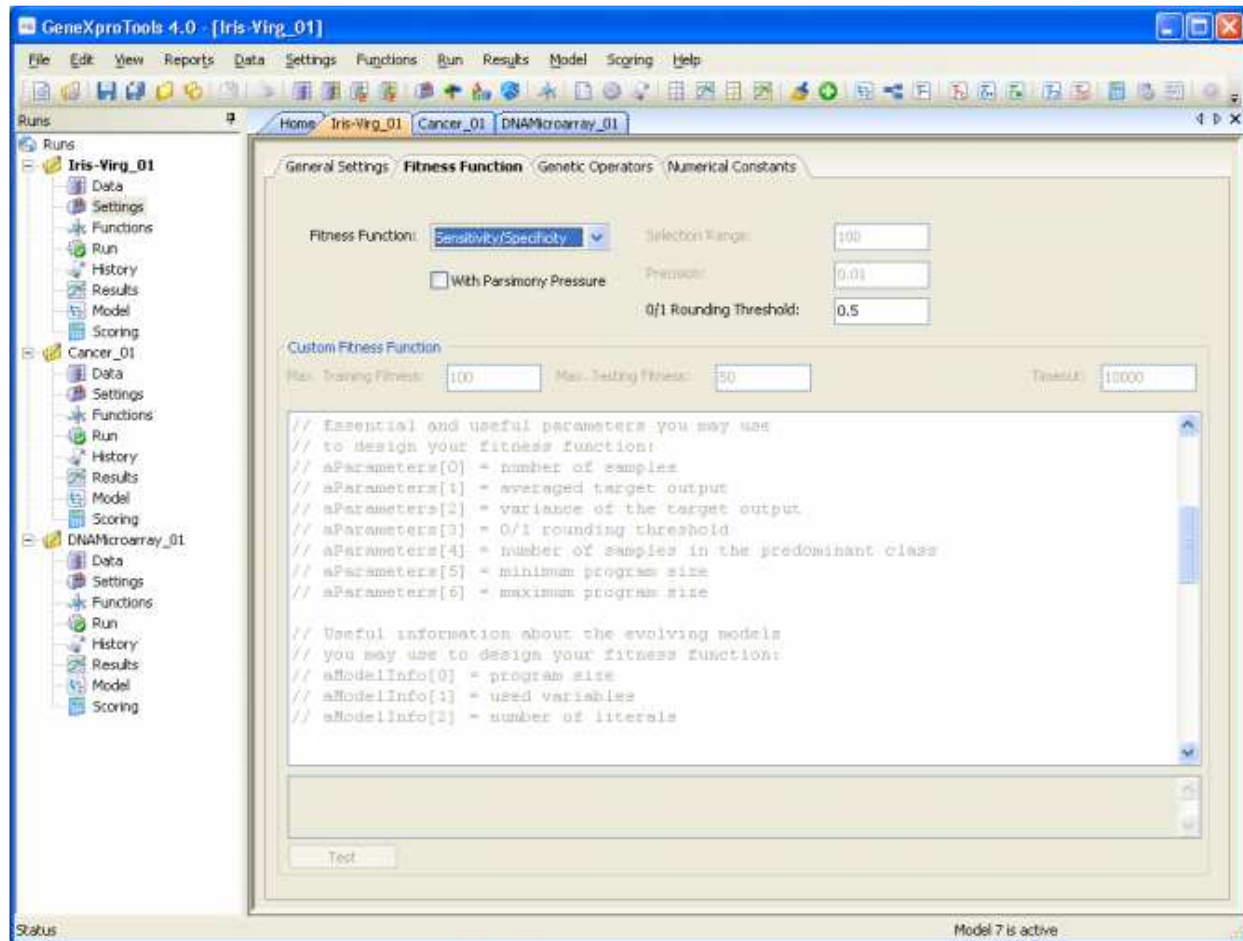


Settings and Features

Settings Panel

The **Settings Panel** of GeneXproTools 4.0 includes the following tabs:

- [General Settings Tab](#)
- [Fitness Function Tab](#)
- [Genetic Operators Tab](#)
- [Numerical Constants Tab](#)



Settings and Features

General Settings Tab

The **General Settings Tab** gives access to the following settings and features:

General Settings Fitness Function Genetic Operators Numerical Constants

Training Samples: 200

Testing Samples: 246

Number of Chromosomes: 30

Head Size: 7

Number of Genes: 3

Linking Function: Addition

☒ Enable Complexity Increase

Generations without Change: 2000

Number of Tries: 3

Max. Complexity: 5

☒ Save Run History

Training Samples

Number of samples used for training.

Reset

Reverts to the total number of training samples in your data.

Testing Samples

Number of samples used for evaluating the generalizing capabilities of the evolved models.

Reset

Reverts to the total number of testing samples in your data.

Number of Chromosomes

Size of the population of models created each generation for breeding new ones.

Head Size

Size of the [head](#) of each gene. This parameter determines the upper limit for the size of the programs encoded in the gene.

Number of Genes

Number of genes per chromosome or, in other words, the number of complex terms in the evolving models.

Linking Function

Function used to link the different terms encoded in the genes of a model. You can choose either addition, subtraction, multiplication, or division to link the [sub-ETs](#) or terms of your models.

Enable Complexity Increase

Check this box if you wish to allow GeneXproTools to automatically increase the number of genes in the evolving models.

Generations without Change

The period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction is created or a neutral gene (an extra term) is automatically added to all evolving models.

Number of Tries

The number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models.

Max. Complexity

The maximum number of terms (genes) you think reasonable for modeling your data. Beyond this threshold no other terms are introduced automatically by the Complexity Increase Engine and GeneXproTools stops evolving new models.



Settings and Features

Fitness Function Tab

The **Fitness Function Tab** gives access to the following settings and features:

General Settings **Fitness Function** Genetic Operators Numerical Constants

Fitness Function: Sensitivity/Specificity Selection Range:

☐ With Parsimony Pressure Precision:

0/1 Rounding Threshold:

Custom Fitness Function

Max. Training Fitness: Max. Testing Fitness: Timeout:

```
// Essential and useful parameters you may use
// to design your fitness function:
// aParameters[0] = number of samples
// aParameters[1] = averaged target output
// aParameters[2] = variance of the target output
// aParameters[3] = 0/1 rounding threshold
// aParameters[4] = number of samples in the predominant class
// aParameters[5] = minimum program size
// aParameters[6] = maximum program size

// Useful information about the evolving models
// you may use to design your fitness function:
// aModelInfo[0] = program size
// aModelInfo[1] = used variables
// aModelInfo[2] = number of literals
```

Test

Fitness Function

Here you select either one of the 30 basic [built-in fitness functions](#) of GeneXproTools 4.0 for Classification problems or a [custom fitness function](#).

With Parsimony Pressure Check Box

By checking this box, you select a fitness function with [parsimony pressure](#).

Selection Range

This box is specific of Function Finding and Time Series Prediction problems, and therefore is disabled in Classification mode.

Precision

This box is specific of Function Finding and Time Series Prediction problems, and therefore is disabled in Classification mode.

0/1 Rounding Threshold

This parameter is essential for all [fitness functions](#), as it is used to convert the value returned by the evolved model into "1" or "0". If the value returned by the evolved model is equal to or greater than the rounding threshold, then the record is classified as "1", "0" otherwise.

Custom Fitness Function Frame

Select Custom in the Fitness Function combo box in order to activate this frame. You can now design or edit your own fitness function.

Custom Fitness Function Code Window

You design or edit the code of your own fitness function in the code window. The code must be in Java Script and can be tested for syntax errors before evolving models with it.

Test

Tests the syntax of the Java Script code of your fitness function for syntax errors. The results of the test are written in the gray window below the code window.

Max. Training Fitness

The maximum fitness corresponding to the custom fitness function on the training set. If maximum fitness depends on the number of training samples, don't forget to update this value every time you change the number of training samples, otherwise you might be creating some inaccuracies or errors. For instance, the plot of the Evolutionary Dynamics on the [Run Panel](#) won't show correctly if your maximum fitness is not correctly set.

Max. Testing Fitness

The maximum fitness corresponding to the custom fitness function on the testing set. If maximum fitness depends on the number of testing samples, don't forget to update this value every time you change the number of testing samples, otherwise you might be creating some inaccuracies.

Timeout

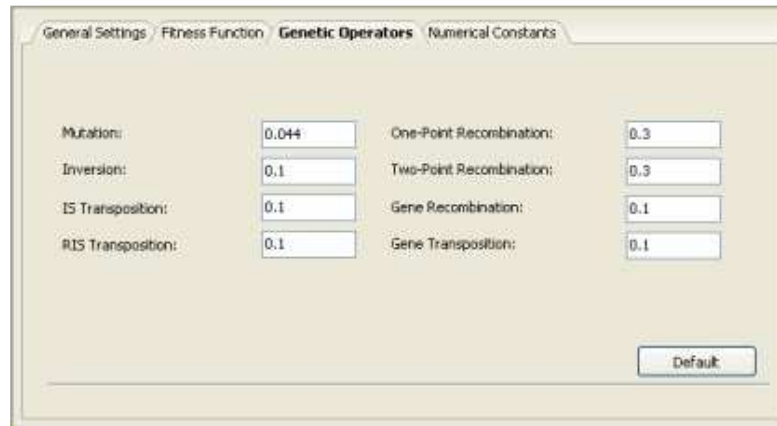
Despite one's best efforts, the code one sometimes writes contains errors that keep computers in infinite loops. This setting gives you a gracious way out of such loops.



Settings and Features

Genetic Operators Tab

The **Genetic Operators Tab** gives access to the following settings and features:



Mutation

The [mutation](#) rate used to create genetic variation. The default is 0.044.

Inversion

The [inversion](#) rate used to create genetic variation. The default is 0.1.

IS Transposition

The [IS transposition](#) rate used to create genetic variation. The default is 0.1.

RIS Transposition

The [RIS transposition](#) rate used to create genetic variation. The default is 0.1.

One-Point Recombination

The [one-point recombination](#) rate used to create genetic variation. The default is 0.3.

Two-Point Recombination

The [two-point recombination](#) rate used to create genetic variation. The default is 0.3.

Gene Recombination

The [gene recombination](#) rate used to create genetic variation. The default is 0.1.

Gene Transposition

The [gene transposition](#) rate used to create genetic variation. The default is 0.1.

Default

Resets the rates of all the above genetic operators to the default values.



Settings and Features

Numerical Constants Tab

The **Numerical Constants Tab** gives access to the following settings and features:

General Settings Fitness Function Genetic Operators **Numerical Constants**

☒ Use Random Numerical Constants

Constants per Gene: 2 Lower Bound: -10

Data Type: Floating-Point Upper Bound: 10

RNC Mutation: 0.01

Dc Mutation: 0.044

Dc Inversion: 0.1

Dc IS Transposition: 0.1

Default

Use Random Numerical Constants

Check this box if you want to use the [GEP-RNC learning algorithm](#) of GeneXproTools. This algorithm has a facility for handling random numerical constants and it evolves models with them.

Constants per Gene

The number of random numerical constants associated with each gene.

Data Type

The data type (integer or floating-point) you think appropriate for the random numerical constants that will be used in the creation of your models. For Integer constants only integer values in the interval $[-2147483648, 2147483647]$ are acceptable. For Floating-point constants a double-precision floating-point value with a range of $-1.79769313486231E308$ to $-4.94065645841247E-324$ for negative values, $4.94065645841247E-324$ to $1.79769313486231E308$ for positive values, and 0 are acceptable.

Lower Bound

Sets the lower bound of the interval from which random numerical constants are drawn.

Upper Bound

Sets the upper bound of the interval from which random numerical constants are drawn.

RNC Mutation

The [RNC mutation](#) rate used to create genetic variation in the arrays of random numerical constants. The default is 0.01.

Dc Mutation

The [Dc mutation](#) rate used to create genetic variation. The default is 0.044.

Dc Inversion

The [Dc inversion](#) rate used to create genetic variation. The default is 0.1.

Dc IS Transposition

The [Dc IS transposition](#) rate used to create genetic variation. The default is 0.1.

Default

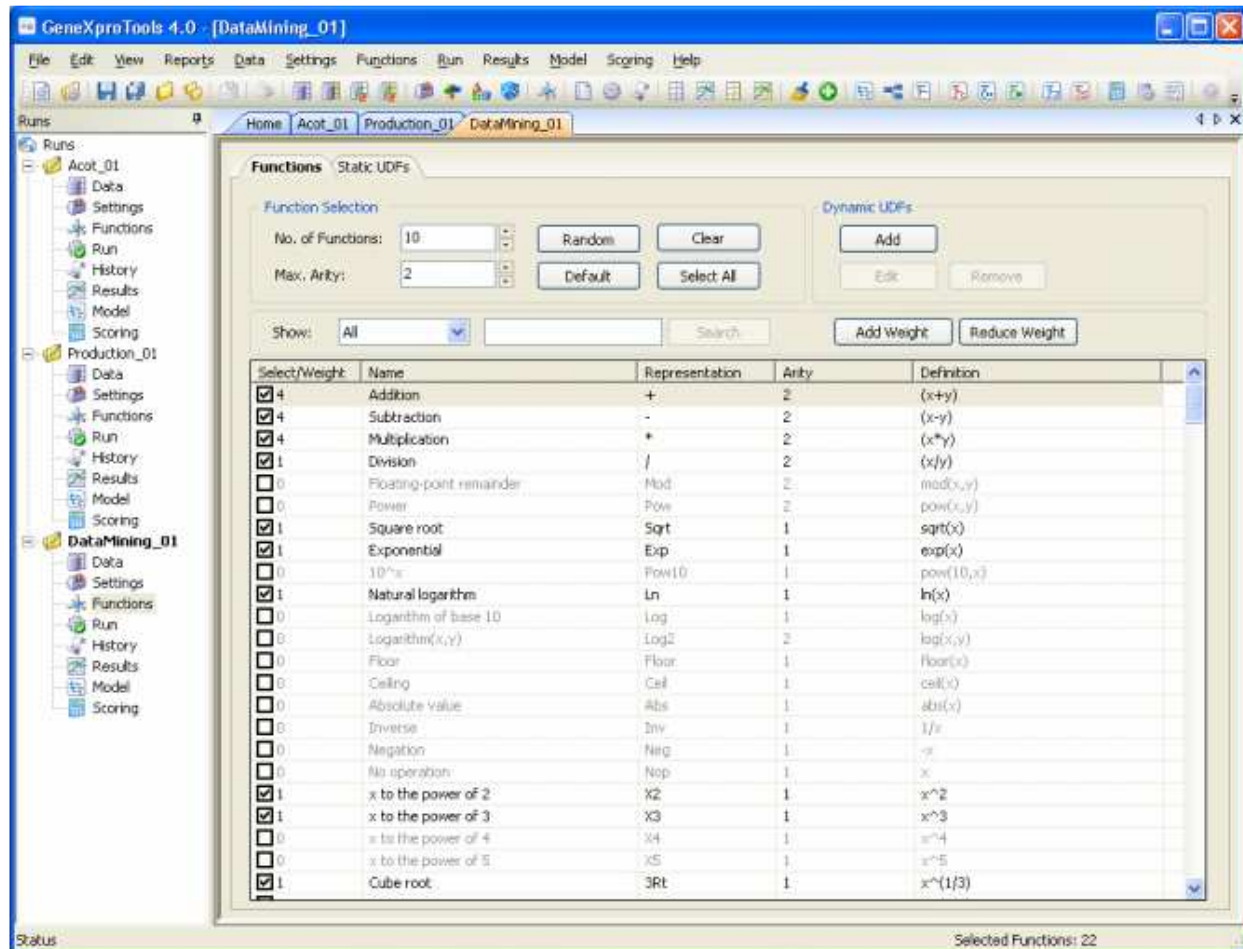
Resets the rates of all the above genetic operators to the default values.

Settings and Features

Functions Panel

The **Functions Panel** of GeneXproTools 4.0 includes the following tabs and windows:

- [Functions Tab](#)
- [Functions Tab – Edit DDF Window](#)
- [Static UDFs Tab](#)
- [Static UDFs Tab – Edit UDF Window](#)



Settings and Features

Functions Tab

The **Functions Tab** lists all the 279 [built-in mathematical functions](#) of GeneXproTools and gives access to the following settings and features:

Functions Selection Framework

Function Selection: No. of Functions: 10, Max. Arity: 2. Buttons: Random, Clear, Add, Edit, Remove, Default, Select All.

Dynamic UDFs: Add, Edit, Remove.

Show: All, Search, Add Weight, Reduce Weight.

Select/Weight	Name	Representation	Arity	Definition
☒ 4	Addition	+	2	(x+y)
☒ 4	Subtraction	-	2	(x-y)
☒ 4	Multiplication	*	2	(x*y)
☒ 1	Division	/	2	(x/y)
☐ 0	Floating-point remainder	Mod	2	mod(x,y)
☐ 0	Power	Pow	2	pow(x,y)
☒ 1	Square root	Sqrt	1	sqrt(x)
☒ 1	Exponential	Exp	1	exp(x)
☐ 0	10^x	Pow10	1	pow(10,x)
☒ 1	Natural logarithm	Ln	1	ln(x)
☐ 0	Logarithm of base 10	Log	1	log(x)
☐ 0	Logarithm(x,y)	Log2	2	log(x,y)
☐ 0	Floor	Floor	1	floor(x)
☐ 0	Ceiling	Ceil	1	ceil(x)
☐ 0	Absolute value	Abs	1	abs(x)
☐ 0	Inverse	Inv	1	1/x
☐ 0	Negation	Neg	1	-x
☐ 0	No operation	Nop	1	x
☒ 1	x to the power of 2	X2	1	x^2
☒ 1	x to the power of 3	X3	1	x^3
☐ 0	x to the power of 4	X4	1	x^4
☐ 0	x to the power of 5	X5	1	x^5
☒ 1	Cube root	3Rt	1	x^(1/3)

Function Selection Framework

Allows the selection of different function sets very quickly through the combination of the Show options with the Random/Default/Clear/Select All buttons plus the Increase/Reduce Weight buttons.

Number of Functions

The number of functions with a certain arity you wish to select randomly by using the Random button.

Maximum Arity

Determines the sub-set of functions from which the Random button will draw a specified number of functions. For instance, by setting maximum arity to 1, only functions of one argument will be drawn; by setting maximum arity to 2, only functions of 1 and 2 arguments will be randomly chosen, and so forth.

Random Button

Selects randomly a specified number of functions belonging to four different sub-sets of functions: functions of just one argument, functions of 1-2 arguments, functions of 1-3 arguments, and functions of 1-4 arguments.

Default Button

Adds the function set of the [run template](#) to the current function set. To revert exactly to the default function set you must press the Clear button before pressing Default.

Clear Button

Clears all the functions from the function set.

Select All Button

Selects all the functions showing currently on the table.

Show List Box

Shows a wide variety of function sub-sets (All functions, Searched functions, Selected Only, Arity 1 Only, Arity 2 Only, Arity 3 Only, Arity 4 Only, and DDFs Only). By selecting Search, the search box

becomes activated and you can enter specific words to find functions that match your criteria in the Name column.

Search Button

Searches, in the Name column, for the functions matching your search words.

Add Weight

Increments by one the weight of each selected function.

Reduce Weight

Reduces by one the weight of each selected function; functions with weight 1, though, remain as they are.

Select/Weight Column

To select a function, check the box on the left. By doing this, the function is included in your function set with the default weight of 1. To increase its weight, click the box below Select/Weight and then introduce its weight.

Name Column

Under this heading is shown the common name of all the [mathematical functions](#) used by GeneXproTools.

Representation Column

Under this heading is shown how all the functions are represented in [Karva notation](#).

Arity Column

Under this heading is shown the arity (number of arguments) of all the functions.

Definition Column

Under this heading is shown the mathematical definition of all GeneXproTools functions and the definition you gave to all the [DDFs](#).

Dynamic UDFs Framework

Allows the introduction of [dynamic UDFs](#) in your function set.

Add Button

Opens the [Edit DDF window](#) for designing a new [DDF](#).

Edit Button

Opens the [Edit DDF window](#) for editing an existing [DDF](#). To edit a DDF, you must first select the DDF you want to modify, and then click the Edit button.

Remove Button

Removes an existing [DDF](#). To remove a DDF, you must first select the DDF you want to remove, and then click the Remove button. A less radical way of removing a DDF, consists of unchecking this function or setting its weight to zero. This way you will be able to use it later if you wish.

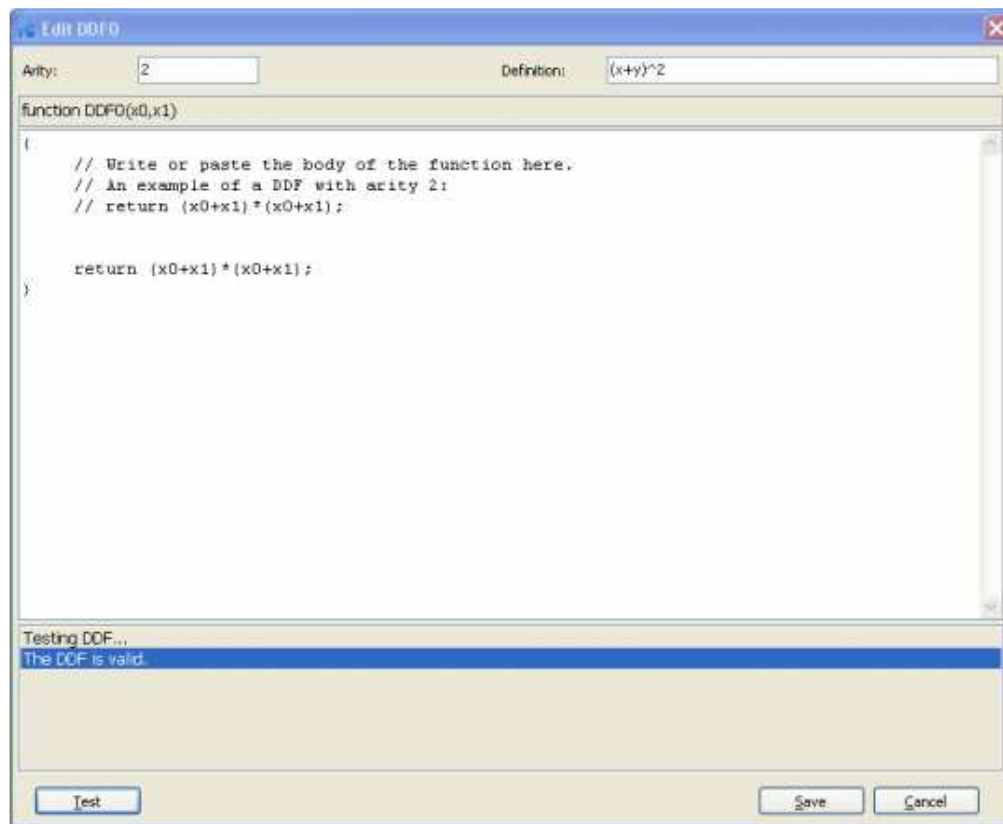


Settings and Features

Functions Tab – Edit DDF Window

The **Edit DDF Window** allows you to introduce your own functions and gives access to the following

settings and features:



Arity

The arity (number of arguments) of your [DDF](#). In GeneXproTools 4.0, the maximum allowed value is 4, and the minimum is obviously 1. After writing the arity, the function header appears in the gray box above the code window.

Definition

Use this box to write a brief description of the [DDF](#) for your future reference. The text you write here will appear in the Definition column on the [Functions Panel](#).

DDF Code Window

You design or edit the code of your own functions ([DDFs](#)) in the code window. The code must be in Java Script and can be tested for syntax errors before including it in the function set.

Test Button

Tests the syntax of the Java Script code of your [DDF](#). The results of the test are written in the gray window below the code window.

Save Button

Saves the [DDF](#) and includes it in your function set with the default weight of 1.

Cancel Button

Closes the Edit DDF window without saving the modifications made there.

Settings and Features

Static UDFs Tab

The **Static UDFs Tab** allows you to design your own [static UDFs](#) and gives access to the following settings and features:

Select/Weight	Name	Representation	Definition
☒ 1	Static UDF0	UDF0	My Function 0
☒ 1	Static UDF1	UDF1	My Function 1
☒ 1	Static UDF2	UDF2	My Function 2
☒ 1	Static UDF3	UDF3	My Function 3

Use UDFs

Check this box if you want to use or design [static UDFs](#).

Function Selection Framework

Allows the selection of different sub-sets of UDFs very quickly through the combination of the Show options with the Random/Clear/Select All buttons plus the Increase/Reduce Weight buttons.

Number of Functions

The number of functions you wish to select randomly by using the Random button.

Maximum Arity

This box is obviously disabled as all UDFs have arity zero.

Random Button

Selects randomly a specified number of functions.

Default Button

This button is obviously disabled as it does not apply to UDFs.

Clear Button

Clears all the functions from the function set.

Select All Button

Selects all the functions showing currently on the table.

Show List Box

Shows three different function sub-sets (All functions, Searched functions, and Selected Only). By selecting Search, the search box becomes activated and you can enter specific words to find functions that match your criteria in the Name column.

Search Button

Searches, in the Name column, for the functions matching your search words.

Add Weight

Increments by one the weight of each selected function.

Reduce Weight

Reduces by one the weight of each selected function; functions with weight 1, though, remain as they are.

Select/Weight Column

To select a [UDF](#), check the box on the left. By doing this, the UDF is included in your function set with the default weight of 1. To increase its weight, click the box below Select/Weight and then introduce its weight.

Name Column

Under this heading is shown the common name of all the [UDFs](#).

Representation Column

Under this heading is shown how all the [UDFs](#) are represented in [Karva notation](#).

Definition Column

Under this heading is shown the definition you gave to all the [UDFs](#).

Static UDFs Framework

Allows the introduction of [static UDFs](#) in your function set.

Add Button

Opens the [Edit UDF Window](#) for designing a new [UDF](#).

Edit Button

Opens the [Edit UDF Window](#) for editing an existing [UDF](#). To edit a UDF, you must first select the UDF you want to modify, and then click the Edit button.

Remove Button

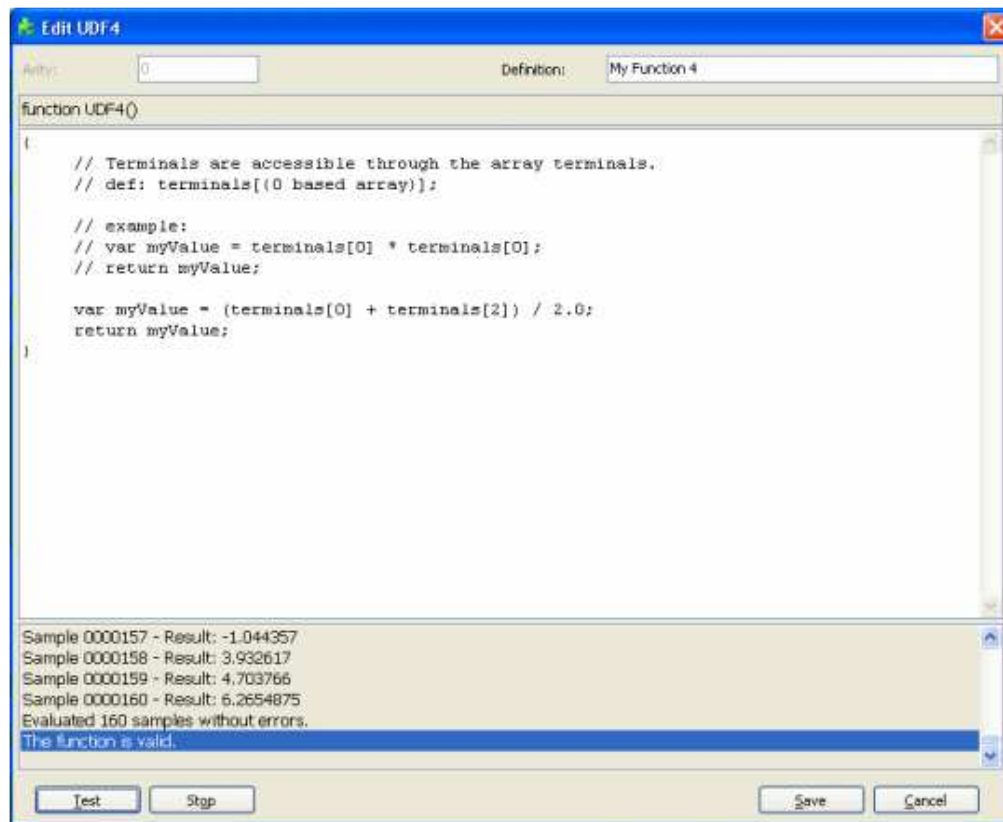
Removes an existing [UDF](#). To remove a UDF, you must first select the UDF you want to remove, and then click the Remove button. A less radical way of removing a UDF, consists of unchecking this function or setting its weight to zero. This way you will be able to use it later if you wish.



Settings and Features

Static UDFs Tab – Edit UDF Window

The **Edit UDF Window** allows you to introduce your own [static UDFs](#) and gives access to the following settings and features:

**Arity**

Reminds you that all the [static UDFs](#) take no parameters and, therefore, have zero arity (notice the function header that appears in the gray box above the code window).

Definition

Use this box to write a brief description of the [UDF](#) for your future reference. The text you write here will appear in the Definition column on the Functions Panel -> Static UDFs.

UDF Code Window

You design or edit the code of your [UDFs](#) in the code window. The code must be in Java Script and can be tested for syntax and calculation errors before including it in the function set.

Test Button

Tests the syntax of the Java Script code of your [UDF](#) and ensures that it is free of calculation errors. The results of the test are written in the gray window below the code window.

Stop Button

Allows you to stop the testing process.

Save Button

Saves the [UDF](#) and includes it in your function set with the default weight of 1.

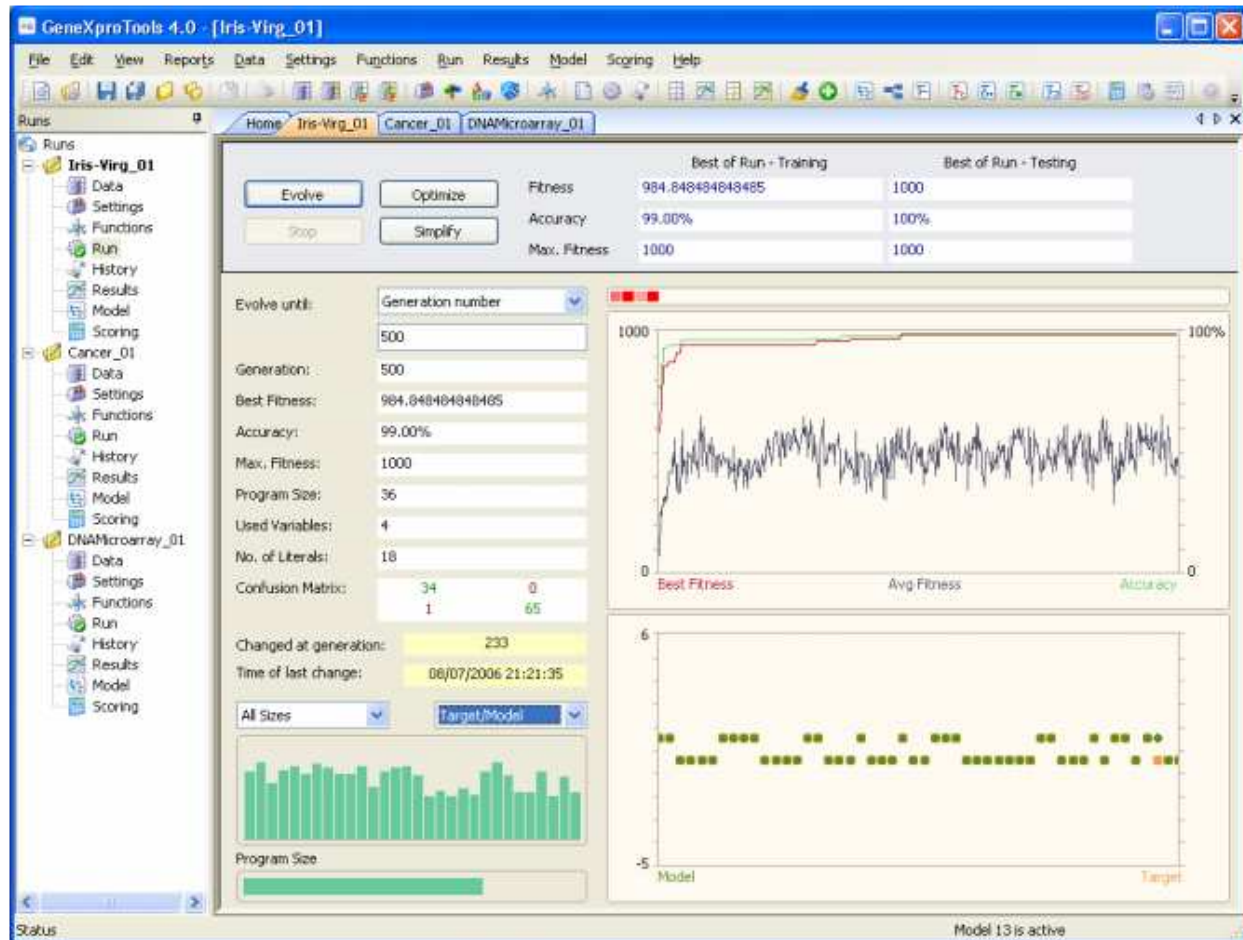
Cancel Button

Closes the Edit UDF window without saving the modifications made there.

Settings and Features

Run Panel

The **Run Panel** allows you to control and monitor the modeling process and gives access to the following settings and features:



Evolve Button

Starts the evolutionary process from scratch, that is, from a completely random initial population.

Stop Button

Stops the evolutionary process.

Optimize Button

Starts the evolutionary process with a [seed model](#), using it to breed better models.

Simplify Button

Also starts an evolutionary process with a seed model, but applies [parsimony pressure](#) to design more compact solutions.

Best of Run – Training

Under this heading are shown the values of [Fitness](#), [Accuracy](#), and Max. Fitness obtained on the training set for the best model of each run. These values are only updated at the end of each run or between Complexity Increase cycles. They are also updated if some relevant setting is changed, say, a different fitness function, a different training set, or a different model introduced through the Change

Seed window, and if the model is afterwards processed in the [Results Panel](#).

Best of Run – Testing

Under this heading are shown the values of [Fitness](#), [Accuracy](#), and Max. Fitness obtained on the testing set for the best model of each run. These values are only updated at the end of each run or between Complexity Increase cycles. They are also updated if some relevant setting is changed, say, a different fitness function, a different testing set, or a different model introduced through the Change Seed window, and if the model is afterwards processed in the [Results Panel](#). If no testing set is used, a dash is shown in all the boxes.

Evolve Until

Sets the stop condition. This stop condition is formulated using two different boxes. In the first you choose the stop condition and, in the second, you set the value for the chosen condition. There are four different stop conditions to choose from. The first allows you to evolve models for a certain number of generations. In this case, you must write in the box underneath the number of generations you want. The second allows you to evolve models until the best fitness is equal to or higher than a certain value. In this case, you must write in the box underneath this fitness value. The third allows you to evolve models until the [accuracy](#) of the best model is equal to or higher than a certain value. In this case, you must write in the box underneath this accuracy value. Obviously, only values higher than 0 and equal to or less than 100 are valid, and if an invalid value is used GeneXproTools uses the value of 100 by default. The fourth allows you to evolve models until the best fitness is equal to maximum fitness. In this case, the correct value of maximum fitness is automatically introduced by GeneXproTools in the box underneath.

Generation

Updates the generation number throughout a run.

Best Fitness

Updates the best fitness of each generation throughout a run.

Accuracy

Updates the [classification accuracy](#) of the best-of-generation model throughout a run.

Max. Fitness

Shows the value of maximum fitness for the current run.

Program Size

Updates the program size of the best-of-generation model throughout a run.

Used Variables

Updates the number of variables used in the best-of-generation model throughout a run.

No. of Literals

Updates the number of literals used in the best-of-generation model throughout a run.

Confusion Matrix

Shows the [confusion matrix](#), where [true positives](#) followed by [false negatives](#) are shown on the first row, and [false positives](#) followed by [true negatives](#) are shown on the second row. For easy reference, the true positives and true negatives are shown in green, whereas the false positives and the false negatives are shown in red.

Changed at Generation

Updates, throughout a run, the number of the generation at which a change in best fitness occurred.

Time of Last Change

Updates, throughout a run, the time when a change in best fitness occurred.

Charts List Boxes

The one on the left allows you to select three different bar plots: Sub-Program Sizes, All Sizes, and All

Fitnesses. And the one on the right allows you to select two different charts: Curve Fitting and Avg/Best Size.

Sub-Program Sizes Chart

This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be visualized any time during evolution by selecting Sub-Program Sizes in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

All Sizes Chart

This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be visualized any time during evolution by selecting All Sizes in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

All Fitnesses Chart

This chart shows the fitnesses of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be visualized any time during evolution by selecting All Fitnesses in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Program Size Chart

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Target/Model Chart

This chart plots the first 50 data points of the training set and shows how well the evolving models are fitting the target. The Target/Model chart can be visualized any time during evolution by selecting Target/Model in the rightmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click. And by placing the cursor over each data point, you can easily identify it and access its value.

Avg/Best Size Chart

This chart plots together the average program size of the population and the program size of the best-of-generation model. This chart is especially useful during simplification and can be visualized any time during evolution by selecting Avg/Best Size in the rightmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click. You can also access the values corresponding to the Max and Min labels that sometimes appear on the Y axis by placing the cursor over them.

Variables Usage Map

This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square gives you access to the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Evolutionary Dynamics Chart

This chart shows the average fitness of the population plus the fitness and [accuracy](#) of the best-of-generation model. The Y axis on the left refers to best and average fitness, whereas the Y axis on the right refers to the [classification accuracy](#). The X axis corresponds obviously to the number of generations, with a complete screen spanning a total of 500 generations. For a good adaptation, the plot for average fitness should never come near the plot for best fitness, otherwise the system is losing

genetic diversity and becoming too uniform for an efficient evolution. Whenever this is happening in your system, you should obviously increase the [genetic diversity](#) of the evolving populations by choosing the right set of genetic operators and adjusting their rates. And as usual, GeneXproTools allows you to copy or print this chart through the mouse right-click.



Settings and Features

History Panel

The **History Panel** lists all the best-of-generation models created in a run and gives access to the following settings and features:

The screenshot shows the GeneXproTools 4.0 interface with the 'Cancer_01' run selected. The History Panel is active, displaying a table of models. The table has the following columns: Model, Generation, Program Size, Training Fitness, Testing Fitness, Training Accuracy, and Testing Accuracy. Model 32 is selected and highlighted in yellow.

Model	Generation	Program Size	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
Model 7	36	64	921.046447002779	936.391206774877	96.00%	97.13%
Model 8	47	63	921.048749503771	966.599791107975	96.00%	98.28%
Model 9	52	57	924.16645873182	966.614289343684	95.71%	98.28%
Model 10	72	57	931.998377873615	948.546732533522	96.00%	97.13%
Model 11	82	55	932.0030375950976	972.547568807339	96.00%	98.28%
Model 12	92	53	932.007697228337	972.55243119266	96.00%	98.28%
Model 13	96	52	932.010027067018	963.379816513762	96.00%	97.70%
Model 14	105	52	936.305004150276	972.554662385321	96.29%	98.28%
Model 15	406	51	936.30734472554	981.732362385321	96.29%	98.85%
Model 16	468	50	936.309685300805	981.734816513761	96.29%	98.85%
Model 17	504	49	936.312025876069	981.737270642202	96.29%	98.85%
Model 18	537	39	936.335431628713	972.586467889908	96.29%	98.28%
Model 19	568	55	940.59292729799	966.61912208092	96.57%	98.28%
Model 20	614	54	940.595278609838	966.621538461538	96.57%	98.28%
Model 21	620	53	940.597629921686	966.623954834157	96.57%	98.28%
Model 22	681	47	940.611737792775	981.742178899083	96.57%	98.85%
Model 23	727	46	940.614089104623	981.744633027523	96.57%	98.85%
Model 24	839	42	940.623494352016	963.40389982569	96.57%	97.70%
Model 25	882	38	940.632899599408	963.413532110092	96.57%	97.70%
Model 26	1464	36	940.637602223104	972.59376146789	96.57%	98.28%
Model 27	1912	48	948.51366703959	975.670018348624	96.86%	98.85%
Model 28	1916	46	948.518409181132	975.674896259704	96.86%	98.85%
Model 29	1930	45	948.520780251904	975.677335215243	96.86%	98.85%
Model 30	1940	35	948.544490959616	975.701724770642	96.86%	98.85%
Model 31	2263	33	948.549233101159	975.706602681722	96.86%	98.85%
Model 32	4295	33	948.549233101159	984.740923076923	96.86%	99.43%

Buttons: Test All, Test Current, Refresh All. Summary: Best Fitness: Model 32: 948.549233101159, Best Accuracy: Model 32: 96.86%, Max. Fitness: 1000.2, Testing: Model 32: 984.740923076923, Model 32: 99.43%, 1000.2.

Test All Button

Evaluates the performance of all the models in the History Panel on the testing set, showing their respective [fitnesses](#) and [accuracies](#). If no testing set is used, then this button is disabled.

Test Current Button

Evaluates the performance of the selected model on the testing set, showing its [fitness](#) and [accuracy](#). If no testing set is used, then this button is disabled.

Refresh All Button

Evaluates the performance of all the models in the History Panel both on the training and testing sets, and updates all the [fitnesses](#) and [accuracies](#). This button is particularly useful when you have changed the fitness function or the training set after a run and would want to evaluate the performance on the training set with this new fitness function or new training set. This new training set is usually introduced for model refreshing, and is always good practice to pre-test all the models in the History Panel with this new data set.

Best of Run – Training

This heading is used to identify the best models of the run on the training set in terms of [fitness](#) and [accuracy](#), showing also the exact values of their fitness and accuracy. For reference, the value of Max. Fitness on the training set is also shown.

Best of Run – Testing

This heading is used to identify the best models of the run on the testing set in terms of [fitness](#) and [accuracy](#), showing also the exact values of their fitness and accuracy. For reference, the value of Max. Fitness on the testing set is also shown. If no testing set is used, then a dash is shown in all the boxes under this heading.

Model Column

In this column are shown the names of all the best-of-generation models created in the run. By clicking a model, you are making it your active model. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order.

Generation Column

In this column is shown the number of the generation in which the model was created. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order.

Program Size Column

In this column is shown the program size of all the best-of-generation models. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their program sizes.

Training Fitness Column

In this column is shown the [fitness](#) obtained by each model on the training set. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their training fitnesses.

Testing Fitness Column

In this column is shown the [fitness](#) obtained by each model on the testing set. Note that, after a run, only the fitness of the best-of-run model is shown; all the others are set to zero. In order to update these values, you must either click on the Test All or Test Current buttons. If no testing set is used, though, a dash is shown in all the entries under this heading. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their testing fitnesses.

Training Accuracy Column

In this column is shown the [accuracy](#) obtained by each model on the training set. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their training accuracies.

Testing Accuracy Column

In this column is shown the [accuracy](#) obtained by each model on the testing set. Note that, after a run, only the accuracy of the best-of-run model is shown; all the others are set to zero. In order to update these values, you must either click on the Test All or Test Current buttons. If no testing set is used, though, a dash is shown in all the entries under this heading. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their testing accuracies.

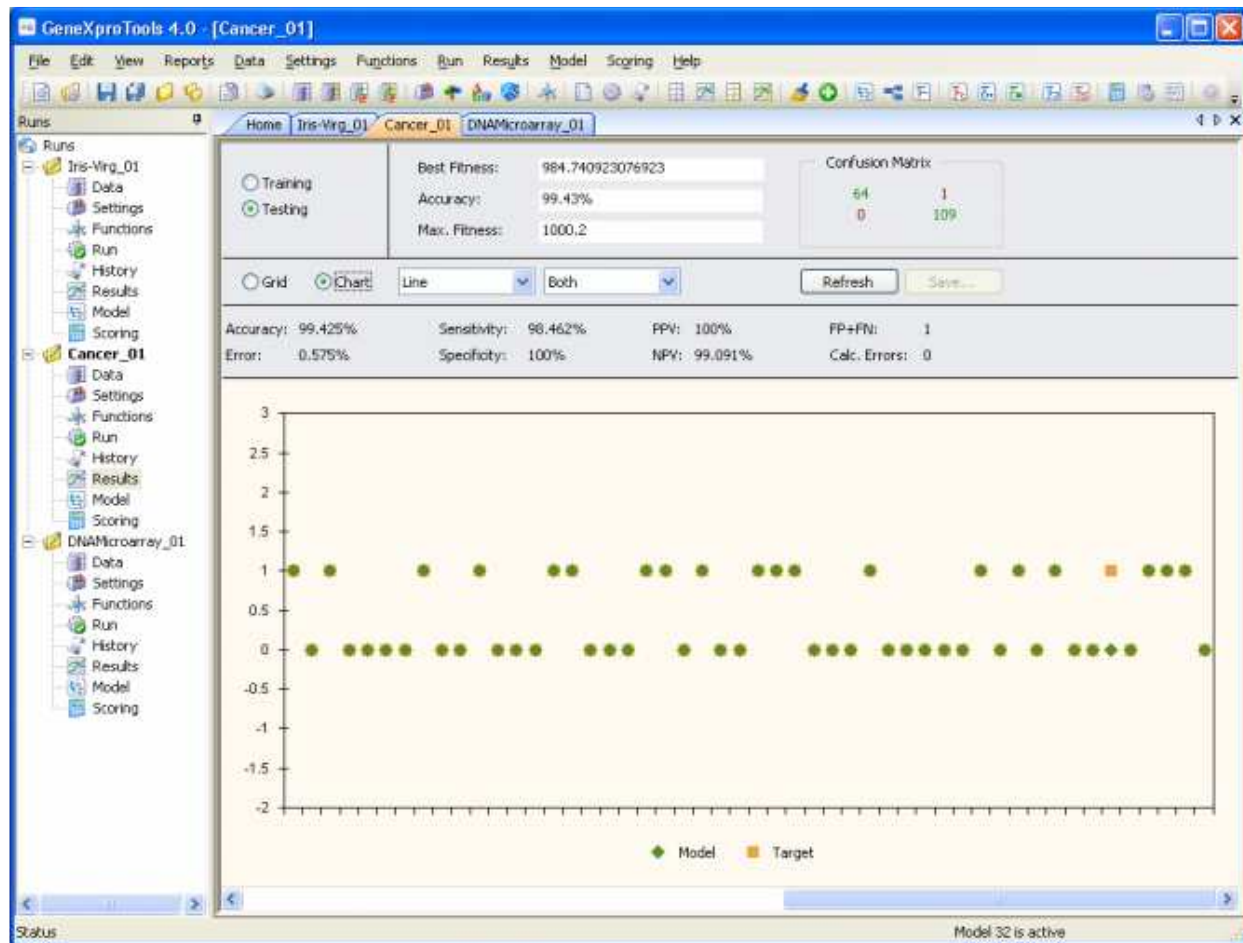


Settings and Features

Results Panel

The **Results Panel** allows you to analyze and evaluate the performance of your models and includes the following options:

- [Training & Testing Results](#)
- [Grid Option](#)
- [Chart Option](#)



Settings and Features

Training & Testing Results

The **Training & Testing Results Windows** allow you to visualize and evaluate the accuracy of your models and gives access to the following settings and features:



Training Option

By selecting this option you have access to all the [statistical evaluations](#) of your model on the training set. You also can compare the output of your model with the target output on the training set using either a [table](#) or different [charts](#).

Testing Option

By selecting this option you have access to all the [statistical evaluations](#) of your model on the testing set. You also can compare the output of your model with the target output on the testing set using either a [table](#) or different [charts](#).

Best Fitness

Shows the [fitness](#) of the active model on the current set.

Accuracy

Shows the [classification accuracy](#) of the active model on the current set.

Max. Fitness

Shows the value of [maximum fitness](#) on the current set.

Confusion Matrix

Shows the [confusion matrix](#), where [true positives](#) followed by [false negatives](#) are shown on the first row, and [false positives](#) followed by [true negatives](#) are shown on the second row. For easy reference, the true positives and true negatives are shown in green, whereas the false positives and the false negatives are shown in red.

Grid Option

By selecting the [Grid Option](#) you can compare the output of your model with the target output on the current set on a spreadsheet where the target and model output are placed side by side.

Chart Option

By selecting the [Chart Option](#) you can compare the output of your model with the target output on the current set using different charts. You choose the chart type with the aid of two boxes. In the first you select the chart type (Line, Area, Bar, Step, 3D Line, 3D Area, 3D Bar, or 3D Step). And in the second you select the data you want to plot (Model, Target, or Both). The plots show only a maximum of 50 points at a time and, therefore, for datasets with more than 50 samples you must move the scroll bar at the bottom of the chart to see all the results. Then, by placing the cursor over the chart, GeneXproTools shows you the exact range used to draw the plot.

Refresh Button

Refreshes the panel. This button is particularly useful when you change your model in the [Change Seed window](#) or change the current set or the fitness function.

Save Button

This button is only active when the [Grid Option](#) is selected as it saves all the results in the table to a file.

Accuracy

Shows the [classification accuracy](#) on the current set.

Error

Shows the value of the [classification error](#) on the current set.

Sensitivity

Shows the value of the [sensitivity](#) on the current set.

Specificity

Shows the value of the [specificity](#) on the current set.

PPV

Shows the value of the [positive predictive value](#) on the current set.

NPV

Shows the value of the [negative predictive value](#) on the current set.

FP+FN

Shows the number of samples wrongly classified on the current set, which corresponds obviously to the sum of [false positives](#) and [false negatives](#). Wrongly classified samples are easily identified by a yellow strip and the identifier WRONG in the table.

Calc. Errors

Shows the number of calculation errors in the current set. Note that, unless you've changed either the training set or the model after a run, no calculation errors should appear in the training set as models returning calculation errors during training have zero fitness and are, therefore, excluded. On the testing set, though, calculation errors might occur and you can see straightaway how many times in the Calculation Errors box. In addition, the rows with calculation errors are easily identified by a red strip and the identifier ERROR. In the charts, you can also easily spot the points returning calculation errors as they are not drawn, creating a discontinuity in the plots.



Settings and Features

Grid Option

The **Grid Option** allows you to compare the output of your model with the target output and is organized as follows:

☒ Training
 ☐ Testing

Best Fitness: 994.848484848485
 Accuracy: 99.00%
 Max. Fitness: 1000

Confusion Matrix

34

0

1

65

☒ Grid
 ☐ Chart

Line

Both

Refresh

Save...

Accuracy: 99.000%
 Error: 1.000%

Sensitivity: 100%
 Specificity: 98.485%

PPV: 97.143%
 NPV: 100%

FP+FN: 1
 Cal. Errors: 0

	Target	Model	Residual
34	0	0	0
35	0	0	0
36	0	0	0
37	1	1	0
38	1	1	0
39	0	0	0
40	0	0	0
41	0	0	0
42	1	1	0
43	0	0	0
44	1	1	0
45	1	1	0
46	0	0	0
47	1	1	0
48	0	1	WRONG
49	0	0	0
50	0	0	0
51	1	1	0
52	1	1	0
53	0	0	0
54	1	1	0
55	0	0	0
56	0	0	0
57	0	0	0

Target

Under this heading is shown the value of the dependent variable (Target) for all the records on the current data set.

Model

Under this heading is shown the output of the model (Model) for all the records on the current data set. In case of calculation errors, a red strip and the identifier ERROR is shown.

Residual

Under this heading is shown the absolute value of the difference between the Target and the Model. For the sake of clarity, though, when the residual is 1 the identifier WRONG is shown instead.



Settings and Features

Chart Option

The **Chart Option** allows you to compare visually the output of your model with the target output using different kinds of curve fitting charts:



Chart Type Box

In this list box you select the chart type: Line, Area, Bar, Step, 3D Line, 3D Area, 3D Bar, or 3D Step.

Chart Data Box

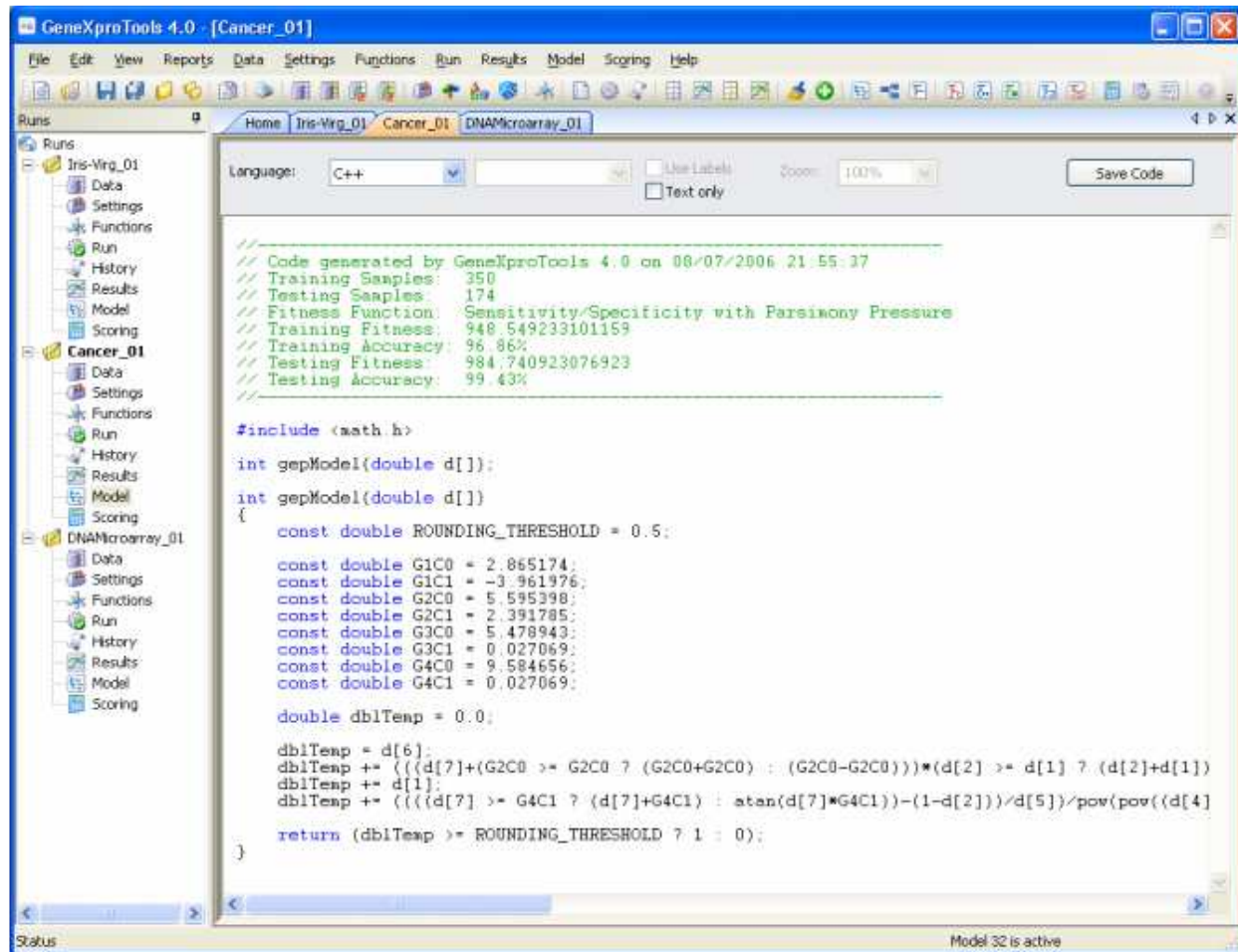
In this list box you select the data you want to plot: Model, Target, or Both. In case of calculation errors, the sample with the calculation error is skipped and a discontinuity is observed in the plot. The plots show only a maximum of 50 points at a time and, therefore, for datasets with more than 50 samples you must move the scroll bar at the bottom of the chart to see all the results. Then, by placing the cursor over the chart, GeneXproTools shows you the exact range used to draw the plot.



Settings and Features

Model Panel

The **Model Panel** allows you to see the structure of your models and translate them into a wide range of [programming languages](#). It gives access to the following settings and features:



Language

In the list box on the left, you can choose different ways of representing your models, including the native [Karva code](#), a universal diagram representation ([Expression Tree](#)), 15 different [programming languages](#) (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL) originally packaged with GeneXproTools 4.0, and any additional languages created by you through the use of [User Defined Grammars](#). The list box on the right is only activated when Karva is selected, and allows you to choose either [Genes](#) or [K-expressions](#). In both cases, though, the elements in the [heads](#) are shown in blue.

Use Labels

This option is only active when you are using data sets with labels, that is, when the columns with each variable are identified by a name. By checking this box, the variables in your code, which are usually represented by the generic $d_0, d_1, \dots$ are replaced by their respective names. And for the sake of clarity, this option is disabled when either Karva or Expression Tree are selected as, in both those cases, the generic representation is always used.

Text Only

By checking this box the default HTML is replaced by simple unformatted text.

Zoom

This option is only active when you are visualizing Expression Trees. It allows you to either choose pre-set values to resize the expression trees (200%, 150%, 100%, 85%, 75%, 50%, and 25%) or enter any other value up to 500%.

Save Code

Saves the selected code to an appropriate format: [Karva code](#) is saved as .txt, [Ada code](#) as .ada, [C code](#) as .c, [C++ code](#) as .hpp, [C# code](#) as .cs, [Fortran code](#) as .f, [Java code](#) as .java, [Java Script code](#) as .js, [Matlab code](#) as .m, [Pascal code](#) as .pas, [Perl code](#) as .pl, [PHP code](#) as .php, [Python code](#) as .py, [Visual Basic code](#) as .bas, [VB.Net code](#) as .vb, and [VHDL code](#) as .vhd.

Save Image

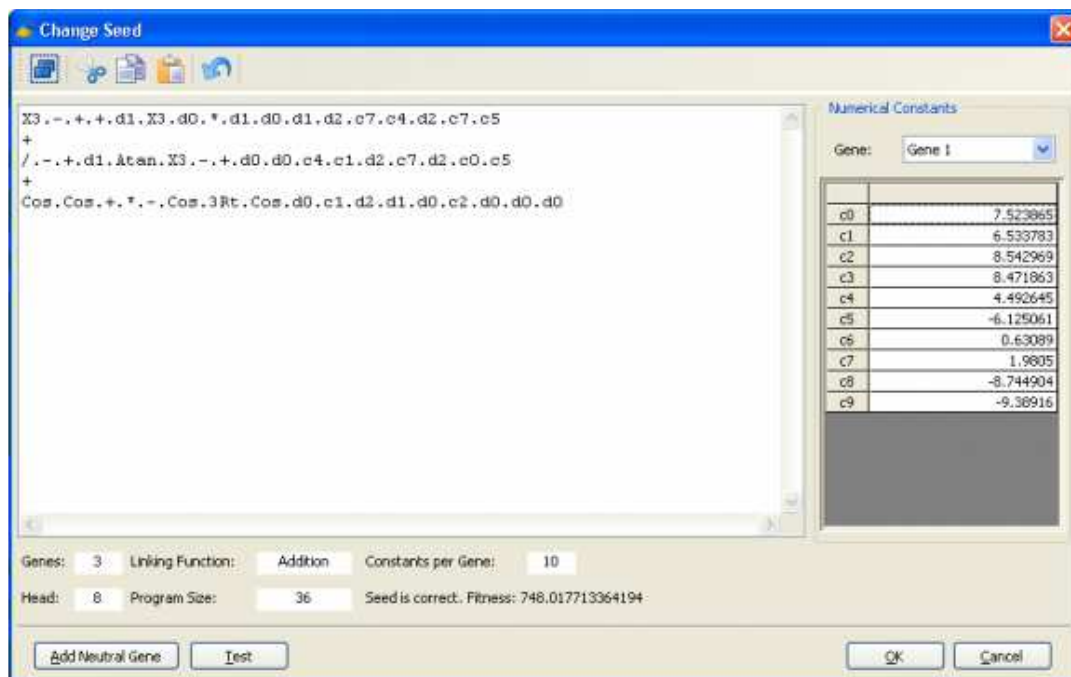
Saves the [Expression Trees](#) as .bmp files.



Settings and Features

Change Seed Window

The **Change Seed Window** allows you to change your model or introduce a completely new one. It gives access to the following settings and features:



Seed Code Window

You design or edit the code of your seed model in the code window. The code must be in [Karva notation](#) and can be tested for structural errors before saving it as seed.

Genes

Shows the current [number of genes](#) in your seed model.

Head

Shows the [head](#) size of your seed model.

Linking Function

Shows the current [linking function](#) in your seed model.

Program Size

Shows the actual size of your seed model, that is, the total number of nodes in all the [sub-ETs](#).

Constants per Gene

This box only appears if [random numerical constants](#) are being used. It shows the number of random numerical constants used per gene in your seed model.

Numerical Constants Framework

This frame only appears if [random numerical constants](#) are being used. Here you have access to all the numerical constants of your seed model by selecting the gene number in the Gene list box. The numerical constants appear in a table where you can edit them. Only values within the range of -1.79769313486231E308 to -4.94065645841247E-324 for negative values, 4.94065645841247E-324 to 1.79769313486231E308 for positive values, and 0 are acceptable.

Add Neutral Gene Button

Adds automatically a neutral gene to your seed model. Neutrality is important for an efficient evolution and, by introducing a neutral gene, you are giving the [learning algorithm](#) more room to play and, hopefully, a better, more complex program will evolve.

Test Button

Tests both the structural soundness and the [fitness](#) of your seed model. It also updates the program size when changes have been made on the seed.

OK Button

Saves the current seed as your seed model so that you can breed new and better models with it. GeneXproTools only saves viable seeds and warns you if there is a structural problem with your seed or if it has zero fitness.

Cancel Button

Cancels all the action on the change seed window.

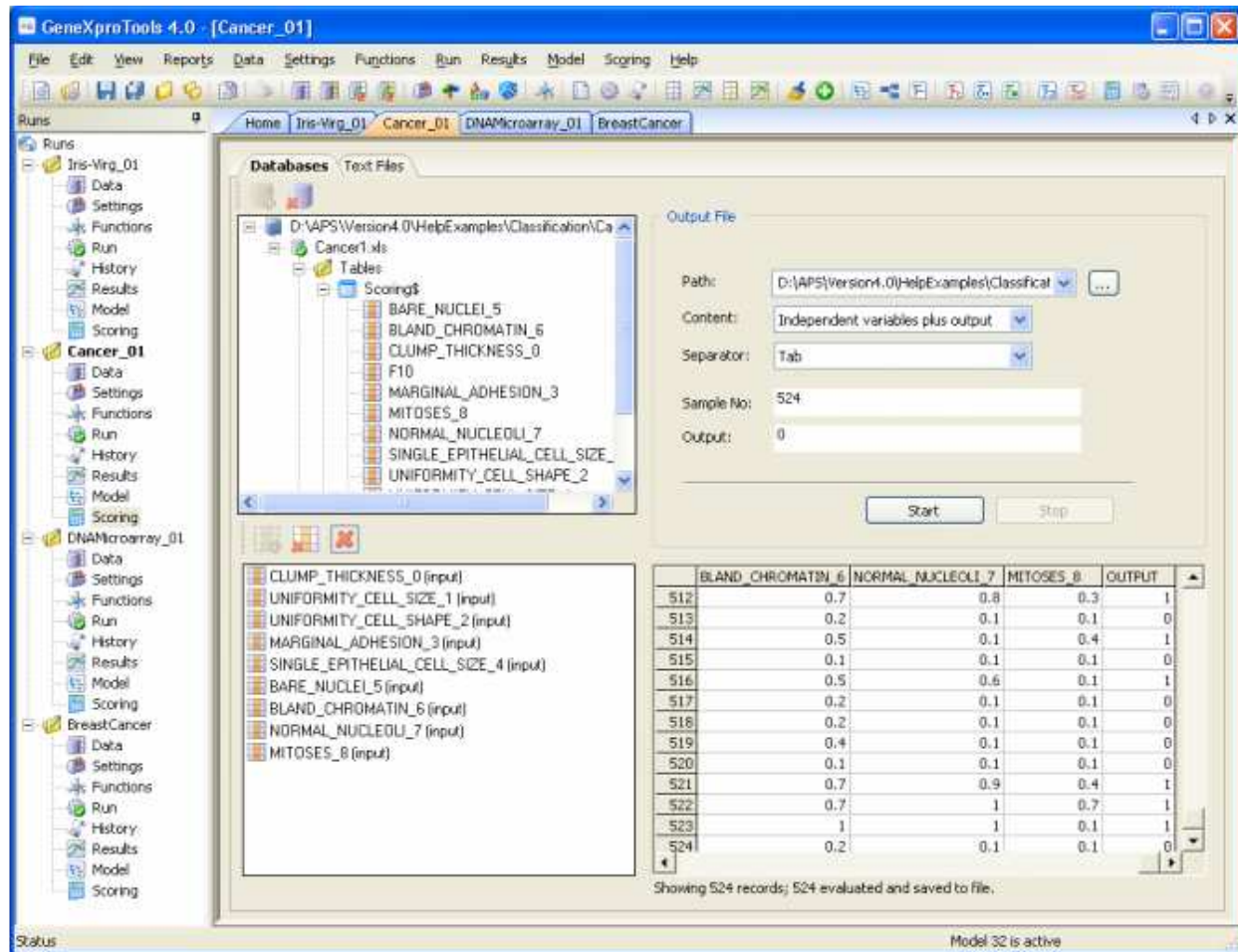


Settings and Features

Scoring Panel

The **Scoring Panel** allows you to score your models immediately without leaving the GeneXproTools modeling environment. It includes the following tabs and windows:

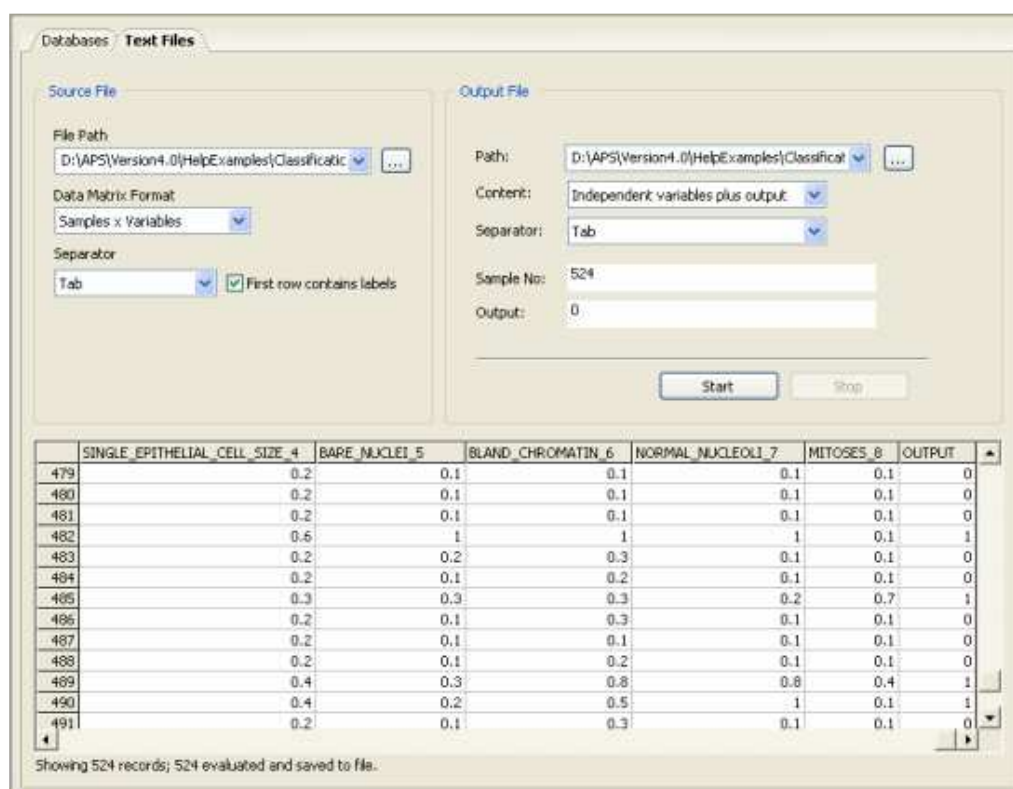
- [Text Files Tab](#)
- [Databases Tab](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Text Files Tab

The **Text Files Tab** allows you to score text files and gives access to the following settings and features:



Source File

File Path: D:\APS\Version4.0\Help\Examples\Classificat... [...]

Data Matrix Format: Samples x Variables

Separator: Tab ☒ First row contains labels

Output File

Path: D:\APS\Version4.0\Help\Examples\Classificat... [...]

Content: Independent variables plus output

Separator: Tab

Sample No: 524

Output: 0

Start Stop

	SINGLE EPITHELIAL_CELL_SIZE_4	BARE_NUCLEI_5	BLAND_CHROMATIN_6	NORMAL_NUCLEOLI_7	MITOSES_8	OUTPUT
479	0.2	0.1	0.1	0.1	0.1	0
480	0.2	0.1	0.1	0.1	0.1	0
481	0.2	0.1	0.1	0.1	0.1	0
482	0.6	1	1	1	0.1	1
483	0.2	0.2	0.3	0.1	0.1	0
484	0.2	0.1	0.2	0.1	0.1	0
485	0.3	0.3	0.3	0.2	0.7	1
486	0.2	0.1	0.3	0.1	0.1	0
487	0.2	0.1	0.1	0.1	0.1	0
488	0.2	0.1	0.2	0.1	0.1	0
489	0.4	0.3	0.8	0.8	0.4	1
490	0.4	0.2	0.5	1	0.1	1
491	0.2	0.1	0.3	0.1	0.1	0

Showing 524 records; 524 evaluated and saved to file.

Source File Framework

In the Source File Framework you enter all the information concerning the source file.

Source File Path

Click the button on the right of the list box to load the scoring file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, you must select the Samples x Variables format. If your samples are in columns and variables in rows, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies.

Source File Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

Output File Framework

In the Output File Framework you enter all the information concerning the output file.

Output File Path

You can either enter the path for the output file or click the button on the right of the list box to browse the Path for the output file.

Output File Content

Here you decide whether you want to include in the output file the columns with the independent variables or just the output.

Output File Separator

Here you choose the separator you want to use in the output file (this is only relevant if you'd choose also to include in the output file the columns with the independent variables).

Sample Number

Shows the number of the sample being processed.

Output

Shows the output obtained for the current sample.

Start Button

Starts the scoring process.

Stop Button

Stops the scoring process.

Scoring Results Table

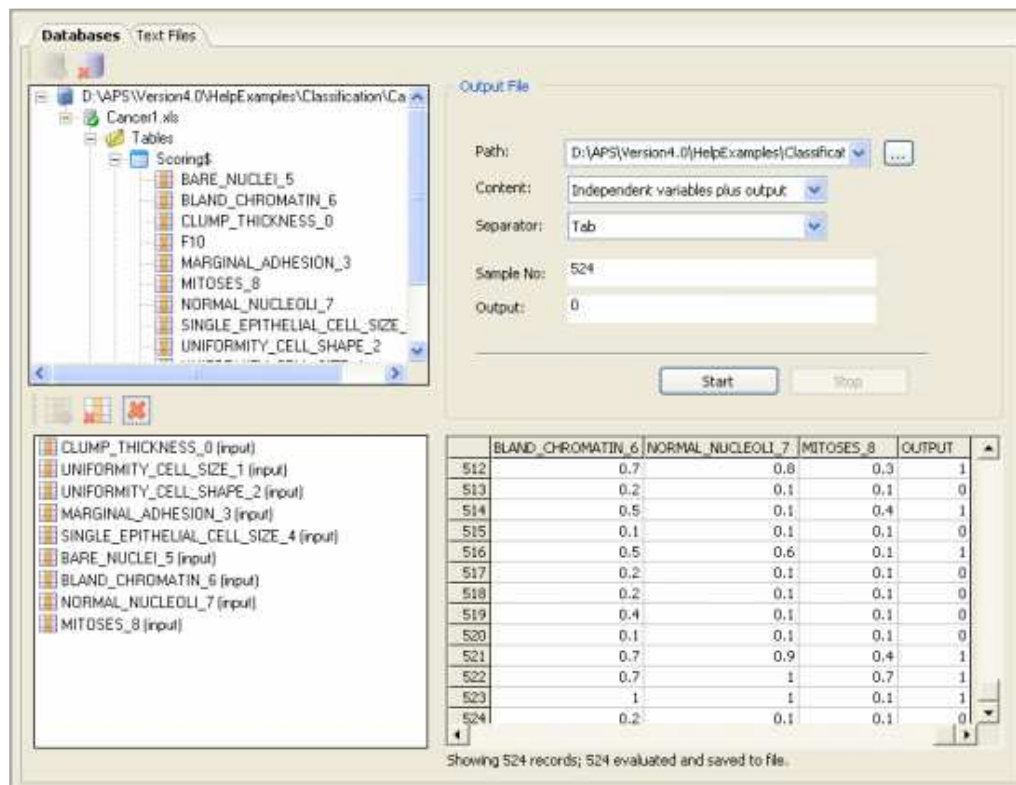
GeneXproTools saves the scoring results to a file, but for small datasets (up to 16 variables and 2000 samples), GeneXproTools also shows the scoring results in their entirety on the Scoring Panel for an immediate assessment. For datasets with more than 16 variables, though, only a single column with the output is shown.



Settings and Features

Databases Tab

The **Databases Tab** allows you to score databases and Excel files and gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Scoring Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Remove All the Columns Icon

Removes all the columns from the Scoring Data Window.

Scoring Data Window

You prepare your scoring data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Remember, however, that your scoring dataset must have exactly the same number of inputs as there are independent variables in your training data.

Output File Framework

In the Output File Framework you enter all the information concerning the output file.

Output File Path

You can either enter the path for the output file or click the button on the right of the list box to browse the path for the output file.

Output File Content

Here you decide whether you want to include in the output file the columns with the independent variables or just the output.

Output File Separator

Here you choose the separator you want to use in the output file (this is only relevant if you'd choose also to include in the output file the columns with the independent variables).

Sample Number

Shows the number of the sample being processed.

Output

Shows the output obtained for the current sample.

Start Button

Starts the scoring process.

Stop Button

Stops the scoring process.

Scoring Results Table

GeneXproTools saves the scoring results to a file, but for small datasets (up to 16 variables and 2000 samples), GeneXproTools also shows the scoring results in their entirety on the Scoring Panel for an immediate assessment. For datasets with more than 16 variables, though, only a single column with the output is shown.

Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:



Driver

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.
Microsoft Access: path to the database file.
Microsoft SQL Server: server name or IP address.
Oracle: server name or IP address.
MySQL: server name or IP address.
DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.
Oracle - Name and database instance (HOST/INSTANCE).
MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

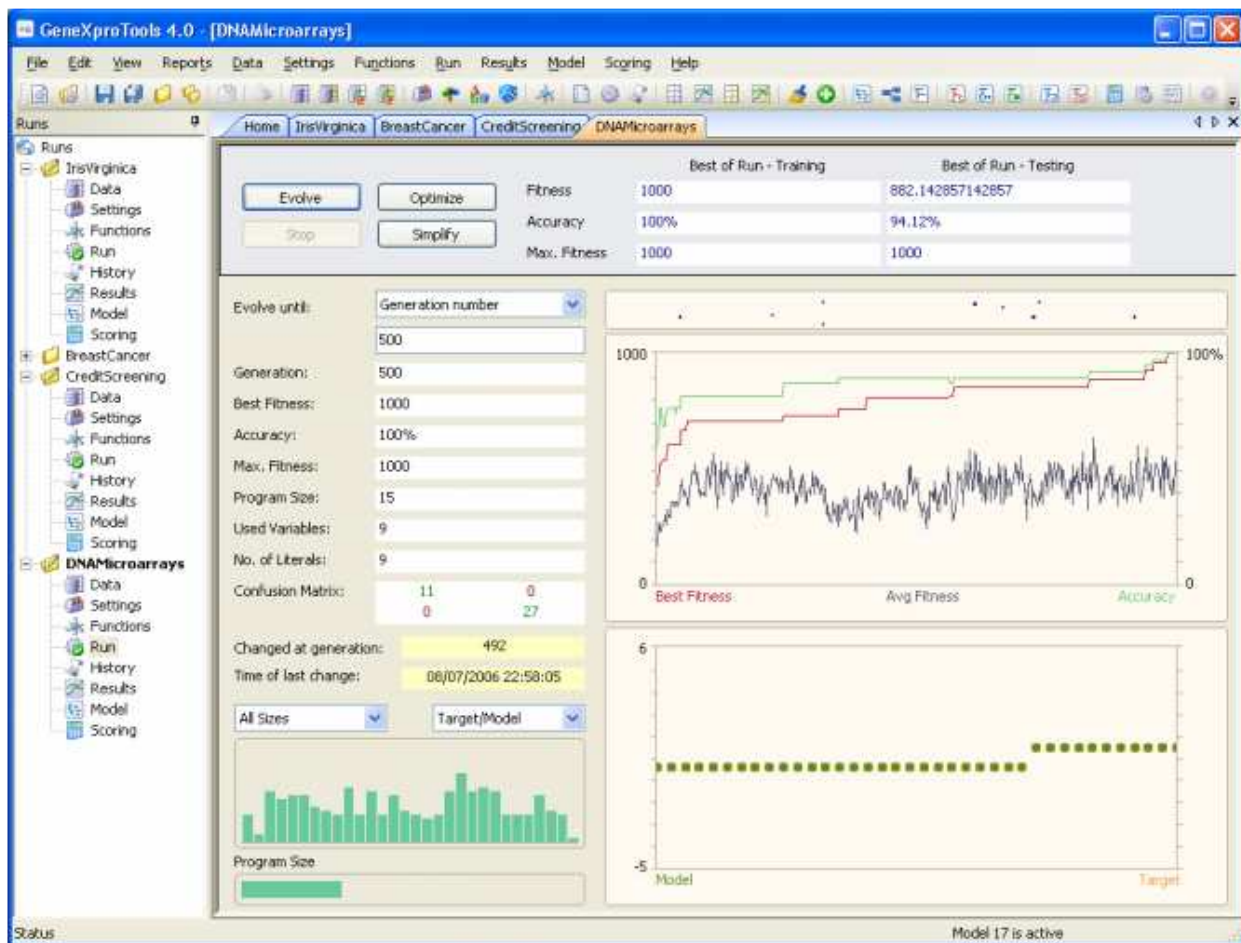
Cancels all the action on the data source connection.

Settings and Features

Menus

For Classification problems, GeneXproTools has the following menus:

- [File](#)
- [Edit](#)
- [View](#)
- [Reports](#)
- [Data](#)
- [Settings](#)
- [Functions](#)
- [Run](#)
- [Results](#)
- [Model](#)
- [Scoring](#)
- [Help](#)



Settings and Features

File

The **File Menu** gives access to the following sub-menus:

New

Creates a new gep file.

Open

Opens a gep file.

Save

Saves the selected run.

Save As

Saves a gep file with a different name.

Save All

Saves all open runs.

Close

Closes the selected run.

Close All

Closes all open runs.

Print

Prints the report, history, code, tree, or charts of the run.

Properties

Loads the identification form for input (Author Name and Comments).

Exit

Closes GeneXproTools.



Settings and Features

Edit

The **Edit Menu** gives access to the following sub-menus:

Copy

Copies the report, tables, code, tree, or charts of the run for pasting elsewhere.

Change Seed

Opens the [Change Seed window](#) for changing exogenous or GeneXproTools evolved models for further optimization.

Add Neutral Gene

Adds automatically a neutral gene to your seed model.

Add DDF

Opens the [Edit DDF window](#) for designing a new [DDF](#).

Add UDF

Opens the [Edit UDF window](#) for designing a new [UDF](#).

Custom Fitness Function

Goes to the [Custom Fitness Function Code window](#) for editing or designing a new [custom fitness function](#).



Settings and Features

View

The **View Menu** gives access to the following sub-menus:

Status Bar

Shows/Hides the Status Bar.

Warnings

Turns on/off warning messages.



Settings and Features

Reports

The **Reports Menu** gives access to the following sub-menus:

Home

Goes to the welcome screen of GeneXproTools.

Run Report

Goes to the [Report Panel](#) of the active run.



Settings and Features

Data

The **Data Menu** gives access to the following sub-menus:

Training Data

Goes to the [Training Set Tab](#) of the selected run.

Testing Data

Goes to the [Testing Set Tab](#) of the selected run.

Change Training Data

Opens the [Change Training Data Window](#) for changing the training set of the selected run.

Change Testing Data

Opens the [Change Testing Data Window](#) for changing the testing set of the selected run.

Add Testing Data

Opens the [Add Testing Data Window](#) for changing the testing set of the selected run.



Settings and Features

Settings

The **Settings Menu** gives access to the following sub-menus:

General Settings

Goes to the [General Settings Tab](#) of the selected run.

Fitness Function

Goes to the [Fitness Function Tab](#) of the selected run.

Genetic Operators

Goes to the [Genetic Operators Tab](#) of the selected run.

Numerical Constants

Goes to the [Numerical Constants Tab](#) of the selected run.



Settings and Features

Functions

The **Functions Menu** gives access to the following sub-menus:

Functions

Goes to the [Functions Tab](#) of the selected run.

Static UDFs

Goes to the [Static UDFs Tab](#) of the selected run.



Settings and Features

Run

The **Run Menu** gives access to the following sub-menus:

Run

Goes to the [Run Panel](#) of the active run.

Evolve

Starts a new run.

Optimize

Starts an optimization process.

Simplify

Starts a simplification process.

Variables Usage

Allows the selection of three different kinds of mapping variables usage during a run: Heat Map, Random Colors, or Monochromatic, and two different showing options: White for None and Grey for None.

Charts

Allows the selection of five different kinds of charts during a run: Target/Model comparison, Avg/Best Size, Sub-Program Sizes, All Sizes, and All Fitnesses.



Settings and Features

Results

The **Results Menu** gives access to the following sub-menus:

History

Goes to the [History Panel](#) of the active run.

Training Results Table

Goes to the [Results Panel -> Training and Grid Options](#) of the active run.

Training Results Chart

Goes to the [Results Panel -> Training and Chart Options](#) of the active run.

Testing Results Table

Goes to the [Results Panel -> Training and Grid Options](#) of the active run.

Testing Results Chart

Goes to the [Results Panel -> Training and Chart Options](#) of the active run.



Settings and Features

Model

The **Model Menu** gives access to the following sub-menus:

Expression Tree

Shows the active model as an [Expression Tree](#).

Karva

Shows the active model in the native [Karva Notation](#).

Ada

Translates the active model into [Ada](#).

C

Translates the active model into [C code](#).

C++

Translates the active model into [C++](#).

C#

Translates the active model into [C#](#).

Fortran

Translates the active model into [Fortran](#).

Java

Translates the active model into [Java](#).

JavaScript

Translates the active model into [JavaScript](#).

Matlab

Translates the active model into [Matlab](#).

Pascal

Translates the active model into [Pascal](#).

Perl

Translates the active model into [Perl](#).

PHP

Translates the active model into [PHP](#).

Python

Translates the active model into [Python](#).

Visual Basic

Translates the active model into [Visual Basic](#).

VB.Net

Translates the active model into [VB.Net](#).

VHDL

Translates the active model into [VHDL](#).



Settings and Features

Scoring

The **Scoring Menu** gives access to the following sub-menus:

Databases

Goes to the [Scoring Panel -> Databases Tab](#) for scoring a database with the active model.

Text Files

Goes to the [Scoring Panel -> Text Files Tab](#) for scoring a database with the active model.



Settings and Features

Help

The **Help Menu** gives access to the following sub-menus:

Contents

Shows the contents of the Help File.

About GeneXproTools

Shows information about GeneXproTools.



Settings and Features

Run Templates

For Classification problems, GeneXproTools 4.0 offers five different **Run Templates** designed so that you can start the modeling process with just a mouse click and yet obtain excellent results. In addition, they also make the discovery of good models quite enjoyable and fast as small populations of just 30 models are used in all of them.

The run templates of GeneXproTools are essentially based on the number of independent variables in your data and are organized as follows (in the Function Set, weights greater than 1 are indicated in parentheses):

Template A

Independent Variables: 1-3
Fitness Function: Sensitivity/Specificity
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, arctangent, NOT, OR1, OR2, AND1, AND2, LT2A12, LT2B12, LT2E12, LT2F12, LT2G12, LT2C12, GOE2A12, GOE2B12, GOE2C12, GOE2E12, and GOE2G12.
Chromosomes: 30
Genes: 3
Head Size: 7
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 5
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template B

Independent Variables: 4-8
Fitness Function: Sensitivity/Specificity
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, arctangent, NOT, OR1, OR2, AND1, AND2, LT2A12, LT2B12, LT2E12, LT2F12, LT2G12, LT2C12, GOE2A12, GOE2B12, GOE2C12, GOE2E12, and GOE2G12.
Chromosomes: 30
Genes: 3
Head Size: 8
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 5
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1

RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template C

Independent Variables: 9-32
Fitness Function: Sensitivity/Specificity
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, arctangent, NOT, OR1, OR2, AND1, AND2, LT2A12, LT2B12, LT2E12, LT2F12, LT2G12, LT2C12, GOE2A12, GOE2B12, GOE2C12, GOE2E12, and GOE2G12.
Chromosomes: 30
Genes: 4
Head Size: 10
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 6
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template D

Independent Variables: 33-64
Fitness Function: Sensitivity/Specificity
Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, arctangent, NOT, OR1, OR2, AND1, AND2, LT2A12, LT2B12, LT2E12, LT2F12, LT2G12, LT2C12, GOE2A12, GOE2B12, GOE2C12, GOE2E12, and GOE2G12.
Chromosomes: 30
Genes: 5
Head Size: 12
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 7
Mutation Rate: 0.044
Inversion Rate: 0.1

IS Transposition Rate: 0.1
 RIS Transposition Rate: 0.1
 One-Point Recombination Rate: 0.3
 Two-Point Recombination Rate: 0.3
 Gene Recombination Rate: 0.1
 Gene Transposition Rate: 0.1
 RNC Mutation Rate: 0.044
 Dc Mutation Rate: 0.1
 Dc Inversion Rate: 0.1
 Dc IS Transposition Rate: 0.1

Template E

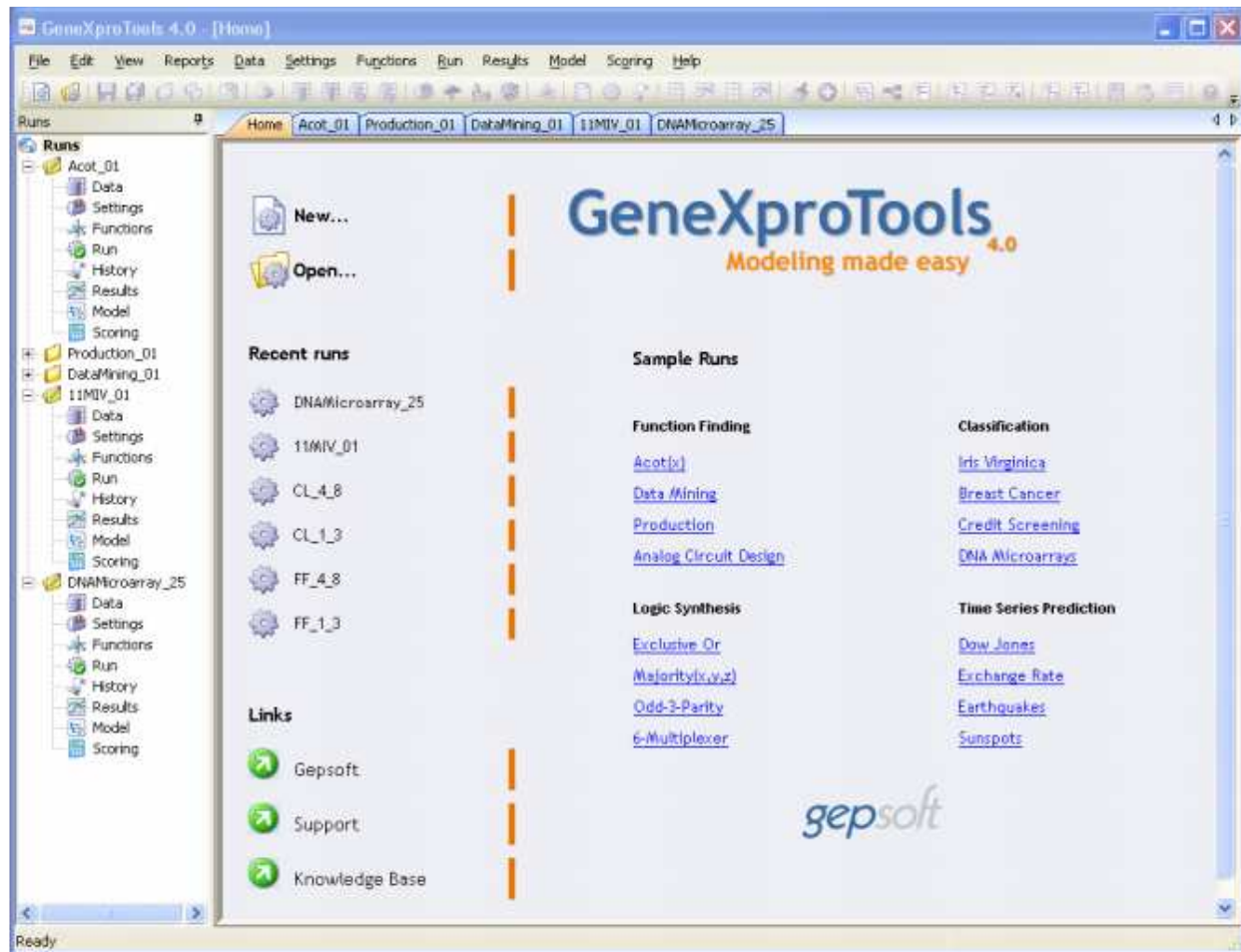
Independent Variables: 65-Unlimited
 Fitness Function: Sensitivity/Specificity
 Function Set: addition (4), subtraction (4), multiplication (4), division, square root, exponential, natural logarithm, x^2 , x^3 , sine, cosine, cube root, arctangent, NOT, OR1, OR2, AND1, AND2, LT2A12, LT2B12, LT2E12, LT2F12, LT2G12, LT2C12, GOE2A12, GOE2B12, GOE2C12, GOE2E12, and GOE2G12.
 Chromosomes: 30
 Genes: 6
 Head Size: 15
 Linking Function: addition
 Random Numerical Constants: Yes
 Constants per Gene: 2
 Data Type: floating-point
 Lower Bound: -10
 Upper Bound: 10
 Complexity Increase: No
 Generations without Change: 2000
 Tries: 3
 Maximum Complexity: 8
 Mutation Rate: 0.044
 Inversion Rate: 0.1
 IS Transposition Rate: 0.1
 RIS Transposition Rate: 0.1
 One-Point Recombination Rate: 0.3
 Two-Point Recombination Rate: 0.3
 Gene Recombination Rate: 0.1
 Gene Transposition Rate: 0.1
 RNC Mutation Rate: 0.044
 Dc Mutation Rate: 0.1
 Dc Inversion Rate: 0.1
 Dc IS Transposition Rate: 0.1



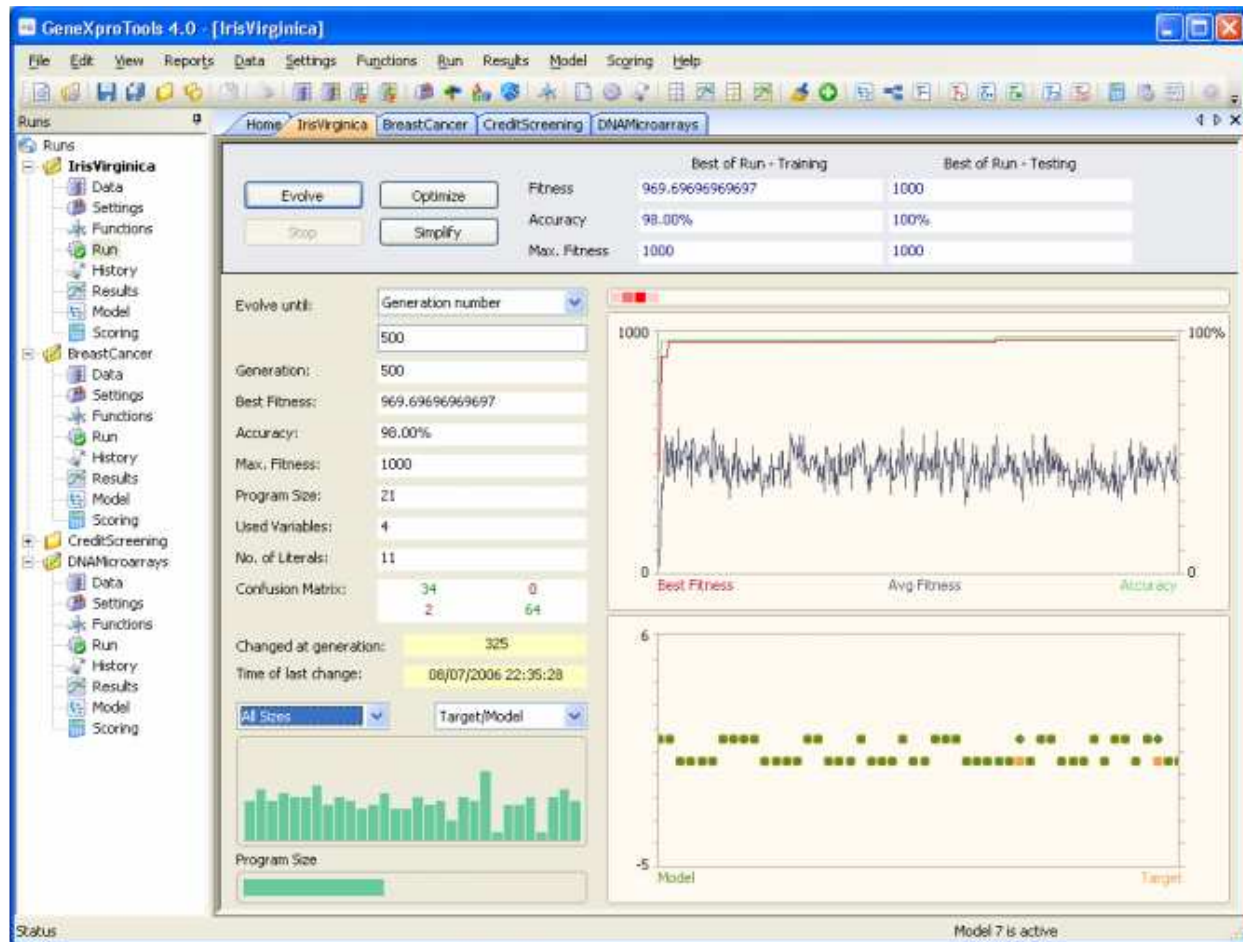
Settings and Features

Sample Runs

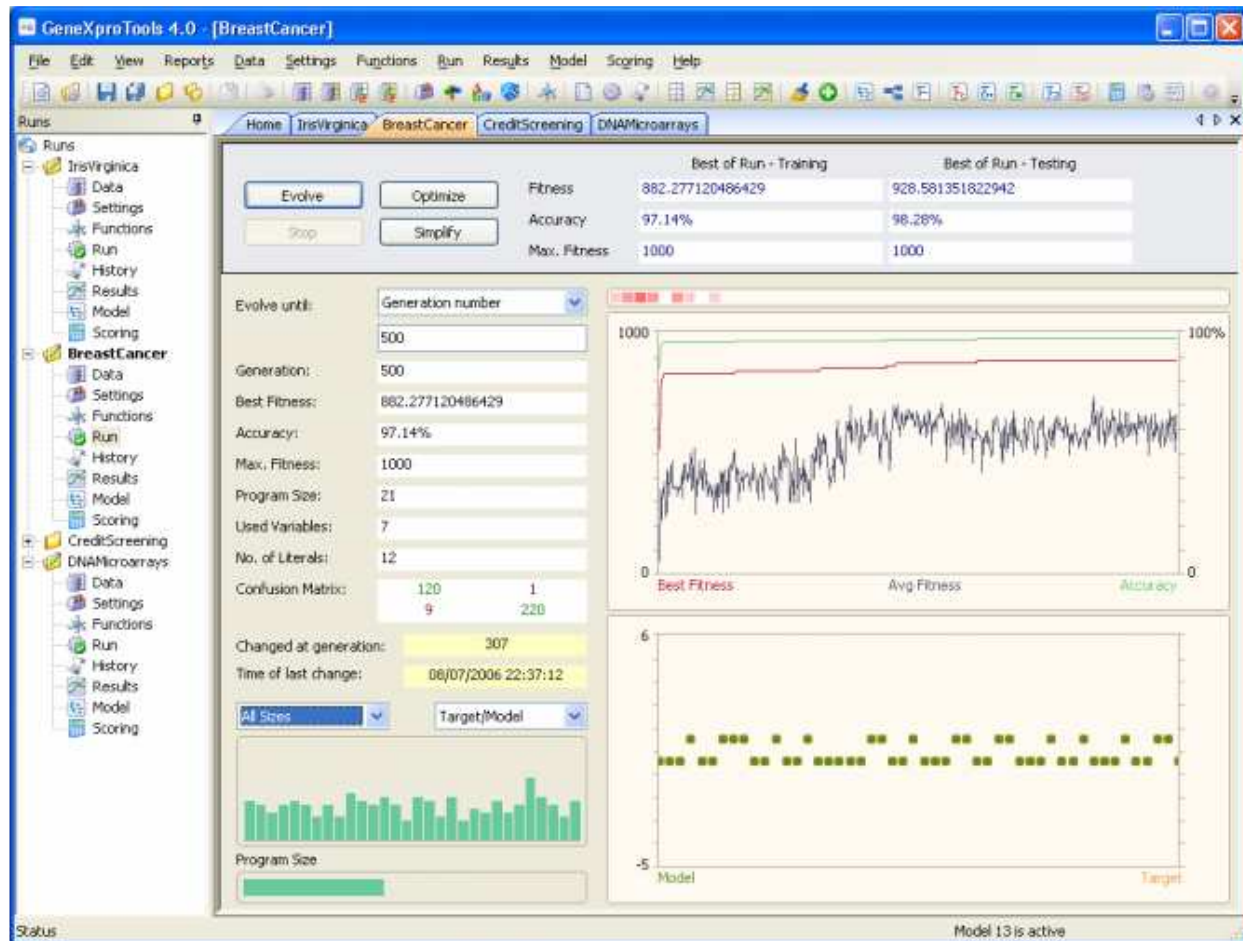
GeneXproTools 4.0 ships with four different **Sample Runs** for Classification: [Iris Virginica](#), [Breast Cancer](#), [Credit Screening](#), and [DNA Microarrays](#). To try any one of them, you just have to click its link on the Welcome Screen of GeneXproTools.



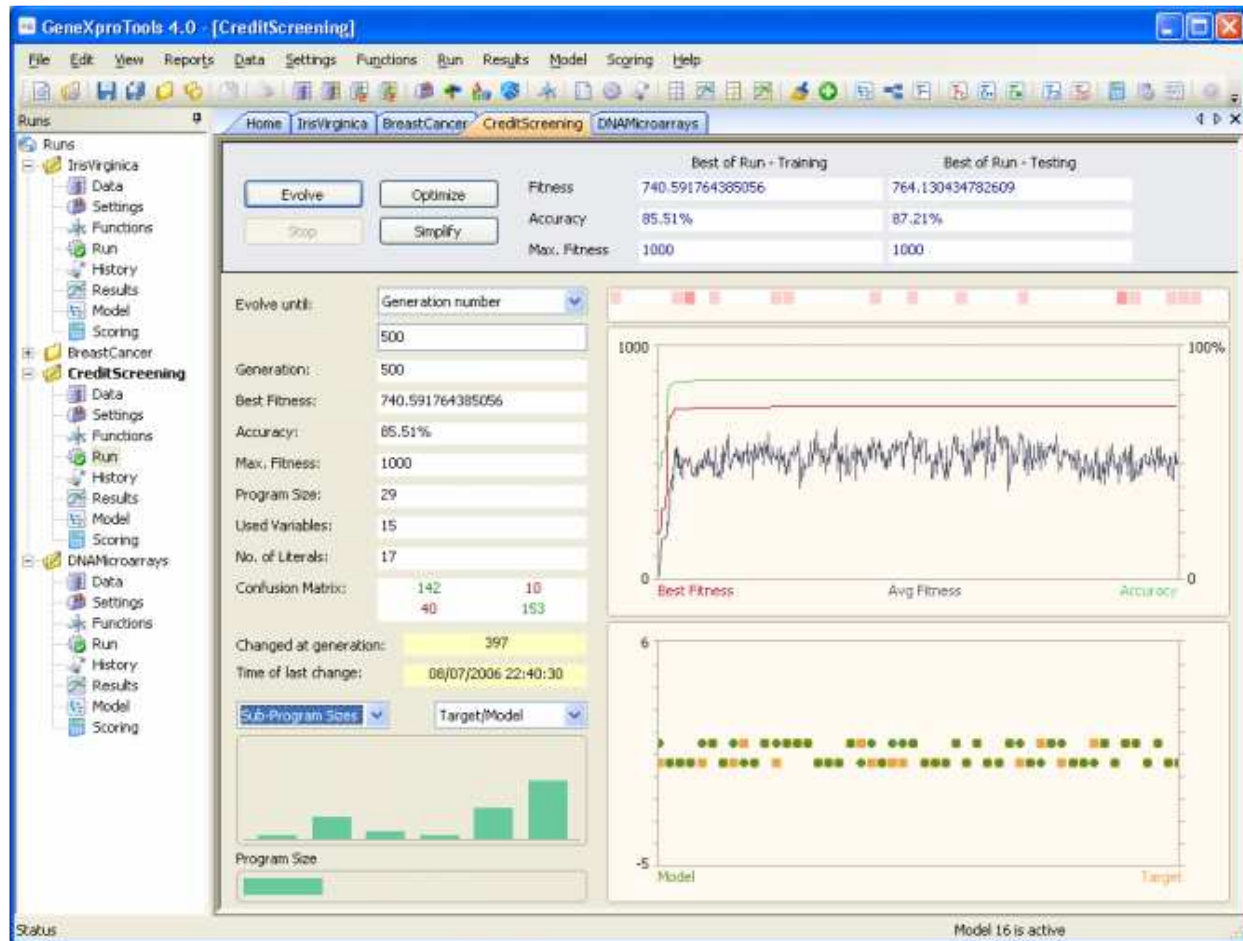
The **Iris Virginica** sample run is a simple real-world problem for distinguishing Iris Virginica from other two irises: Iris Setosa and Iris Versicolor. The original iris dataset contains fifty examples each of the three types of iris. In this sample run, though, the sub-problem Virginica versus Not Virginica is analyzed, where 100 randomly chosen samples are used for training and the remaining 50 for testing. You will see that this is an easy problem for GeneXproTools and exceptionally good models with 100% accuracy on the testing set can be easily created.



The **Breast Cancer** sample run is a complex real-world problem for diagnosing breast cancer based on nine different cell analysis (clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses). The dataset cancer1 used in this sample run was obtained from [PROBEN1](#), with 350 samples used for training and 174 for testing. And again you will see that GeneXproTools is able to create exceptionally good models for diagnosing breast cancer with predictive accuracies around 99%.



The **Credit Screening** sample run is a high-dimensional real-world problem. And the goal here is to decide whether to approve or not a customer's request for a credit card. Each sample in the dataset represents a real credit card application and the output describes whether the bank granted the credit card or not. This problem has 51 input attributes, all of them unexplained in the original dataset for confidentiality reasons. The dataset card1 used in this sample run was obtained from [PROBEN1](#), with 345 samples used for training and 172 for testing. And here you will see that GeneXproTools is able to create very good models with high predictive accuracy.



The **DNA Microarrays** sample run is an extremely complex real-world problem with thousands of variables. The training dataset consists of 38 bone marrow samples (27 ALL and 11 AML), over 7129 probes from 6817 human genes. And the testing dataset consists of 34 samples, with 20 ALL and 14 AML. In this sample run, "0" was used to represent "ALL" and "1" to represent "AML". The 7129 genes were numbered d_0 - d_{7128} .

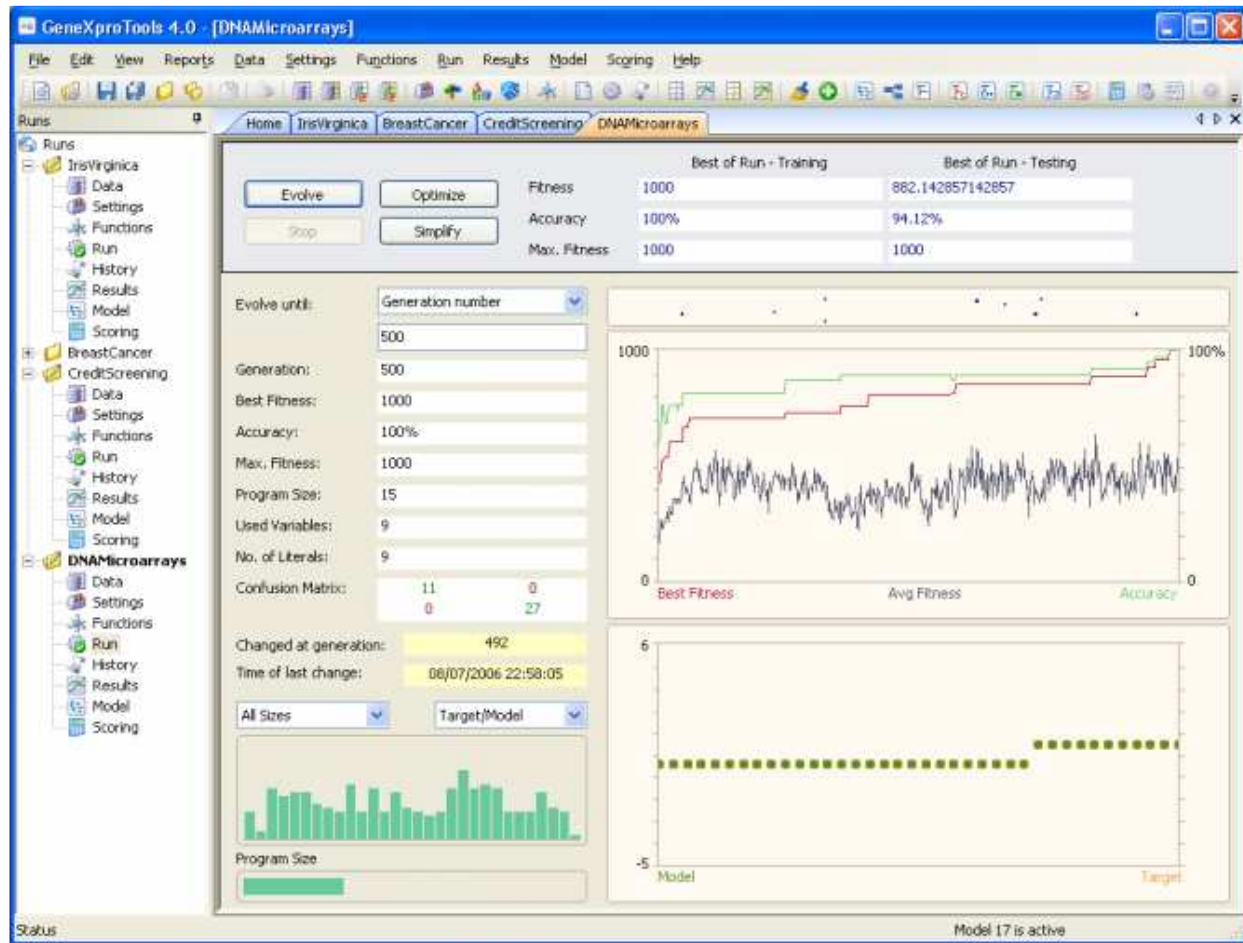
Thus, like in all DNA microarrays problems, in the [ALL-AML Leukemia problem](#), the number of samples available both for training and testing is quite small, which obviously poses some challenges.

First, it is common to try and narrow down the search space with sophisticated discretization algorithms to filter out the noise or irrelevant genes. And although good results have been reported (data size reductions of 50%-98%), the problem still remains: hundreds or thousands of variables to be mined using just a few dozen samples.

The great advantage of using GeneXproTools in DNA microarray studies is that you can use the **raw data** (obviously you can also use the filtered data) and still obtain excellent results. Obviously, not all the models with a good accuracy on the training data will have good predictive accuracy, but by making several runs one can select the top 10-20 models and then cross-reference the genes (attributes) used in all of them. You can then select and copy these most important genes from the [Data Panel](#) and create a much smaller dataset for creating the final model.

For instance, for the ALL-AML Leukemia problem, a total of 308 promising genes were identified in 25 good runs, that is, runs with 100% training accuracy and testing accuracies between 91.18%-97.06%. Of these 308 promising genes, only 11 (genes 759, 1881, 2287, 2407, 4362, 4846, 5485, 6040, 6587, 6638, and 6854) appeared in two or more models; and of these, only five (genes 759, 1881, 2287, 4846, and 6854) appeared in more than three models, with the most prevalent being genes 1881,

2287, and 4846, with eight, six, and four appearances, respectively. So, it is a good guess that the genes mostly to be involved in ALL-AML leukemia are genes 1881, 2287, and 4846, which is an exceptionally good starting point for tackling leukemia.



References

Golub, T. R., D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfeld, and E. S. Lander, 1999. Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science*, 286:531-537.

Prechelt, L., 1994. PROBEN1 - A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Univ. Karlsruhe, Germany.



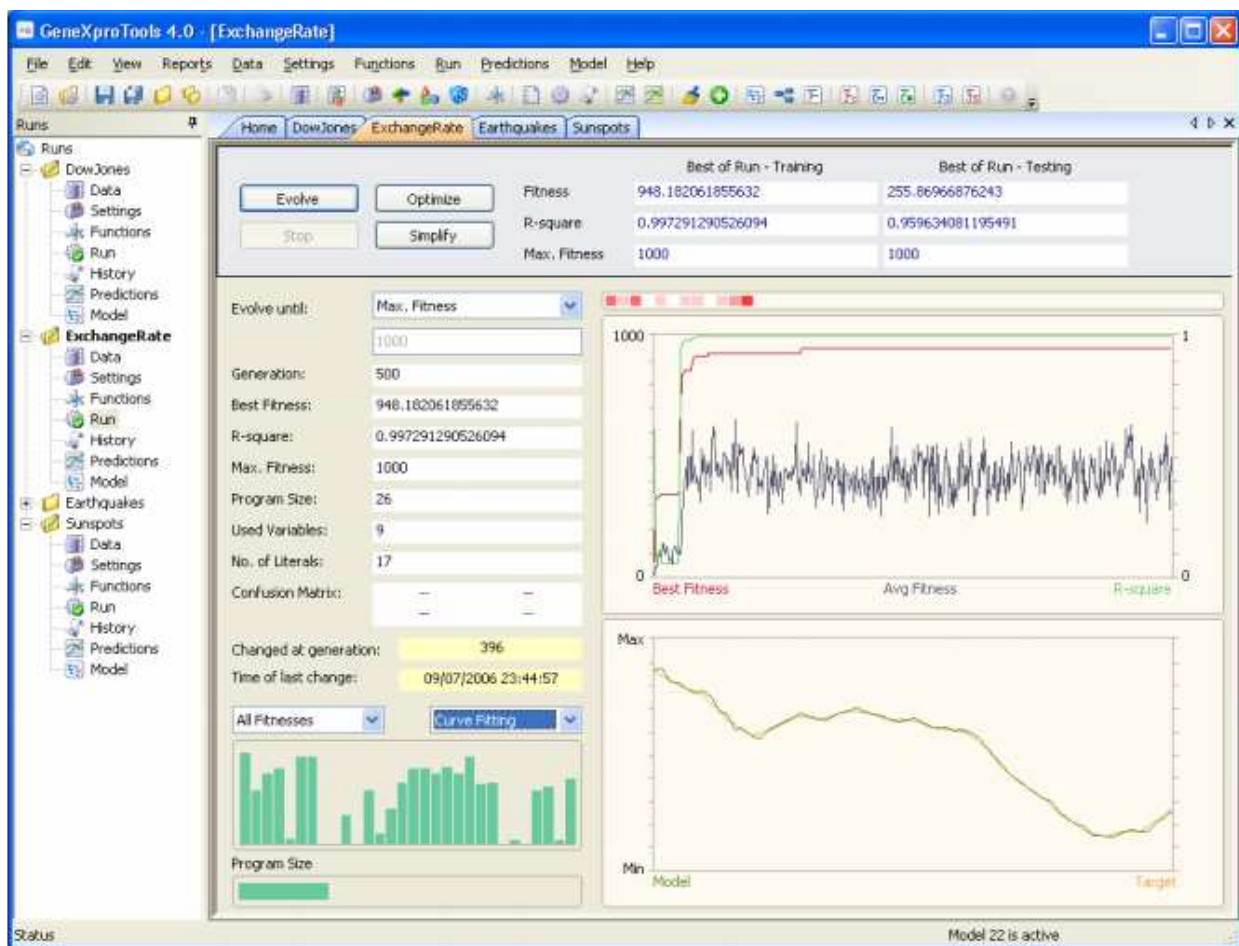
Settings and Features

Settings and Features for Time Series Prediction Problems

The settings and features of GeneXproTools 4.0 available for Time Series Prediction problems are

organized into the following panels and windows:

- New Run Wizard
- Report Panel
- Data Panel
- Change Time Series Engine
- Settings Panel
- Functions Panel
- Run Panel
- History Panel
- Predictions Panel
- Model Panel
- Change Seed Window
- Menus
- Run Templates
- Sample Runs

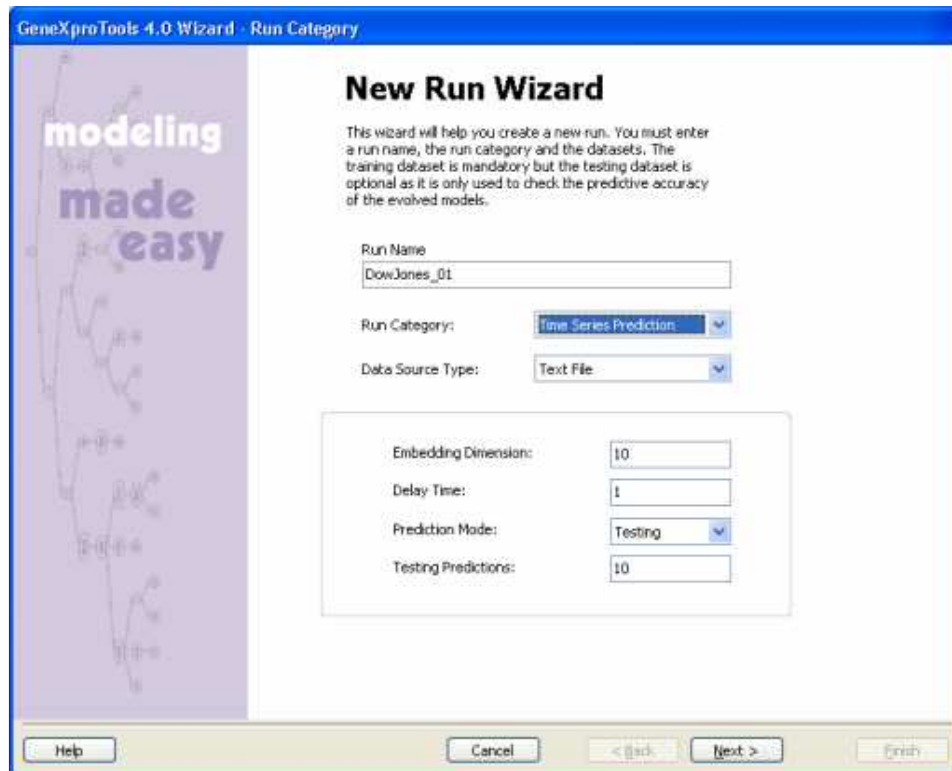


Settings and Features

New Run Wizard

The **New Run Wizard** of GeneXproTools 4.0 includes the following windows:

- [Run Category Window](#)
- [Training Data Window - Text Files](#)
- [Training Data Window - Excel & Databases](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Run Category Window

The **Run Category Window** gives access to the following settings and features:

Run Name

The name of the .gep file in which the run will be saved.

Run Category

For time series prediction problems, you must obviously choose Time Series Prediction and you'll notice that the Time Series Transformation Framework lights up.

Data Source Type

The type of data source you are using for modeling: Text Files or Excel & Databases.

Embedding Dimension

The value of the embedding dimension, which corresponds to the number of independent variables after the transformation of the time series.

Delay Time

The value of the delay time τ , which determines how data are processed, that is, continuously if $\tau = 1$ or at τ intervals.

Prediction Mode

Lists the two kinds of recursive predictions you can do in GeneXproTools: Prediction and Testing. The first one is used for making predictions about the future. And the second is used for evaluating the predictive power of the evolved models on past known events. In both cases, predictions are made recursively by evaluating the forecast at $t+1$, then using it to forecast $t+2$, and so on.

Testing Predictions

This box becomes active when Testing is selected in the Prediction Mode list box. Here you must enter the number of samples you want to reserve for the recursive test.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Next Button

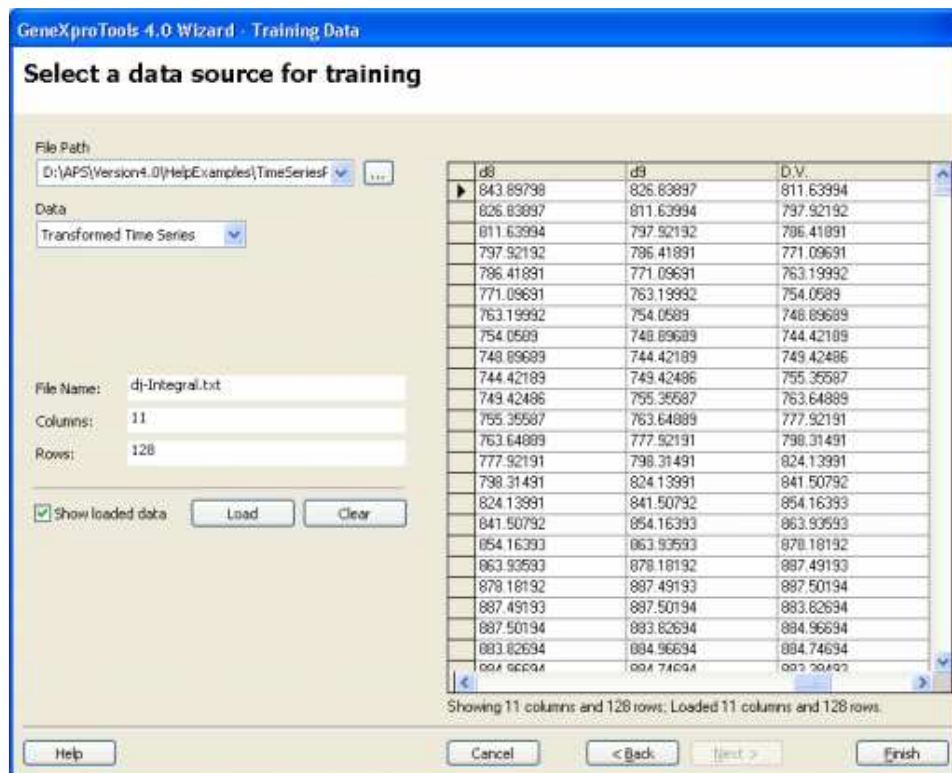
Takes you to the next window of the New Run Wizard.



Settings and Features

Training Data Window - Text Files

The **Training Data Window** gives access to the following settings and features:

**File Path**

Click the button on the right of the list box to load the time series file. You'll notice that GeneXproTools automatically transforms your time series according to the parameters entered on the [Run Category Window](#) (embedding dimension, delay time, and prediction mode).

Data

In this list box you can choose to see on the table on the right the Transformed Time Series, the Original Time Series, and the Time Series Testing Samples when in Testing Mode.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns in the current dataset.

Rows

GeneXproTools shows in this box the number of rows in the current dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads and transforms the time series.

Clear Button

Clears the datasets.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Finish Button

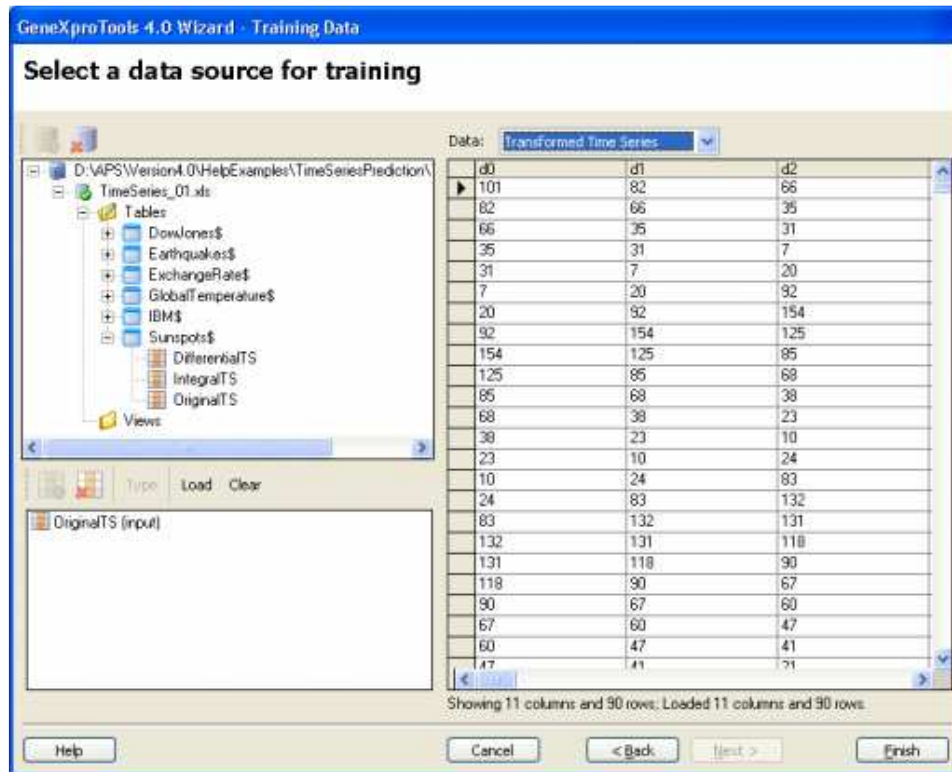
Opens the Save As dialog box to save the new run.



Settings and Features

Training Data Window - Excel & Databases

The **Training Data Window** gives access to the following settings and features:

**Add Data Source Icon**

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Time Series Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column, in this case, the single column with the time series.

Type Menu

This menu is disabled in Time Series Prediction as GeneXproTools automatically transforms the times series and automatically sets the predictable.

Load Button

Loads and transforms the time series.

Clear Button

Removes the column with the time series from the Time Series Window.

Time Series Window

You prepare your training data here by dragging and dropping the column with your time series from the Data Source Window. And, after clicking the Load button, GeneXproTools will automatically transform your time series according to the parameters entered on the [Run Category Window](#) (embedding dimension, delay time, and prediction mode).

Data

In this list box you can choose to see on the table below the Transformed Time Series, the Original Time Series, and the Time Series Testing Samples when in Testing Mode. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar at the bottom of the table.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Finish Button

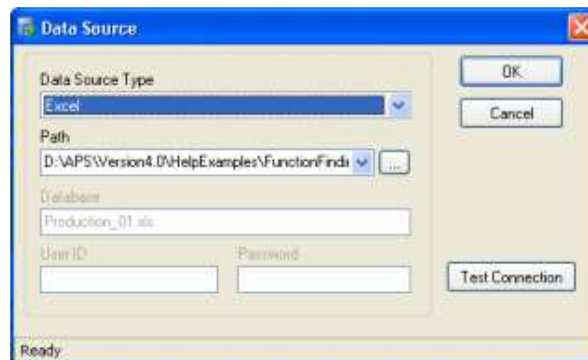
Opens the Save As dialog box to save the new run.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:

**Driver**

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.

Microsoft Access: path to the database file.

Microsoft SQL Server: server name or IP address.

Oracle: server name or IP address.

MySQL: server name or IP address.

DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.

Oracle - Name and database instance (HOST/INSTANCE).

MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

Cancels all the action on the data source connection.



Settings and Features

Report Panel

The **Report Panel** of GeneXproTools 4.0 keeps a summary of all the relevant information of a run, and is organized into the following headings:



Statistics – Training

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, number of calculation errors (they should always be zero, unless the seed was modified or the training set was changed after the model had been created), and number of outliers (when the fitness function is one of the following: [Relative with SR](#), [Relative/Hits](#), [Absolute with SR](#), and [Absolute/Hits](#)).

file://C:\Documents and Settings\candida\Local Settings\Temp\~hh4BEF.htm

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	24	9	d0(5), d1(3), d2(1)	847.262339126746	859.152192900566	0.960006472679246	0.937587454632183
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Sqrt} + \text{Atan} X3} \cdot d0 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d0 \cdot d2 \cdot d0 \\ & + \\ & \sqrt{X2} / \sqrt{\text{Sqrt} + \text{Cos} d1} \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d0 \cdot d0 \\ & + \\ & X3 \cdot \text{Atan} d0 \cdot d0 \cdot \text{Cos} \cdot \text{Ln} \cdot \text{Sqrt} \cdot d0 \cdot d2 \cdot d2 \cdot d0 \cdot d1 \cdot d1 \cdot d0 \cdot d1 \end{aligned}$							
12	25	9	d0(4), d1(4), d2(1)	852.285418790807	864.046489990001	0.970670596225273	0.976046016260914
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Sqrt} + \text{Atan} X3} \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d1 \cdot d0 \\ & + \\ & \sqrt{X2} / \sqrt{\text{Sqrt} + \text{Cos} d1} \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d2 \cdot d2 \cdot d0 \\ & + \\ & \text{Atan} X3 \cdot \text{Atan} d0 \cdot d0 \cdot \text{Cos} \cdot \text{Exp} d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d1 \cdot d2 \cdot d1 \end{aligned}$							
23	25	9	d0(3), d1(4), d2(2)	864.701706874743	864.588575207565	0.97254807136764	0.976472602618102
$\begin{aligned} & \sqrt{X3} / \sqrt{\text{Sqrt} + \text{Atan} X3} \cdot d1 \cdot d2 \cdot d2 \cdot d1 \cdot d0 \cdot d2 \cdot d2 \cdot d0 \\ & + \\ & \sqrt{X2} / \sqrt{\text{Sqrt} + \text{Cos} d1} \cdot d0 \cdot d0 \cdot d1 \cdot d2 \cdot d2 \cdot d0 \cdot d0 \cdot d0 \\ & + \\ & \text{Atan} X3 \cdot \text{Atan} d0 \cdot d0 \cdot \text{Cos} \cdot \text{Exp} d1 \cdot d2 \cdot d2 \cdot d2 \cdot d1 \cdot d2 \cdot d0 \cdot d0 \end{aligned}$							

Statistics – Testing

Under this heading are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the testing set, that is, the efficiency of your model at predicting unknown behavior: [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), and [R-square](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, number of calculation errors, and number of outliers (when the fitness function is one of the following: [Relative with SR](#), [Relative/Hits](#), [Absolute with SR](#), and [Absolute/Hits](#)).

Note again that all the parameters listed here refer to the last evaluated model in the [Predictions Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	11	4	d0(1), d1(2), d2(1)	850.429030313836	870.616009249946	0.976590014908023	0.982091633615242
$\begin{aligned} & d1 \cdot \text{Sqrt} \cdot \text{Exp} - \text{Atan} X3 \cdot d2 \cdot c1 \cdot c1 \cdot c1 \cdot d0 \cdot d2 \cdot d0 \cdot d0 \cdot d0 \cdot c1 \\ & + \\ & \text{Atan} X3 / \text{Atan} \cdot \text{Exp} \cdot 3 \cdot \text{Rt} d1 \cdot * d2 \cdot d0 \cdot d2 \cdot d2 \cdot c1 \cdot d0 \cdot d0 \cdot c1 \cdot c0 \end{aligned}$							
Numerical Constants: Gene 1 c0 = 7.749878 c1 = 8.115112 Gene 2 c0 = -9.974549 c1 = -6.347351							
2	10	3	d1(2), d2(1)	859.910797156278	872.16215754529	0.975751560710861	0.980563947942955
$\begin{aligned} & d1 \cdot \text{Sqrt} \cdot \text{Exp} - d2 \cdot d2 \cdot X3 \cdot d2 \cdot c1 \cdot c1 \cdot c1 \cdot d0 \cdot d2 \cdot d0 \cdot d0 \cdot d0 \cdot c1 \\ & + \\ & \text{Atan} X3 / \text{Atan} \cdot \text{Exp} \cdot 3 \cdot \text{Rt} d1 \cdot X2 \cdot d2 \cdot d0 \cdot c1 \cdot d2 \cdot c1 \cdot d0 \cdot d0 \cdot c0 \cdot c0 \end{aligned}$							
Numerical Constants: Gene 1 c0 = 7.749878 c1 = 8.115112 Gene 2 c0 = -9.974549 c1 = -6.347351							

Data

Under this heading are listed the number of independent variables after the transformation of the time series, the number of samples in the training data, and the number of records in the time series.

Program Structure

The program size, the number of literals, and a detailed list of all used variables and respective weights are shown here. This information refers to the active model highlighted in yellow in the History heading.

Program Structure:	
Program Size:	11
Literals:	4
Used Variables:	$dx(t)$, $dx(2)$, $dx(1)$

Function Set

Under this heading are listed all the functions used to create your model, together with the symbols used to represent them in [Karva notation](#), their weights and arities.

Function Set		
Function	Symbol	Weight Arity
Addition	+	1 2
Subtraction	-	1 2
Multiplication	*	1 2
Division	/	1 2
Square root	Sqrt	1 1
Exponential	Exp	1 1
Natural logarithm	Ln	1 1
x to the power of 2	X2	1 1
x to the power of 3	X3	1 1
Sine	Sin	1 1
Cosine	Cos	1 1
Cube root	3Rt	1 1
Arctangent	Atan	1 1

Note that the highest value of arity in your function set determines the structure of each gene, making the length of the [tail](#) t grow according to the equation:

$$t = h (n-1) + 1$$

where h is the length of the [head](#) and n is maximum arity.

Settings

Under this heading there are six entries: General Settings, Time Series Settings, Complexity Increase, Fitness Function, Genetic Operators, and Numerical Constants. Under General Settings are listed the number of chromosomes, number of genes, head size, tail size, Dc size, gene size, and linking function. Under Time Series Settings are listed the embedding dimension, delay time, prediction mode, and number of testing predictions. Under Complexity Increase are listed the number of Generations Without Change, the Number of Tries, and Max. Complexity. The name of the fitness function and all the relevant parameters for its design are given below the entry Fitness Function. Under Genetic Operators are listed the rates of all the genetic operators used to create genetic variation. And, finally, all the parameters relevant to the manipulation of random numerical constants are listed under Numerical Constants.

Settings:	
General	
Chromosomes:	30
Genes:	3
Head Size:	8
Tail Size:	9
Dc Size:	9
Gene Size:	26
Linking Function:	Addition
Complexity Increase	
Generations without Change:	2000
Number of Tries:	3
Max. Complexity:	5
Fitness Function	
Error Type:	RRSE
Precision:	--
Selection Range:	--
Genetic Operators	
Mutation Rate:	0.044
Inversion Rate:	0.1
IS Transposition Rate:	0.1
RIS Transposition Rate:	0.1
One-Point Recombination Rate:	0.3
Two-Point Recombination Rate:	0.3
Gene Recombination Rate:	0.1
Gene Transposition Rate:	0.1
Numerical Constants	
Constants per Gene:	2
Data Type:	Floating-Point
Lower Bound:	-10
Upper Bound:	10
RNC Mutation:	0.01
Dc Mutation:	0.044
Dc Inversion:	0.1
Dc IS Transposition:	0.1

Data Sources

Here is registered the path to the original time series used to create your model so that its origin could be easily identified.

File Path

Under this heading the path to the gep file that was used to save your model is given so that it can be easily traced for future reference.

History

Under this heading are listed all the best-of-generation models, including obviously the best-of-run model, discovered during the evolutionary process. Note, however, that the notation "best-of-run" and "best-of-generation" refers to the performance of these models on the training set, which could be slightly different on the testing set. This is why GeneXproTools 4.0 allows you to analyze the performance of all these models on the testing set and, if appropriate, choose the best model based on these results. GeneXproTools 4.0 gives you the possibility of highlighting in yellow one of these models (the active model) and we advise you to use this feature to highlight the best one. Note also that, for a quick reference, not only the program structure (program size, number of literals, and list of used variables and respective weights) but also the fitness and the R-square of all these models both on the training and testing sets are also given, although, for the testing set, they must be evaluated in the [History Panel](#).

History

Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
0	24	9	di(5), di(3), di(1)	847.282339126748	859.152192900566	0.960006472679246	0.973587484532183
/.X3 / Sqrt + Atan X3 . d0 . d0 . d2 . d1 . d0 . d0 . d2 . d0 + /.X2 / Sqrt + Cos d1 . d0 . d0 . d1 . d2 . d2 . d2 . d0 . d0 + X3 . Atan d0 . d0 . Cos Ln Sqrt . d0 . d2 . d2 . d0 . d1 . d1 . d0 . d1							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
12	25	9	di(4), di(4), di(1)	852.285419790807	864.046489990801	0.970678596225273	0.975046816260914
/.X3 / Sqrt + Atan X3 . d1 . d0 . d2 . d1 . d0 . d2 . d1 . d0 + /.X2 / Sqrt + Cos d1 . d0 . d0 . d1 . d2 . d2 . d2 . d2 . d0 . d0 + Atan X3 . Atan d0 . d0 . Cos Exp d1 . d2 . d2 . d2 . d1 . d1 . d2 . d1							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training R-square	Testing R-square
23	25	9	di(3), di(4), di(2)	854.701706874743	864.588575207565	0.97254807136764	0.976472602618102
/.X3 / Sqrt + Atan X3 . d1 . d2 . d2 . d1 . d0 . d2 . d2 . d0 + /.X2 / Sqrt + Cos d1 . d0 . d0 . d1 . d2 . d2 . d0 . d0 . d0 + Atan X3 . Atan d0 . d0 . Cos Exp d1 . d2 . d2 . d2 . d1 . d2 . d0 . d0							

For brevity sake, all the best-of-generation models are represented in the compact [Karva notation](#), but you can translate any one of them into a more canonical representation by activating any of these models in the [History Panel](#) and then translate it into the language of your choice in the [Model Panel](#).

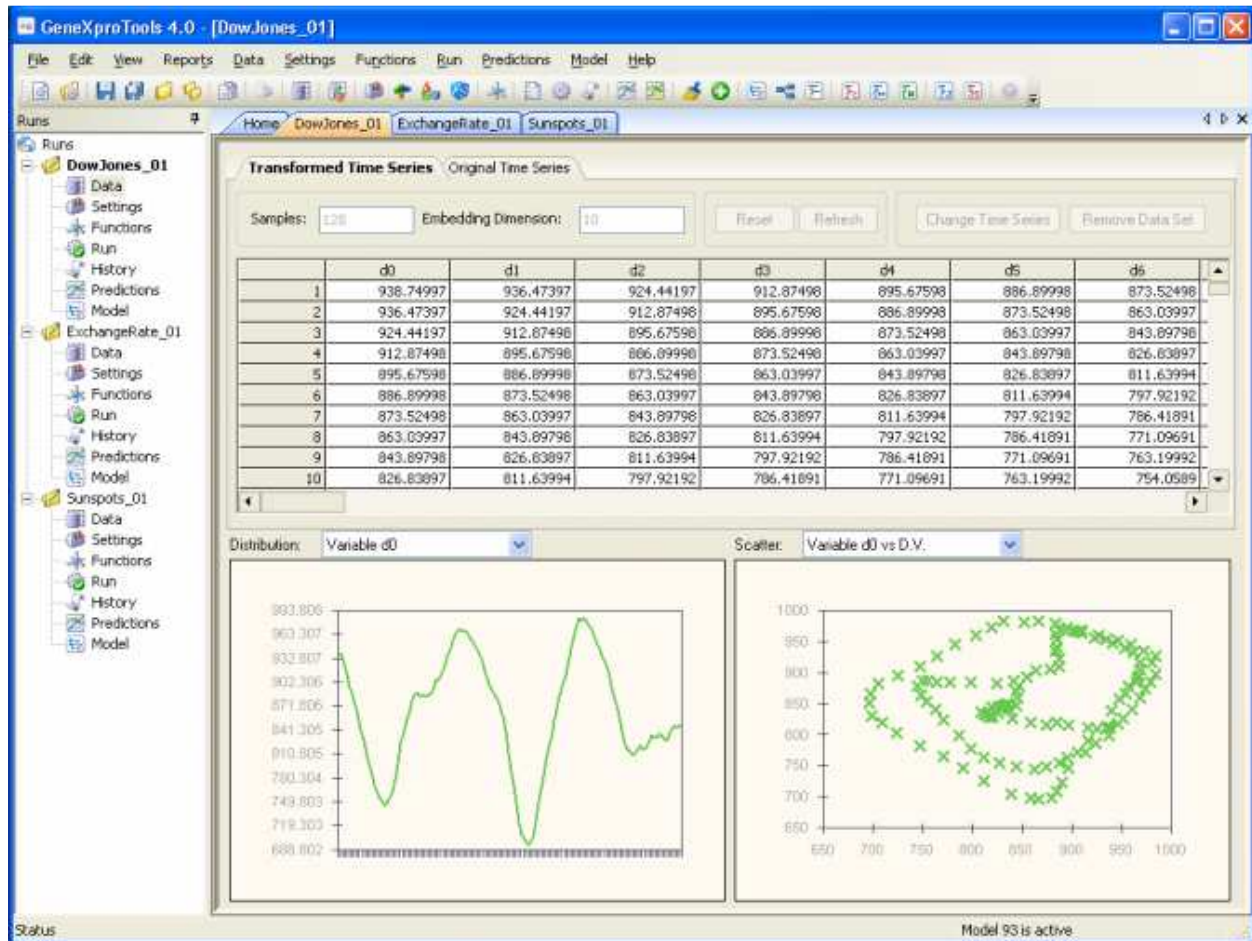


Settings and Features

Data Panel

The **Data Panel** of GeneXproTools 4.0 includes the following tabs:

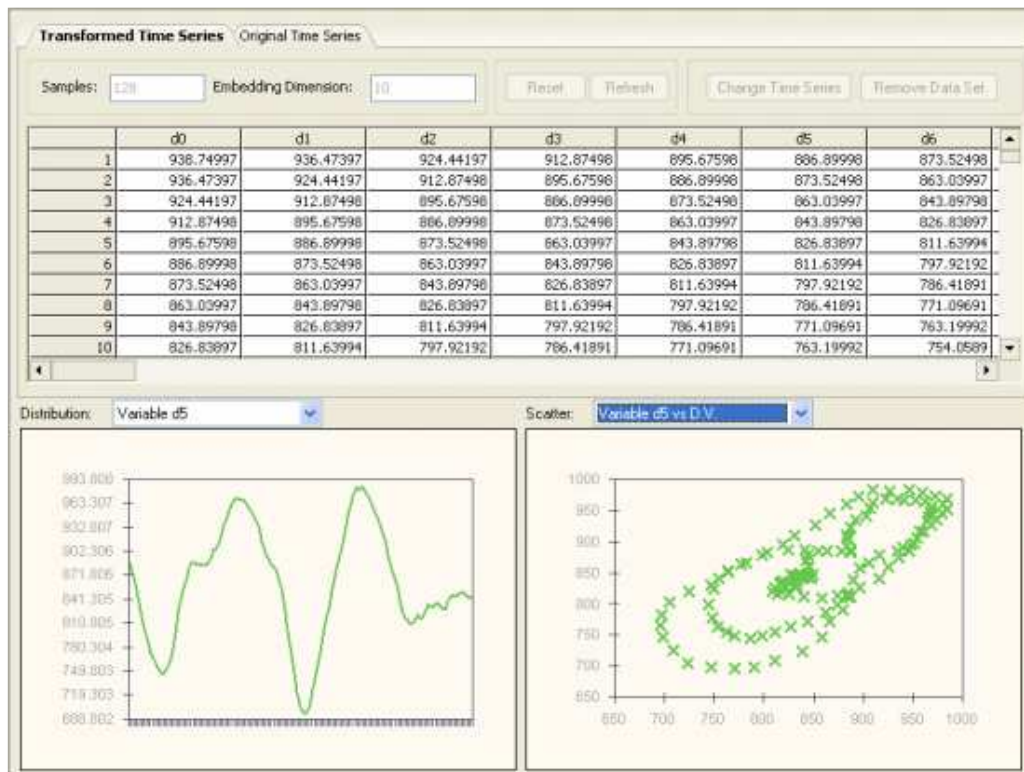
- [Transformed Time Series Tab](#)
- [Original Time Series Tab](#)



Settings and Features

Transformed Time Series Tab

The **Transformed Time Series Tab** gives access to the following settings and features:



Samples

This box is informative only and shows the number of samples used for training.

Embedding Dimension

This box is informative only and shows the embedding dimension (number of independent variables) used to create the training set.

Reset Button

Disabled in Time Series Prediction.

Refresh Button

Disabled in Time Series Prediction.

Change Time Series Button

Disabled in the Transformed Time Series Tab; you can only change the time series from the [Original Time Series Tab](#).

Remove Data Set Button

Disabled in Time Series Prediction.

Table

Shows all the variables in the training set. The entire table can be copied to the clipboard by choosing Copy on the Edit Menu. You can also copy selected columns to the clipboard with the mouse right-click.

Distribution

Allows you to visualize the distribution of values for each variable in the training set. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.

Scatter

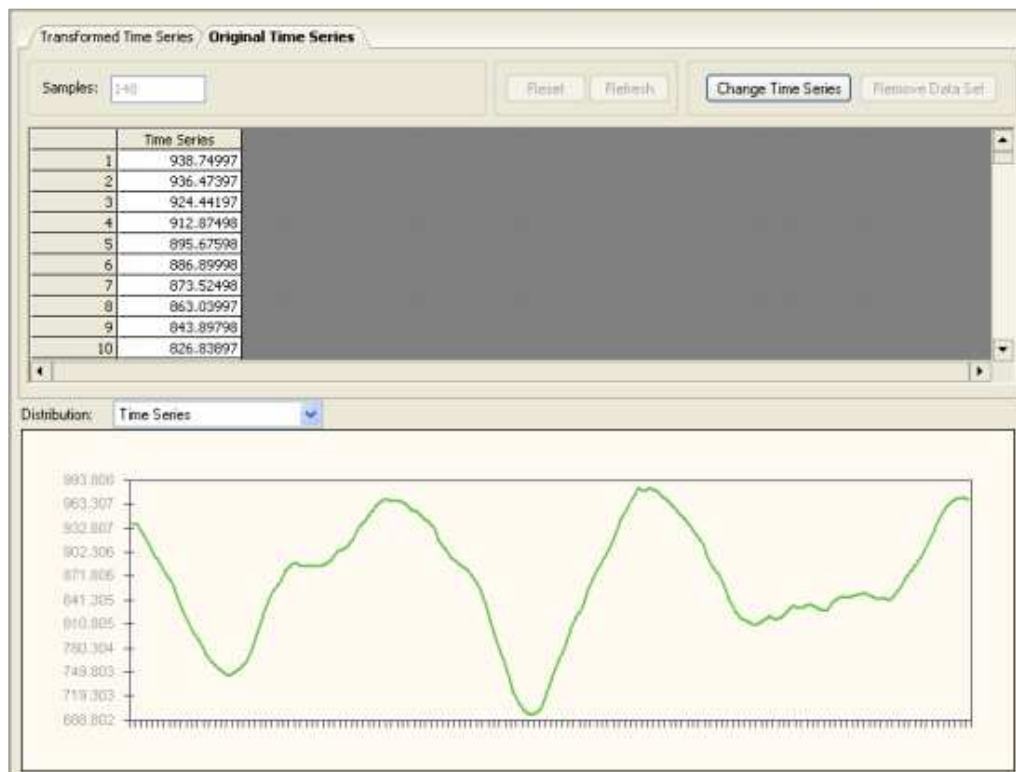
Allows you to visualize the scatter plot of each independent variable in the training set against the dependent variable. This plot shows a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.



Settings and Features

Original Time Series Tab

The **Original Time Series Tab** gives access to the following settings and features:



Samples

This box is informative only and shows the number of observations in the time series.

Reset Button

Disabled in Time Series Prediction.

Refresh Button

Disabled in Time Series Prediction.

Change Time Series Button

Allows you to choose another time series for modeling.

Remove Data Set Button

Disabled in Time Series Prediction.

Table

Shows the original time series. The entire table can be copied to the clipboard by choosing Copy on the Edit Menu.

Distribution

Allows you to visualize the distribution of the time series. This plot shows a maximum of 250 points at a time and, therefore, for time series with more than 250 records you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.



Settings and Features

Change Time Series Engine

The **Change Time Series Engine** of GeneXproTools 4.0 includes the following windows:

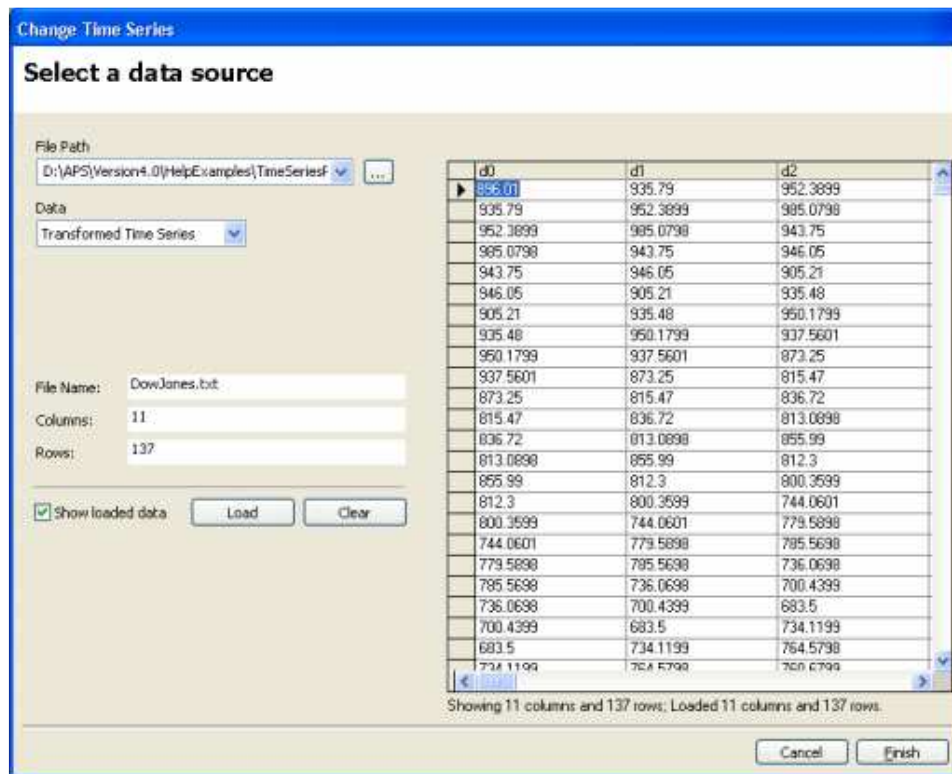
- [Change Time Series Window - Text Files](#)
- [Change Time Series Window - Excel & Databases](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Change Time Series Window - Text Files

The **Change Time Series Window - Text Files** allows you to change the time series and gives access to the following settings and features:



File Path

Click the button on the right of the list box to load the time series file. You'll notice that GeneXproTools automatically transforms your time series according to the embedding dimension, delay time, and prediction mode you chose for the run.

Data

In this list box you can choose to see on the table on the right the Transformed Time Series, the Original Time Series, and the Time Series Testing Samples when in Testing Mode.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns in the current dataset.

Rows

GeneXproTools shows in this box the number of rows in the current dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns in the Change Time Series Window, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads and transforms the time series.

Clear Button

Clears the datasets.

Cancel Button

Cancels all the action on the replacement of the time series.

Finish Button

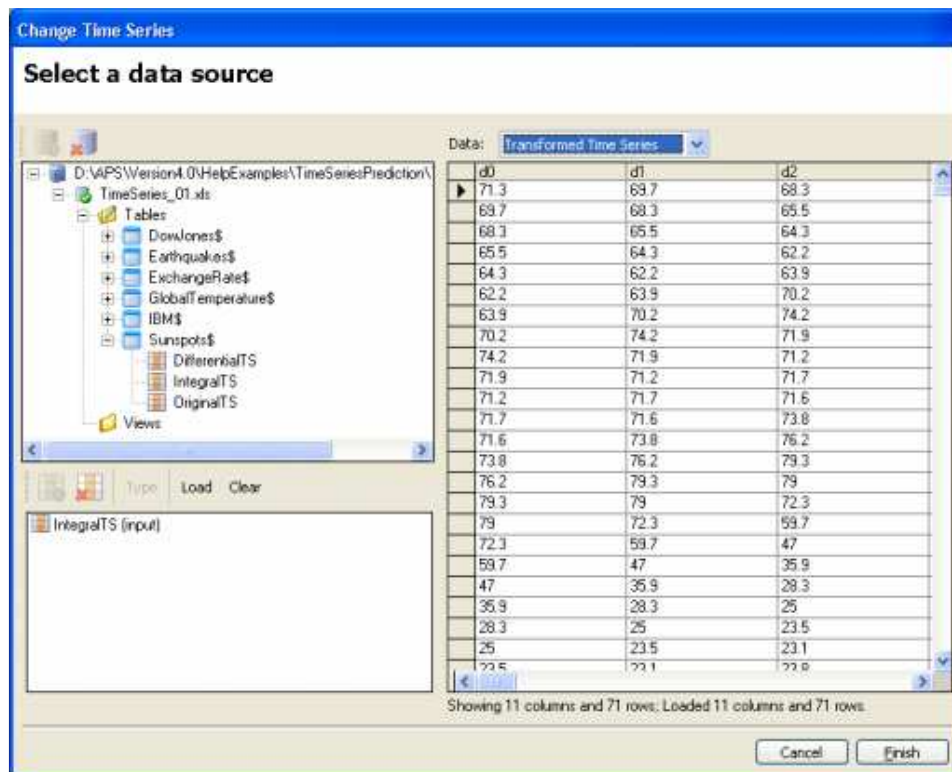
Opens the Save As dialog box to save the new run.



Settings and Features

Change Time Series Window - Excel & Databases

The **Change Time Series Window - Excel & Databases** allows you to change the time series and gives access to the following settings and features:

**Add Data Source Icon**

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Time Series Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column, in this case, the single column with the time series.

Type Menu

This menu is disabled in Time Series Prediction as GeneXproTools automatically transforms the times series and automatically sets the predictable.

Load Button

Loads and transforms the time series.

Clear Button

Removes the column with the time series from the Time Series Window.

Time Series Window

You prepare your training data here by dragging and dropping the column with your time series from the Data Source Window. And, after clicking the Load button, GeneXproTools will automatically transform your time series according to the embedding dimension, delay time, and prediction mode you chose for the run.

Data

In this list box you can choose to see on the table below the Transformed Time Series, the Original Time Series, and the Time Series Testing Samples when in Testing Mode. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar at the bottom of the table.

Cancel Button

Cancels all the action on the creation of a new run.

Finish Button

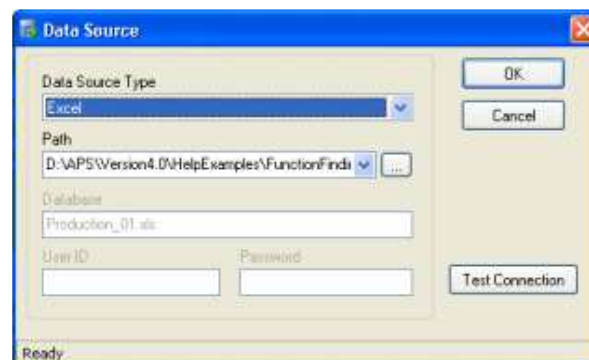
Opens the Save As dialog box to save the new run.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:



Driver

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database

engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.

Microsoft Access: path to the database file.

Microsoft SQL Server: server name or IP address.

Oracle: server name or IP address.

MySQL: server name or IP address.

DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.

Oracle - Name and database instance (HOST/INSTANCE).

MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

Cancels all the action on the data source connection.

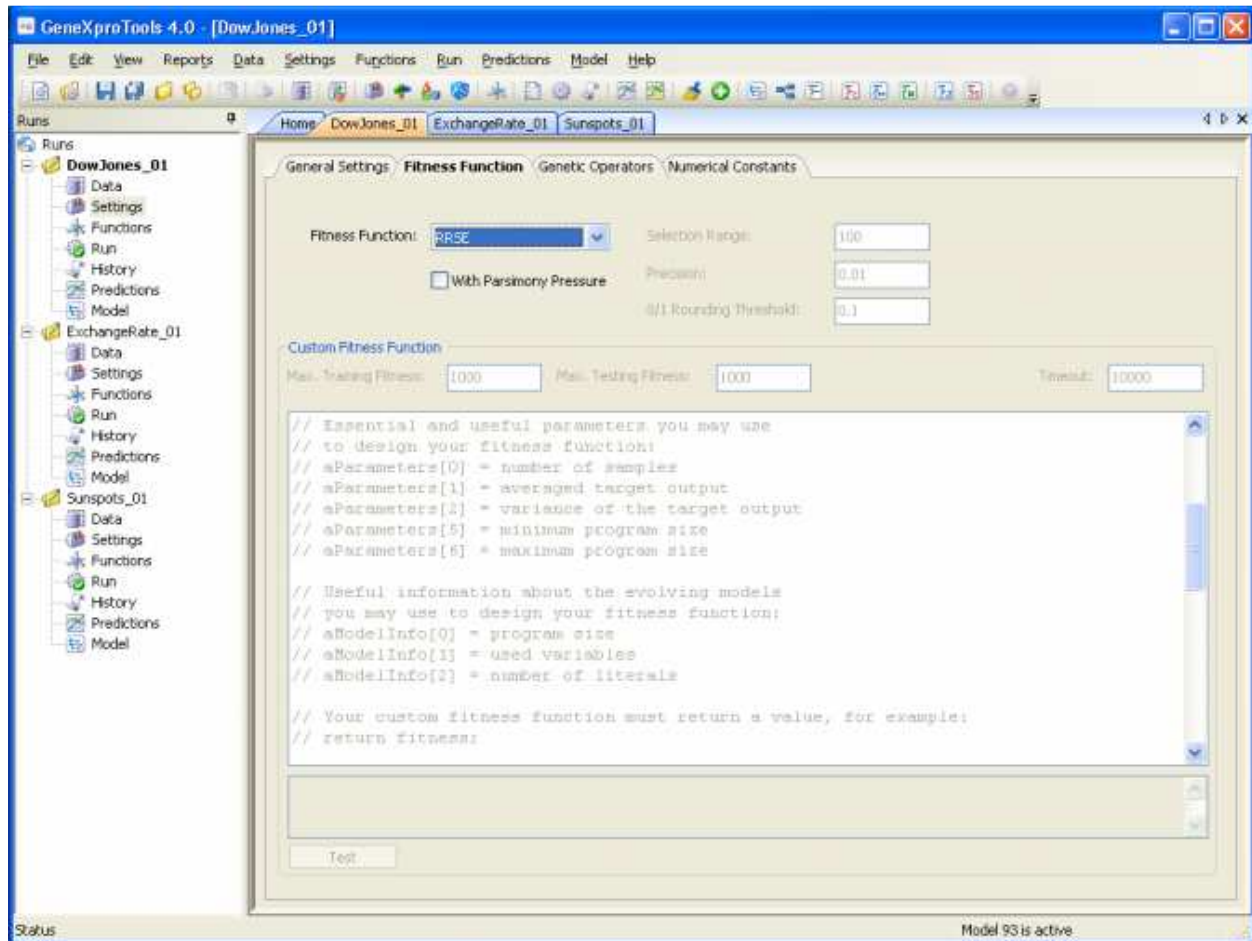


Settings and Features

Settings Panel

The **Settings Panel** of GeneXproTools 4.0 includes the following tabs:

- [General Settings Tab](#)
- [Fitness Function Tab](#)
- [Genetic Operators Tab](#)
- [Numerical Constants Tab](#)



Settings and Features

General Settings Tab

The **General Settings Tab** gives access to the following settings and features:

General Settings | Fitness Function | Genetic Operators | Numerical Constants

Training Samples: 128 [Reset]

Records in Time Series: 148 [Reset]

Number of Chromosomes: 30 [v]

Embedding Dimension: 10 [v]

Head Size: 8 [v]

Delay Time: 1 [v]

Number of Genes: 5 [v]

Prediction Mode: Testing [v]

Linking Function: Addition [v]

Testing Predictions: 10 [v]

☒ Enable Complexity Increase

Generations without Change: 2000 [v]

Number of Tries: 3 [v]

Max. Complexity: 8 [v]

☒ Save Run History

Training Samples

Number of samples used for training. This box is informative only as the number of training samples depends on the parameters chosen to transform the time series such as Embedding Dimension, Delay Time, Prediction Mode, and the number of Testing Predictions.

Records in Time Series

This box is informative only and shows the number of records in the current time series.

Number of Chromosomes

Size of the population of models created each generation for breeding new ones.

Head Size

Size of the [head](#) of each gene. This parameter determines the upper limit for the size of the programs encoded in the gene.

Number of Genes

Number of genes per chromosome or, in other words, the number of complex terms in the evolving models.

Linking Function

Function used to link the different terms encoded in the genes of a model. You can choose either addition, subtraction, multiplication, or division to link the [sub-ETs](#) of your models.

Embedding Dimension

The embedding dimension used to transform the time series for training. It corresponds to the number of independent variables after the transformation of the time series.

Delay Time

The value of the delay time τ , which determines how data are processed, that is, continuously if $\tau = 1$ or at τ intervals.

Prediction Mode

Lists the two kinds of recursive predictions you can do in GeneXproTools: Prediction and Testing. The first one is used for making predictions about the future. And the second is used for evaluating the predictive power of the evolved models on past known events. In both cases, predictions are made recursively by evaluating the forecast at $t+1$, then using it to forecast $t+2$, and so on.

Testing Predictions

This box becomes active when Testing is selected in the Prediction Mode list box. Here you must enter the number of samples you want to reserve for the recursive test.

Enable Complexity Increase

Check this box if you wish to allow GeneXproTools to automatically increase the number of genes in the evolving models.

Generations without Change

The period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction is created or a neutral gene (an extra term) is automatically added to all evolving models.

Number of Tries

The number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models.

Max. Complexity

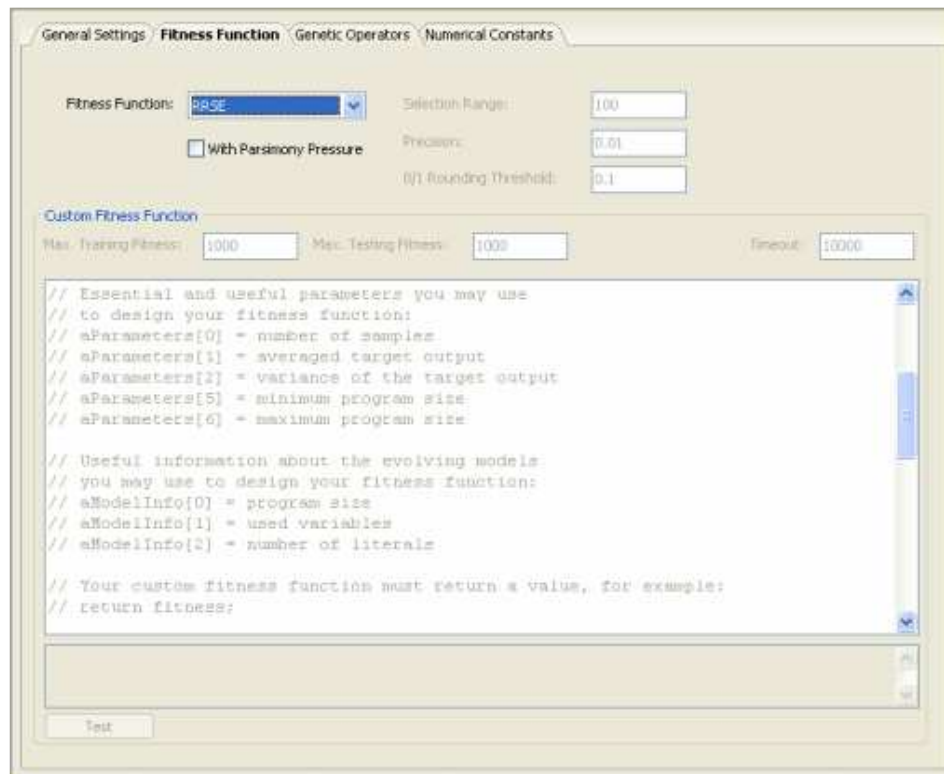
The maximum number of terms (genes) you think reasonable for modeling your data. Beyond this threshold no other terms are introduced automatically by the Complexity Increase Engine and GeneXproTools stops evolving new models.



Settings and Features

Fitness Function Tab

The **Fitness Function Tab** gives access to the following settings and features:



Fitness Function

Here you select either one of the 36 basic [built-in fitness functions](#) of GeneXproTools 4.0 for Time Series Prediction or a [custom fitness function](#).

With Parsimony Pressure Check Box

By checking this box, you select a fitness function with [parsimony pressure](#).

Selection Range

The Selection Range (SR) relates exclusively to the fitness functions [Relative with SR](#) and [Absolute with SR](#) and is used as a limit for selection to operate, above which the performance of a program on a particular fitness case contributes nothing to its fitness.

Precision

The Precision in GeneXproTools 4.0 is associated with four different fitness functions. The Precision related to the fitness functions [Relative with SR](#) and [Absolute with SR](#) corresponds to the limit for improvement, for the closer solutions are to this limit the better they fit the data. The Precision related to the fitness functions [Relative/Hits](#) and [Absolute/Hits](#) corresponds to the error (relative or absolute) below which the performance of a program on a particular fitness case counts as a hit.

0/1 Rounding Threshold

This box is specific of Classification problems, and therefore is disabled in Function Finding.

Custom Fitness Function Frame

Select Custom in the Fitness Function combo box in order to activate this frame. You can now design or edit your own fitness function.

Custom Fitness Function Code Window

You design or edit the code of your own fitness function in the code window. The code must be in Java Script and can be tested for syntax errors before evolving models with it.

Test

Tests the syntax of the Java Script code of your fitness function for syntax errors. The results of the test are written in the gray window below the code window.

Max. Training Fitness

The maximum fitness corresponding to the custom fitness function on the training set. If maximum fitness depends on the number of training samples, don't forget to update this value every time you change the number of training samples, otherwise you might be creating some inaccuracies or errors. For instance, the plot of the Evolutionary Dynamics on the [Run Panel](#) won't show correctly if your maximum fitness is not correctly set.

Max. Testing Fitness

The maximum fitness corresponding to the custom fitness function on the testing set. If maximum fitness depends on the number of testing samples, don't forget to update this value every time you change the number of testing samples, otherwise you might be creating some inaccuracies.

Timeout

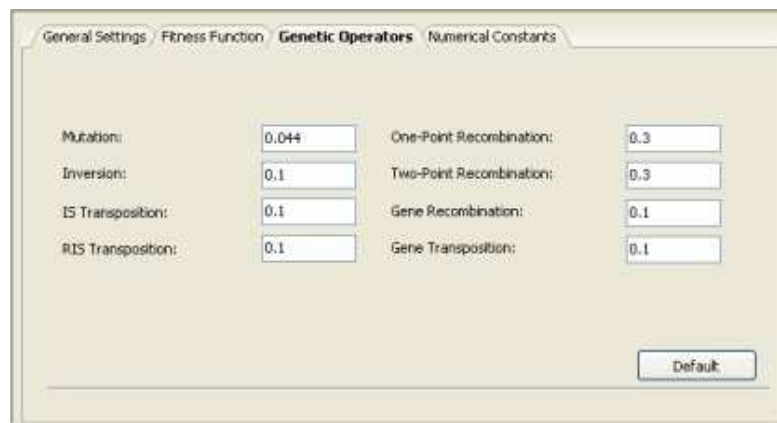
Despite one's best efforts, the code one sometimes writes contains errors that keep computers in infinite loops. This setting gives you a gracious way out of such loops.



Settings and Features

Genetic Operators Tab

The **Genetic Operators Tab** gives access to the following settings and features:



Mutation

The [mutation](#) rate used to create genetic variation. The default is 0.044.

Inversion

The [inversion](#) rate used to create genetic variation. The default is 0.1.

IS Transposition

The [IS transposition](#) rate used to create genetic variation. The default is 0.1.

RIS Transposition

The [RIS transposition](#) rate used to create genetic variation. The default is 0.1.

One-Point Recombination

The [one-point recombination](#) rate used to create genetic variation. The default is 0.3.

Two-Point Recombination

The [two-point recombination](#) rate used to create genetic variation. The default is 0.3.

Gene Recombination

The [gene recombination](#) rate used to create genetic variation. The default is 0.1.

Gene Transposition

The [gene transposition](#) rate used to create genetic variation. The default is 0.1.

Default

Resets the rates of all the above genetic operators to the default values.



Settings and Features

Numerical Constants Tab

The **Numerical Constants Tab** gives access to the following settings and features:

Use Random Numerical Constants

Check this box if you want to use the [GEP-RNC learning algorithm](#) of GeneXproTools. This algorithm has a facility for handling random numerical constants and it evolves models with them.

Constants per Gene

The number of random numerical constants associated with each gene.

Data Type

The data type (integer or floating-point) you think appropriate for the random numerical constants that will be used in the creation of your models. For Integer constants only integer values in the interval [-2147483648, 2147483647] are acceptable. For Floating-point constants a double-precision floating-

point value with a range of -1.79769313486231E308 to -4.94065645841247E-324 for negative values, 4.94065645841247E-324 to 1.79769313486231E308 for positive values, and 0 are acceptable.

Lower Bound

Sets the lower bound of the interval from which random numerical constants are drawn.

Upper Bound

Sets the upper bound of the interval from which random numerical constants are drawn.

RNC Mutation

The [RNC mutation](#) rate used to create genetic variation in the arrays of random numerical constants. The default is 0.01.

Dc Mutation

The [Dc mutation](#) rate used to create genetic variation. The default is 0.044.

Dc Inversion

The [Dc inversion](#) rate used to create genetic variation. The default is 0.1.

Dc IS Transposition

The [Dc IS transposition](#) rate used to create genetic variation. The default is 0.1.

Default

Resets the rates of all the above genetic operators to the default values.

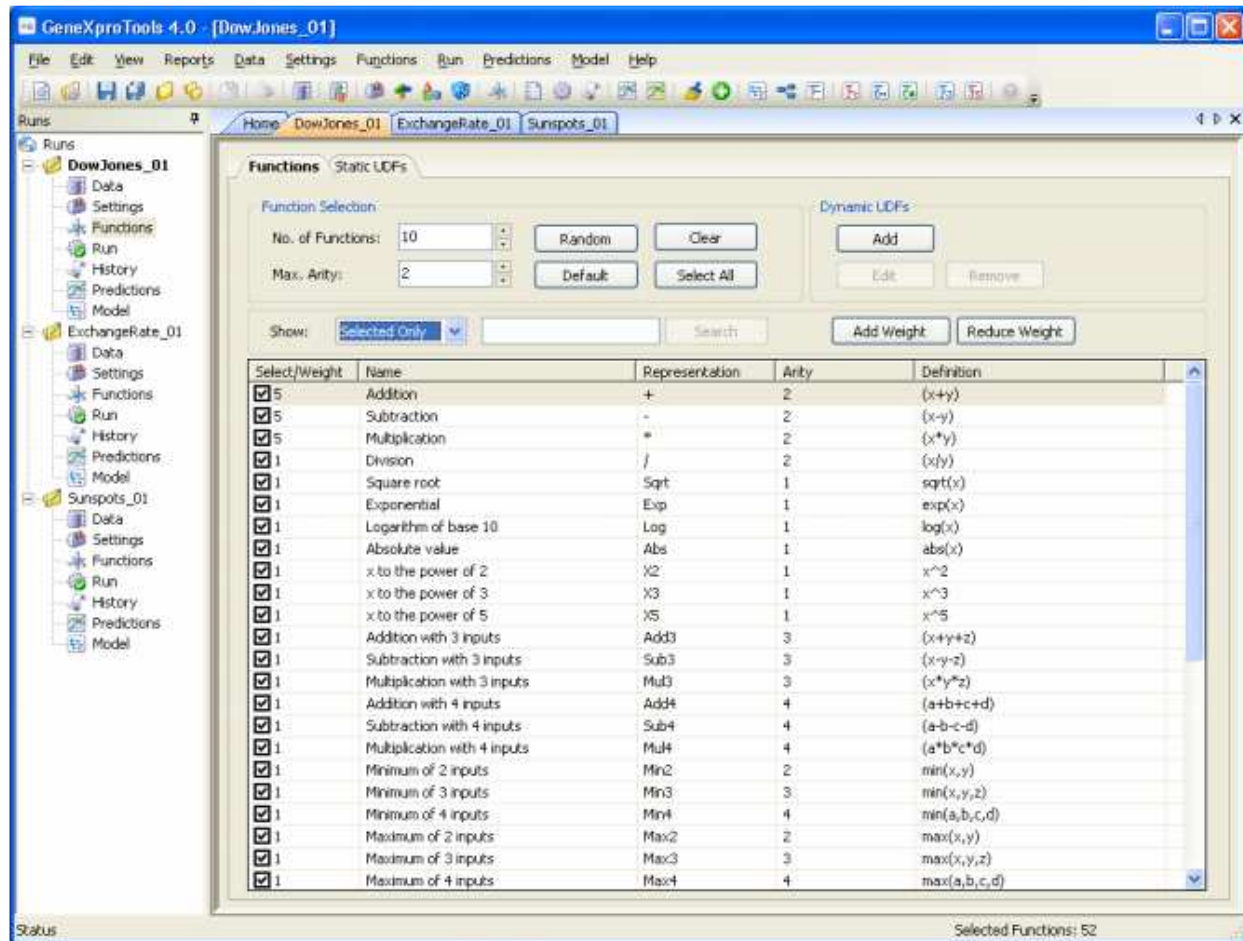


Settings and Features

Functions Panel

The **Functions Panel** of GeneXproTools 4.0 includes the following tabs and windows:

- [Functions Tab](#)
- [Functions Tab – Edit DDF Window](#)
- [Static UDFs Tab](#)
- [Static UDFs Tab – Edit UDF Window](#)



Settings and Features

Functions Tab

The **Functions Tab** lists all the 279 [built-in mathematical functions](#) of GeneXproTools and gives access to the following settings and features:

Functions Static UDFs

Function Selection:

No. of Functions: Random Clear

Max. Arity: Default Select All

Dynamic UDFs: Add Edit Remove

Show: Search Add Weight Reduce Weight

Select/Weight	Name	Representation	Arity	Definition
☒ 4	Addition	+	2	(x+y)
☒ 4	Subtraction	-	2	(x-y)
☒ 4	Multiplication	*	2	(x*y)
☒ 1	Division	/	2	(x/y)
☐ 0	Floating-point remainder	Mod	2	mod(x,y)
☐ 0	Power	Pow	2	pow(x,y)
☒ 1	Square root	Sqrt	1	sqrt(x)
☒ 1	Exponential	Exp	1	exp(x)
☐ 0	10^x	Pow10	1	pow(10,x)
☒ 1	Natural logarithm	Ln	1	ln(x)
☐ 0	Logarithm of base 10	Log	1	log(x)
☐ 0	Logarithm(x,y)	Log2	2	log(x,y)
☐ 0	Floor	Floor	1	floor(x)
☐ 0	Ceiling	Ceil	1	ceil(x)
☐ 0	Absolute value	Abs	1	abs(x)
☐ 0	Inverse	Inv	1	1/x
☐ 0	Negation	Neg	1	-x
☐ 0	No operation	Nop	1	x
☒ 1	x to the power of 2	X2	1	x^2
☒ 1	x to the power of 3	X3	1	x^3
☐ 0	x to the power of 4	X4	1	x^4
☐ 0	x to the power of 5	X5	1	x^5
☒ 1	Cube root	3Rt	1	x^(1/3)

Function Selection Framework

Allows the selection of different function sets very quickly through the combination of the Show options with the Random/Default/Clear/Select All buttons plus the Increase/Reduce Weight buttons.

Number of Functions

The number of functions with a certain arity you wish to select randomly by using the Random button.

Maximum Arity

Determines the sub-set of functions from which the Random button will draw a specified number of functions. For instance, by setting maximum arity to 1, only functions of one argument will be drawn; by setting maximum arity to 2, only functions of 1 and 2 arguments will be randomly chosen, and so forth.

Random Button

Selects randomly a specified number of functions belonging to four different sub-sets of functions: functions of just one argument, functions of 1-2 arguments, functions of 1-3 arguments, and functions of 1-4 arguments.

Default Button

Adds the function set of the [run template](#) to the current function set. To revert exactly to the default function set you must press the Clear button before pressing Default.

Clear Button

Clears all the functions from the function set.

Select All Button

Selects all the functions showing currently on the table.

Show List Box

Shows a wide variety of function sub-sets (All functions, Searched functions, Selected Only, Arity 1 Only, Arity 2 Only, Arity 3 Only, Arity 4 Only, and DDFs Only). By selecting Search, the search box

becomes activated and you can enter specific words to find functions that match your criteria in the Name column.

Search Button

Searches, in the Name column, for the functions matching your search words.

Add Weight

Increments by one the weight of each selected function.

Reduce Weight

Reduces by one the weight of each selected function; functions with weight 1, though, remain as they are.

Select/Weight Column

To select a function, check the box on the left. By doing this, the function is included in your function set with the default weight of 1. To increase its weight, click the box below Select/Weight and then introduce its weight.

Name Column

Under this heading is shown the common name of all the [mathematical functions](#) used by GeneXproTools.

Representation Column

Under this heading is shown how all the functions are represented in [Karva notation](#).

Arity Column

Under this heading is shown the arity (number of arguments) of all the functions.

Definition Column

Under this heading is shown the mathematical definition of all GeneXproTools functions and the definition you gave to all the [DDFs](#).

Dynamic UDFs Framework

Allows the introduction of [dynamic UDFs](#) in your function set.

Add Button

Opens the [Edit DDF window](#) for designing a new [DDF](#).

Edit Button

Opens the [Edit DDF window](#) for editing an existing [DDF](#). To edit a DDF, you must first select the DDF you want to modify, and then click the Edit button.

Remove Button

Removes an existing [DDF](#). To remove a DDF, you must first select the DDF you want to remove, and then click the Remove button. A less radical way of removing a DDF, consists of unchecking this function or setting its weight to zero. This way you will be able to use it later if you wish.

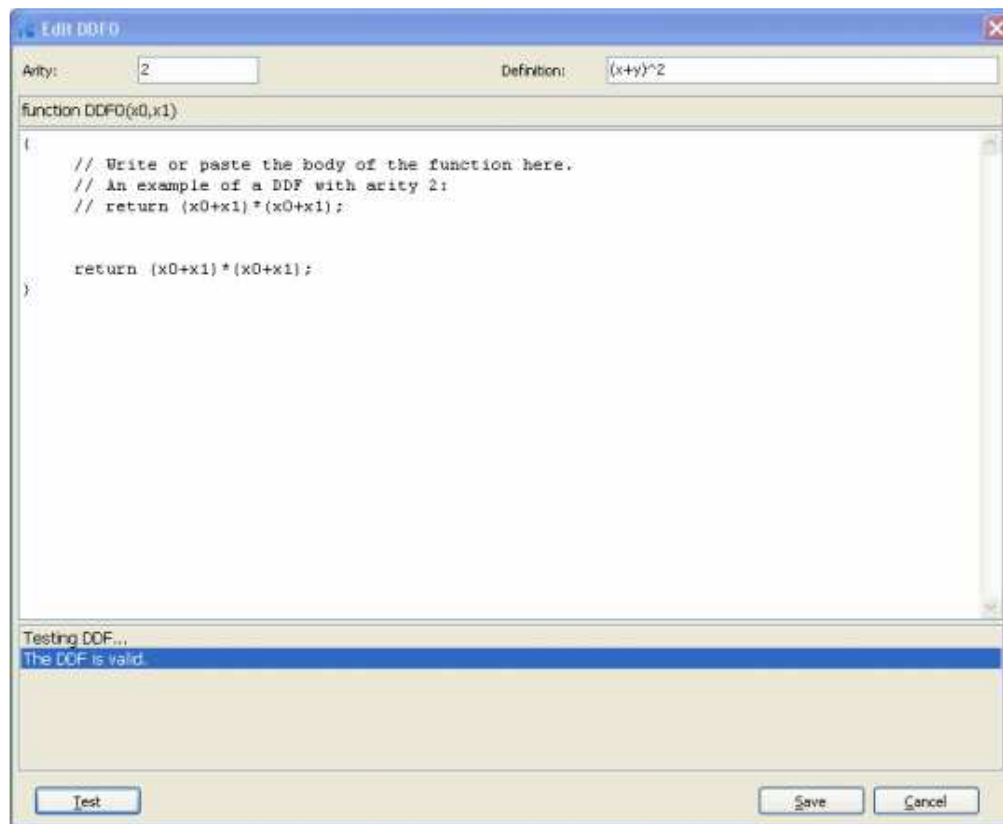


Settings and Features

Functions Tab – Edit DDF Window

The **Edit DDF Window** allows you to introduce your own functions and gives access to the following

settings and features:



Arity

The arity (number of arguments) of your [DDF](#). In GeneXproTools 4.0, the maximum allowed value is 4, and the minimum is obviously 1. After writing the arity, the function header appears in the gray box above the code window.

Definition

Use this box to write a brief description of the [DDF](#) for your future reference. The text you write here will appear in the Definition column on the [Functions Panel](#).

DDF Code Window

You design or edit the code of your own functions ([DDFs](#)) in the code window. The code must be in Java Script and can be tested for syntax errors before including it in the function set.

Test Button

Tests the syntax of the Java Script code of your [DDF](#). The results of the test are written in the gray window below the code window.

Save Button

Saves the [DDF](#) and includes it in your function set with the default weight of 1.

Cancel Button

Closes the Edit DDF window without saving the modifications made there.

Settings and Features

Static UDFs Tab

The **Static UDFs Tab** allows you to design your own [static UDFs](#) and gives access to the following settings and features:

Use UDFs

Check this box if you want to use or design [static UDFs](#).

Function Selection Framework

Allows the selection of different sub-sets of UDFs very quickly through the combination of the Show options with the Random/Clear/Select All buttons plus the Increase/Reduce Weight buttons.

Number of Functions

The number of functions you wish to select randomly by using the Random button.

Maximum Arity

This box is obviously disabled as all UDFs have arity zero.

Random Button

Selects randomly a specified number of functions.

Default Button

This button is obviously disabled as it does not apply to UDFs.

Clear Button

Clears all the functions from the function set.

Select All Button

Selects all the functions showing currently on the table.

Show List Box

Shows three different function sub-sets (All functions, Searched functions, and Selected Only). By selecting Search, the search box becomes activated and you can enter specific words to find functions that match your criteria in the Name column.

Search Button

Searches, in the Name column, for the functions matching your search words.

Add Weight

Increments by one the weight of each selected function.

Reduce Weight

Reduces by one the weight of each selected function; functions with weight 1, though, remain as they are.

Select/Weight Column

To select a [UDF](#), check the box on the left. By doing this, the UDF is included in your function set with the default weight of 1. To increase its weight, click the box below Select/Weight and then introduce its weight.

Name Column

Under this heading is shown the common name of all the [UDFs](#).

Representation Column

Under this heading is shown how all the [UDFs](#) are represented in [Karva notation](#).

Definition Column

Under this heading is shown the definition you gave to all the [UDFs](#).

Static UDFs Framework

Allows the introduction of [static UDFs](#) in your function set.

Add Button

Opens the [Edit UDF window](#) for designing a new [UDF](#).

Edit Button

Opens the [Edit UDF window](#) for editing an existing [UDF](#). To edit a UDF, you must first select the UDF you want to modify, and then click the Edit button.

Remove Button

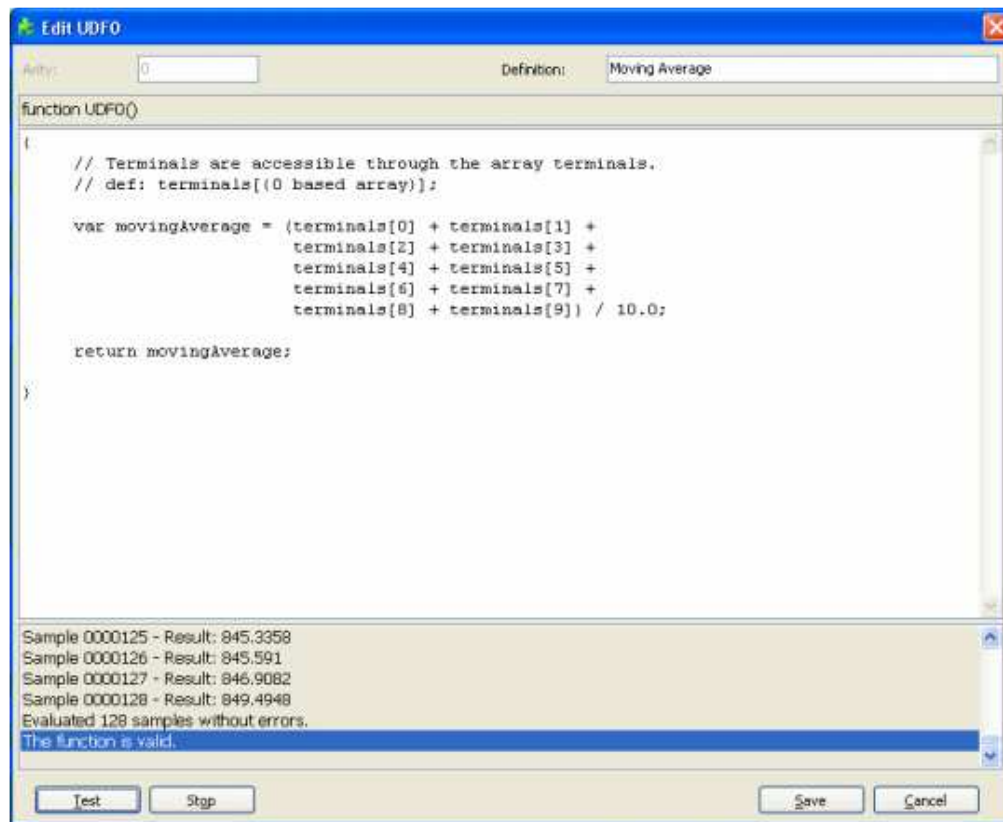
Removes an existing [UDF](#). To remove a UDF, you must first select the UDF you want to remove, and then click the Remove button. A less radical way of removing a UDF, consists of unchecking this function or setting its weight to zero. This way you will be able to use it later if you wish.



Settings and Features

Static UDFs Tab – Edit UDF Window

The **Edit UDF Window** allows you to introduce your own [static UDFs](#) and gives access to the following settings and features:

**Arity**

Reminds you that all the [static UDFs](#) take no parameters and, therefore, have zero arity (notice the function header that appears in the gray box above the code window).

Definition

Use this box to write a brief description of the [UDF](#) for your future reference. The text you write here will appear in the Definition column on the Functions Panel -> Static UDFs.

UDF Code Window

You design or edit the code of your [UDFs](#) in the code window. The code must be in Java Script and can be tested for syntax and calculation errors before including it in the function set.

Test Button

Tests the syntax of the Java Script code of your [UDF](#) and ensures that it is free of calculation errors. The results of the test are written in the gray window below the code window.

Stop Button

Allows you to stop the testing process.

Save Button

Saves the [UDF](#) and includes it in your function set with the default weight of 1.

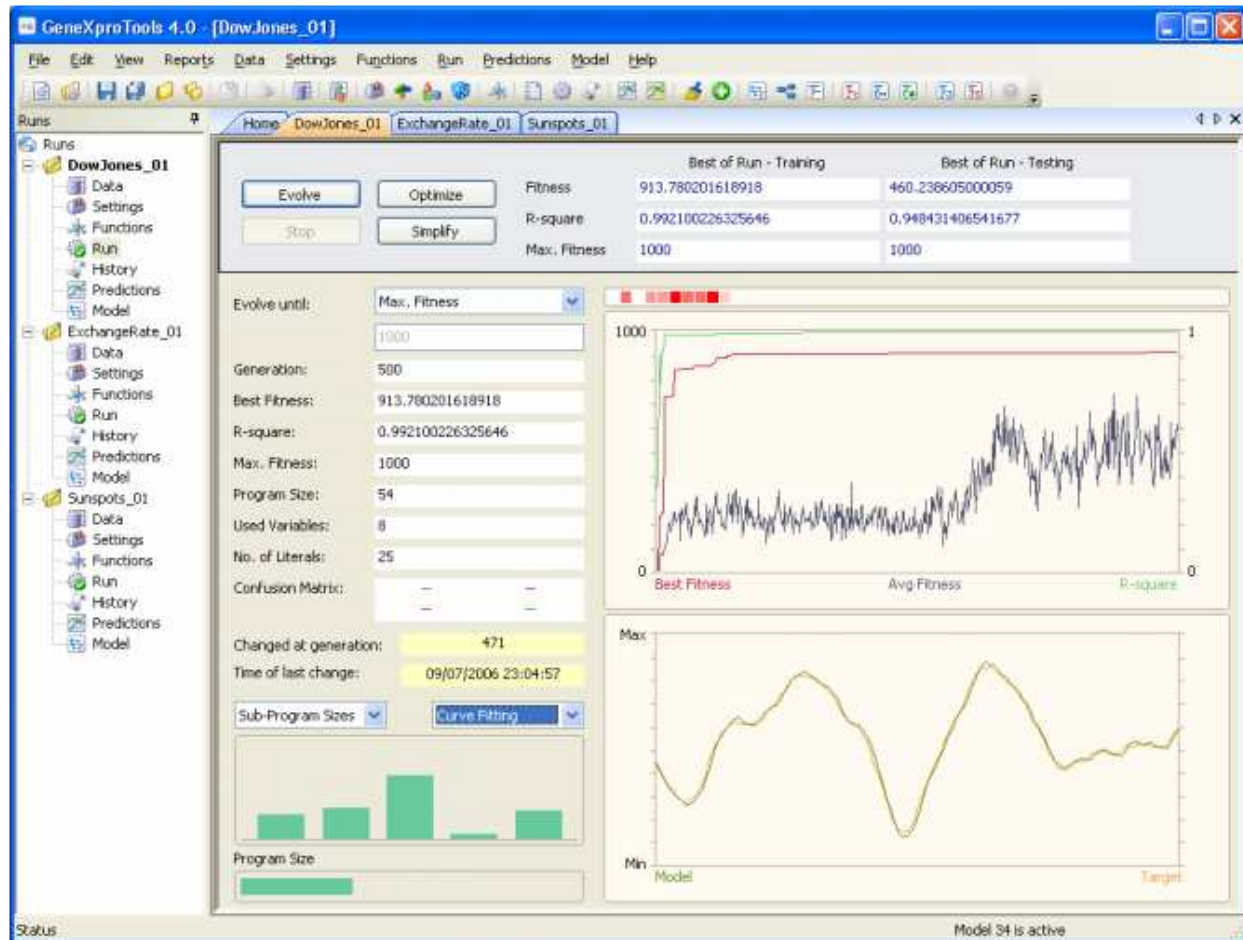
Cancel Button

Closes the Edit UDF window without saving the modifications made there.

Settings and Features

Run Panel

The **Run Panel** allows you to control and monitor the modeling process and gives access to the following settings and features:



Evolve Button

Starts the evolutionary process from scratch, that is, from a completely random initial population.

Stop Button

Stops the evolutionary process.

Optimize Button

Starts the evolutionary process with a [seed model](#), using it to breed better models.

Simplify Button

Also starts an evolutionary process with a seed model, but applies [parsimony pressure](#) to design more compact solutions.

Best of Run – Training

Under this heading are shown the values of [Fitness](#), [R-square](#), and Max. Fitness obtained on the training set for the best model of each run. These values are only updated at the end of each run or between Complexity Increase cycles. They are also updated if some relevant setting is changed, say, a different fitness function, a different training set, or a different model introduced through the Change

Seed window, and if the model is afterwards processed in the [Predictions Panel](#).

Best of Run – Testing

Under this heading are shown the values of [Fitness](#), [R-square](#), and Max. Fitness obtained on the testing set for the best model of each run. These values are only updated at the end of each run or between Complexity Increase cycles. They are also updated if some relevant setting is changed, say, a different fitness function, or a different model introduced through the Change Seed window, and if the model is afterwards processed in the [Predictions Panel](#). When in Prediction Mode, a dash is shown in all the boxes.

Evolve Until

Sets the stop condition. This stop condition is formulated using two different boxes. In the first you choose the stop condition and, in the second, you set the value for the chosen condition. There are four different stop conditions to choose from. The first allows you to evolve models for a certain number of generations. In this case, you must write in the box underneath the number of generations you want. The second allows you to evolve models until the best fitness is equal to or higher than a certain value. In this case, you must write in the box underneath this fitness value. The third allows you to evolve models until the [R-square](#) of the best model is equal to or higher than a certain value. In this case, you must write in the box underneath this R-square value. Obviously, only values higher than 0 and equal to or less than 1 are valid, and if an invalid value is used GeneXproTools uses the value of 1 by default. The fourth allows you to evolve models until the best fitness is equal to maximum fitness. In this case, the correct value of maximum fitness is automatically introduced by GeneXproTools in the box underneath.

Generation

Updates the generation number throughout a run.

Best Fitness

Updates the best fitness of each generation throughout a run.

R-square

Updates the [R-square](#) of the best-of-generation model throughout a run.

Max. Fitness

Shows the value of maximum fitness for the current run.

Program Size

Updates the program size of the best-of-generation model throughout a run.

Used Variables

Updates the number of variables used in the best-of-generation model throughout a run.

No. of Literals

Updates the number of literals used in the best-of-generation model throughout a run.

Confusion Matrix

This box is specific of Classification problems, and therefore only dashes are shown in Time Series Prediction.

Changed at Generation

Updates, throughout a run, the number of the generation at which a change in best fitness occurred.

Time of Last Change

Updates, throughout a run, the time when a change in best fitness occurred.

Charts List Boxes

The one on the left allows you to select three different bar plots: Sub-Program Sizes, All Sizes, and All Fitnesses. And the one on the right allows you to select two different charts: Curve Fitting and Avg/Best Size.

Sub-Program Sizes Chart

This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be visualized any time during evolution by selecting Sub-Program Sizes in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

All Sizes Chart

This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be visualized any time during evolution by selecting All Sizes in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

All Fitnesses Chart

This chart shows the fitnesses of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be visualized any time during evolution by selecting All Fitnesses in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Program Size Chart

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Curve Fitting Chart

This chart plots the first 250 data points of the training set and shows how well the evolving models are fitting the target. The Curve Fitting chart can be visualized any time during evolution by selecting Curve Fitting in the rightmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click. And by placing the cursor over the line, you can access the value of that point. You can also access the values corresponding to the Max and Min labels that sometimes appear on the Y axis by placing the cursor over them.

Avg/Best Size Chart

This chart plots together the average program size of the population and the program size of the best-of-generation model. This chart is especially useful during simplification and can be visualized any time during evolution by selecting Avg/Best Size in the rightmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click. You can also access the values corresponding to the Max and Min labels that sometimes appear on the Y axis by placing the cursor over them.

Variables Usage Map

This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square gives you access to the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic. And as usual, GeneXproTools allows you to copy or print this chart through the mouse right-click.

Evolutionary Dynamics Chart

This chart shows the average fitness of the population plus the [fitness](#) and [R-square](#) of the best-of-generation model. The Y axis on the left refers to best and average fitness, whereas the Y axis on the right refers to the [R-square](#). The X axis corresponds obviously to the number of generations, with a complete screen spanning a total of 500 generations. For a good adaptation, the plot for average fitness should never come near the plot for best fitness, otherwise the system is losing genetic diversity and becoming too uniform for an efficient evolution. Whenever this is happening in your

system, you should obviously increase the [genetic diversity](#) of the evolving populations by choosing the right set of genetic operators and adjusting their rates. GeneXproTools allows you to copy or print this chart through the mouse right-click.



Settings and Features

History Panel

The **History Panel** lists all the best-of-generation models created in a run and gives access to the following settings and features:

Test All Button

Evaluates the performance of all the models in the History Panel on the testing set, showing their respective [fitnesses](#) and [R-squares](#). When in Prediction Mode, then this button is disabled.

Test Current Button

Evaluates the performance of the selected model on the testing set, showing its [fitness](#) and [R-square](#). When in Prediction Mode, then this button is disabled.

Refresh All Button

Evaluates the performance of all the models in the History Panel both on the training and testing sets, and updates all the [fitnesses](#) and [R-squares](#). For instance, this button is particularly useful when you have changed the fitness function after a run and would want to evaluate the performance on the training set with this new fitness function.

Best of Run – Training

This heading is used to identify the best models of the run on the training set in terms of [fitness](#) and [R-square](#), showing also the exact values of their fitness and R-square. For reference, the value of Max. Fitness on the training set is also shown.

Best of Run – Testing

This heading is used to identify the best models of the run on the testing set in terms of [fitness](#) and [R-square](#), showing also the exact values of their fitness and R-square. For reference, the value of Max. Fitness on the testing set is also shown. When in Prediction Mode, a dash is shown in all the boxes under this heading.

Model Column

In this column are shown the names of all the best-of-generation models created in the run. By clicking a model, you are making it your active model. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order.

Generation Column

In this column is shown the number of the generation in which the model was created. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order.

Program Size Column

In this column is shown the program size of all the best-of-generation models. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their program sizes.

Training Fitness Column

In this column is shown the [fitness](#) obtained by each model on the training set. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their training fitnesses.

Testing Fitness Column

In this column is shown the [fitness](#) obtained by each model on the testing set. Note that, after a run, only the fitness of the best-of-run model is shown; all the others are set to zero. In order to update these values, you must either click on the Test All or Test Current buttons. When in Prediction Mode, though, a dash is shown in all the entries under this heading. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their testing fitnesses.

Training R-square Column

In this column is shown the [R-square](#) obtained by each model on the training set. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their training R-squares.

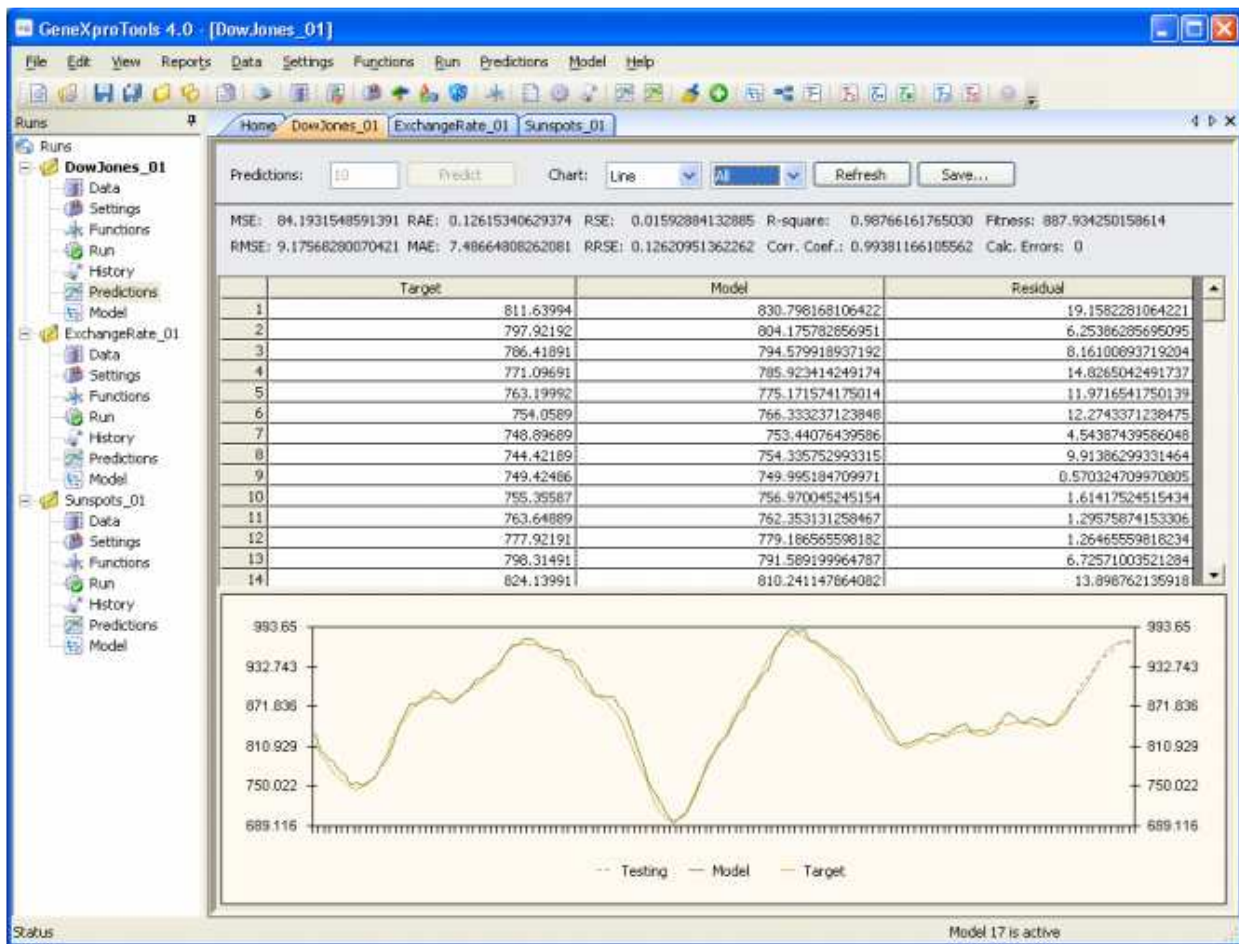
Testing R-square Column

In this column is shown the [R-square](#) obtained by each model on the testing set. Note that, after a run, only the R-square of the best-of-run model is shown; all the others are set to zero. In order to update these values, you must either click on the Test All or Test Current buttons. When in Prediction Mode, though, a dash is shown in all the entries under this heading. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their testing R-squares.

Settings and Features

Predictions Panel

The **Predictions Panel** allows you to visualize and evaluate the accuracy of your models and gives access to the following settings and features:



Predictions

The number of recursive predictions you want to make with the evolved model. The maximum number allowed is 9,999. When you are working on Testing Mode, this box is disabled.

Predict

Press this button to make the recursive predictions. The results are immediately written to the table and chart of this panel. When you are working on Testing Mode, this button is disabled.

Chart Type Box

In this combo box you select the chart type: Line, Area, Bar, Step, 3D Line, 3D Area, 3D Bar, or 3D Step.

Chart Data Box

In this combo box you select the data you want to plot: Model, Target, Testing, or All if you are in Testing Mode; or Model, Target, Prediction, or All if you are in Prediction Mode. In case of calculation errors, the sample with the error is skipped and a discontinuity is observed in the plot. The plots show

only a maximum of 250 points at a time and, therefore, for datasets with more than 250 samples you must move the scroll bar at the bottom of the chart to see all the results. Then, by placing the cursor over the chart, GeneXproTools shows you the exact range used to draw the plot.

Refresh

Refreshes the panel. This button is particularly useful when you change your model in the [Change Seed window](#).

Save

Saves all the results in the table to a file.

Statistics

By selecting Testing in the Chart Data box, you have access to a wide set of [statistical evaluations](#) of your model performed on the recursive testing. By selecting any of the other options in the Chart Data box (Model, Target, Prediction, or All), the statistics shown in the panel refer to the results obtained on the training set.

MSE

Shows the value of the [Mean Squared Error](#) on the current set.

RMSE

Shows the value of the [Root Mean Squared Error](#) on the current set.

RAE

Shows the value of the [Relative Absolute Error](#) on the current set.

MAE

Shows the value of the [Mean Absolute Error](#) on the current set.

RSE

Shows the value of the [Relative Squared Error](#) on the current set.

RRSE

Shows the value of the [Root Relative Squared Error](#) on the current set.

Corr. Coef.

Shows the value of the [Correlation Coefficient](#) on the current set. This parameter is not shown when the fitness function is [Relative With SR](#), [Relative/Hits](#), [Absolute With SR](#), or [Absolute/Hits](#). In this case the number of outliers is shown instead.

Outliers

Shows the number of outliers in the current set. This parameter refers only to the fitness functions [Relative With SR](#), [Relative/Hits](#), [Absolute With SR](#), or [Absolute/Hits](#). The rows with the outliers are easily identified by a yellow strip.

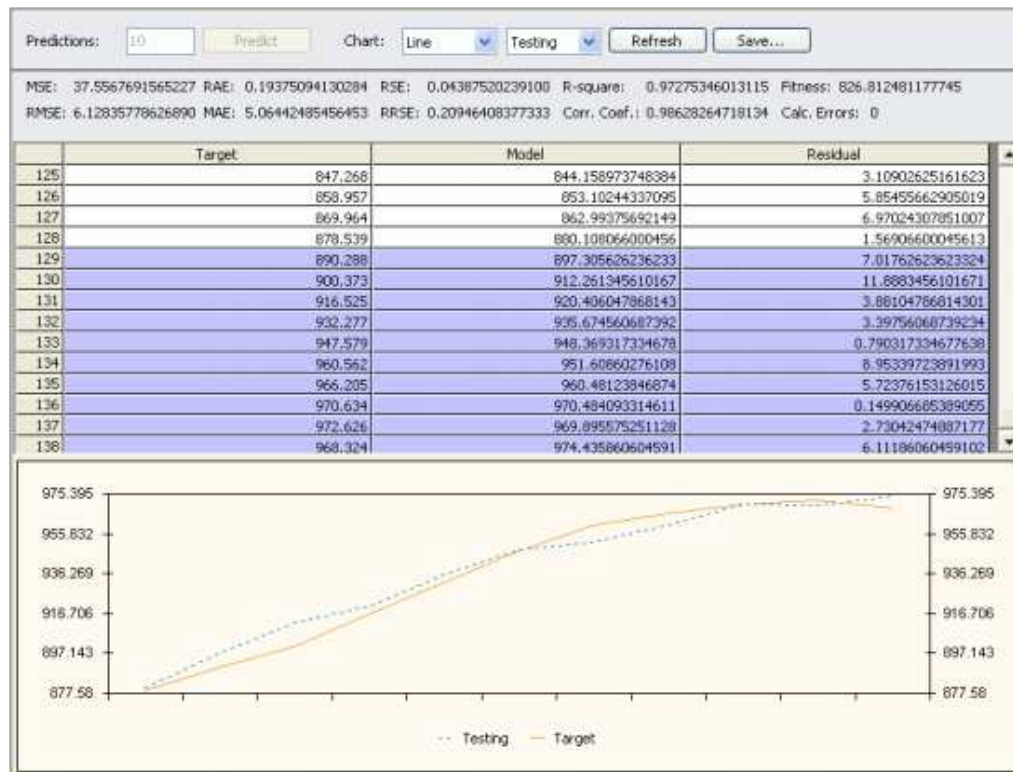
Calc. Errors

Shows the number of calculation errors in the current set. Note that, unless you've changed either the training set or the model after a run, no calculation errors should appear in the training set as models returning calculation errors during training have zero fitness and are, therefore, excluded. However, on the testing set or while making predictions, calculation errors might occur and you can see straightaway how many times in the Calculation Errors box. In addition, the rows with calculation errors are easily identified by a red strip and the identifier ERROR. In the charts, you can also easily spot the points returning calculation errors as they are not drawn, creating a discontinuity in the plots.

Table

The table allows you to compare the output of your model with the target output on the training set. If you are in Testing Mode, it also allows you to compare the results of your model on the recursive testing with the target output. The results obtained on the recursive testing are shown immediately

after the results obtained on the training set and are highlighted in blue. And if you are in Prediction Mode, it shows you the recursive predictions made with your model. The recursive predictions are shown immediately after the results obtained on the training set and are also highlighted in blue.



Target

Under this heading is shown the value of the dependent variable (Target) for all the records.

Model

Under this heading is shown the output of the model (Model) for all the records. In case of calculation errors, a red strip and the identifier ERROR is used.

Residual

Under this heading is shown the absolute value of the difference between the Target and the Model.

This heading has another name (Relative Error) when the fitness function is either [Relative With SR](#) or [Relative/Hits](#).

Relative Error

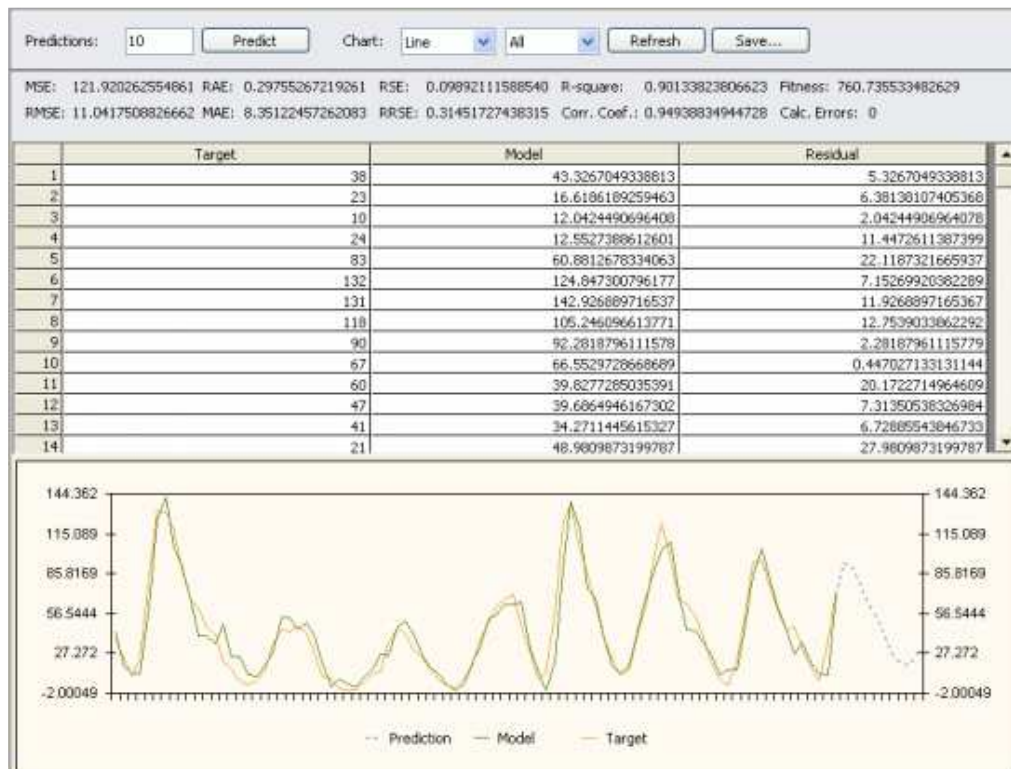
Under this heading is shown the absolute value of the difference between the Target and the Model.

This heading has another name (Residual) when the fitness function is other than [Relative With SR](#) or [Relative/Hits](#). In case of divisions by zero during the evaluation of the relative error, a green strip and the identifier UNDEFINED is used.

Chart

The charts allow you to compare the output of your model with the target output on the training set using different curve fitting charts. If you are in Testing Mode, they also allow you to compare the results of your model on the recursive testing with the target output. And if you are in Prediction Mode, they allow you to analyze the recursive predictions made with your model. In addition, you may choose to observe All the plots together, or you may choose to plot the Model, the Target, the Testing or the Prediction separately. The plots show only a maximum of 250 points at a time and, therefore, for more than 250 points you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the chart, GeneXproTools

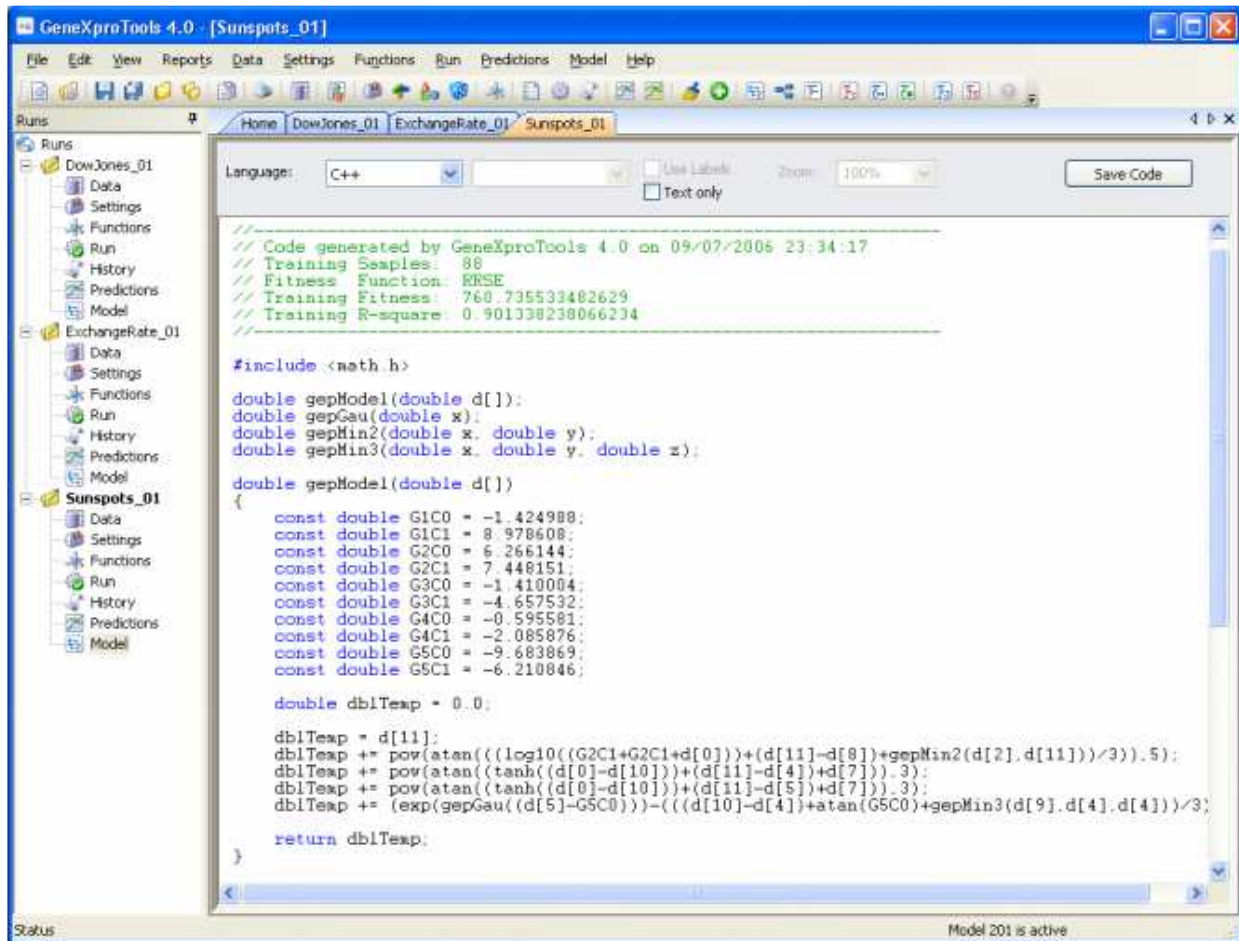
shows you the exact range used to draw the plot. All the plots can be copied to the clipboard by clicking in the plot area and pressing the right button of the mouse.



Settings and Features

Model Panel

The **Model Panel** allows you to see the structure of your models and translate them into a wide range of [programming languages](#). It gives access to the following settings and features:



Language

In the list box on the left, you can choose different ways of representing your models, including the native [Karva code](#), a universal diagram representation ([Expression Tree](#)), 15 different [programming languages](#) (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, and VHDL) originally packaged with GeneXproTools 4.0, and any additional languages created by you through the use of [User Defined Grammars](#). The list box on the right is only activated when Karva is selected, and allows you to choose either [Genes](#) or [K-expressions](#). In both cases, though, the elements in the [heads](#) are shown in blue.

Use Labels

This option is disabled in Time Series Prediction.

Text Only

By checking this box the default HTML is replaced by simple unformatted text.

Zoom

This option is only active when you are visualizing Expression Trees. It allows you to either choose pre-set values to resize the expression trees (200%, 150%, 100%, 85%, 75%, 50%, and 25%) or enter any other value up to 500%.

Save Code

Saves the selected code to an appropriate format: [Karva code](#) is saved as .txt, [Ada](#) code as .ada, [C](#) code as .c, [C++](#) code as .hpp, [C#](#) code as .cs, [Fortran](#) code as .f, [Java](#) code as .java, [Java Script](#) code as .js, [Matlab](#) code as .m, [Pascal](#) code as .pas, [Perl](#) code as .pl, [PHP](#) code as .php, [Python](#) code as .py, [Visual Basic](#) code as .bas, [VB.Net](#) code as .vb, and [VHDL](#) code as .vhd.

Save Image

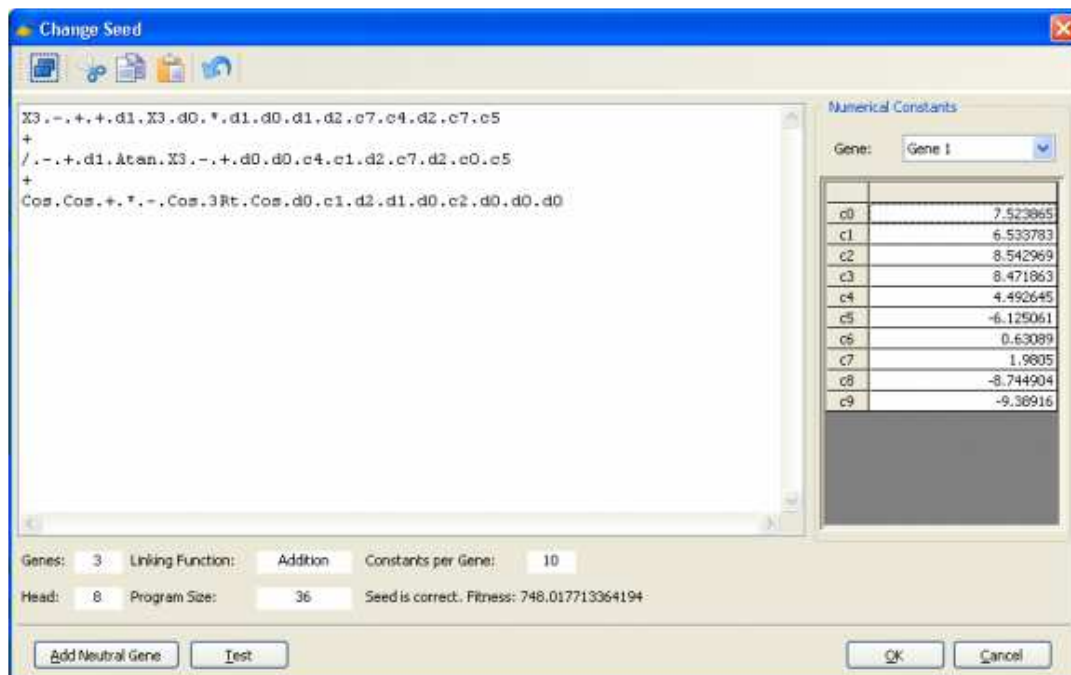
Saves the [Expression Trees](#) as .bmp files.



Settings and Features

Change Seed Window

The **Change Seed Window** allows you to change your model or introduce a completely new one. It gives access to the following settings and features:

**Seed Code Window**

You design or edit the code of your seed model in the code window. The code must be in [Karva notation](#) and can be tested for structural errors before saving it as seed.

Genes

Shows the current [number of genes](#) in your seed model.

Head

Shows the [head](#) size of your seed model.

Linking Function

Shows the current [linking function](#) in your seed model.

Program Size

Shows the actual size of your seed model, that is, the total number of nodes in all the [sub-ETs](#).

Constants per Gene

This box only appears if [random numerical constants](#) are being used. It shows the number of random

numerical constants used per gene in your seed model.

Numerical Constants Framework

This frame only appears if [random numerical constants](#) are being used. Here you have access to all the numerical constants of your seed model by selecting the gene number in the Gene list box. The numerical constants appear in a table where you can edit them. Only values within the range of -1.79769313486231E308 to -4.94065645841247E-324 for negative values, 4.94065645841247E-324 to 1.79769313486231E308 for positive values, and 0 are acceptable.

Add Neutral Gene Button

Adds automatically a neutral gene to your seed model. Neutrality is important for an efficient evolution and, by introducing a neutral gene, you are giving the [learning algorithm](#) more room to play and, hopefully, a better, more complex program will evolve.

Test Button

Tests both the structural soundness and the [fitness](#) of your seed model. It also updates the program size when changes have been made on the seed.

OK Button

Saves the current seed as your seed model so that you can breed new and better models with it. GeneXproTools only saves viable seeds and warns you if there is a structural problem with your seed or if it has zero fitness.

Cancel Button

Cancels all the action on the change seed window.

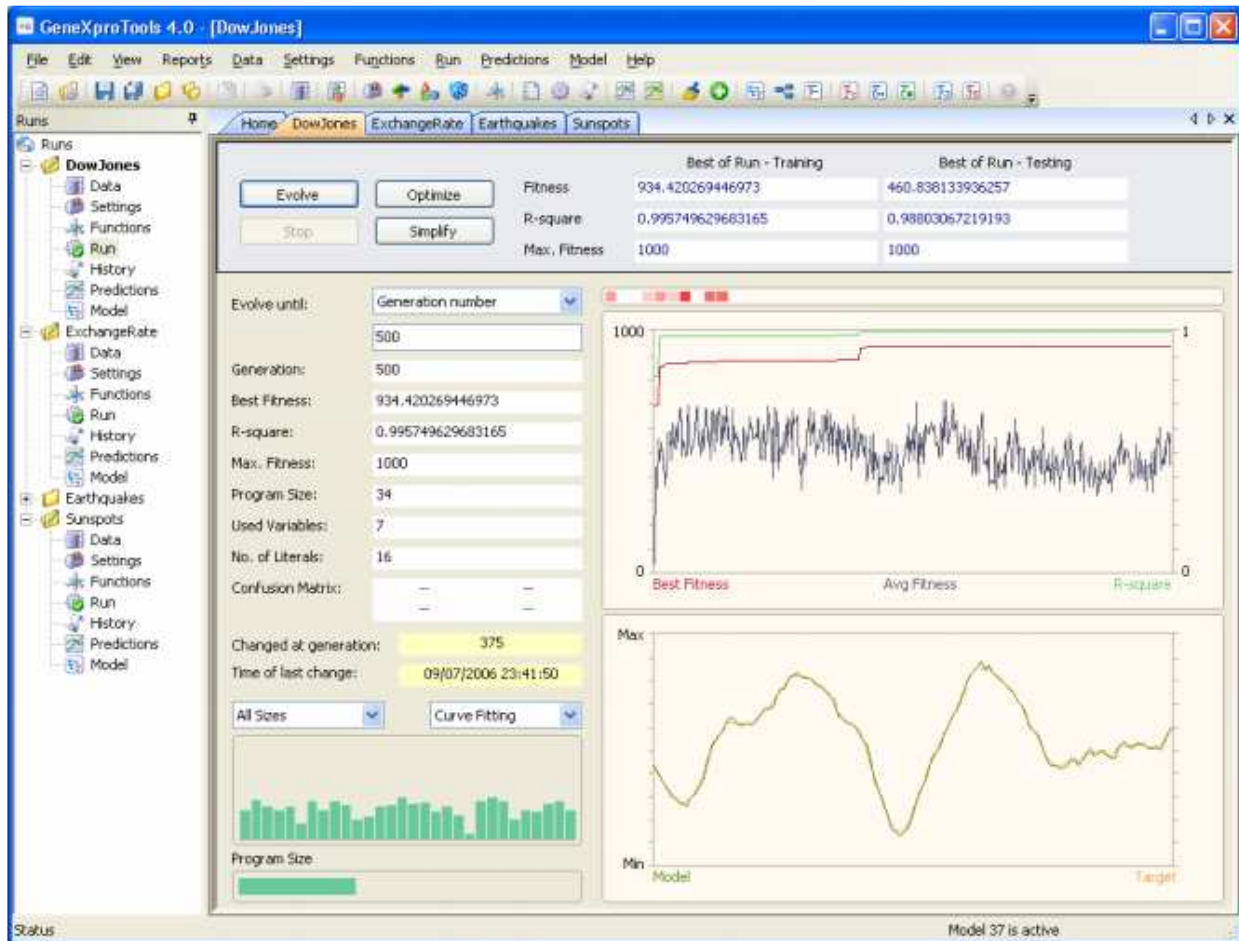


Settings and Features

Menus

For Time Series Prediction problems, GeneXproTools has the following menus:

- [File](#)
- [Edit](#)
- [View](#)
- [Reports](#)
- [Data](#)
- [Settings](#)
- [Functions](#)
- [Run](#)
- [Predictions](#)
- [Model](#)
- [Help](#)



◀ | ▶

Settings and Features

File

The **File Menu** gives access to the following sub-menus:

New

Creates a new gep file.

Open

Opens a gep file.

Save

Saves the selected run.

Save As

Saves a gep file with a different name.

Save All

Saves all open runs.

Close

Closes the selected run.

Close All

Closes all open runs.

Print

Prints the report, history, code, tree, or charts of the run.

Properties

Loads the identification form for input (Author Name and Comments).

Exit

Closes GeneXproTools.



Settings and Features

Edit

The **Edit Menu** gives access to the following sub-menus:

Copy

Copies the report, tables, code, tree, or charts of the run for pasting elsewhere.

Change Seed

Opens the [Change Seed window](#) for changing exogenous or GeneXproTools evolved models for further optimization.

Add Neutral Gene

Adds automatically a neutral gene to your seed model.

Add DDF

Opens the [Edit DDF window](#) for designing a new [DDF](#).

Add UDF

Opens the [Edit UDF window](#) for designing a new [UDF](#).

Custom Fitness Function

Goes to the [Custom Fitness Function Code window](#) for editing or designing a new [custom fitness function](#).



Settings and Features

View

The **View Menu** gives access to the following sub-menus:

Status Bar

Shows/Hides the Status Bar.

Warnings

Turns on/off warning messages.



Settings and Features

Reports

The **Reports Menu** gives access to the following sub-menus:

Home

Goes to the welcome screen of GeneXproTools.

Run Report

Goes to the [Report Panel](#) of the active run.



Settings and Features

Data

The **Data Menu** gives access to the following sub-menus:

Training Data

Goes to the [Transformed Time Series Tab](#) of the selected run.

Time Series

Goes to the [Original Time Series Tab](#) of the selected run.

Change Time Series

Opens the [Change Time Series Window](#) for changing the time series of the selected run.



Settings and Features

Settings

The **Settings Menu** gives access to the following sub-menus:

General Settings

Goes to the [General Settings Tab](#) of the selected run.

Fitness Function

Goes to the [Fitness Function Tab](#) of the selected run.

Genetic Operators

Goes to the [Genetic Operators Tab](#) of the selected run.

Numerical Constants

Goes to the [Numerical Constants Tab](#) of the selected run.



Settings and Features

Functions

The **Functions Menu** gives access to the following sub-menus:

Functions

Goes to the [Functions Tab](#) of the selected run.

Static UDFs

Goes to the [Static UDFs Tab](#) of the selected run.



Settings and Features

Run

The **Run Menu** gives access to the following sub-menus:

Run

Goes to the [Run Panel](#) of the active run.

Evolve

Starts a new run.

Optimize

Starts an optimization process.

Simplify

Starts a simplification process.

Variables Usage

Allows the selection of three different kinds of mapping variables usage during a run: Heat Map, Random Colors, or Monochromatic, and two different showing options: White for None and Grey for

None.

Charts

Allows the selection of five different kinds of charts during a run: Curve Fitting, Avg/Best Size, Sub-Program Sizes, All Sizes, and All Fitnesses.



Settings and Features

Predictions

The **Predictions Menu** gives access to the following sub-menus:

History

Goes to the [History Panel](#) of the active run.

Training Results

Goes to the [Predictions Panel -> All Option](#) of the active run.

Predictions

Goes to the [Predictions Panel -> Prediction Option](#) of the active run.

Recursive Testing

Goes to the [Predictions Panel -> Testing Option](#) of the active run.



Settings and Features

Model

The **Model Menu** gives access to the following sub-menus:

Expression Tree

Shows the active model as an [Expression Tree](#).

Karva

Shows the active model in the native [Karva Notation](#).

Ada

Translates the active model into [Ada](#).

C

Translates the active model into [C code](#).

C++

Translates the active model into [C++](#).

C#

Translates the active model into [C#](#).

Fortran

Translates the active model into [Fortran](#).

Java

Translates the active model into [Java](#).

JavaScript

Translates the active model into [JavaScript](#).

Matlab

Translates the active model into [Matlab](#).

Pascal

Translates the active model into [Pascal](#).

Perl

Translates the active model into [Perl](#).

PHP

Translates the active model into [PHP](#).

Python

Translates the active model into [Python](#).

Visual Basic

Translates the active model into [Visual Basic](#).

VB.Net

Translates the active model into [VB.Net](#).

VHDL

Translates the active model into [VHDL](#).



Settings and Features

Help

The **Help Menu** gives access to the following sub-menus:

Contents

Shows the contents of the Help File.

About GeneXproTools

Shows information about GeneXproTools.



Settings and Features

Run Templates

For Time Series Prediction problems, GeneXproTools 4.0 offers five different **Run Templates** designed so that you can start the modeling process with just a mouse click and yet obtain excellent results. In addition, they also make the discovery of good models quite enjoyable and fast as small populations of just 30 models are used in all of them.

The run templates of GeneXproTools are essentially based on the number of independent variables in your data and are organized as follows (in the Function Set, weights greater than 1 are indicated in parentheses):

Template A

Independent Variables (Embedding Dimension): 1-5

Fitness Function: RRSE

Function Set: addition (5), subtraction (5), multiplication (5), division, Add3, Sub3, Mul3, Add4, Sub4, Mul4, Sqrt, Log, Exp, Abs, x^2 , x^3 , x^5 , Logistic, Gaussian, Sin, Cos, Cot, Tan, Asin, Atan, Acsc, Tanh, Csch, Sech, Pi, NOT, Min2, Min3, Min4, Max2, Max3, Max4, Avg2, Avg3, and Avg4.

Chromosomes: 30

Genes: 4

Head Size: 7

Linking Function: addition

Random Numerical Constants: Yes

Constants per Gene: 2

Data Type: floating-point

Lower Bound: -10

Upper Bound: 10

Complexity Increase: No

Generations without Change: 2000

Tries: 3

Maximum Complexity: 9

Mutation Rate: 0.044

Inversion Rate: 0.1

IS Transposition Rate: 0.1

RIS Transposition Rate: 0.1

One-Point Recombination Rate: 0.3

Two-Point Recombination Rate: 0.3

Gene Recombination Rate: 0.1

Gene Transposition Rate: 0.1

RNC Mutation Rate: 0.044

Dc Mutation Rate: 0.1

Dc Inversion Rate: 0.1

Dc IS Transposition Rate: 0.1

Template B

Independent Variables (Embedding Dimension): 6-10

Fitness Function: RRSE

Function Set: addition (5), subtraction (5), multiplication (5), division, Add3, Sub3, Mul3, Add4, Sub4, Mul4, Sqrt, Log, Exp, Abs, x^2 , x^3 , x^5 , Logistic, Gaussian, Sin, Cos, Cot, Tan, Asin, Atan, Acsc, Tanh, Csch, Sech, Pi, NOT, Min2, Min3, Min4, Max2, Max3, Max4, Avg2, Avg3, and Avg4.

Chromosomes: 30

Genes: 5

Head Size: 8

Linking Function: addition

Random Numerical Constants: Yes

Constants per Gene: 2

Data Type: floating-point

Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 10
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template C

Independent Variables (Embedding Dimension): 11-32
Fitness Function: RRSE
Function Set: addition (5), subtraction (5), multiplication (5), division, Add3, Sub3, Mul3, Add4, Sub4, Mul4, Sqrt, Log, Exp, Abs, x^2 , x^3 , x^5 , Logistic, Gaussian, Sin, Cos, Cot, Tan, Asin, Atan, Acsc, Tanh, Csch, Sech, Pi, NOT, Min2, Min3, Min4, Max2, Max3, Max4, Avg2, Avg3, and Avg4.
Chromosomes: 30
Genes: 5
Head Size: 10
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 10
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template D

Independent Variables (Embedding Dimension): 33-64
Fitness Function: RRSE
Function Set: addition (5), subtraction (5), multiplication (5), division, Add3, Sub3, Mul3, Add4, Sub4, Mul4, Sqrt, Log, Exp, Abs, x^2 , x^3 , x^5 , Logistic, Gaussian, Sin, Cos, Cot, Tan, Asin, Atan, Acsc, Tanh, Csch, Sech, Pi, NOT, Min2, Min3, Min4, Max2, Max3, Max4, Avg2, Avg3, and Avg4.
Chromosomes: 30

Genes: 6
Head Size: 12
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 10
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

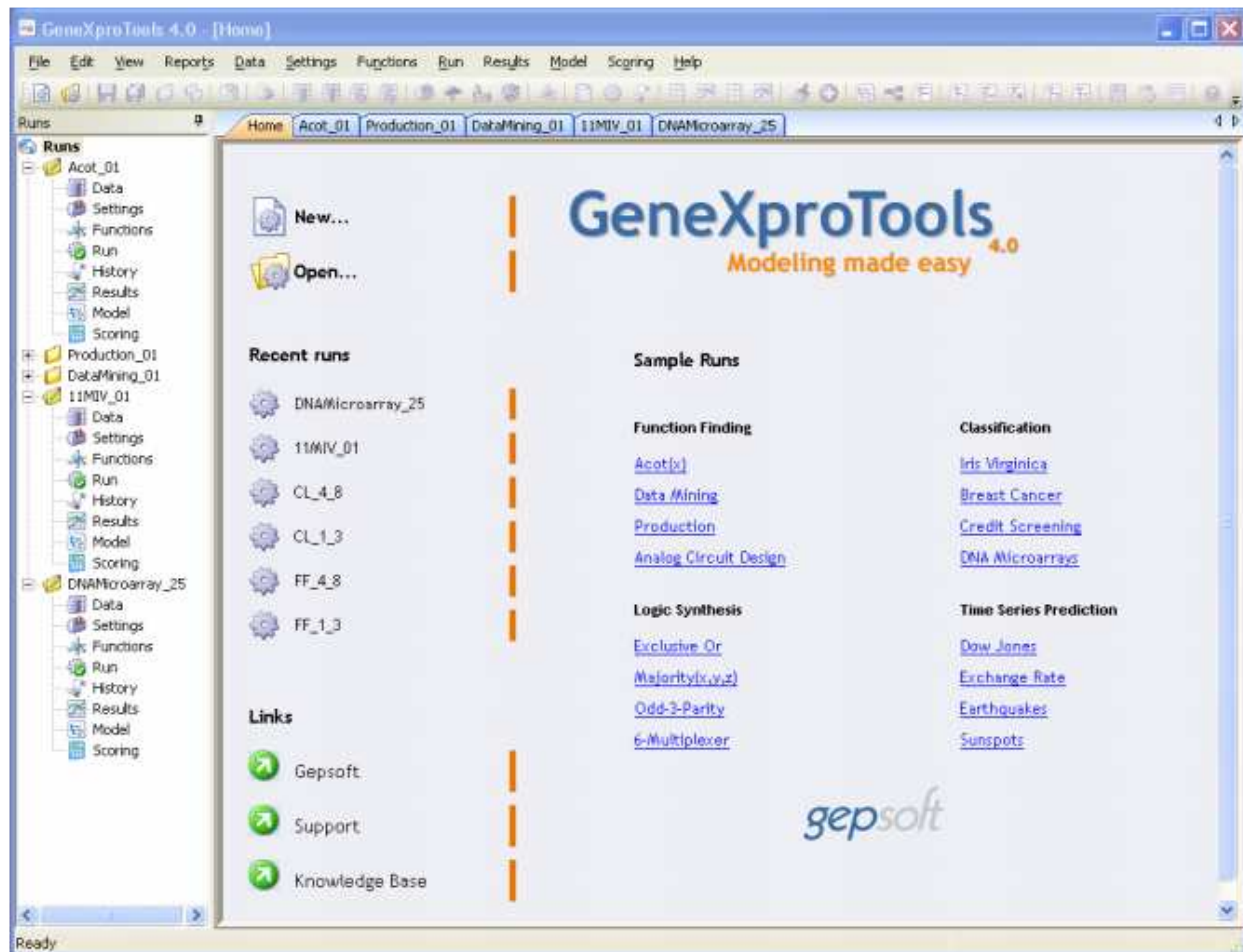
Template E

Independent Variables (Embedding Dimension): 65-Unlimited
Fitness Function: RRSE
Function Set: addition (5), subtraction (5), multiplication (5), division, Add3, Sub3, Mul3, Add4, Sub4, Mul4, Sqrt, Log, Exp, Abs, x^2 , x^3 , x^5 , Logistic, Gaussian, Sin, Cos, Cot, Tan, Asin, Atan, Acsc, Tanh, Csch, Sech, Pi, NOT, Min2, Min3, Min4, Max2, Max3, Max4, Avg2, Avg3, and Avg4.
Chromosomes: 30
Genes: 6
Head Size: 15
Linking Function: addition
Random Numerical Constants: Yes
Constants per Gene: 2
Data Type: floating-point
Lower Bound: -10
Upper Bound: 10
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 10
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Settings and Features

Sample Runs

GeneXproTools 4.0 ships with four different **Sample Runs** for Time Series Prediction: [Dow Jones](#), [Exchange Rate](#), [Earthquakes](#), and [Sunsports](#). To try any one of them, you just have to click its link on the Welcome Screen of GeneXproTools.



All these four sample runs use real-world data and you will see that, like with all complex phenomena, it takes some cunning to design very good models: some times good results can be obtained using raw data, but other times by smoothing the data you can obtain better results. In three of the sample runs ([Dow Jones](#), [Exchange Rate](#), and [Earthquakes](#)) an integral smoothing algorithm was used, and if you compare both the plots and behavior of these time series with the [Sunsports](#) series (in this case, the original raw data was used), you will see that, as a rule, it is much easier to create good models using smoothed data.

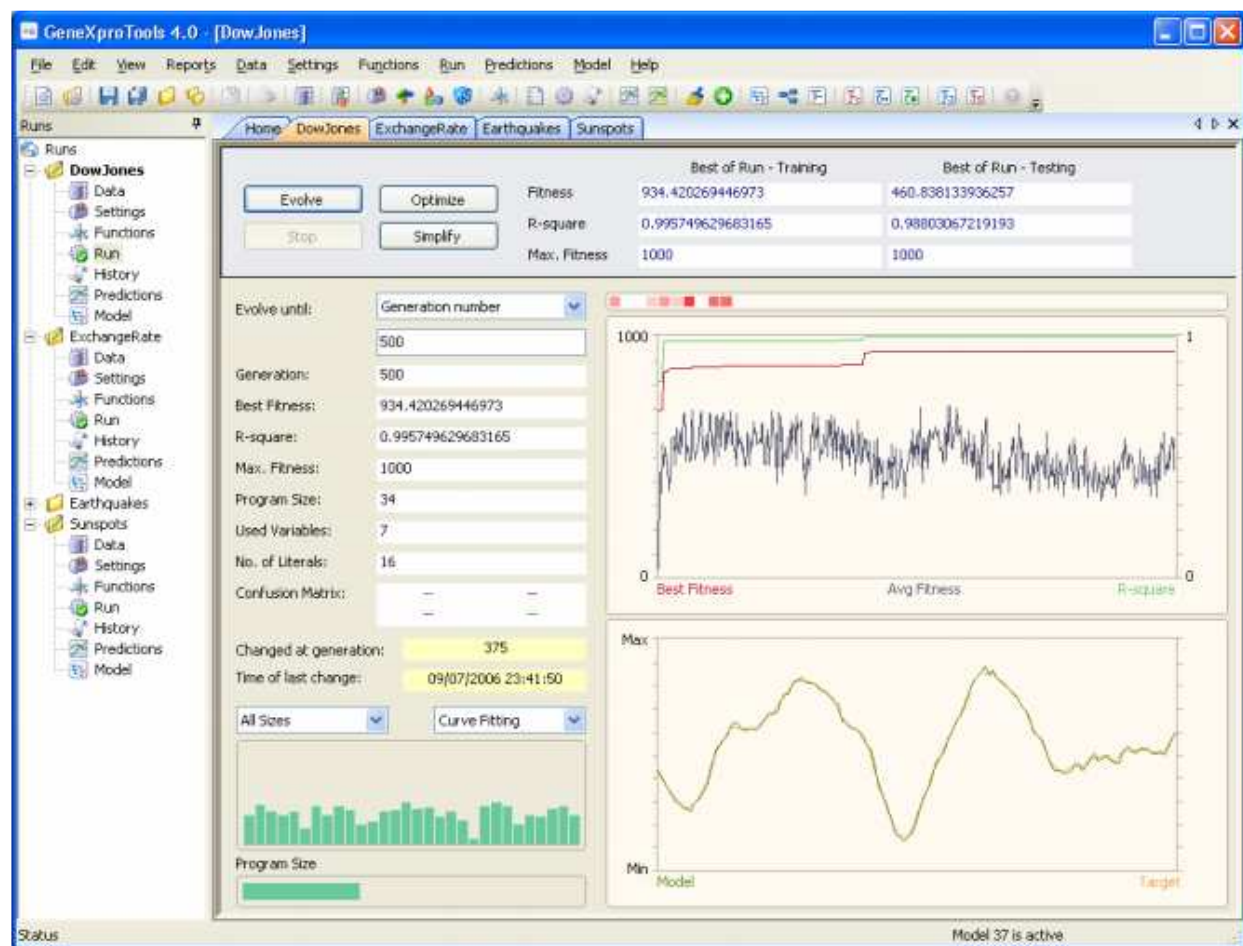
Notwithstanding, GeneXproTools has exceptional tools to assist you in creating good predictive models. The most important is the vast number of [built-in functions](#) that allow you to create really complex models. As an example, see the default functions we recommend for Time Series Prediction in the [Run Templates](#). Try experimenting with different function sets by mixing some functions chosen by hand with others picked randomly with the Random button in the [Functions Panel](#).

Also important is being able to make predictions quickly with all the best-of-generation models created

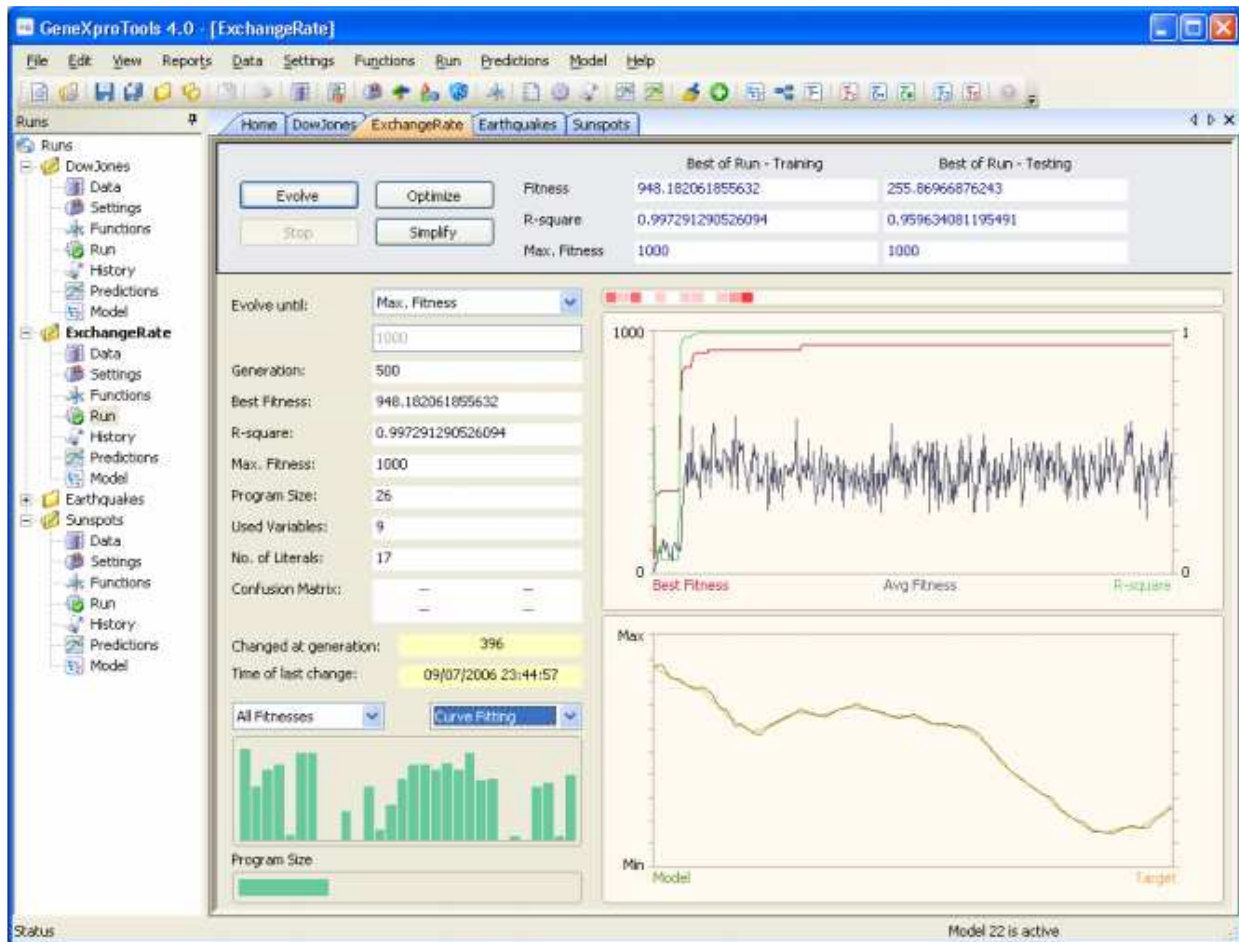
in a run (they are all kept in the [History Panel](#)) and then choosing the best one. For that you just have to select the model you want to test in the [History Panel](#) and then make predictions with it in the [Predictions Panel](#).

These are some of the tips and strategies you might want to try with the four sample runs GeneXproTools offers for Time Series Prediction:

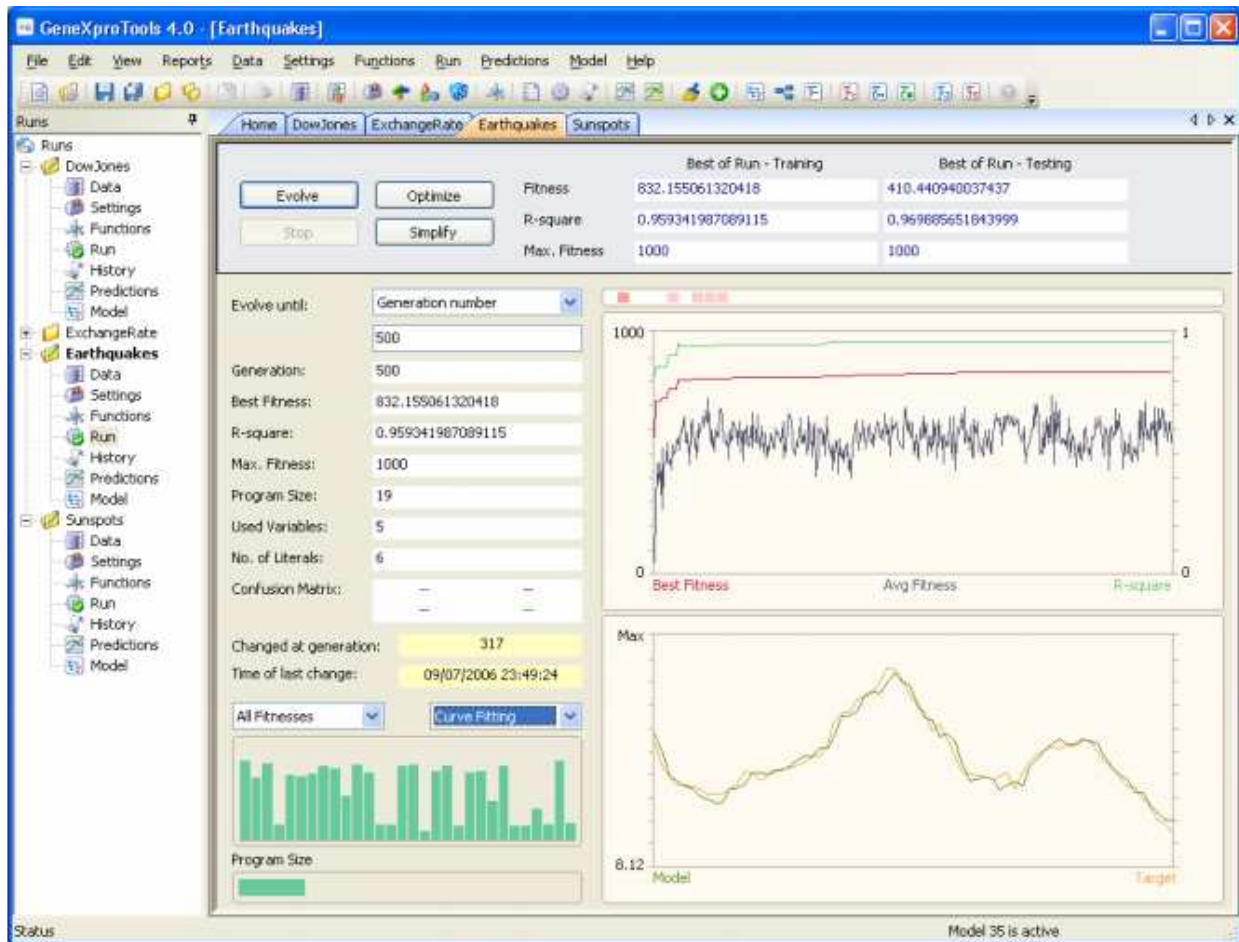
The **Dow Jones** sample run is an interesting time series application that uses monthly closings of the Dow-Jones industrial index. For this sample run, each value from the original raw time series (data file dj.dat obtained from the [Time Series Data Library](#)) was replaced by its moving average using a smoothing period $p = 10$. You will see that GeneXproTools is able to make very good predictions about the monthly closings of the Dow-Jones if you choose Testing Mode and then compare past known behavior with your models' predictions in the [Predictions Panel](#).



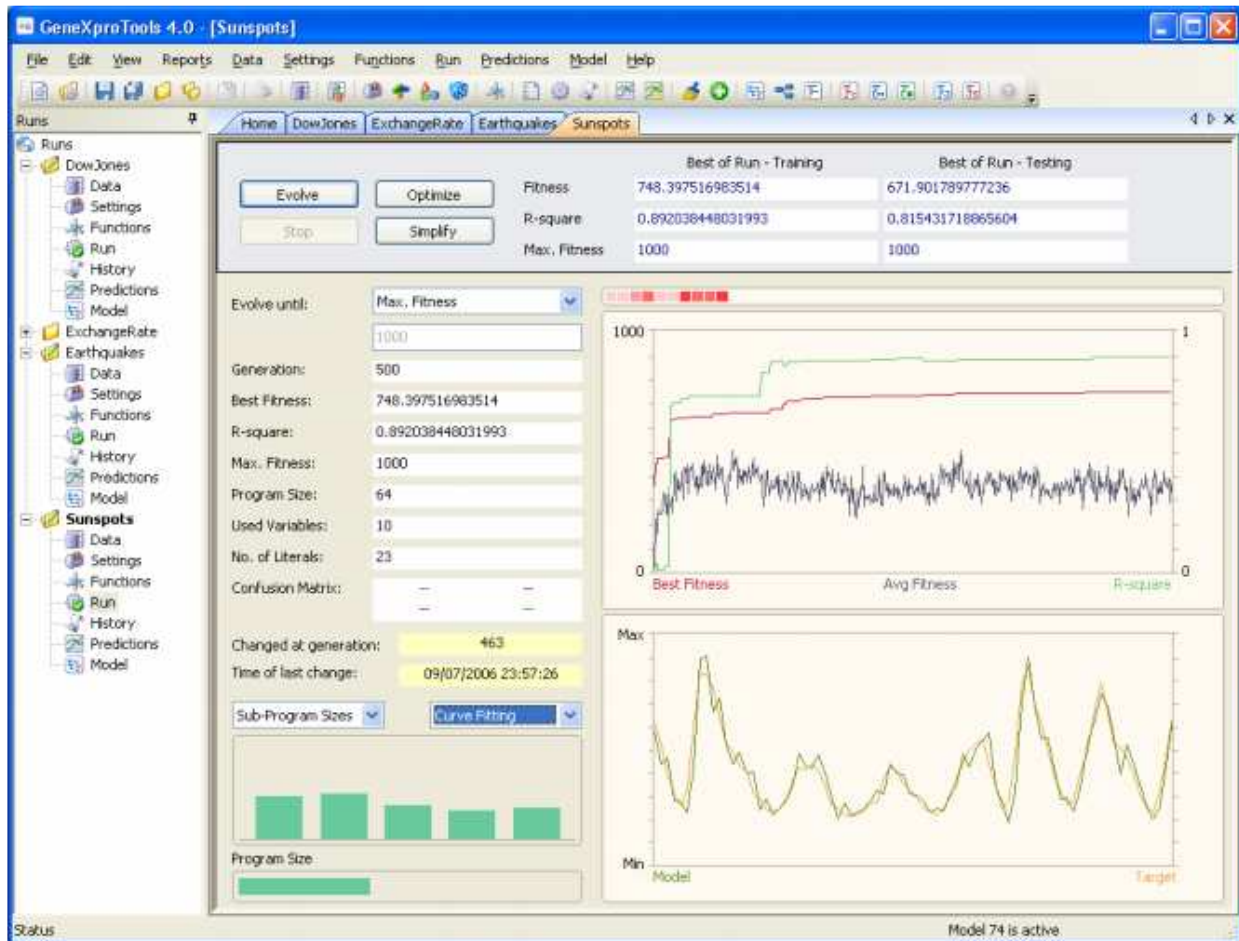
The **Exchange Rate** sample run uses the monthly average exchange rate of Australian dollar for US dollar. For this sample run, each value from the original raw time series (data file usexchm.dat obtained from the [Time Series Data Library](#)) was replaced by its moving average using a smoothing period $p = 12$. You will see that GeneXproTools is able to make very good predictions about exchange rates if you choose Testing Mode and then compare past known behavior with your models' predictions in the [Predictions Panel](#).



In the **Earthquakes** sample run a time series with the number of earthquakes per year with magnitude 7.0 or greater in the period 1900-1998 is used. And again, each value from the original raw time series (data file earthq.dat obtained from the [Time Series Data Library](#)) was replaced by its moving average using a smoothing period $p = 10$. You will see that GeneXproTools is able to make very good predictions about earthquakes if you choose Testing Mode and then compare past known behavior with your models' predictions in the [Predictions Panel](#).



The **Sunspots** sample run uses the well-known Wolfer sunspots data. For this sample run, 100 observations of the Wolfer sunspots series were used with an embedding dimension of 10 and a delay time of one. You will see that GeneXproTools is able to make very good predictions about sunspots if you choose Testing Mode and then compare past known behavior with your models' predictions in the [Predictions Panel](#).



◀ | ▶

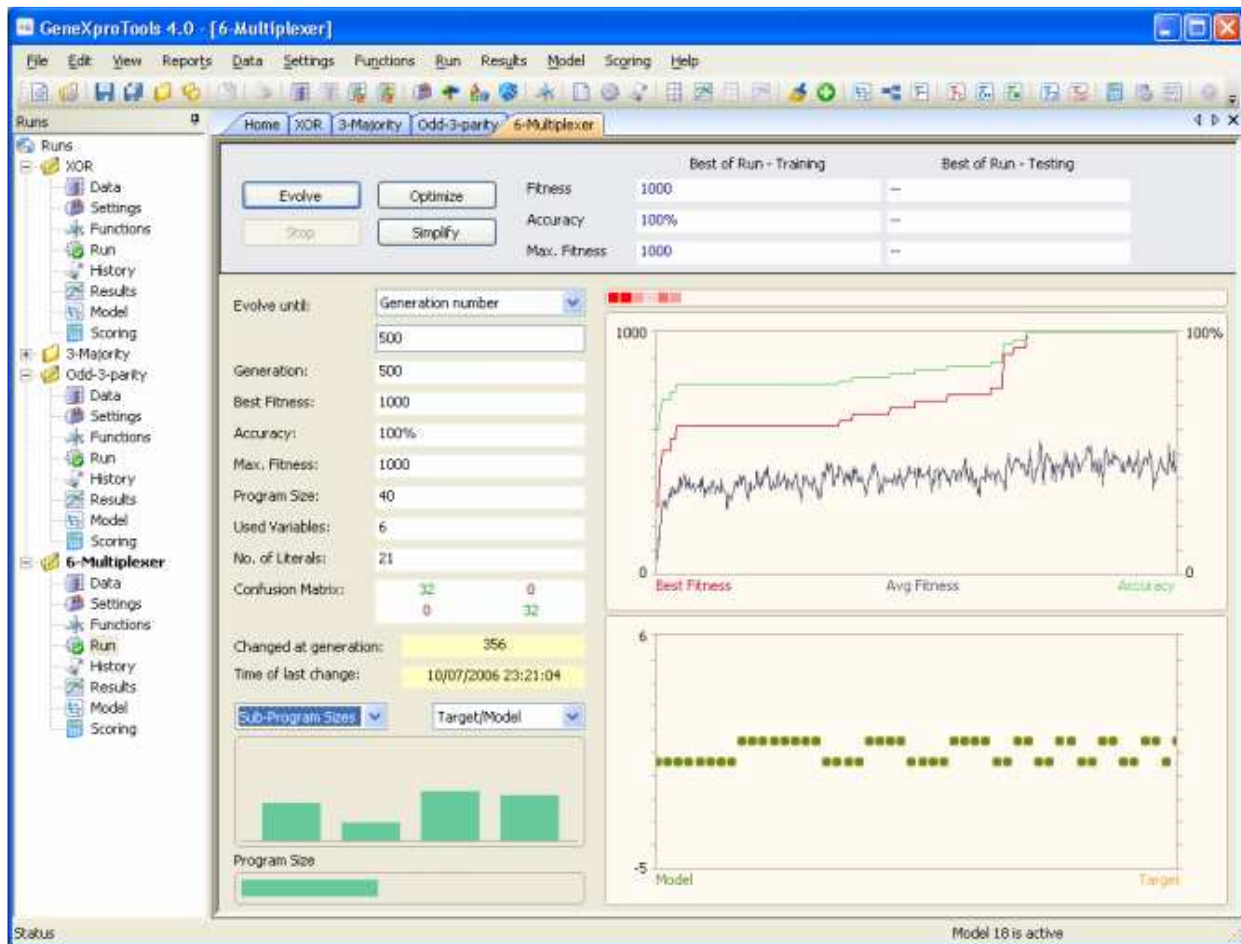
Settings and Features

Settings and Features for Logic Synthesis Problems

The settings and features of GeneXproTools 4.0 available for Logic Synthesis problems are organized as follows:

- [New Run Wizard](#)
- [Report Panel](#)
- [Data Panel](#)
- [Change/Add Dataset Engine](#)
- [Settings Panel](#)
- [Functions Panel](#)
- [Run Panel](#)
- [History Panel](#)
- [Results Panel](#)
- [Model Panel](#)
- [Change Seed Window](#)

- Scoring Panel
- Menus
- Run Templates
- Sample Runs



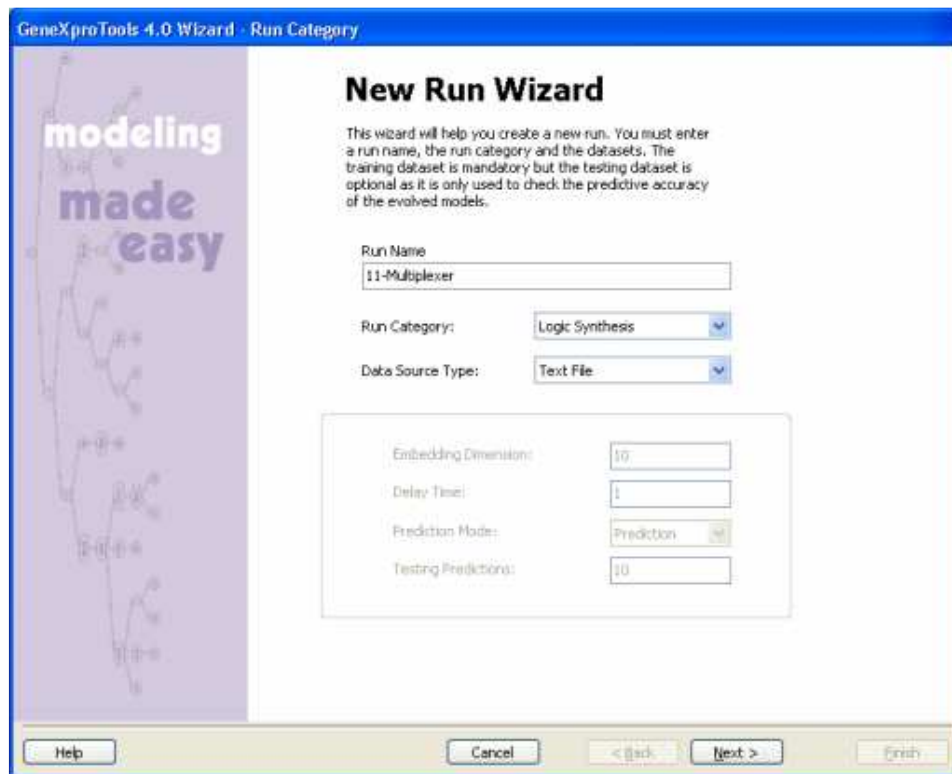
◀ | ▶

Settings and Features

New Run Wizard

The **New Run Wizard** of GeneXproTools 4.0 includes the following windows:

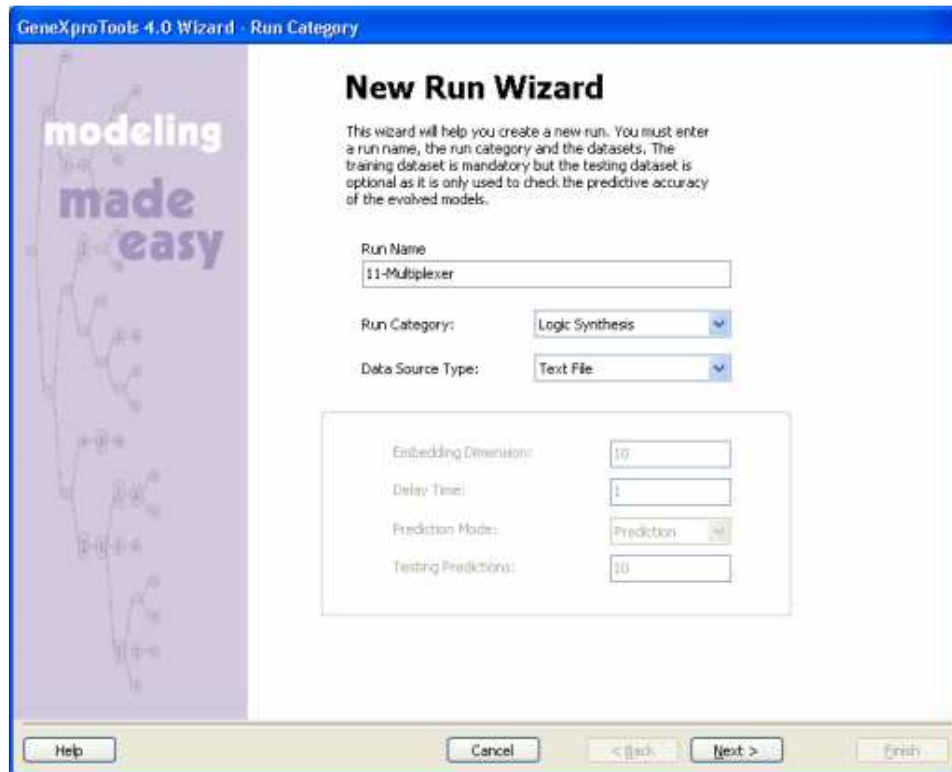
- Run Category Window
- Training Data Window - Text Files
- Testing Data Window - Text Files
- Training Data Window - Excel & Databases
- Testing Data Window - Excel & Databases
- Data Source Dialog Box - Excel & Databases



Settings and Features

Run Category Window

The **Run Category Window** gives access to the following settings and features:



Run Name

The name of the .gep file in which the run will be saved.

Run Category

For Logic Synthesis problems, you must obviously choose Logic Synthesis.

Data Source Type

The type of data source you are using for modeling: Text Files or Excel & Databases.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Next Button

Takes you to the next window of the New Run Wizard.



Settings and Features

Training Data Window - Text Files

The **Training Data Window** gives access to the following settings and features:

GeneXproTools 4.0 Wizard - Training Data

Select a data source for training

File Path: D:\AP5\Version4.0\HelpExamples\LogicSynth...

Data Matrix Format: Samples x Variables

Separator: Tab ☐ First row contains labels

File Name: 11-Multiplexer.txt

Columns: 12

Rows: 2,048

☒ Show loaded data

	d4	d5	d6	d7	d8	d9	d10	D.V.
1	1	0	1	0	0	0	0	0
1	1	0	1	0	0	1	1	1
1	1	0	1	0	1	0	0	0
1	1	0	1	0	1	1	1	1
1	1	0	1	1	0	0	0	0
1	1	0	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0
1	1	1	0	0	0	1	1	1
1	1	1	0	0	1	0	0	0
1	1	1	0	0	1	1	1	1
1	1	1	0	0	0	0	0	0
1	1	1	0	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	0	1	1	1	1	1
1	1	1	1	0	0	0	0	0
1	1	1	1	0	0	1	1	1
1	1	1	1	0	1	0	0	0
1	1	1	1	0	1	1	1	1
1	1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	1	1
1	1	1	1	1	0	1	0	0
1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1

Showing 12 columns and 2048 rows; Loaded 12 columns and 2048 rows.

File Path

Click the button on the right of the list box to load the training file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable or function output occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable or function output occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. Remember that only the values of False and True (case insensitive) or "0" and "1" are valid as input.

Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the training dataset.

Rows

GeneXproTools shows in this box the number of rows (samples) in the training dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the training set.

Clear Button

Clears the training set.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

Opens the Save As dialog box to save the new run.



Settings and Features

Testing Data Window - Text Files

The **Testing Data Window** gives access to the following settings and features:

GeneXproTools 4.0 Wizard - Testing Data

Select a data source for testing (optional)

File Path: ...

Data Matrix Format:

Separator: ☐ First row contains labels

File Name:

Columns:

Rows:

☒ Show loaded data

d0	d1	d2	d3	d4	d5	d6	d7	d8
0	0	0	1	1	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	1	0
1	1	0	0	1	0	0	1	1
1	1	0	1	0	0	1	1	1
0	1	1	1	1	0	0	1	1
0	1	0	0	0	1	0	1	1
1	1	1	1	0	1	1	0	1
1	1	0	1	0	1	1	0	0
0	1	1	0	1	1	0	0	0
0	0	0	0	1	1	0	1	1
0	1	0	1	0	0	1	0	1
1	1	0	1	0	1	0	1	1
1	0	1	1	0	0	1	1	0
1	0	0	0	1	1	0	0	0
0	1	0	0	0	0	1	1	0
1	0	0	0	1	0	1	1	1
1	0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	0	0
1	0	0	1	0	1	1	0	0
1	0	1	1	0	0	1	1	0
1	0	0	1	0	0	1	0	1
0	0	0	1	0	0	1	0	1
0	1	0	0	0	0	0	0	1

Showing 12 columns and 1000 rows; Loaded 12 columns and 1000 rows.

File Path

Click the button on the right of the list box to load the testing file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable or function output occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable or function output occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. Remember that only the values of False and True (case insensitive) or "0" and "1" are valid as input.

Separator

GeneXproTools identifies automatically the separator in the testing file and shows it in the combo box under this label. And if the variables in your testing data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the testing dataset.

Rows

GeneXproTools shows in this box the number of rows (samples) in the testing dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the testing set.

Clear Button

Clears the testing set.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

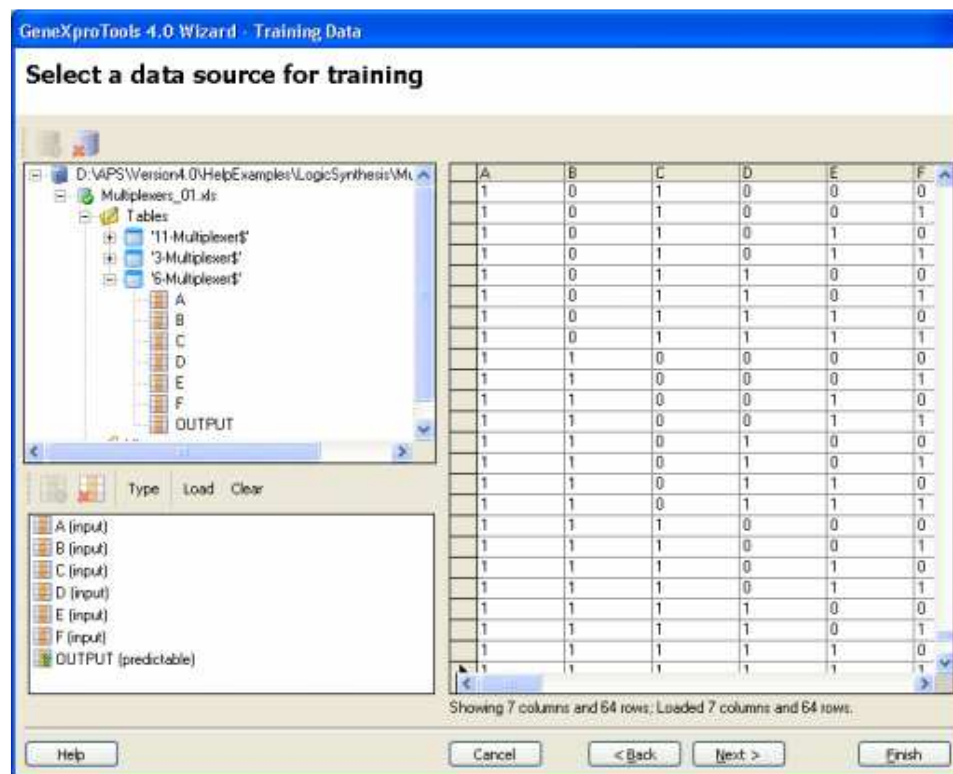
Opens the Save As dialog box to save the new run.



Settings and Features

Training Data Window - Excel & Databases

The **Training Data Window** gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Training Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Training Data Window.

Training Data Window

You prepare your training data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Remember that only the values of False and True (case insensitive) or "0" and "1" are valid as input.

Training Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

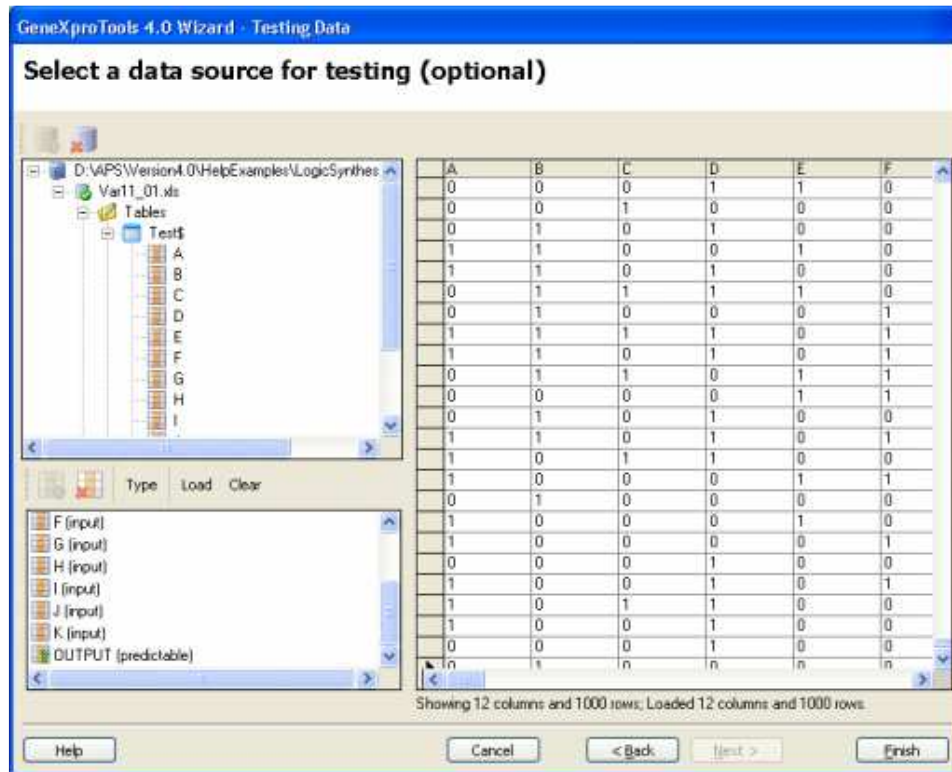
Opens the Save As dialog box to save the new run.



Settings and Features

Testing Data Window - Excel & Databases

The **Testing Data Window** gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Testing Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Testing Data Window.

Testing Data Window

You prepare your testing data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Remember that only the values of False and True (case insensitive) or "0" and "1" are valid as input.

Testing Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns in the New Run Wizard, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Help Button

Opens the Help File.

Cancel Button

Cancels all the action on the creation of a new run.

Back Button

Takes you to the previous window of the New Run Wizard.

Next Button

Takes you to the next window of the New Run Wizard.

Finish Button

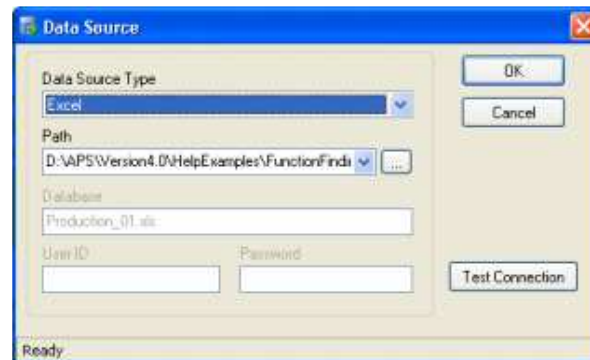
Opens the Save As dialog box to save the new run.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:

**Driver**

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.

Microsoft Access: path to the database file.

Microsoft SQL Server: server name or IP address.

Oracle: server name or IP address.

MySQL: server name or IP address.

DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.

Oracle - Name and database instance (HOST/INSTANCE).

MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

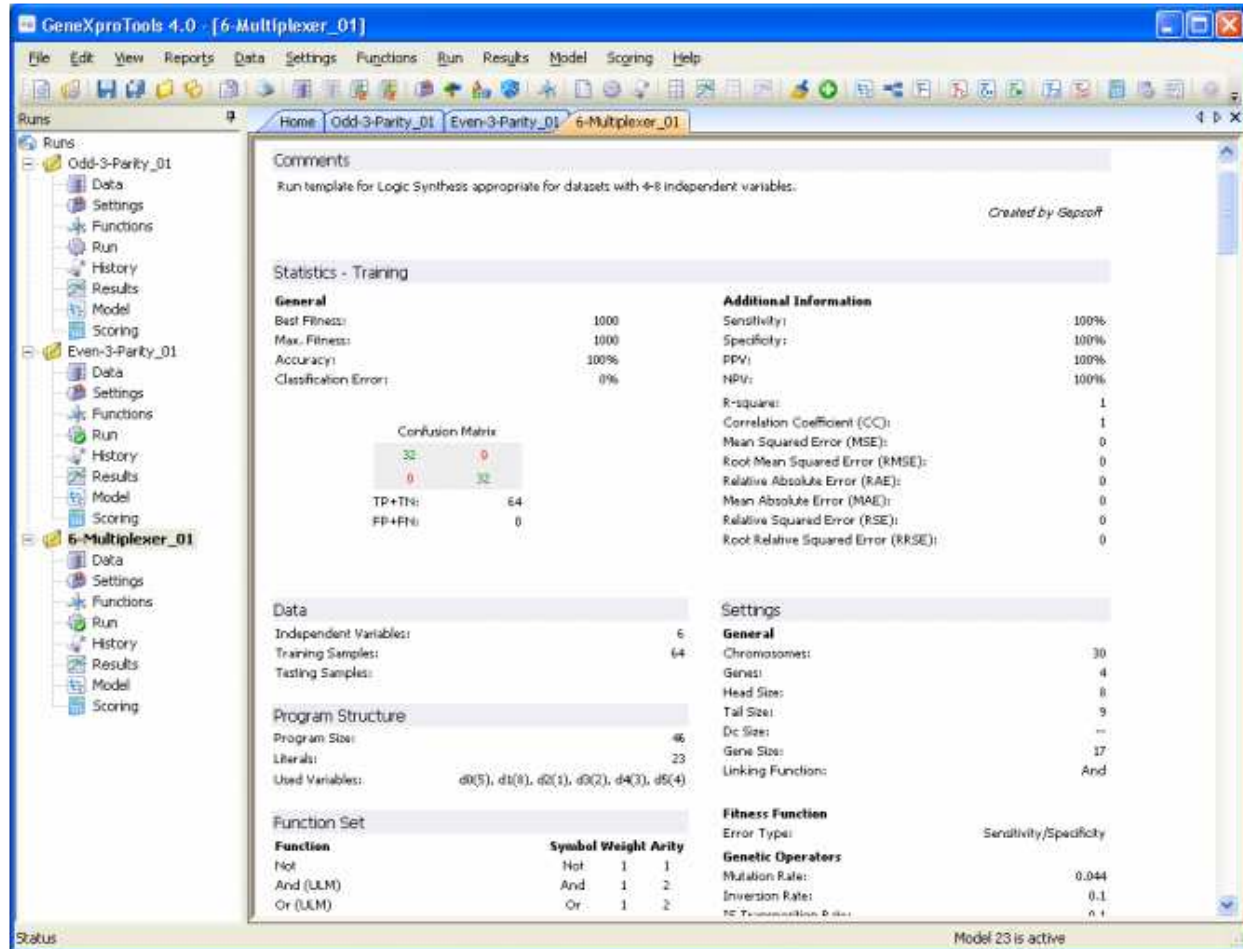
Cancels all the action on the data source connection.



Settings and Features

Report Panel

The **Report Panel** of GeneXproTools 4.0 keeps a summary of all the relevant information of a run, and is organized into the following headings:



Comments

This heading is usually used to give a small description of the problem, the objectives, the kind of data used, their source, etc. Both the comments and the author's name are introduced by choosing Properties on the File Menu.

Statistics – Training

Under this heading are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the training set: [classification error](#), [classification accuracy](#), [confusion matrix](#) (true positives, true negatives, false positives, and false negatives), [sensitivity](#), [specificity](#), [positive predictive value](#), [negative predictive value](#), [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), and [R-square](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, the number of [true positives](#) plus [true negatives](#), and the number of [false positives](#) plus [false negatives](#).

A note of warning, though: All the parameters listed here refer to the last evaluated model in the [Results Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
0	19	11	d0(2), d1(3), d2(6)	562.5	--	75.00%	--
And Or d2 And Or Or Or d2 d0 d0 d1 d1 d2 d2 d0 Or Or d2 d2 Not d2 d2 d0 d2 d0 d1 d2 d0 d2 d2 d0 Or And d1 d2 d1 d2 Or Not d1 d0 d0 d1 d2 d2 d1 d1							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
1	21	11	d0(3), d1(3), d2(5)	750	--	87.50%	--
Not Not And d2 d2 d2 And d1 d0 d1 d2 d0 d0 d0 d0 Or And Or And d0 d0 d1 d1 d2 d0 d1 d2 d0 d2 d2 d0 Or And And d2 And d1 d2 And d2 d0 d0 d0 d2 d1 d2 d2							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
4	13	8	d0(4), d1(2), d2(2)	1000	--	100%	--
And d1 d2 d0 d0 d2 Or d2 d2 d2 d2 d2 d2 d1 d2 Or And d0 And And d1 d0 d0 d2 d2 d2 d0 d0 d0 d1 Or And d2 d0 Not d2 And Or d1 d1 d1 d2 d0 d1 d0 d2							

Statistics – Testing

This heading only appears when a testing set was used. Here are listed the values of all statistical functions used by GeneXproTools 4.0 to evaluate the accuracy of your model on the testing set, that is, the efficiency of your model at predicting unknown behavior: [classification error](#), [classification accuracy](#), [confusion matrix](#) (true positives, true negatives, false positives, and false negatives), [sensitivity](#), [specificity](#), [positive predictive value](#), [negative predictive value](#), [mean squared error](#), [root mean squared error](#), [mean absolute error](#), [relative squared error](#), [root relative squared error](#), [relative absolute error](#), [correlation coefficient](#), and [R-square](#).

Also listed under this heading are more general run parameters such as best fitness, maximum fitness, the number of [true positives](#) plus [true negatives](#), and the number of [false positives](#) plus [false negatives](#).

Note again that all the parameters listed here refer to the last evaluated model in the [Results Panel](#), so make sure that they do refer to the model you are interested in (usually the active model, highlighted in yellow in the History heading).

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
0	37	19	d0(5), d1(4), d2(1), d3(2), d4(4), d5(3)	1000.096875	--	100%	--
Or d5 .Not .And .Or .And .d4 .d0 .d1 .d0 .d5 .d2 .d2 .d2 .d4 .d1 .d4 And Or .Or .d4 .Or .And .d4 .Not d5 .d1 .d0 .d0 .d1 .d3 .d1 .d0 .d3 .d0 And Or .Or .And .Not .And .d0 .d5 .d1 .d3 .d3 .d0 .d0 .d3 .d0 .d5 .d4 .d0 And Or d1 .Or d2 .And .d4 .d0 .d2 .d1 .d2 .d4 .d5 .d0 .d4 .d2 .d3 .d4							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
46	35	18	d0(5), d1(5), d2(1), d3(1), d4(3), d5(3)	1000.103125	--	100%	--
Or .Or .And .d3 .Not d0 .d5 .d1 .d3 .d5 .d0 .d0 .d3 .d0 .d1 .d4 .d0 And Or d5 .Not .And .Or .And .d4 .d0 .d1 .d0 .d1 .d2 .d0 .d2 .d3 .d2 .d1 And Or .Or .d4 .Or .And .d1 .Not d5 .d1 .d0 .d2 .d1 .d3 .d1 .d4 .d3 .d0 And Or d1 .Or d2 .And .d4 .d0 .d2 .d1 .d2 .d4 .d5 .d4 .d4 .d2 .d3 .d1							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
280	35	18	d0(4), d1(6), d2(1), d3(3), d4(2), d5(2)	1000.103125	--	100%	--
Or d5 .Not .And .And .And .d3 .d0 .d1 .d3 .d5 .d1 .d4 .d5 .d2 .d1 .d1 And Or .Or d2 .d1 .And .d4 .d0 .And d1 .d3 .d5 .d3 .d3 .d2 .d5 .d0 .d2 And Or .Or .Or d4 .And .d1 .Not d1 .d1 .d0 .d5 .d2 .d2 .d5 .d1 .d5 .d2 And Or .Or .And .Not d3 .d0 .d5 .d1 .d1 .d3 .d1 .d3 .d2 .d4 .d1 .d0 .d5							

Data

Under this heading are listed the number of samples in the training set actually used during training (remember that GeneXproTools 4.0 allows you to choose a smaller sub-set of the training set to use during training), the number of testing samples used to test the generalizing capabilities of your model (again, not all the samples on the testing set were necessarily used to test your model), and the number of attributes or independent variables in your data.

Program Structure

The program size, the number of literals, and a detailed list of all used variables and respective weights are shown here. This information refers to the active model highlighted in yellow in the History heading.

Program Structure	
Program Size:	37
Literals:	19
Used Variables:	d0(5), d1(4), d2(1), d3(2), d4(4), d5(3)

Function Set

Under this heading are listed all the functions used to create your model, together with the symbols used to represent them in [Karva notation](#), their weights and arities.

Function Set		
Function	Symbol	Weight Arity
Not	Not	1 1
And (ULM)	And	1 2
Or (ULM)	Or	1 2

Note that the highest value of arity in your function set determines the structure of each gene, making the length of the [tail](#) t grow according to the equation:

$$t = h(n-1) + 1$$

where h is the length of the [head](#) and n is maximum arity.

Settings

Under this heading there are five entries: General Settings, Complexity Increase, Fitness Function, Genetic Operators, and Numerical Constants. Under General Settings are listed the number of chromosomes, number of genes, head size, tail size, Dc size, gene size, and linking function. Under Complexity Increase are listed the number of Generations Without Change, the Number of Tries, and Max. Complexity. The name of the fitness function is shown below the entry Fitness Function. Under Genetic Operators are listed the rates of all the genetic operators used to create genetic variation. And, finally, all the parameters relevant to the manipulation of random numerical constants are listed under Numerical Constants.

Settings	
General	
Chromosomes:	30
Genes:	4
Head Size:	8
Tail Size:	9
Dc Size:	9
Gene Size:	26
Linking Function:	And
Complexity Increase	
Generations without Change:	2000
Number of Tries:	5
Max. Complexity:	10
Fitness Function	
Error Type:	Sensitivity/Specificity
Genetic Operators	
Mutation Rate:	0.044
Inversion Rate:	0.1
IS Transposition Rate:	0.1
RIS Transposition Rate:	0.1
One-Point Recombination Rate:	0.3
Two-Point Recombination Rate:	0.3
Gene Recombination Rate:	0.1
Gene Transposition Rate:	0.1
Numerical Constants	
Constants per Gene:	2
Data Type:	Integer
Lower Bound:	0
Upper Bound:	1
RNC Mutation:	0.01
Dc Mutation:	0.044
Dc Inversion:	0.1
Dc IS Transposition:	0.1

Data Sources

Under this heading the paths to the input files (training and testing files) used to create your model are given so that their origin could be easily identified.

File Path

Under this heading the path to the gep file that was used to save your model is given so that it can be easily traced for future reference.

History

Under this heading are listed all the best-of-generation models, including obviously the best-of-run model, discovered during the evolutionary process. Note, however, that the notation "best-of-run" and "best-of-generation" refers to the performance of these models on the training set, which could be slightly different on the testing set. This is why GeneXproTools 4.0 allows you to analyze the performance of all these models on the testing set and, if appropriate, choose the best model based on these results. GeneXproTools 4.0 gives you the possibility of highlighting in yellow one of these models (the active model) and we advise you to use this feature to highlight the best one. Note also that, for a quick reference, not only the program structure (program size, number of literals, and list of used variables and respective weights) but also the fitness and the [accuracy](#) of all these models both on the training and testing sets are also given, although, for the testing set, they must be evaluated in the

History Panel.

History							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
0	19	11	d0(2), d1(3), d2(6)	562.5	--	75.00%	--
And Or d2 And Or Or Or d2 d0 d0 d1 d1 d2 d2 d0 Or Or d2 d2 Not d2 d2 d0 d2 d0 d1 d2 d0 d2 d2 d0 Or And d1 d2 d1 d2 Or Not d1 d0 d0 d1 d2 d2 d1 d1							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
1	21	11	d0(3), d1(3), d2(5)	750	--	87.50%	--
Not Not And d2 d2 d2 And d1 d0 d1 d2 d0 d0 d0 d0 Or And Or And d0 d0 d1 d1 d2 d0 d1 d2 d0 d2 d2 d0 Or And And d2 And d1 d2 And d2 d0 d0 d0 d2 d1 d2 d2							
Generation	Program Size	Literals	Used Variables	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
4	13	8	d0(4), d1(2), d2(2)	1000	--	100%	--
And d1 d2 d0 d0 d2 Or d2 d2 d2 d2 d2 d2 d1 d2 Or And d0 And And d1 d0 d0 d2 d2 d2 d0 d0 d0 d1 Or And d2 d0 Not d2 And Or d1 d1 d1 d2 d0 d1 d0 d2							

For brevity sake, all the best-of-generation models are represented in the compact [Karva notation](#), but you can translate any one of them into a more canonical representation by activating any of these models in the [History Panel](#) and then translate it into the language of your choice in the [Model Panel](#).

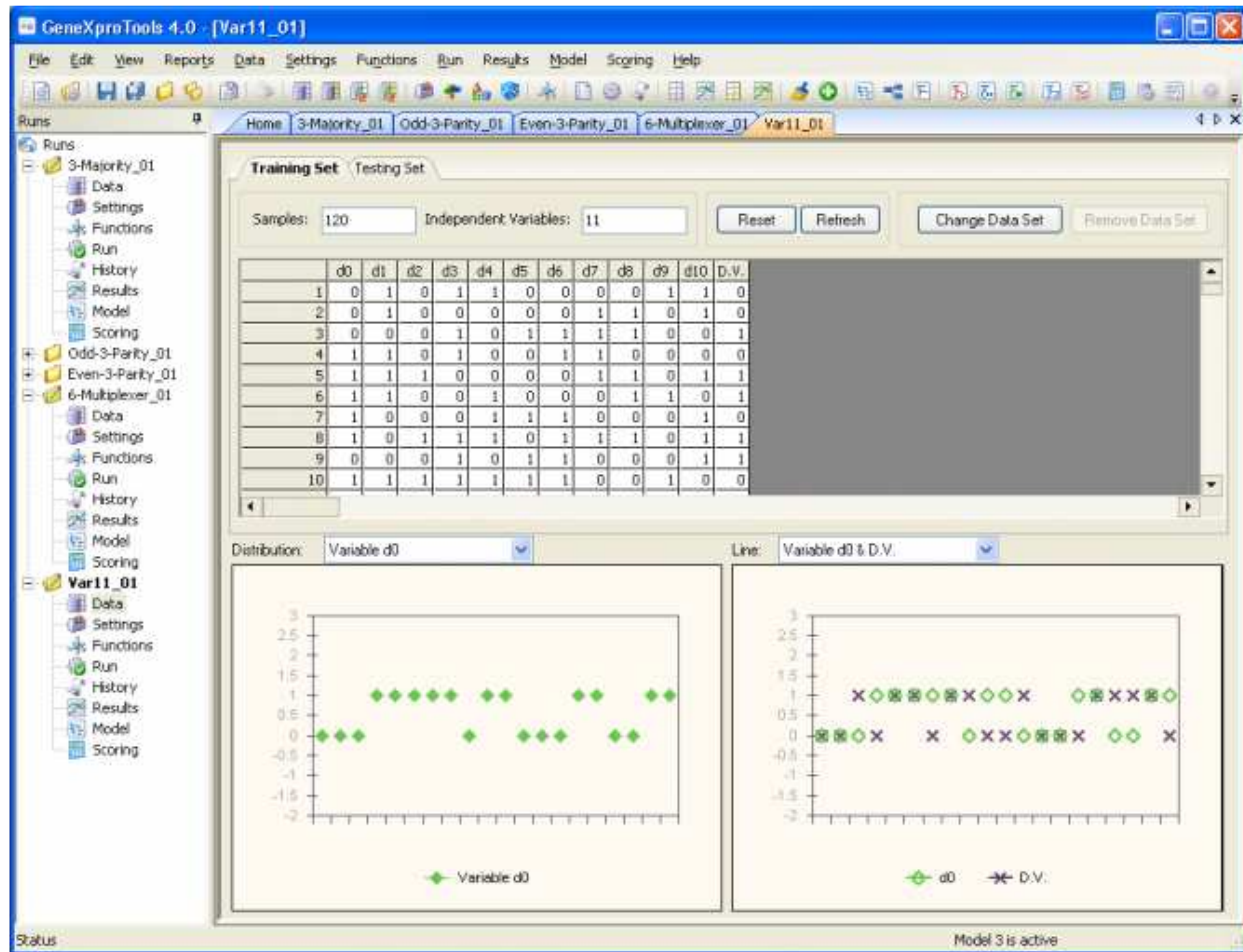


Settings and Features

Data Panel

The **Data Panel** of GeneXproTools 4.0 includes the following tabs:

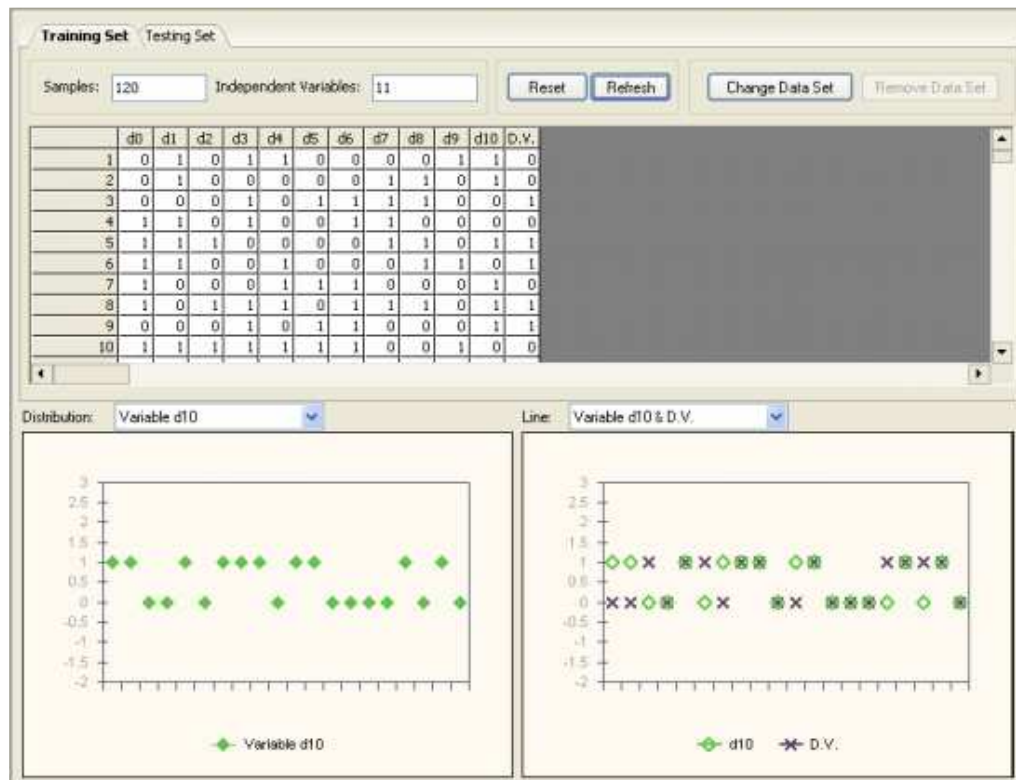
- [Training Set Tab](#)
- [Testing Set Tab](#)



Settings and Features

Training Set Tab

The **Training Set Tab** gives access to the following settings and features:



Samples

Allows you to select a subset of samples for training.

Independent Variables

This box is informative only and shows the number of independent variables in the training set.

Reset Button

Reverts to the total number of training samples in your data.

Refresh Button

Refreshes the panel after changing the training data set.

Change Data Set Button

Allows you to choose another training set for modeling. Note that GeneXproTools 4.0 not only allows you to change a training set by another with the same structure, that is, the same number of independent variables, but also allows you to replace the training set by another one with a different structure. Obviously, in this case, all the models in the run history will be invalidated and the testing set, if there is one, will be automatically removed.

Table

Shows all the variables in the training set. The entire table can be copied to the clipboard by choosing Copy on the Edit Menu. You can also copy selected columns to the clipboard with the mouse right-click.

Distribution

Allows you to visualize the distribution of values for each variable in the training set. This plot shows a maximum of 20 points at a time and, therefore, for datasets with more than 20 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.

Line

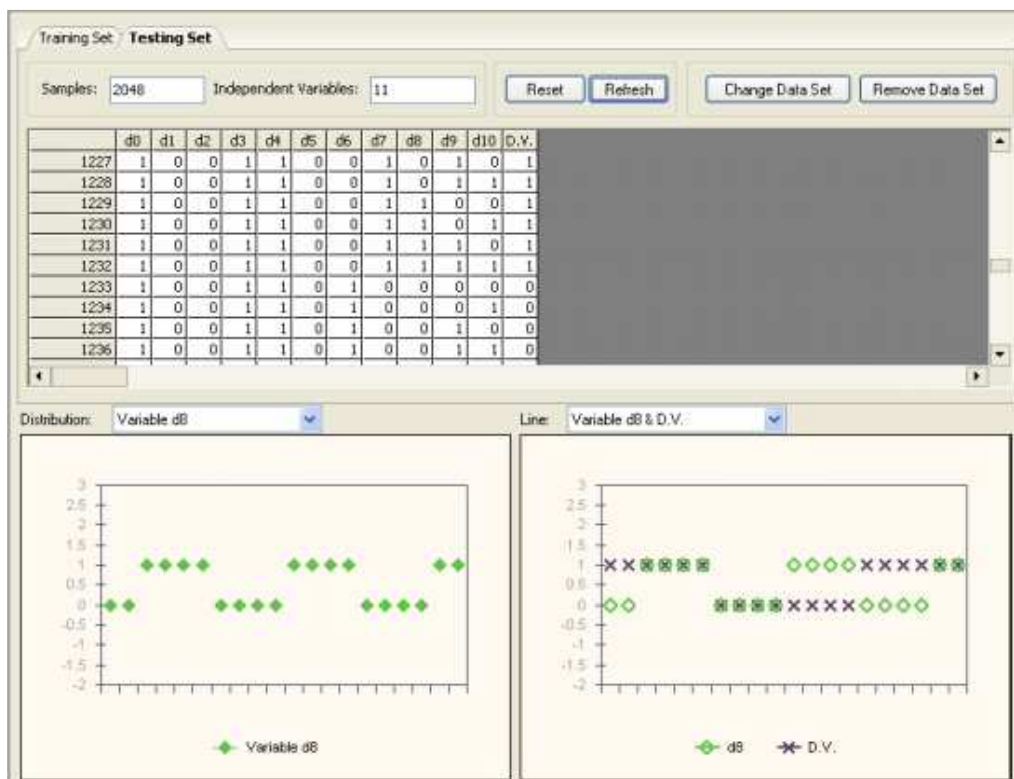
Plots each independent variable together with the dependent variable or function output. This plot shows a maximum of 20 points at a time and, therefore, for datasets with more than 20 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.



Settings and Features

Testing Set Tab

The **Testing Set Tab** gives access to the following settings and features:



Samples

Allows you to select a subset of samples for testing.

Independent Variables

This box is informative only and shows the number of independent variables or attributes in the testing set.

Reset Button

Reverts to the total number of testing samples in your data.

Refresh Button

Refreshes the panel after changing the testing data set.

Change Data Set Button

Allows you to choose another testing set for evaluating the generalizing capabilities of your models. The new testing set must obviously have the same structure of the training set, that is, the same number of independent variables.

Remove Data Set Button

Allows you to remove the testing set (you can always add it later by selecting Add Testing Data on the Data Menu).

Table

Shows all the variables in the testing set. The entire table can be copied to the clipboard by choosing Copy on the Edit Menu. You can also copy selected columns to the clipboard with the mouse right-click.

Distribution

Allows you to visualize the distribution of values for each variable in the testing set. This plot shows a maximum of 20 points at a time and, therefore, for datasets with more than 20 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.

Line

Plots each independent variable together with the dependent variable or function output. This plot shows a maximum of 20 points at a time and, therefore, for datasets with more than 20 samples you must move the scroll bar on the right of the table and GeneXproTools will redraw the plot starting at the first visible row. Then, by placing the cursor over the plot, GeneXproTools shows you the range used to draw the plot. GeneXproTools allows you to copy or print this plot through the mouse right-click.



Settings and Features

Change/Add Dataset Engine

The **Change/Add Dataset Engine** of GeneXproTools 4.0 includes the following windows:

- [Change Training Data Window - Text Files](#)
- [Change/Add Testing Data Window - Text Files](#)
- [Change Training Data Window - Excel & Databases](#)
- [Change/Add Testing Data Window - Excel & Databases](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Change Training Data Window - Text Files

The **Change Training Data Window** allows you to change the training set and gives access to the following settings and features:

File Path

Click the button on the right of the list box to load the training file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable or function output occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable or function output occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. Remember that only the values of False and True (case insensitive) or "0" and "1" are valid as input.

Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the training dataset. You can replace the old training set by another one with a different number of variables, but obviously, in this case, all the models in the run history will be invalidated and the testing set (if there is one) will be automatically removed by GeneXproTools.

Rows

GeneXproTools shows in this box the number of rows (samples) in the training dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns on the table of the Change Training Data Window, but obviously loads them all,

and you can see the exact number on the Columns box above.

Load Button

Loads the training set.

Clear Button

Clears the training set.

Cancel Button

Cancels all the action on the replacement of the training set.

Finish Button

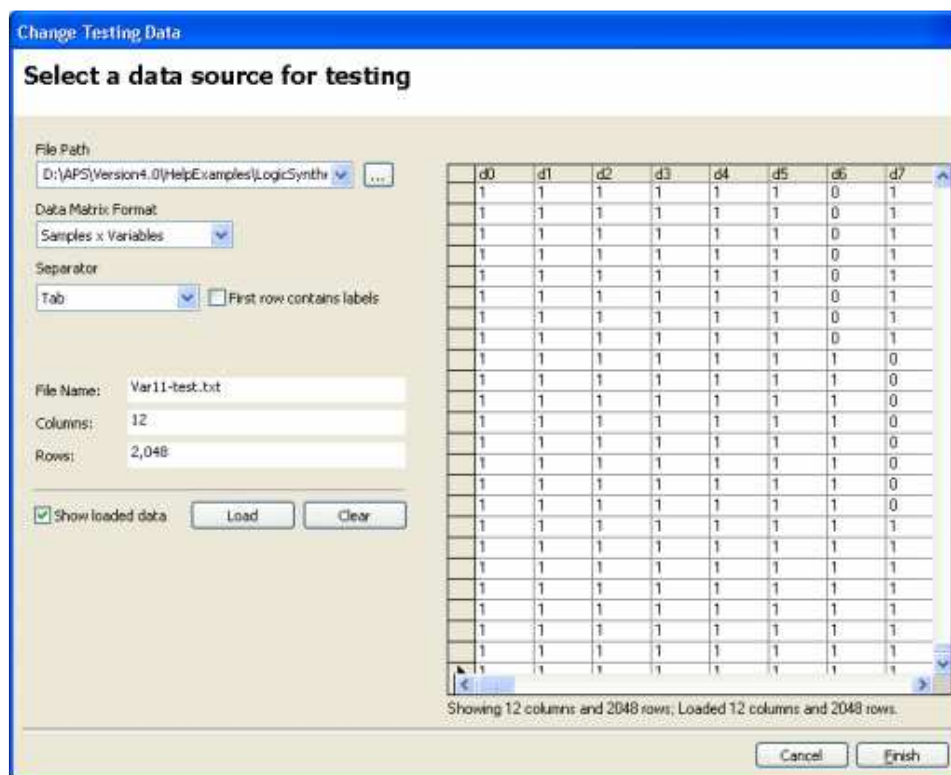
Saves the new training set.



Settings and Features

Change/Add Testing Data Window - Text Files

The **Change/Add Testing Data Window** allows you to change the testing set and gives access to the following settings and features:



File Path

Click the button on the right of the list box to load the testing file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, with the dependent variable or function output occupying the rightmost position, you must select the Samples x Variables format. If your samples are in columns and variables in rows, with the dependent variable or function output occupying the topmost position, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. Remember that only the values of False and True (case insensitive) or "0" and "1" are valid as input.

Separator

GeneXproTools identifies automatically the separator in the testing file and shows it in the combo box under this label. And if the variables in your testing data are identified by a name, GeneXproTools also ticks the box on the right.

File Name

GeneXproTools shows you the file name in this box.

Columns

GeneXproTools shows in this box the number of columns (variables) in the testing dataset. Your new testing set must have the same number of variables as the training set. If you want to change the number of variables, you must first [change the training set](#).

Rows

GeneXproTools shows in this box the number of rows (samples) in the testing dataset.

Show Loaded Data

Check this box if you want to see the loaded dataset on the table on the right. GeneXproTools only shows 2048 columns on the table of the Change Testing Data Window, but obviously loads them all, and you can see the exact number on the Columns box above.

Load Button

Loads the testing set.

Clear Button

Clears the testing set.

Cancel Button

Cancels all the action on the replacement of the testing set.

Finish Button

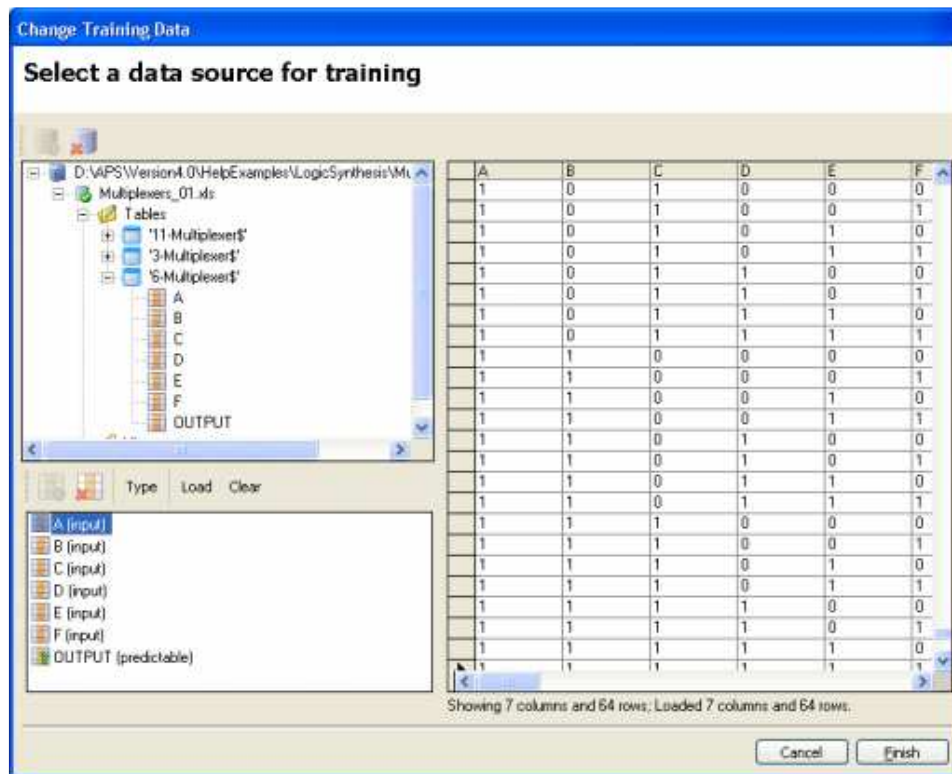
Saves the new testing set.



Settings and Features

Change Training Data Window - Excel & Databases

The **Change Training Data Window** allows you to change the training set and gives access to the following settings and features:

**Add Data Source Icon**

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Training Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Training Data Window.

Training Data Window

You prepare your training data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Recall that only the values of False and True (case insensitive) or "0" and "1" are valid as input. GeneXproTools allows you to replace the old training set by another one with a

different number of variables, but obviously, in this case, all the models in the run history will be invalidated and the testing set (if there is one) will be automatically removed by GeneXproTools.

Training Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns on this table, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Cancel Button

Cancels all the action on the replacement of the training set.

Finish Button

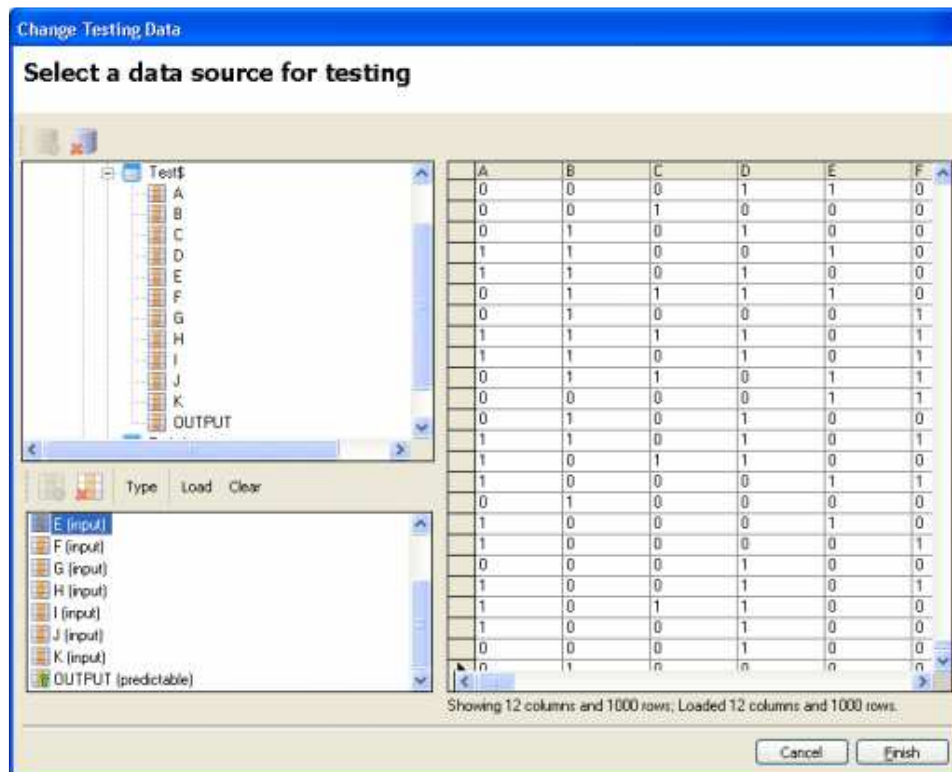
Saves the new training set.



Settings and Features

Change/Add Testing Data Window - Excel & Databases

The **Change/Add Testing Data Window** allows you to change the testing set (or add a new one) and gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Testing Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Type Menu

Sets the type (input or predictable) of the selected column. You can also set the type through the mouse right-click.

Load Button

Loads/shows the selected columns' contents.

Clear Button

Removes all the columns from the Testing Data Window.

Testing Data Window

You prepare your testing data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Recall that only the values of False and True (case insensitive) or "0" and "1" are valid as input. Your new testing set must have the same number of variables as the training set. If you want to change the number of variables, you must first [change the training set](#).

Testing Data Table

Shows the loaded columns. GeneXproTools only shows 2048 columns on this table, but obviously loads them all, and you can see the exact number of loaded/shown columns on the status bar below.

Cancel Button

Cancels all the action on the replacement of the testing set.

Finish Button

Saves the new testing set.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:



Driver

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.

Microsoft Access: path to the database file.

Microsoft SQL Server: server name or IP address.

Oracle: server name or IP address.

MySQL: server name or IP address.

DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.

Oracle - Name and database instance (HOST/INSTANCE).

MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

Cancels all the action on the data source connection.



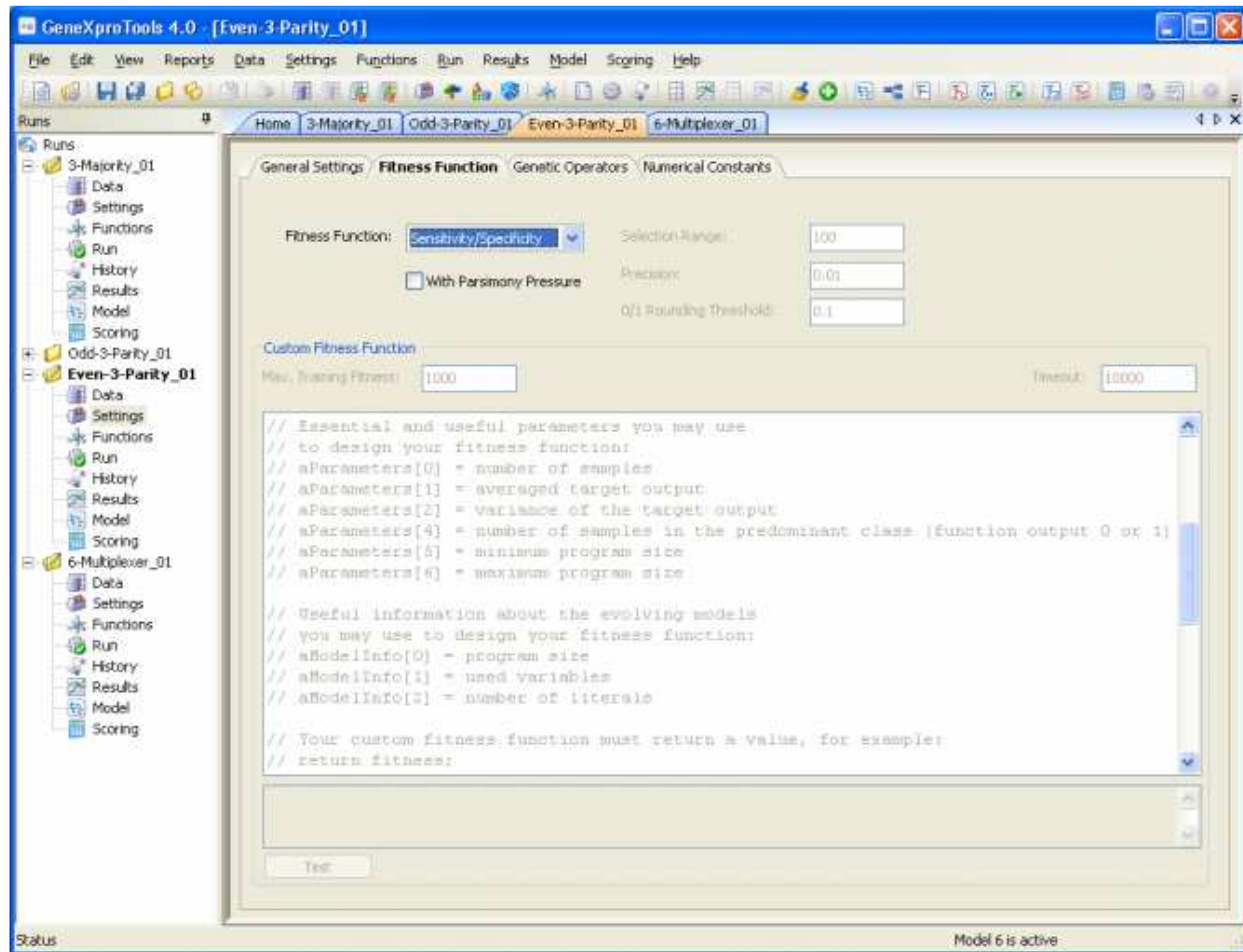
Settings and Features

Settings Panel

The **Settings Panel** of GeneXproTools 4.0 includes the following tabs:

- [General Settings Tab](#)

- Fitness Function Tab
- Genetic Operators Tab
- Numerical Constants Tab



Settings and Features

General Settings Tab

The **General Settings Tab** gives access to the following settings and features:

The screenshot shows the 'General Settings' tab of the GeneXproTools application. It contains several input fields and checkboxes for configuring the evolutionary process. The 'Training Samples' field is set to 120, and the 'Testing Samples' field is set to 2048, both with 'Reset' buttons. Below these, 'Number of Chromosomes' is 30, 'Head Size' is 10, 'Number of Genes' is 7, and 'Linking Function' is set to 'And'. A section titled 'Enable Complexity Increase' is checked, with 'Generations without Change' set to 2000, 'Number of Tries' set to 3, and 'Max. Complexity' set to 7. At the bottom, the 'Save Run History' checkbox is also checked.

Parameter	Value	Action
Training Samples	120	Reset
Testing Samples	2048	Reset
Number of Chromosomes	30	Spinners
Head Size	10	Spinners
Number of Genes	7	Spinners
Linking Function	And	Dropdown
Enable Complexity Increase	☒	Checkbox
Generations without Change	2000	Spinners
Number of Tries	3	Spinners
Max. Complexity	7	Spinners
Save Run History	☒	Checkbox

Training Samples

Number of samples used for training.

Reset

Reverts to the total number of training samples in your data.

Testing Samples

Number of samples used for evaluating the generalizing capabilities of the evolved models.

Reset

Reverts to the total number of testing samples in your data.

Number of Chromosomes

Size of the population of models created each generation for breeding new ones.

Head Size

Size of the [head](#) of each gene. This parameter determines the upper limit for the size of the programs encoded in the gene.

Number of Genes

Number of genes per chromosome or, in other words, the number of complex terms in the evolving models.

Linking Function

Function used to link the different terms encoded in the genes of a model. You can choose either addition, subtraction, multiplication, or division to link the [sub-ETs](#) or terms of your models.

Enable Complexity Increase

Check this box if you wish to allow GeneXproTools to automatically increase the number of genes in the evolving models.

Generations without Change

The period of time you think acceptable for evolution to occur without improvement in best fitness, after which a mass extinction is created or a neutral gene (an extra term) is automatically added to all evolving models.

Number of Tries

The number of consecutive evolutionary epochs (defined by the parameter Generations Without Change) you will allow before a neutral gene is introduced in all evolving models.

Max. Complexity

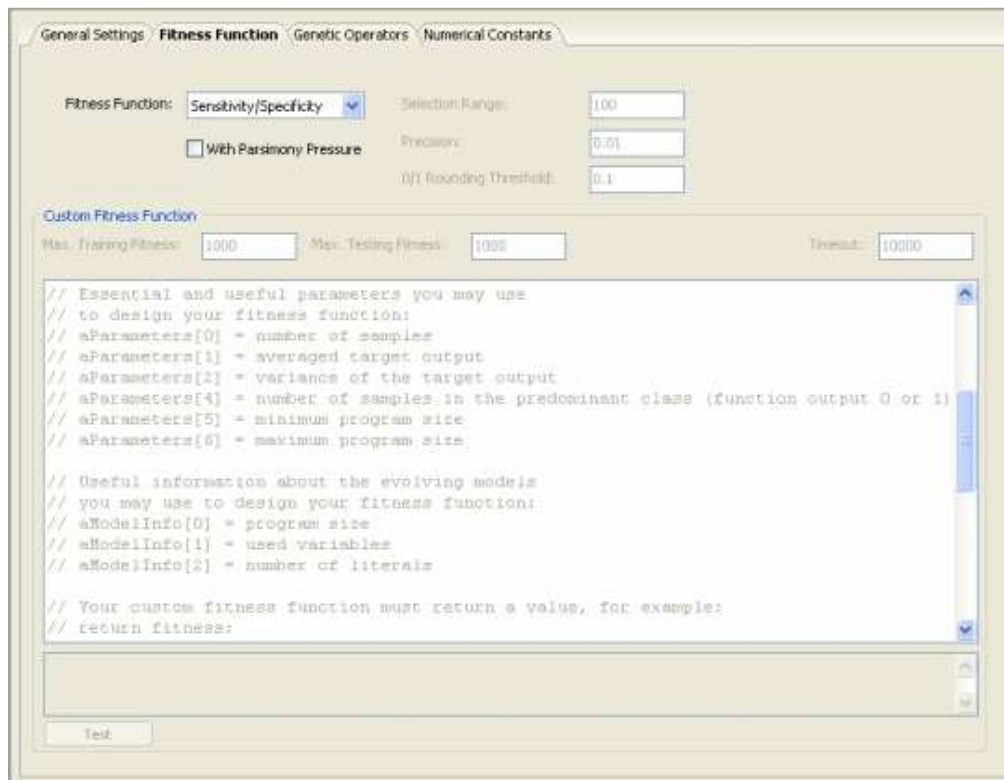
The maximum number of terms (genes) you think reasonable for modeling your data. Beyond this threshold no other terms are introduced automatically by the Complexity Increase Engine and GeneXproTools stops evolving new models.



Settings and Features

Fitness Function Tab

The **Fitness Function Tab** gives access to the following settings and features:



Fitness Function

Here you select either one of the 30 basic [built-in fitness functions](#) of GeneXproTools 4.0 for Classification problems or a [custom fitness function](#).

With Parsimony Pressure Check Box

By checking this box, you select a fitness function with [parsimony pressure](#).

Selection Range

This box is specific of Function Finding and Time Series Prediction problems, and therefore is disabled in Logic Synthesis.

Precision

This box is specific of Function Finding and Time Series Prediction problems, and therefore is disabled in Logic Synthesis.

0/1 Rounding Threshold

This box is specific of Classification problems, and therefore is disabled in Logic Synthesis.

Custom Fitness Function Frame

Select Custom in the Fitness Function combo box in order to activate this frame. You can now design or edit your own fitness function.

Custom Fitness Function Code Window

You design or edit the code of your own fitness function in the code window. The code must be in Java Script and can be tested for syntax errors before evolving models with it.

Test

Tests the syntax of the Java Script code of your fitness function for syntax errors. The results of the test are written in the gray window below the code window.

Max. Training Fitness

The maximum fitness corresponding to the custom fitness function on the training set. If maximum fitness depends on the number of training samples, don't forget to update this value every time you change the number of training samples, otherwise you might be creating some inaccuracies or errors. For instance, the plot of the Evolutionary Dynamics on the [Run Panel](#) won't show correctly if your maximum fitness is not correctly set.

Max. Testing Fitness

The maximum fitness corresponding to the custom fitness function on the testing set. If maximum fitness depends on the number of testing samples, don't forget to update this value every time you change the number of testing samples, otherwise you might be creating some inaccuracies.

Timeout

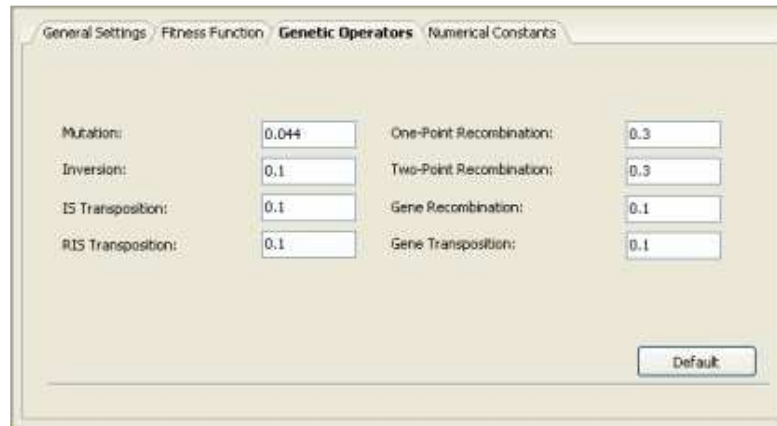
Despite one's best efforts, the code one sometimes writes contains errors that keep computers in infinite loops. This setting gives you a gracious way out of such loops.



Settings and Features

Genetic Operators Tab

The **Genetic Operators Tab** gives access to the following settings and features:



Mutation

The [mutation](#) rate used to create genetic variation. The default is 0.044.

Inversion

The [inversion](#) rate used to create genetic variation. The default is 0.1.

IS Transposition

The [IS transposition](#) rate used to create genetic variation. The default is 0.1.

RIS Transposition

The [RIS transposition](#) rate used to create genetic variation. The default is 0.1.

One-Point Recombination

The [one-point recombination](#) rate used to create genetic variation. The default is 0.3.

Two-Point Recombination

The [two-point recombination](#) rate used to create genetic variation. The default is 0.3.

Gene Recombination

The [gene recombination](#) rate used to create genetic variation. The default is 0.1.

Gene Transposition

The [gene transposition](#) rate used to create genetic variation. The default is 0.1.

Default

Resets the rates of all the above genetic operators to the default values.



Settings and Features

Numerical Constants Tab

The **Numerical Constants Tab** gives access to the following settings and features:

Use Random Numerical Constants

Check this box if you want to use the [GEP-RNC learning algorithm](#) of GeneXproTools. This algorithm has a facility for handling random numerical constants, in this case just 0's and 1's, and it evolves models with them.

Constants per Gene

The number of random numerical constants associated with each gene.

Data Type

The Data Type list box is disabled in Logic Synthesis, as GeneXproTools automatically sets the data type to Integer.

Lower Bound

Sets the lower bound of the interval from which random numerical constants are drawn, which obviously can only take two values: 0 or 1.

Upper Bound

Sets the upper bound of the interval from which random numerical constants are drawn, which obviously can only take two values: 0 or 1.

RNC Mutation

The [RNC mutation](#) rate used to create genetic variation in the arrays of random numerical constants. The default is 0.01.

Dc Mutation

The [Dc mutation](#) rate used to create genetic variation. The default is 0.044.

Dc Inversion

The [Dc inversion](#) rate used to create genetic variation. The default is 0.1.

Dc IS Transposition

The [Dc IS transposition](#) rate used to create genetic variation. The default is 0.1.

Default

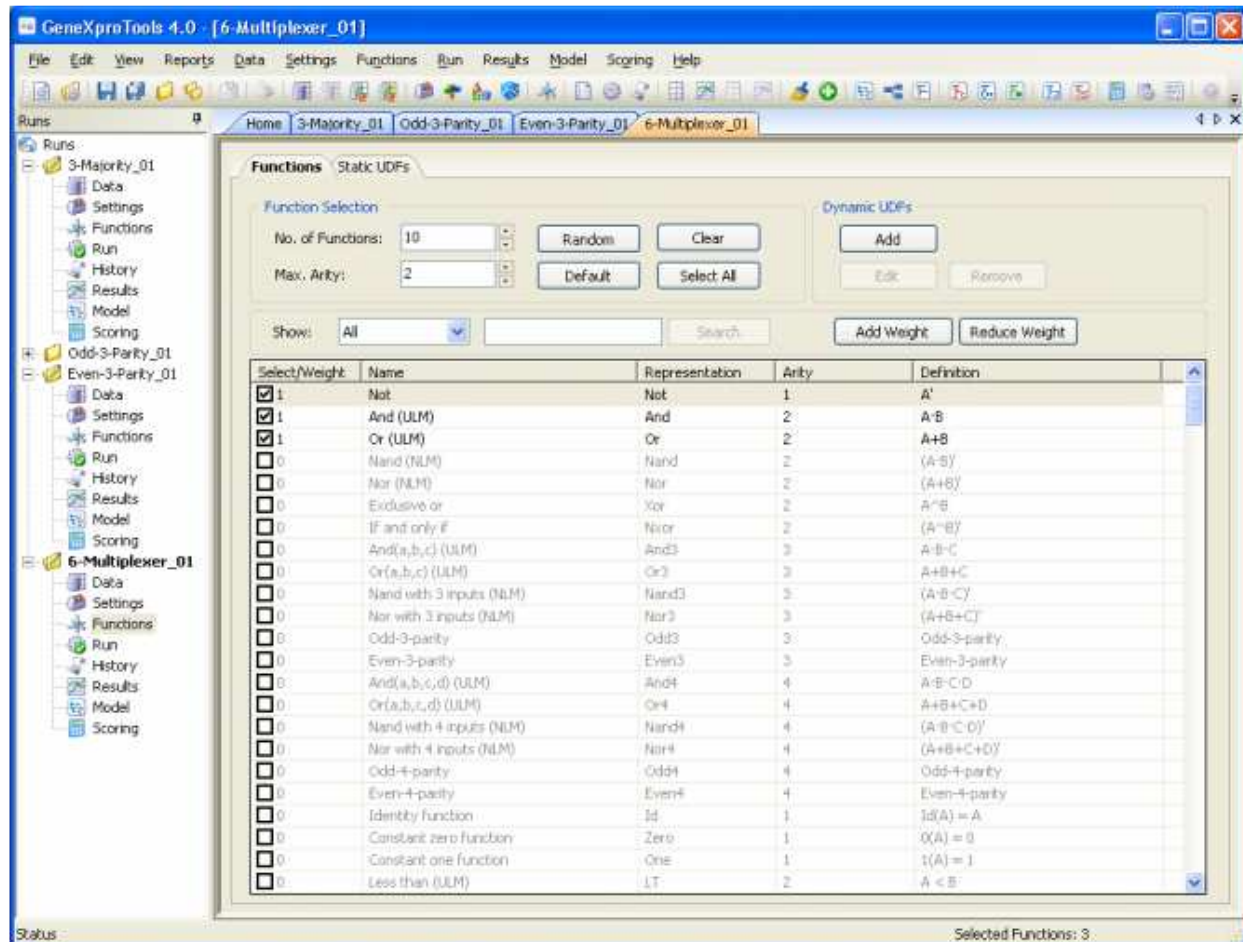
Resets the rates of all the above genetic operators to the default values.

Settings and Features

Functions Panel

The **Functions Panel** of GeneXproTools 4.0 includes the following tabs and windows:

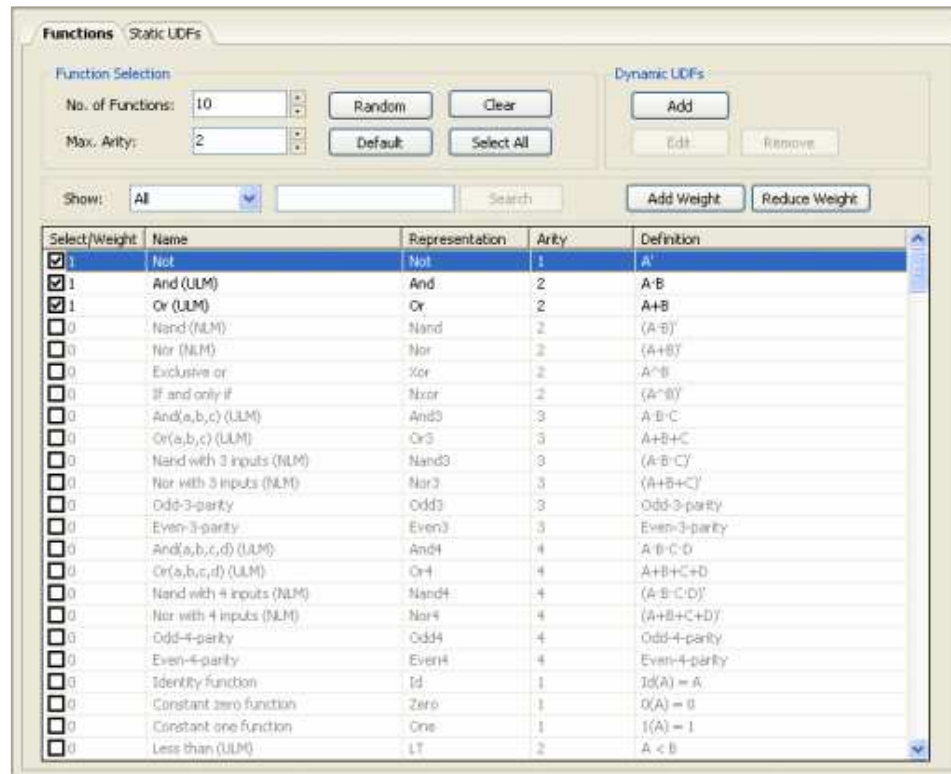
- [Functions Tab](#)
- [Functions Tab – Edit DDF Window](#)
- [Static UDFs Tab](#)
- [Static UDFs Tab – Edit UDF Window](#)



Settings and Features

Functions Tab

The **Functions Tab** lists all the 258 [built-in logical functions](#) of GeneXproTools and gives access to the following settings and features:



Function Selection Framework

Allows the selection of different function sets very quickly through the combination of the Show options with the Random/Default/Clear/Select All buttons plus the Increase/Reduce Weight buttons.

Number of Functions

The number of functions with a certain arity you wish to select randomly by using the Random button.

Maximum Arity

Determines the sub-set of functions from which the Random button will draw a specified number of functions. For instance, by setting maximum arity to 1, only functions of one argument will be drawn; by setting maximum arity to 2, only functions of 1 and 2 arguments will be randomly chosen, and so forth.

Random Button

Selects randomly a specified number of functions belonging to four different sub-sets of functions: functions of just one argument, functions of 1-2 arguments, functions of 1-3 arguments, and functions of 1-4 arguments.

Default Button

Adds the function set of the [run template](#) to the current function set. To revert exactly to the default function set you must press the Clear button before pressing Default.

Clear Button

Clears all the functions from the function set.

Select All Button

Selects all the functions showing currently on the table.

Show List Box

Shows a wide variety of function sub-sets (All functions, Searched functions, Selected Only, Arity 1 Only, Arity 2 Only, Arity 3 Only, Arity 4 Only, and DDFs Only). By selecting Search, the search box

becomes activated and you can enter specific words to find functions that match your criteria in the Name column.

Search Button

Searches, in the Name column, for the functions matching your search words.

Add Weight

Increments by one the weight of each selected function.

Reduce Weight

Reduces by one the weight of each selected function; functions with weight 1, though, remain as they are.

Select/Weight Column

To select a function, check the box on the left. By doing this, the function is included in your function set with the default weight of 1. To increase its weight, click the box below Select/Weight and then introduce its weight.

Name Column

Under this heading is shown the common name of all the [logical functions](#) used by GeneXproTools.

Representation Column

Under this heading is shown how all the functions are represented in [Karva notation](#).

Arity Column

Under this heading is shown the arity (number of arguments) of all the functions.

Definition Column

Under this heading is shown the logical definition of all GeneXproTools functions and the definition you gave to all the [DDFs](#).

Dynamic UDFs Framework

Allows the introduction of [dynamic UDFs](#) in your function set.

Add Button

Opens the [Edit DDF window](#) for designing a new [DDF](#).

Edit Button

Opens the [Edit DDF window](#) for editing an existing [DDF](#). To edit a DDF, you must first select the DDF you want to modify, and then click the Edit button.

Remove Button

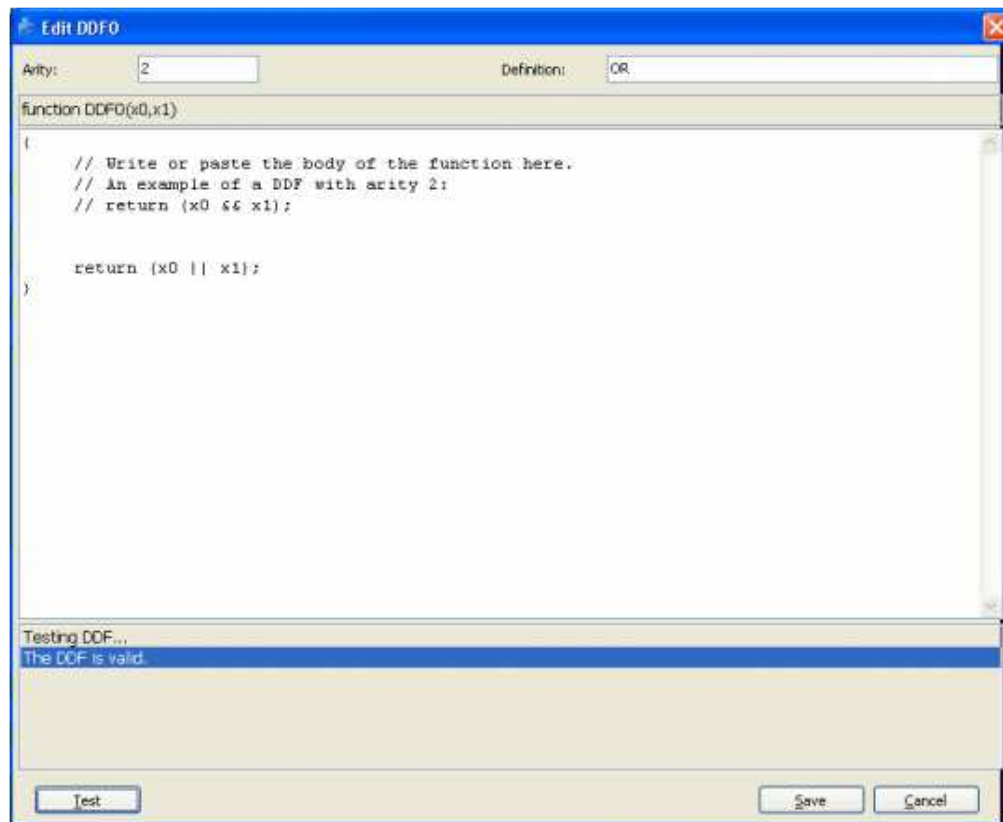
Removes an existing [DDF](#). To remove a DDF, you must first select the DDF you want to remove, and then click the Remove button. A less radical way of removing a DDF, consists of unchecking this function or setting its weight to zero. This way you will be able to use it later if you wish.



Settings and Features

Functions Tab – Edit DDF Window

The **Edit DDF Window** allows you to introduce your own functions and gives access to the following settings and features:

**Arity**

The arity (number of arguments) of your [DDF](#). In GeneXproTools 4.0, the maximum allowed value is 4, and the minimum is obviously 1. After writing the arity, the function header appears in the gray box above the code window.

Definition

Use this box to write a brief description of the [DDF](#) for your future reference. The text you write here will appear in the Definition column on the [Functions Panel](#).

DDF Code Window

You design or edit the code of your own functions ([DDFs](#)) in the code window. The code must be in Java Script and can be tested for syntax errors before including it in the function set.

Test Button

Tests the syntax of the Java Script code of your [DDF](#). The results of the test are written in the gray window below the code window.

Save Button

Saves the [DDF](#) and includes it in your function set with the default weight of 1.

Cancel Button

Closes the Edit DDF window without saving the modifications made there.



Settings and Features

Static UDFs Tab

The **Static UDFs Tab** allows you to design your own [static UDFs](#) and gives access to the following settings and features:

Select/Weight	Name	Representation	Definition
☒ 1	Static UDF0	UDF0	My Function 0
☒ 1	Static UDF1	UDF1	My Function 1
☒ 1	Static UDF2	UDF2	My Function 2
☒ 1	Static UDF3	UDF3	My Function 3

Use UDFs

Check this box if you want to use or design [static UDFs](#).

Function Selection Framework

Allows the selection of different sub-sets of UDFs very quickly through the combination of the Show options with the Random/Clear/Select All buttons plus the Increase/Reduce Weight buttons.

Number of Functions

The number of functions you wish to select randomly by using the Random button.

Maximum Arity

This box is obviously disabled as all UDFs have arity zero.

Random Button

Selects randomly a specified number of functions.

Default Button

This button is obviously disabled as it does not apply to UDFs.

Clear Button

Clears all the functions from the function set.

Select All Button

Selects all the functions showing currently on the table.

Show List Box

Shows three different function sub-sets (All functions, Searched functions, and Selected Only). By selecting Search, the search box becomes activated and you can enter specific words to find functions that match your criteria in the Name column.

Search Button

Searches, in the Name column, for the functions matching your search words.

Add Weight

Increments by one the weight of each selected function.

Reduce Weight

Reduces by one the weight of each selected function; functions with weight 1, though, remain as they are.

Select/Weight Column

To select a [UDF](#), check the box on the left. By doing this, the UDF is included in your function set with the default weight of 1. To increase its weight, click the box below Select/Weight and then introduce its weight.

Name Column

Under this heading is shown the common name of all the [UDFs](#).

Representation Column

Under this heading is shown how all the [UDFs](#) are represented in [Karva notation](#).

Definition Column

Under this heading is shown the definition you gave to all the [UDFs](#).

Static UDFs Framework

Allows the introduction of [static UDFs](#) in your function set.

Add Button

Opens the [Edit UDF Window](#) for designing a new [UDF](#).

Edit Button

Opens the [Edit UDF Window](#) for editing an existing [UDF](#). To edit a UDF, you must first select the UDF you want to modify, and then click the Edit button.

Remove Button

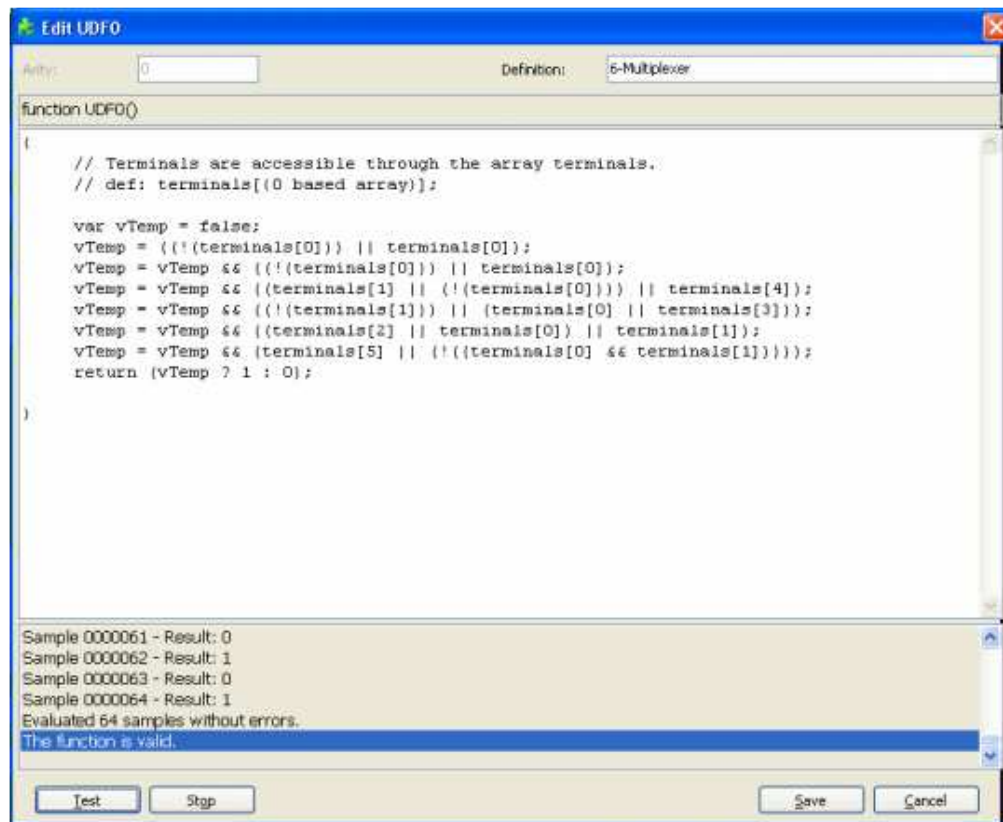
Removes an existing [UDF](#). To remove a UDF, you must first select the UDF you want to remove, and then click the Remove button. A less radical way of removing a UDF, consists of unchecking this function or setting its weight to zero. This way you will be able to use it later if you wish.



Settings and Features

Static UDFs Tab – Edit UDF Window

The **Edit UDF Window** allows you to introduce your own [static UDFs](#) and gives access to the following settings and features:



Arity

Reminds you that all the [static UDFs](#) take no parameters and, therefore, have zero arity (notice the function header that appears in the gray box above the code window).

Definition

Use this box to write a brief description of the [UDF](#) for your future reference. The text you write here will appear in the Definition column on the Functions Panel -> Static UDFs.

UDF Code Window

You design or edit the code of your [UDFs](#) in the code window. The code must be in Java Script and can be tested for syntax and calculation errors before including it in the function set.

Test Button

Tests the syntax of the Java Script code of your [UDF](#) and ensures that it is free of calculation errors. The results of the test are written in the gray window below the code window.

Stop Button

Allows you to stop the testing process.

Save Button

Saves the [UDF](#) and includes it in your function set with the default weight of 1.

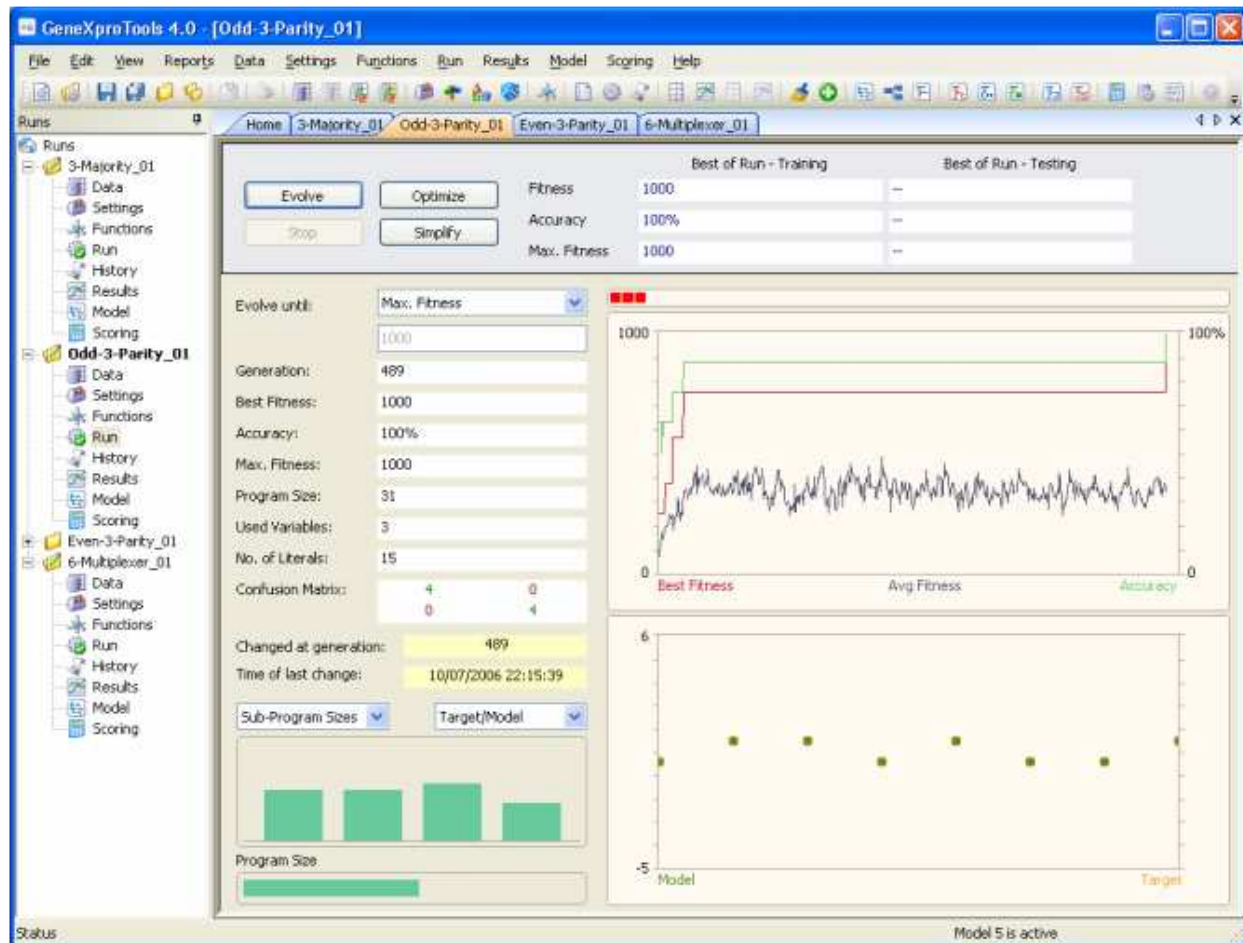
Cancel Button

Closes the Edit UDF window without saving the modifications made there.

Settings and Features

Run Panel

The **Run Panel** allows you to control and monitor the modeling process and gives access to the following settings and features:



Evolve Button

Starts the evolutionary process from scratch, that is, from a completely random initial population.

Stop Button

Stops the evolutionary process.

Optimize Button

Starts the evolutionary process with a [seed model](#), using it to breed better models.

Simplify Button

Also starts an evolutionary process with a seed model, but applies [parsimony pressure](#) to design more compact solutions.

Best of Run – Training

Under this heading are shown the values of [Fitness](#), [Accuracy](#), and Max. Fitness obtained on the training set for the best model of each run. These values are only updated at the end of each run or between Complexity Increase cycles. They are also updated if some relevant setting is changed, say, a different fitness function, a different training set, or a different model introduced through the Change

Seed window, and if the model is afterwards processed in the [Results Panel](#).

Best of Run – Testing

Under this heading are shown the values of [Fitness](#), [Accuracy](#), and Max. Fitness obtained on the testing set for the best model of each run. These values are only updated at the end of each run or between Complexity Increase cycles. They are also updated if some relevant setting is changed, say, a different fitness function, a different testing set, or a different model introduced through the Change Seed window, and if the model is afterwards processed in the [Results Panel](#). If no testing set is used, a dash is shown in all the boxes.

Evolve Until

Sets the stop condition. This stop condition is formulated using two different boxes. In the first you choose the stop condition and, in the second, you set the value for the chosen condition. There are four different stop conditions to choose from. The first allows you to evolve models for a certain number of generations. In this case, you must write in the box underneath the number of generations you want. The second allows you to evolve models until the best fitness is equal to or higher than a certain value. In this case, you must write in the box underneath this fitness value. The third allows you to evolve models until the [accuracy](#) of the best model is equal to or higher than a certain value. In this case, you must write in the box underneath this accuracy value. Obviously, only values higher than 0 and equal to or less than 100 are valid, and if an invalid value is used GeneXproTools uses the value of 100 by default. The fourth allows you to evolve models until the best fitness is equal to maximum fitness. In this case, the correct value of maximum fitness is automatically introduced by GeneXproTools in the box underneath.

Generation

Updates the generation number throughout a run.

Best Fitness

Updates the best fitness of each generation throughout a run.

Accuracy

Updates the [classification accuracy](#) of the best-of-generation model throughout a run.

Max. Fitness

Shows the value of maximum fitness for the current run.

Program Size

Updates the program size of the best-of-generation model throughout a run.

Used Variables

Updates the number of variables used in the best-of-generation model throughout a run.

No. of Literals

Updates the number of literals used in the best-of-generation model throughout a run.

Confusion Matrix

Shows the [confusion matrix](#), where [true positives](#) followed by [false negatives](#) are shown on the first row, and [false positives](#) followed by [true negatives](#) are shown on the second row. For easy reference, the true positives and true negatives are shown in green, whereas the false positives and the false negatives are shown in red.

Changed at Generation

Updates, throughout a run, the number of the generation at which a change in best fitness occurred.

Time of Last Change

Updates, throughout a run, the time when a change in best fitness occurred.

Charts List Boxes

The one on the left allows you to select three different bar plots: Sub-Program Sizes, All Sizes, and All

Fitnesses. And the one on the right allows you to select two different charts: Curve Fitting and Avg/Best Size.

Sub-Program Sizes Chart

This chart shows the sizes of all the sub-programs in the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of all sub-programs in your model. The Sub-Program Sizes chart can be visualized any time during evolution by selecting Sub-Program Sizes in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

All Sizes Chart

This chart shows the sizes of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the size of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Sizes chart can be visualized any time during evolution by selecting All Sizes in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

All Fitnesses Chart

This chart shows the fitnesses of all the models in the population. Not only after a run but also during evolution, by placing the cursor over each bar you can access the fitness of a particular model; the best-of-generation model always occupies the first position so you can also easily see how it fares relatively to the others. The All Fitnesses chart can be visualized any time during evolution by selecting All Fitnesses in the leftmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Program Size Chart

This chart shows the size of the best-of-generation model. Not only after a run but also during evolution, by placing the cursor over the horizontal bar you can access the size of the best model. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Target/Model Chart

This chart plots the first 50 data points of the training set and shows how well the evolving models are fitting the target. The Target/Model chart can be visualized any time during evolution by selecting Target/Model in the rightmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click. And by placing the cursor over each data point, you can easily identify it and access its value.

Avg/Best Size Chart

This chart plots together the average program size of the population and the program size of the best-of-generation model. This chart is especially useful during simplification and can be visualized any time during evolution by selecting Avg/Best Size in the rightmost Chart List Box. GeneXproTools allows you to copy or print this chart through the mouse right-click. You can also access the values corresponding to the Max and Min labels that sometimes appear on the Y axis by placing the cursor over them.

Variables Usage Map

This map shows not only the variables that are being used by the best-of-generation models but also their weight. Not only after a run but also during evolution, by placing the cursor over each square gives you access to the variable ID and the number of times it appears in the current model. And by clicking the right button of your mouse, you can also change its appearance: Heat Map, Random Colors, or Monochromatic. GeneXproTools allows you to copy or print this chart through the mouse right-click.

Evolutionary Dynamics Chart

This chart shows the average fitness of the population plus the [fitness](#) and [accuracy](#) of the best-of-generation model. The Y axis on the left refers to best and average fitness, whereas the Y axis on the right refers to the [classification accuracy](#). The X axis corresponds obviously to the number of generations, with a complete screen spanning a total of 500 generations. For a good adaptation, the plot for average fitness should never come near the plot for best fitness, otherwise the system is losing

genetic diversity and becoming too uniform for an efficient evolution. Whenever this is happening in your system, you should obviously increase the [genetic diversity](#) of the evolving populations by choosing the right set of genetic operators and adjusting their rates. And as usual, GeneXproTools allows you to copy or print this chart through the mouse right-click.



Settings and Features

History Panel

The **History Panel** lists all the best-of-generation models created in a run and gives access to the following settings and features:

The screenshot shows the GeneXproTools 4.0 interface with the History Panel open. The panel displays a table of models with columns for Model, Generation, Program Size, Training Fitness, Testing Fitness, Training Accuracy, and Testing Accuracy. Model 57 is selected and highlighted.

Model	Generation	Program Size	Training Fitness	Testing Fitness	Training Accuracy	Testing Accuracy
Model 32	150	46	738.404690625	--	85.94%	--
Model 33	165	45	738.40528125	--	85.94%	--
Model 34	186	44	738.405871875	--	85.94%	--
Model 35	220	40	765.7566875	--	87.50%	--
Model 36	271	39	765.7573	--	87.50%	--
Model 37	308	37	765.758525	--	87.50%	--
Model 38	315	29	765.763425	--	87.50%	--
Model 39	317	26	765.7652625	--	87.50%	--
Model 40	577	36	820.45621875	--	90.63%	--
Model 41	591	35	820.456875	--	90.63%	--
Model 42	873	35	875.154	--	93.75%	--
Model 43	974	34	875.1547	--	93.75%	--
Model 44	1793	31	875.1568	--	93.75%	--
Model 45	1975	29	875.1582	--	93.75%	--
Model 46	2031	27	875.1596	--	93.75%	--
Model 47	2094	25	875.161	--	93.75%	--
Model 48	2971	35	937.665	--	96.88%	--
Model 49	2977	48	968.910425	--	98.44%	--
Model 50	3000	37	1000.1744	--	100%	--
Model 51	3002	36	1000.1752	--	100%	--
Model 52	3008	34	1000.1768	--	100%	--
Model 53	3076	33	1000.1776	--	100%	--
Model 54	3164	31	1000.1792	--	100%	--
Model 55	3254	29	1000.1808	--	100%	--
Model 56	3588	28	1000.1816	--	100%	--
Model 57	4683	28	1000.1816	--	100%	--

Test All Button

Evaluates the performance of all the models in the History Panel on the testing set, showing their respective [fitnesses](#) and [accuracies](#). If no testing set is used, then this button is disabled.

Test Current Button

Evaluates the performance of the selected model on the testing set, showing its [fitness](#) and [accuracy](#). If no testing set is used, then this button is disabled.

Refresh All Button

Evaluates the performance of all the models in the History Panel both on the training and testing sets, and updates all the [fitnesses](#) and [accuracies](#). This button is particularly useful when you have changed the fitness function or the training set after a run and would want to evaluate the performance on the training set with this new fitness function or new training set.

Best of Run – Training

This heading is used to identify the best models of the run on the training set in terms of [fitness](#) and [accuracy](#), showing also the exact values of their fitness and accuracy. For reference, the value of Max. Fitness on the training set is also shown.

Best of Run – Testing

This heading is used to identify the best models of the run on the testing set in terms of [fitness](#) and [accuracy](#), showing also the exact values of their fitness and accuracy. For reference, the value of Max. Fitness on the testing set is also shown. If no testing set is used, then a dash is shown in all the boxes under this heading.

Model Column

In this column are shown the names of all the best-of-generation models created in the run. By clicking a model, you are making it your active model. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order.

Generation Column

In this column is shown the number of the generation in which the model was created. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order.

Program Size Column

In this column is shown the program size of all the best-of-generation models. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their program sizes.

Training Fitness Column

In this column is shown the [fitness](#) obtained by each model on the training set. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their training fitnesses.

Testing Fitness Column

In this column is shown the [fitness](#) obtained by each model on the testing set. Note that, after a run, only the fitness of the best-of-run model is shown; all the others are set to zero. In order to update these values, you must either click on the Test All or Test Current buttons. If no testing set is used, though, a dash is shown in all the entries under this heading. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their testing fitnesses.

Training Accuracy Column

In this column is shown the [accuracy](#) obtained by each model on the training set. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their training accuracies.

Testing Accuracy Column

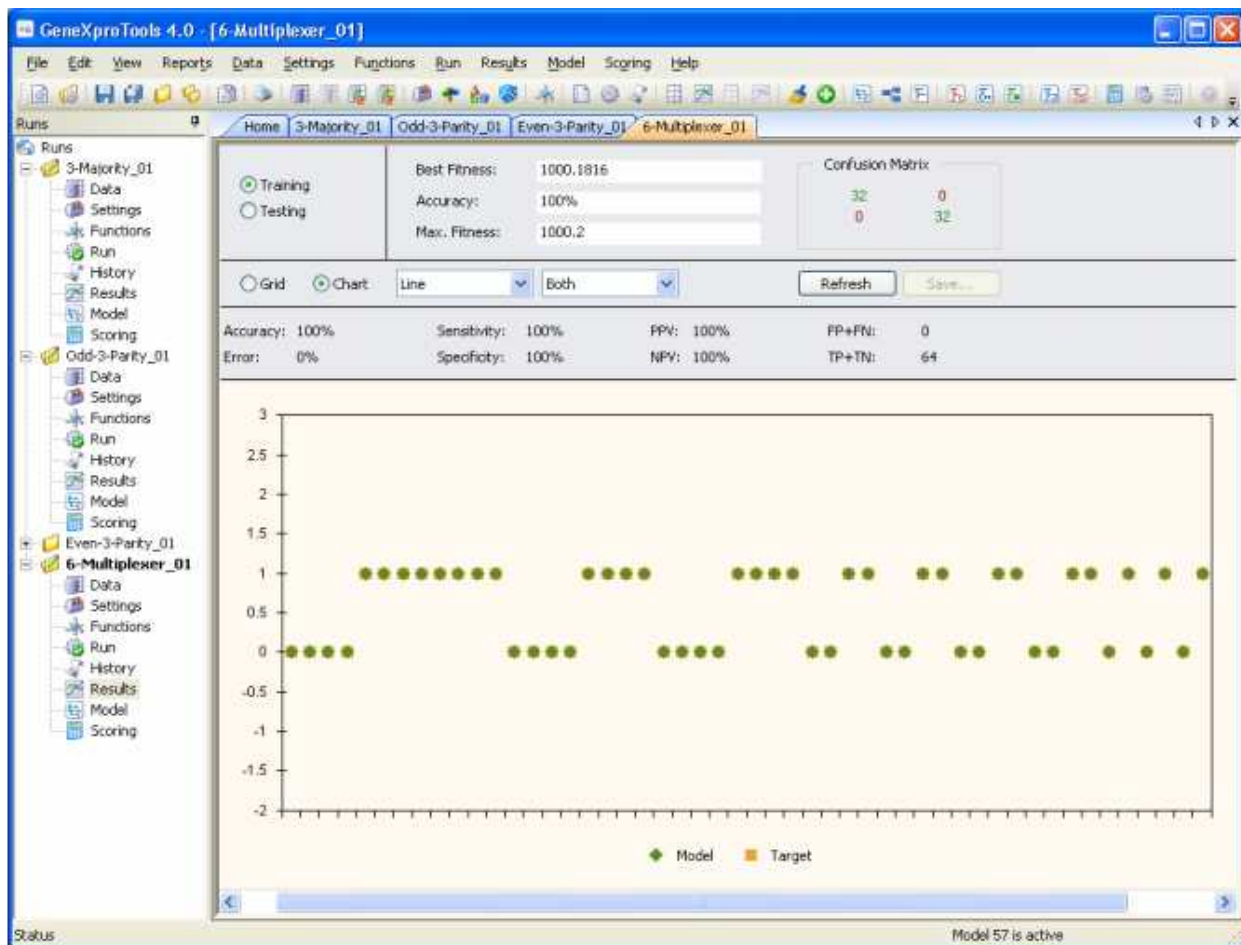
In this column is shown the [accuracy](#) obtained by each model on the testing set. Note that, after a run, only the accuracy of the best-of-run model is shown; all the others are set to zero. In order to update these values, you must either click on the Test All or Test Current buttons. If no testing set is used, though, a dash is shown in all the entries under this heading. By clicking the column heading, GeneXproTools allows you to sort the models in ascending or descending order of their testing accuracies.

Settings and Features

Results Panel

The **Results Panel** allows you to analyze and evaluate the performance of your models and includes the following options:

- [Training & Testing Results](#)
- [Grid Option](#)
- [Chart Option](#)



Settings and Features

Training & Testing Results

The **Training & Testing Results Windows** allow you to visualize and evaluate the accuracy of your models and gives access to the following settings and features:



Training Option

By selecting this option you have access to all the [statistical evaluations](#) of your model on the training set. You also can compare the output of your model with the target output on the training set using either a [table](#) or different [charts](#).

Testing Option

By selecting this option you have access to all the [statistical evaluations](#) of your model on the testing set. You also can compare the output of your model with the target output on the testing set using either a [table](#) or different [charts](#).

Best Fitness

Shows the [fitness](#) of the active model on the current set.

Accuracy

Shows the [classification accuracy](#) of the active model on the current set.

Max. Fitness

Shows the value of [maximum fitness](#) on the current set.

Confusion Matrix

Shows the [confusion matrix](#), where [true positives](#) followed by [false negatives](#) are shown on the first row, and [false positives](#) followed by [true negatives](#) are shown on the second row. For easy reference, the true positives and true negatives are shown in green, whereas the false positives and the false negatives are shown in red.

Grid Option

By selecting the [Grid Option](#) you can compare the output of your model with the target output on the current set on a spreadsheet where the target and model output are placed side by side.

Chart Option

By selecting the [Chart Option](#) you can compare the output of your model with the target output on the current set using different charts. You choose the chart type with the aid of two boxes. In the first you select the chart type (Line, Area, Bar, Step, 3D Line, 3D Area, 3D Bar, or 3D Step). And in the second you select the data you want to plot (Model, Target, or Both). The plots show only a maximum of 50 points at a time and, therefore, for datasets with more than 50 samples you must move the scroll bar at the bottom of the chart to see all the results. Then, by placing the cursor over the chart, GeneXproTools shows you the exact range used to draw the plot.

Refresh Button

Refreshes the panel. This button is particularly useful when you change your model in the [Change Seed window](#) or change the current set or the fitness function.

Save Button

This button is only active when the [Grid Option](#) is selected as it saves all the results in the table to a file.

Accuracy

Shows the [classification accuracy](#) on the current set.

Error

Shows the value of the [classification error](#) on the current set.

Sensitivity

Shows the value of the [sensitivity](#) on the current set.

Specificity

Shows the value of the [specificity](#) on the current set.

PPV

Shows the value of the [positive predictive value](#) on the current set.

NPV

Shows the value of the [negative predictive value](#) on the current set.

FP+FN

Shows the number of samples wrongly classified on the current set, which corresponds obviously to the sum of [false positives](#) and [false negatives](#). Wrongly classified samples are easily identified by a yellow strip and the identifier WRONG in the table.

TP+TN

Shows the number of samples correctly classified on the current set, which corresponds obviously to the sum of [true positives](#) and [true negatives](#).



Settings and Features

Grid Option

The **Grid Option** allows you to compare the output of your model with the target output and is organized as follows:

☒ Training ☐ Testing		Best Fitness: 1000.1816 Accuracy: 100% Max. Fitness: 1000.2	Confusion Matrix 32 0 0 32	
☒ Grid ☐ Chart		Line Both	Refresh Save...	
Accuracy: 100% Error: 0%		Sensitivity: 100% Specificity: 100%	PPV: 100% NPV: 100%	FP+FN: 0 TP+TN: 64
	Target	Model	Residual	
7	0	0	0	
8	0	0	0	
9	1	1	0	
10	1	1	0	
11	1	1	0	
12	1	1	0	
13	1	1	0	
14	1	1	0	
15	1	1	0	
16	1	1	0	
17	0	0	0	
18	0	0	0	
19	0	0	0	
20	0	0	0	
21	1	1	0	
22	1	1	0	
23	1	1	0	
24	1	1	0	
25	0	0	0	
26	0	0	0	
27	0	0	0	
28	0	0	0	
29	1	1	0	
30	1	1	0	

Target

Under this heading is shown the value of the dependent variable (Target) for all the records on the current data set.

Model

Under this heading is shown the output of the model (Model) for all the records on the current data set.

Residual

Under this heading is shown the absolute value of the difference between the Target and the Model. For the sake of clarity, though, when the residual is 1 the identifier WRONG is shown instead.



Settings and Features

Chart Option

The **Chart Option** allows you to compare visually the output of your model with the target output using different kinds of curve fitting charts:

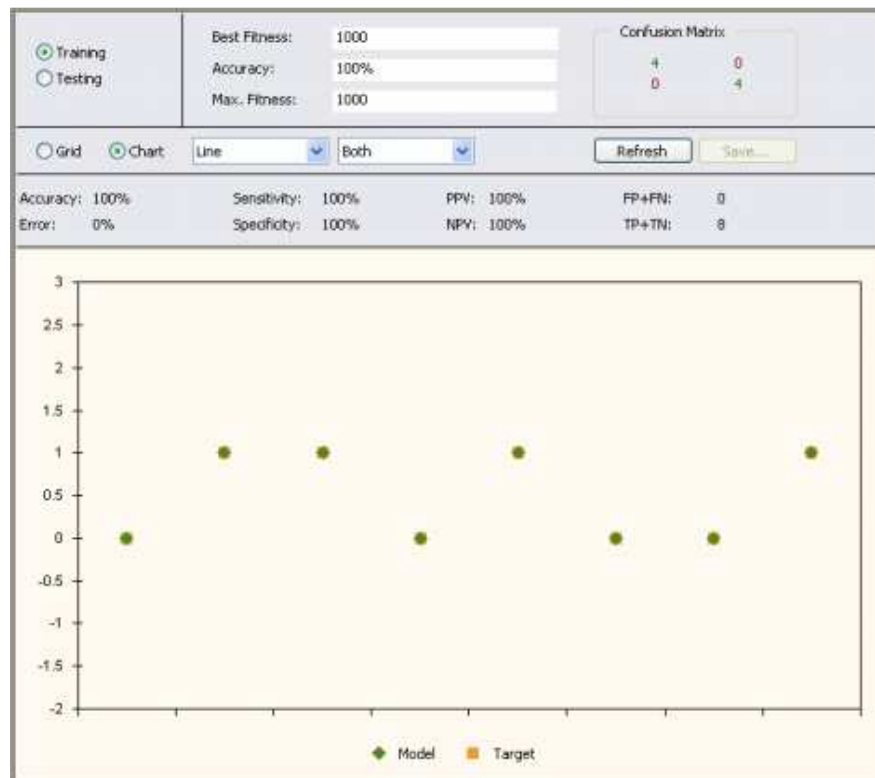


Chart Type Box

In this list box you select the chart type: Line, Area, Bar, Step, 3D Line, 3D Area, 3D Bar, or 3D Step.

Chart Data Box

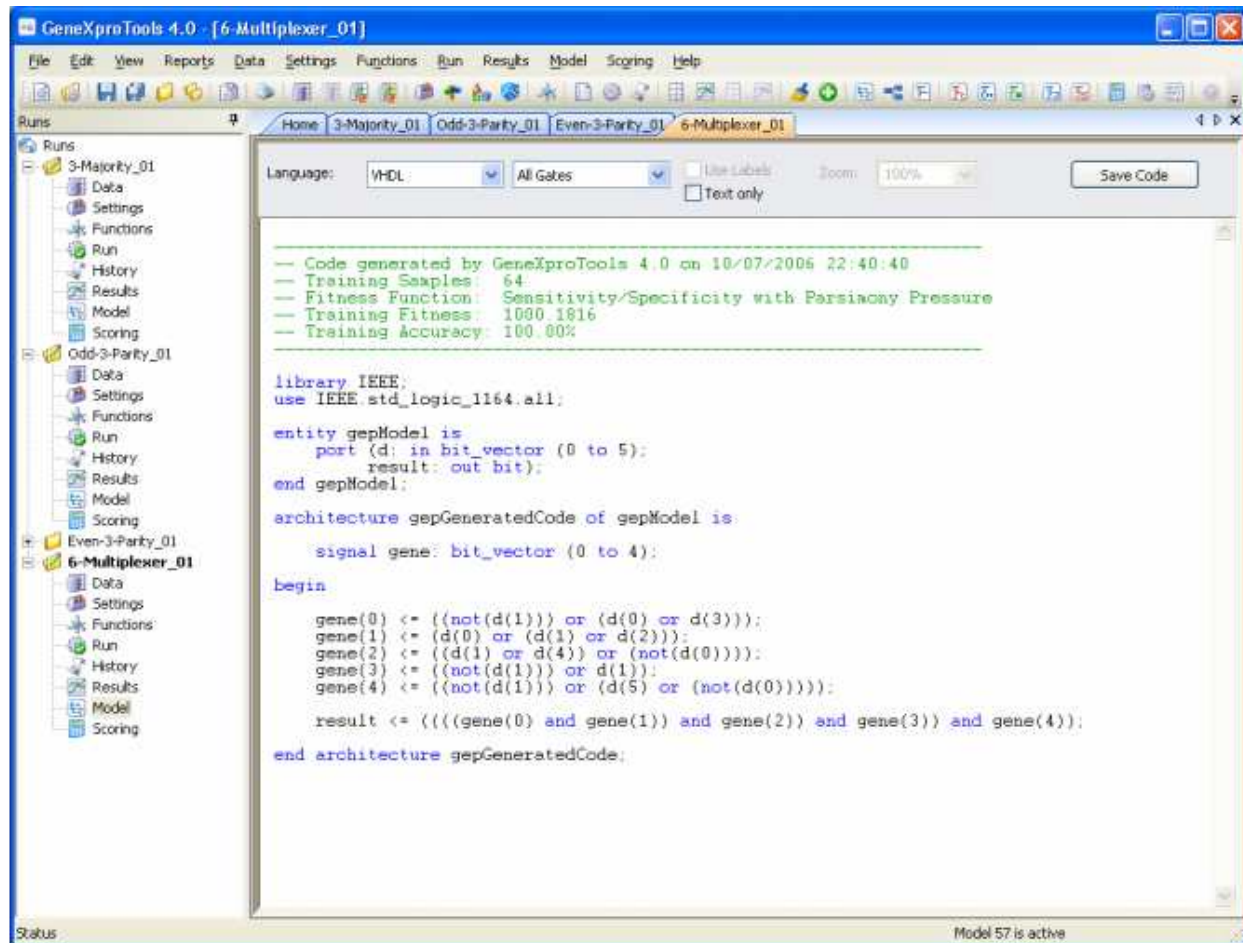
In this list box you select the data you want to plot: Model, Target, or Both. In case of calculation errors, the sample with the calculation error is skipped and a discontinuity is observed in the plot. The plots show only a maximum of 50 points at a time and, therefore, for datasets with more than 50 samples you must move the scroll bar at the bottom of the chart to see all the results. Then, by placing the cursor over the chart, GeneXproTools shows you the exact range used to draw the plot.



Settings and Features

Model Panel

The **Model Panel** allows you to see the structure of your models and translate them into a wide range of [programming languages](#). It gives access to the following settings and features:



Language

In the list box on the left, you can choose different ways of representing your models, including the native [Karva code](#), a universal diagram representation ([Expression Tree](#)), 16 different [programming languages](#) (Ada, C, C++, C#, Fortran, Java, Java Script, Matlab, Pascal, Perl, PHP, Python, Visual Basic, VB.Net, Verilog, and VHDL) originally packaged with GeneXproTools 4.0, and any additional languages created by you through the use of [User Defined Grammars](#). On the list box on the right and if Karva is selected, you can choose either [Genes](#) or [K-expressions](#). In both cases, though, the elements in the [heads](#) are shown in blue. And when any of the [programming languages](#) GeneXproTools supports is selected, you can choose to express your logic circuits using either the basic [All Gates Grammars](#) or five different [logical systems](#): Not-And-Or Only, Nand Only, Nor Only, Mux System, and Reed-Muller System.

Use Labels

This option is only active when you are using data sets with labels, that is, when the columns with each variable are identified by a name. By checking this box, the variables in your code, which are usually represented by the generic $d_0, d_1, \dots$ are replaced by their respective names. And for the sake of clarity, this option is disabled when either Karva or Expression Tree are selected as, in both those cases, the generic representation is always used.

Text Only

By checking this box the default HTML is replaced by simple unformatted text.

Zoom

This option is only active when you are visualizing Expression Trees. It allows you to either choose pre-set values to resize the expression trees (200%, 150%, 100%, 85%, 75%, 50%, and 25%) or enter any other value up to 500%.

Save Code

Saves the selected code to an appropriate format: [Karva code](#) is saved as .txt, [Ada code](#) as .ada, [C code](#) as .c, [C++ code](#) as .hpp, [C# code](#) as .cs, [Fortran code](#) as .f, [Java code](#) as .java, [Java Script code](#) as .js, [Matlab code](#) as .m, [Pascal code](#) as .pas, [Perl code](#) as .pl, [PHP code](#) as .php, [Python code](#) as .py, [Visual Basic code](#) as .bas, [VB.Net code](#) as .vb, [Verilog code](#) as .v, and [VHDL code](#) as .vhd.

Save Image

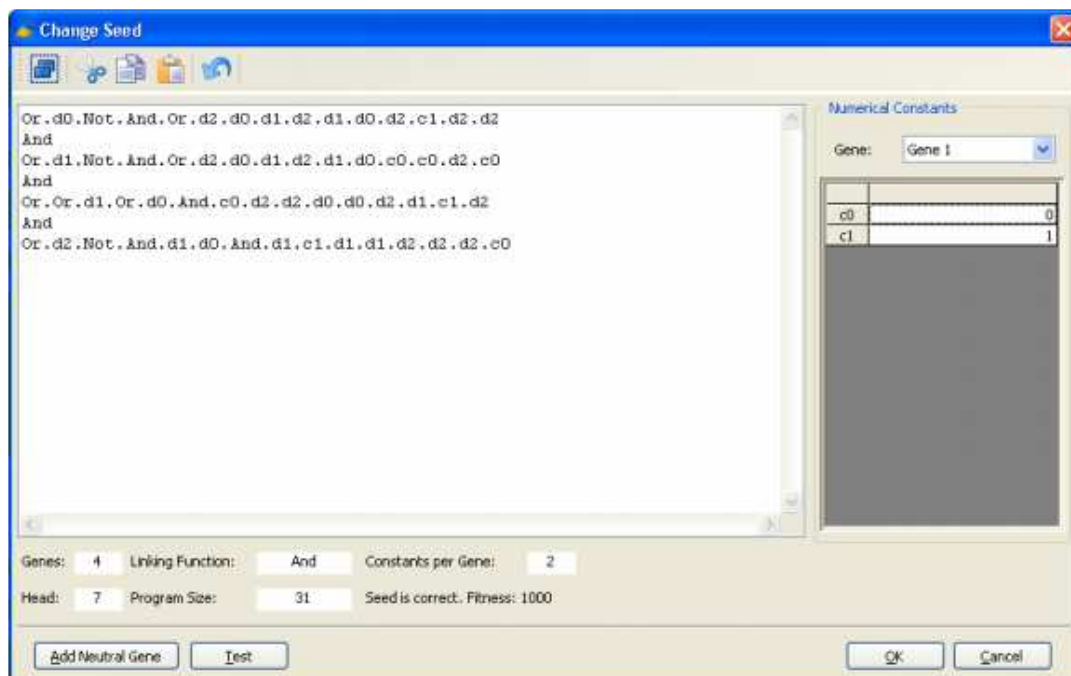
Saves the [Expression Trees](#) as .bmp files.



Settings and Features

Change Seed Window

The **Change Seed Window** allows you to change your model or introduce a completely new one. It gives access to the following settings and features:

**Seed Code Window**

You design or edit the code of your seed model in the code window. The code must be in [Karva notation](#) and can be tested for structural errors before saving it as seed.

Genes

Shows the current [number of genes](#) in your seed model.

Head

Shows the [head](#) size of your seed model.

Linking Function

Shows the current [linking function](#) in your seed model.

Program Size

Shows the actual size of your seed model, that is, the total number of nodes in all the [sub-ETs](#).

Constants per Gene

This box only appears if [random numerical constants](#) are being used. It shows the number of random numerical constants used per gene in your seed model.

Numerical Constants Framework

This frame only appears if [random numerical constants](#) are being used. Here you have access to all the numerical constants of your seed model by selecting the gene number in the Gene list box. The numerical constants appear in a table where you can edit them. Obviously, only 0's and 1's are acceptable.

Add Neutral Gene Button

Adds automatically a neutral gene to your seed model. Neutrality is important for an efficient evolution and, by introducing a neutral gene, you are giving the [learning algorithm](#) more room to play and, hopefully, a better, more complex program will evolve.

Test Button

Tests both the structural soundness and the [fitness](#) of your seed model. It also updates the program size when changes have been made on the seed.

OK Button

Saves the current seed as your seed model so that you can breed new and better models with it. GeneXproTools only saves viable seeds and warns you if there is a structural problem with your seed or if it has zero fitness.

Cancel Button

Cancels all the action on the change seed window.

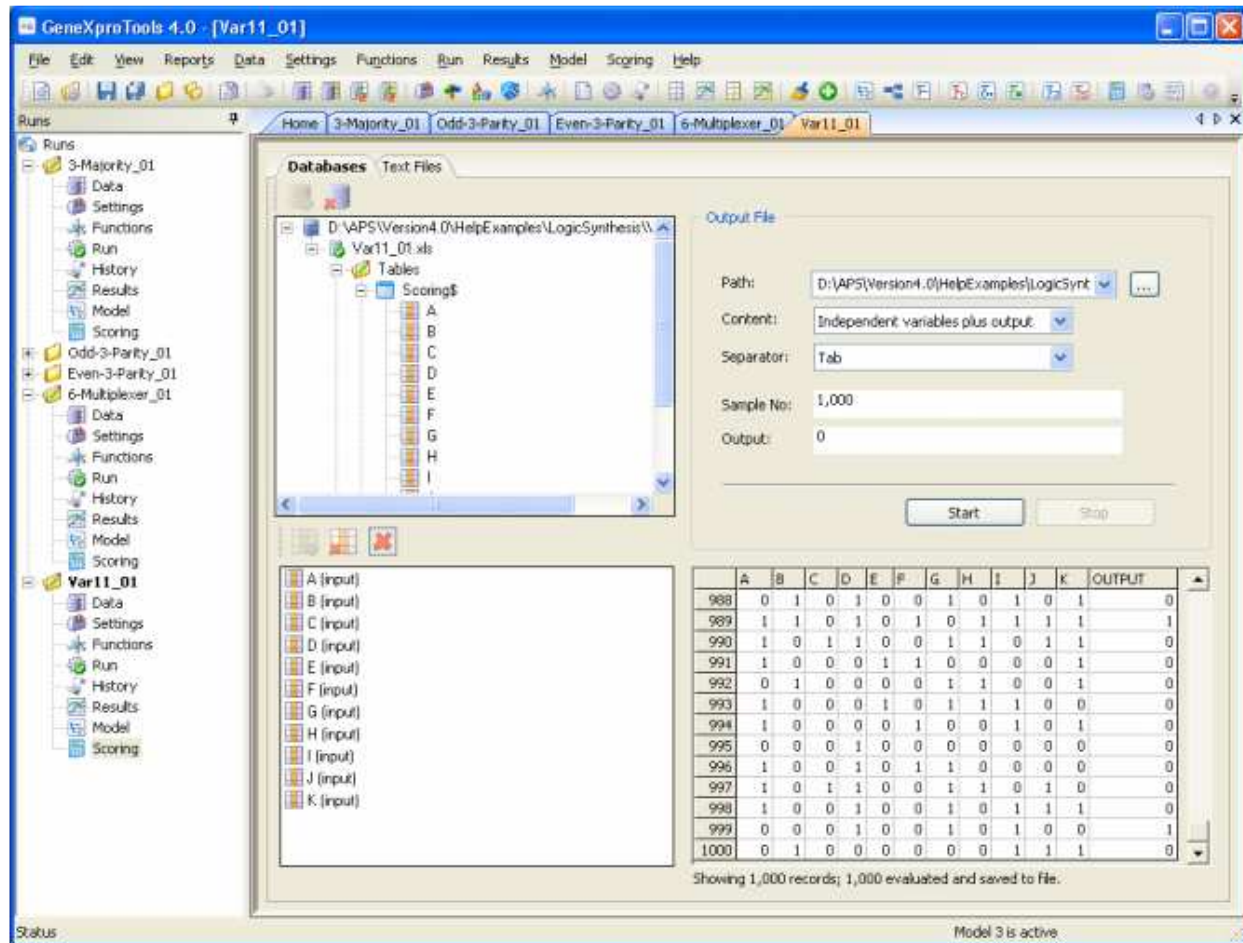


Settings and Features

Scoring Panel

The **Scoring Panel** allows you to score your models immediately without leaving the GeneXproTools modeling environment. It includes the following tabs and windows:

- [Text Files Tab](#)
- [Databases Tab](#)
- [Data Source Dialog Box - Excel & Databases](#)



Settings and Features

Text Files Tab

The **Text Files Tab** allows you to score text files and gives access to the following settings and features:

	A	B	C	D	E	F	G	H	I	J	K	OUTPUT
1	0	1	0	1	1	0	0	0	0	1	1	0
2	0	1	0	0	0	0	0	1	1	1	0	1
3	0	0	0	1	0	1	1	1	1	0	0	1
4	1	1	0	1	0	0	1	1	0	0	0	0
5	1	1	1	0	0	0	0	1	1	0	1	1
6	1	1	0	0	1	0	0	0	1	1	0	1
7	1	0	0	0	1	1	1	0	0	0	1	0
8	1	0	1	1	1	0	1	1	1	0	1	1
9	0	0	0	1	0	1	1	0	0	0	1	0
10	1	1	1	1	1	1	1	0	0	1	0	0
11	1	1	0	1	1	1	1	0	0	0	1	0
12	0	0	0	1	0	1	0	1	1	1	1	1
13	0	1	0	1	0	0	0	0	0	1	0	0

Showing 1,000 records; 1,000 evaluated and saved to file.

Source File Framework

In the Source File Framework you enter all the information concerning the source file.

Source File Path

Click the button on the right of the list box to load the scoring file.

Data Matrix Format

Choose the data matrix format from the list box. If your samples are in rows and variables in columns, you must select the Samples x Variables format. If your samples are in columns and variables in rows, you must select the Gene Expression Matrix format commonly used in DNA microarrays studies. Recall that only the values of False and True (case insensitive) or "0" and "1" are valid as input.

Source File Separator

GeneXproTools identifies automatically the separator in the training file and shows it in the combo box under this label. And if the variables in your training data are identified by a name, GeneXproTools also ticks the box on the right.

Output File Framework

In the Output File Framework you enter all the information concerning the output file.

Output File Path

You can either enter the path for the output file or click the button on the right of the list box to browse the path for the output file.

Output File Content

Here you decide whether you want to include in the output file the columns with the independent variables or just the output.

Output File Separator

Here you choose the separator you want to use in the output file (this is only relevant if you'd choose also to include in the output file the columns with the independent variables).

Sample Number

Shows the number of the sample being processed.

Output

Shows the output obtained for the current sample.

Start Button

Starts the scoring process.

Stop Button

Stops the scoring process.

Scoring Results Table

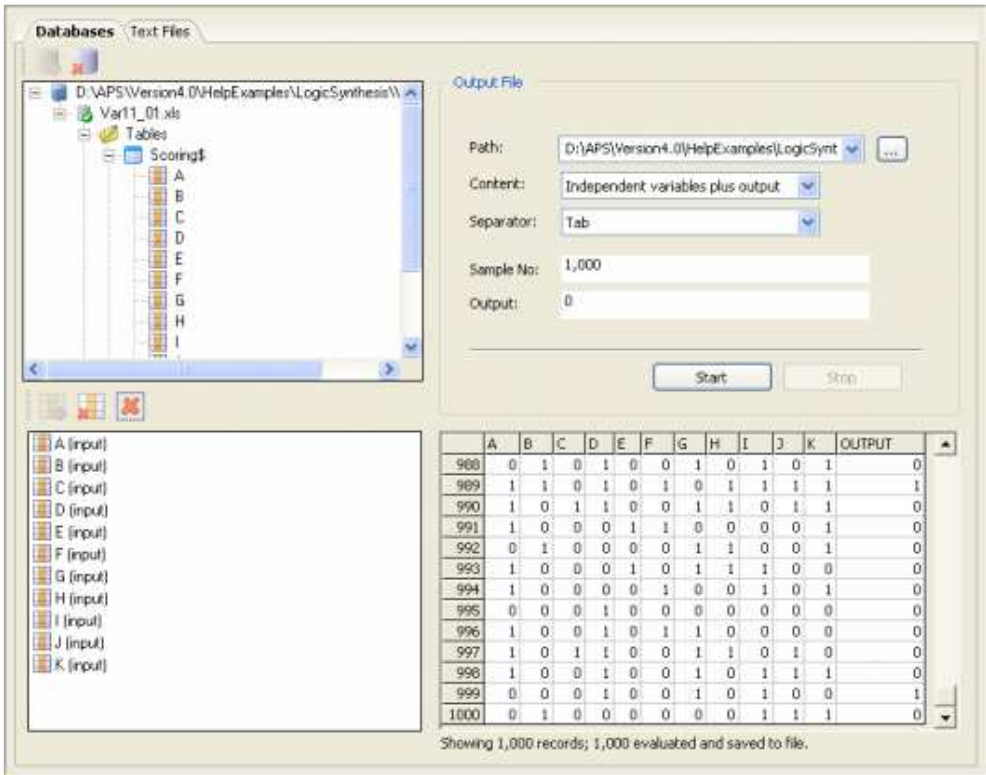
GeneXproTools saves the scoring results to a file, but for small datasets (up to 16 variables and 2000 samples), GeneXproTools also shows the scoring results in their entirety on the Scoring Panel for an immediate assessment. For datasets with more than 16 variables, though, only a single column with the output is shown.



Settings and Features

Databases Tab

The **Databases Tab** allows you to score databases and Excel files and gives access to the following settings and features:



Add Data Source Icon

Opens the [Data Source Dialog Box](#).

Remove Data Source Icon

Removes the data source.

Data Source Window

Shows the data source with its tables and columns. Columns and entire tables can be dragged and dropped onto the Scoring Data Window below.

Add Column Icon

Adds a column selected from the Data Source Window above.

Remove Column Icon

Removes a selected column from the dataset.

Remove All the Columns Icon

Removes all the columns from the Scoring Data Window.

Scoring Data Window

You prepare your scoring data here by dragging and dropping the columns with your variables from the Data Source Window. You can also reorganize your dataset by moving a selected column up or down with the mouse. Remember, however, that your scoring dataset must have exactly the same number of inputs as there are independent variables in your training data. Remember also that only the values of False and True (case insensitive) or "0" and "1" are valid as input.

Output File Framework

In the Output File Framework you enter all the information concerning the output file.

Output File Path

You can either enter the path for the output file or click the button on the right of the list box to browse the path for the output file.

Output File Content

Here you decide whether you want to include in the output file the columns with the independent variables or just the output.

Output File Separator

Here you choose the separator you want to use in the output file (this is only relevant if you'd choose also to include in the output file the columns with the independent variables).

Sample Number

Shows the number of the sample being processed.

Output

Shows the output obtained for the current sample.

Start Button

Starts the scoring process.

Stop Button

Stops the scoring process.

Scoring Results Table

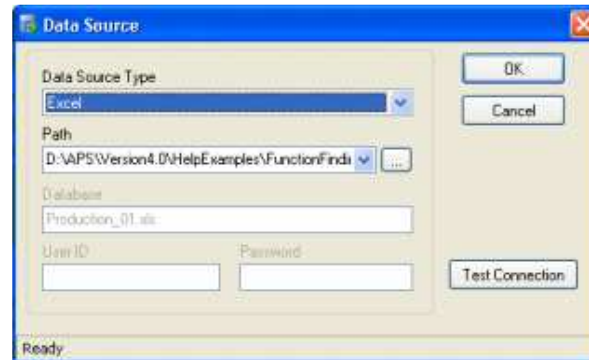
GeneXproTools saves the scoring results to a file, but for small datasets (up to 16 variables and 2000 samples), GeneXproTools also shows the scoring results in their entirety on the Scoring Panel for an immediate assessment. For datasets with more than 16 variables, though, only a single column with the output is shown.



Settings and Features

Data Source Dialog Box - Excel & Databases

The **Data Source Dialog Box** gives access to the following settings and features:



Driver

Lists the supported database drivers. This version of GeneXproTools uses specific drivers to connect to Microsoft Access, Microsoft SQL Server, Oracle and MySQL databases. To connect to other database engines GeneXproTools uses the generic ODBC driver.

Database Path/Server Name/DSN

Microsoft Excel: path to the Excel file.
 Microsoft Access: path to the database file.
 Microsoft SQL Server: server name or IP address.
 Oracle: server name or IP address.
 MySQL: server name or IP address.
 DSN: the DSN of the chosen database.

User ID

Username when needed.

Password

Username password when needed.

Database

Microsoft SQL Server - Name of the database.
 Oracle - Name and database instance (HOST/INSTANCE).
 MySQL - Name of the database (cannot contain spaces or MySQL special characters).

Test Connection Button

Tries to open a connection to the selected database.

OK Button

Adds a data source.

Cancel Button

Cancels all the action on the data source connection.

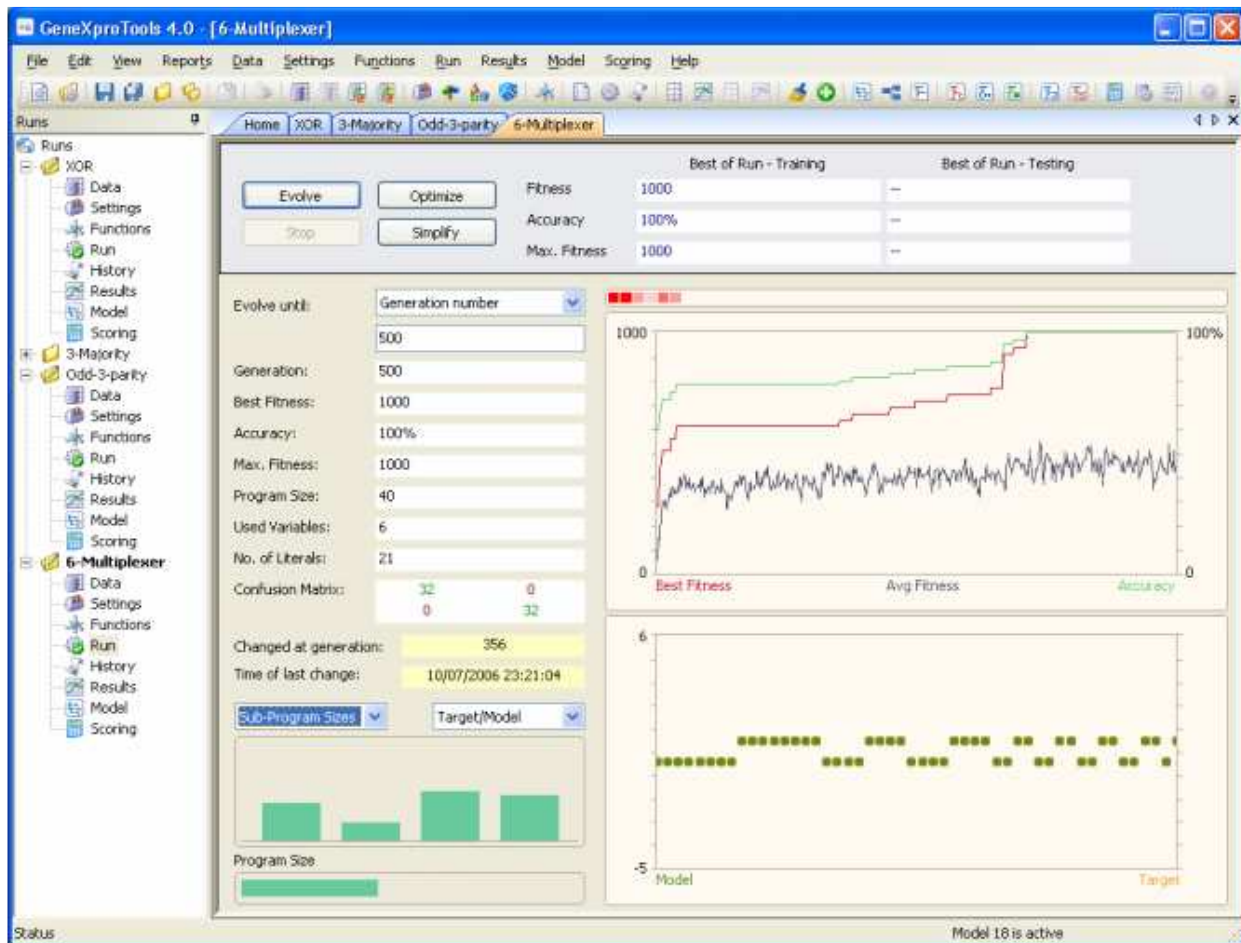


Settings and Features

Menus

For Logic Synthesis problems, GeneXproTools has the following menus:

- [File](#)
- [Edit](#)
- [View](#)
- [Reports](#)
- [Data](#)
- [Settings](#)
- [Functions](#)
- [Run](#)
- [Results](#)
- [Model](#)
- [Scoring](#)
- [Help](#)



Settings and Features

File

The **File Menu** gives access to the following sub-menus:

New

Creates a new gep file.

Open

Opens a gep file.

Save

Saves the selected run.

Save As

Saves a gep file with a different name.

Save All

Saves all open runs.

Close

Closes the selected run.

Close All

Closes all open runs.

Print

Prints the report, history, code, tree, or charts of the run.

Properties

Loads the identification form for input (Author Name and Comments).

Exit

Closes GeneXproTools.



Settings and Features

Edit

The **Edit Menu** gives access to the following sub-menus:

Copy

Copies the report, tables, code, tree, or charts of the run for pasting elsewhere.

Change Seed

Opens the [Change Seed window](#) for changing exogenous or GeneXproTools evolved models for further optimization.

Add Neutral Gene

Adds automatically a neutral gene to your seed model.

Add DDF

Opens the [Edit DDF window](#) for designing a new [DDF](#).

Add UDF

Opens the [Edit UDF window](#) for designing a new [UDF](#).

Custom Fitness Function

Goes to the [Custom Fitness Function Code window](#) for editing or designing a new [custom fitness function](#).



Settings and Features

View

The **View Menu** gives access to the following sub-menus:

Status Bar

Shows/Hides the Status Bar.

Warnings

Turns on/off warning messages.



Settings and Features

Reports

The **Reports Menu** gives access to the following sub-menus:

Home

Goes to the welcome screen of GeneXproTools.

Run Report

Goes to the [Report Panel](#) of the active run.



Settings and Features

Data

The **Data Menu** gives access to the following sub-menus:

Training Data

Goes to the [Training Set Tab](#) of the selected run.

Testing Data

Goes to the [Testing Set Tab](#) of the selected run.

Change Training Data

Opens the [Change Training Data Window](#) for changing the training set of the selected run.

Change Testing Data

Opens the [Change Testing Data Window](#) for changing the testing set of the selected run.

Add Testing Data

Opens the [Add Testing Data Window](#) for changing the testing set of the selected run.



Settings and Features

Settings

The **Settings Menu** gives access to the following sub-menus:

General Settings

Goes to the [General Settings Tab](#) of the selected run.

Fitness Function

Goes to the [Fitness Function Tab](#) of the selected run.

Genetic Operators

Goes to the [Genetic Operators Tab](#) of the selected run.

Numerical Constants

Goes to the [Numerical Constants Tab](#) of the selected run.



Settings and Features

Functions

The **Functions Menu** gives access to the following sub-menus:

Functions

Goes to the [Functions Tab](#) of the selected run.

Static UDFs

Goes to the [Static UDFs Tab](#) of the selected run.



Settings and Features

Run

The **Run Menu** gives access to the following sub-menus:

Run

Goes to the [Run Panel](#) of the active run.

Evolve

Starts a new run.

Optimize

Starts an optimization process.

Simplify

Starts a simplification process.

Variables Usage

Allows the selection of three different kinds of mapping variables usage during a run: Heat Map, Random Colors, or Monochromatic, and two different showing options: White for None and Grey for None.

Charts

Allows the selection of five different kinds of charts during a run: Target/Model comparison, Avg/Best Size, Sub-Program Sizes, All Sizes, and All Fitnesses.



Settings and Features

Results

The **Results Menu** gives access to the following sub-menus:

History

Goes to the [History Panel](#) of the active run.

Training Results Table

Goes to the [Results Panel -> Training and Grid Options](#) of the active run.

Training Results Chart

Goes to the [Results Panel -> Training and Chart Options](#) of the active run.

Testing Results Table

Goes to the [Results Panel -> Training and Grid Options](#) of the active run.

Testing Results Chart

Goes to the [Results Panel -> Training and Chart Options](#) of the active run.



Settings and Features

Model

The **Model Menu** gives access to the following sub-menus:

Expression Tree

Shows the active model as an [Expression Tree](#).

Karva

Shows the active model in the native [Karva Notation](#).

Ada

Translates the active model into [Ada](#).

C

Translates the active model into [C code](#).

C++

Translates the active model into [C++](#).

C#

Translates the active model into [C#](#).

Fortran

Translates the active model into [Fortran](#).

Java

Translates the active model into [Java](#).

JavaScript

Translates the active model into [JavaScript](#).

Matlab

Translates the active model into [Matlab](#).

Pascal

Translates the active model into [Pascal](#).

Perl

Translates the active model into [Perl](#).

PHP

Translates the active model into [PHP](#).

Python

Translates the active model into [Python](#).

Visual Basic

Translates the active model into [Visual Basic](#).

VB.Net

Translates the active model into [VB.Net](#).

Verilog

Translates the active model into [Verilog](#).

VHDL

Translates the active model into [VHDL](#).



Settings and Features

Scoring

The **Scoring Menu** gives access to the following sub-menus:

Databases

Goes to the [Scoring Panel -> Databases Tab](#) for scoring a database with the active model.

Text Files

Goes to the [Scoring Panel -> Text Files Tab](#) for scoring a database with the active model.



Settings and Features

Help

The **Help Menu** gives access to the following sub-menus:

Contents

Shows the contents of the Help File.

About GeneXproTools

Shows information about GeneXproTools.



Settings and Features

Run Templates

For Logic Synthesis problems, GeneXproTools 4.0 offers five different **Run Templates** designed so that you can start the modeling process with just a mouse click and yet obtain excellent results. In addition, they also make the discovery of good models quite enjoyable and fast as small populations of just 30 models are used in all of them.

The run templates of GeneXproTools are essentially based on the number of independent variables in your data and are organized as follows (in the Function Set, weights greater than 1 are indicated in parentheses):

Template A

Independent Variables: 1-3
Fitness Function: Sensitivity/Specificity
Function Set: And, Or, Not.
Chromosomes: 30
Genes: 3
Head Size: 7
Linking Function: Or
Random Numerical Constants: No
Constants per Gene: 2
Data Type: integer
Lower Bound: 0
Upper Bound: 1
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 5
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template B

Independent Variables: 4-8
Fitness Function: Sensitivity/Specificity
Function Set: And, Or, Not.
Chromosomes: 30
Genes: 4
Head Size: 8
Linking Function: Or
Random Numerical Constants: No
Constants per Gene: 2
Data Type: integer
Lower Bound: 0
Upper Bound: 1
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 6
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1

Dc IS Transposition Rate: 0.1

Template C

Independent Variables: 9-32

Fitness Function: Sensitivity/Specificity

Function Set: And, Or, Not.

Chromosomes: 30

Genes: 5

Head Size: 10

Linking Function: Or

Random Numerical Constants: No

Constants per Gene: 2

Data Type: integer

Lower Bound: 0

Upper Bound: 1

Complexity Increase: No

Generations without Change: 2000

Tries: 3

Maximum Complexity: 7

Mutation Rate: 0.044

Inversion Rate: 0.1

IS Transposition Rate: 0.1

RIS Transposition Rate: 0.1

One-Point Recombination Rate: 0.3

Two-Point Recombination Rate: 0.3

Gene Recombination Rate: 0.1

Gene Transposition Rate: 0.1

RNC Mutation Rate: 0.044

Dc Mutation Rate: 0.1

Dc Inversion Rate: 0.1

Dc IS Transposition Rate: 0.1

Template D

Independent Variables: 33-64

Fitness Function: Sensitivity/Specificity

Function Set: And, Or, Not.

Chromosomes: 30

Genes: 6

Head Size: 15

Linking Function: Or

Random Numerical Constants: No

Constants per Gene: 2

Data Type: integer

Lower Bound: 0

Upper Bound: 1

Complexity Increase: No

Generations without Change: 2000

Tries: 3

Maximum Complexity: 8

Mutation Rate: 0.044

Inversion Rate: 0.1

IS Transposition Rate: 0.1

RIS Transposition Rate: 0.1

One-Point Recombination Rate: 0.3

Two-Point Recombination Rate: 0.3

Gene Recombination Rate: 0.1

Gene Transposition Rate: 0.1

RNC Mutation Rate: 0.044

Dc Mutation Rate: 0.1

Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1

Template E

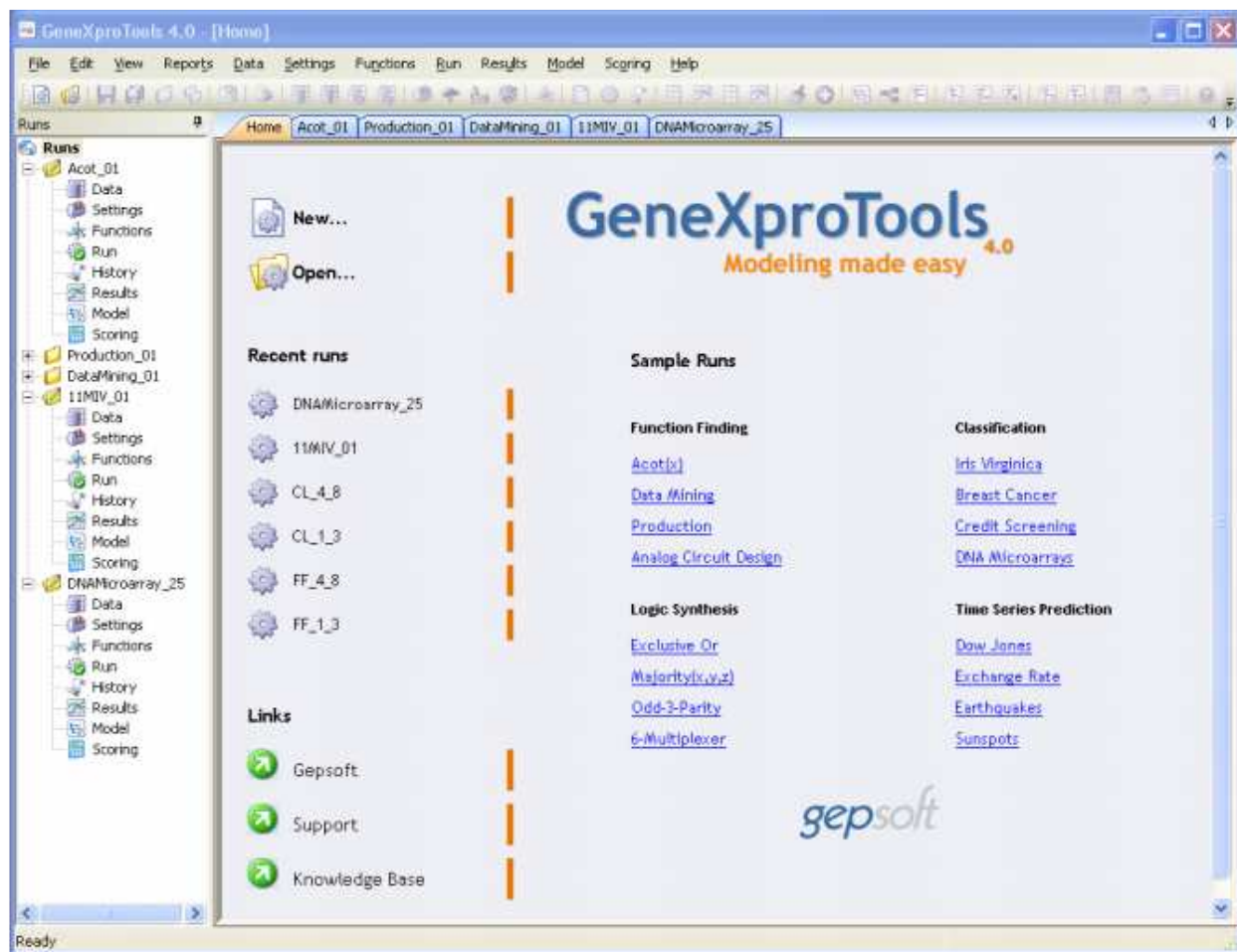
Independent Variables: 65-Unlimited
Fitness Function: Sensitivity/Specificity
Function Set: And, Or, Not.
Chromosomes: 30
Genes: 7
Head Size: 20
Linking Function: Or
Random Numerical Constants: No
Constants per Gene: 2
Data Type: integer
Lower Bound: 0
Upper Bound: 1
Complexity Increase: No
Generations without Change: 2000
Tries: 3
Maximum Complexity: 9
Mutation Rate: 0.044
Inversion Rate: 0.1
IS Transposition Rate: 0.1
RIS Transposition Rate: 0.1
One-Point Recombination Rate: 0.3
Two-Point Recombination Rate: 0.3
Gene Recombination Rate: 0.1
Gene Transposition Rate: 0.1
RNC Mutation Rate: 0.044
Dc Mutation Rate: 0.1
Dc Inversion Rate: 0.1
Dc IS Transposition Rate: 0.1



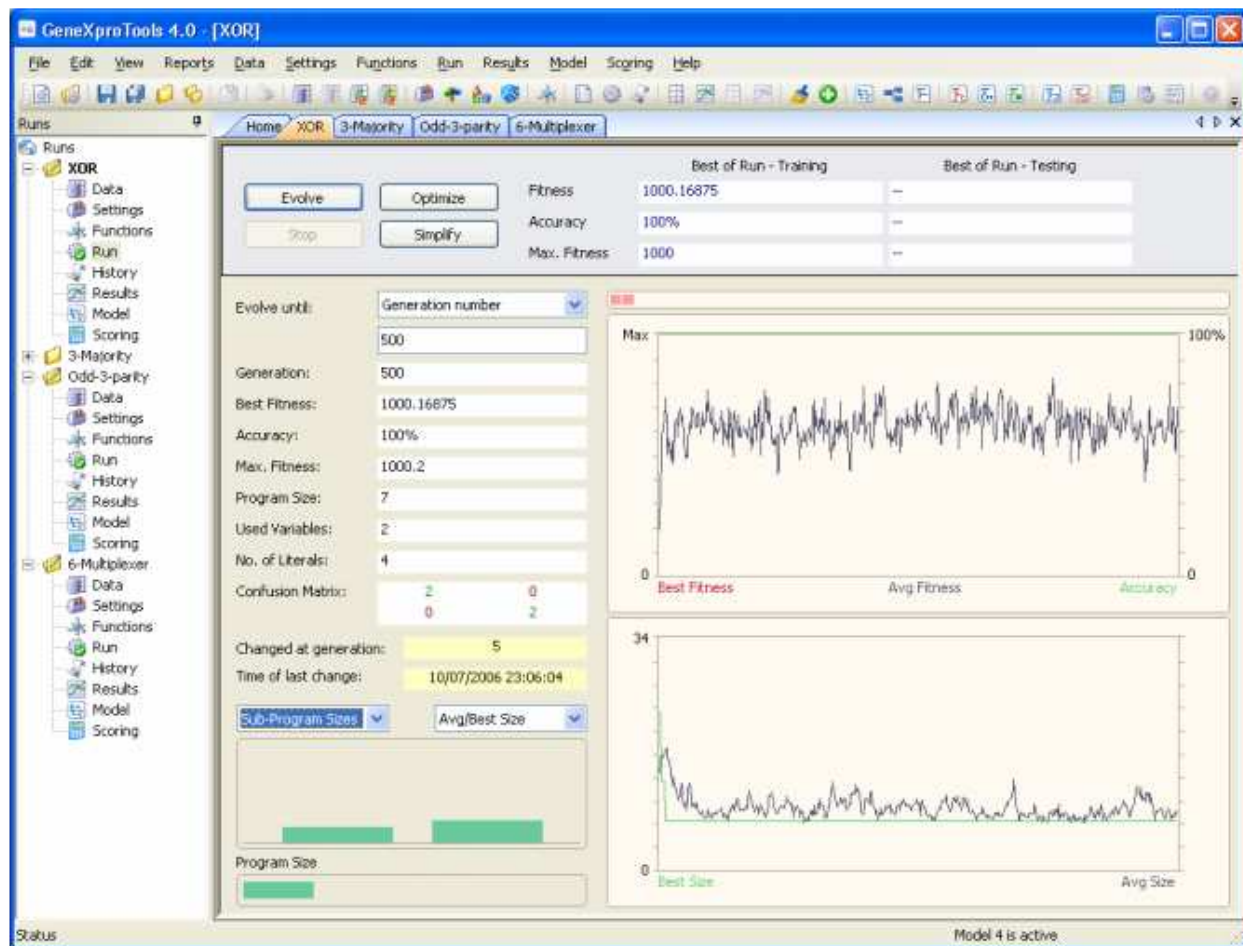
Settings and Features

Sample Runs

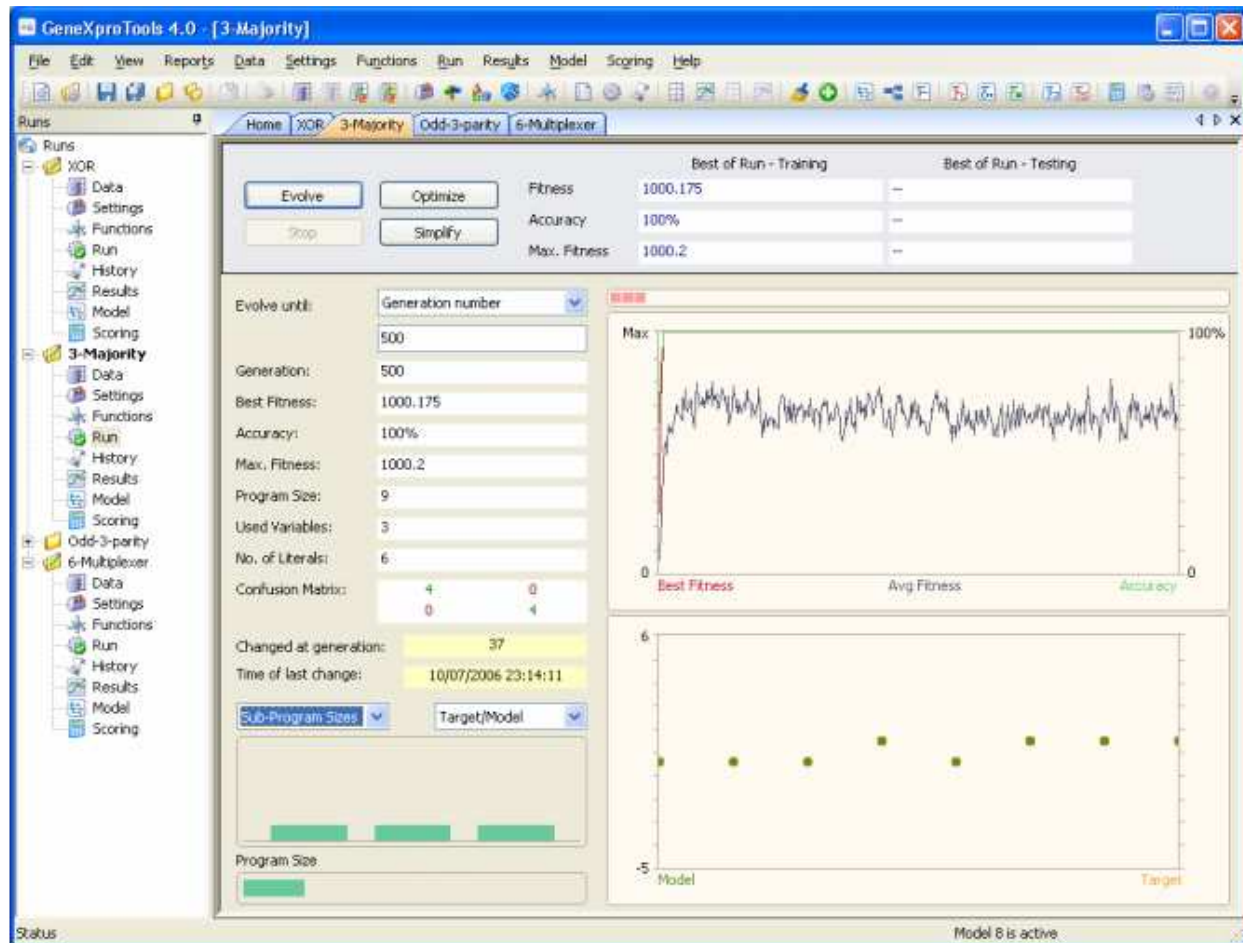
GeneXproTools 4.0 ships with four different **Sample Runs** for Logic Synthesis: [XOR](#), [3-Majority](#), [Odd-3-Parity](#), and [6-Multiplexer](#). To try any one of them, you just have to click its link on the Welcome Screen of GeneXproTools.



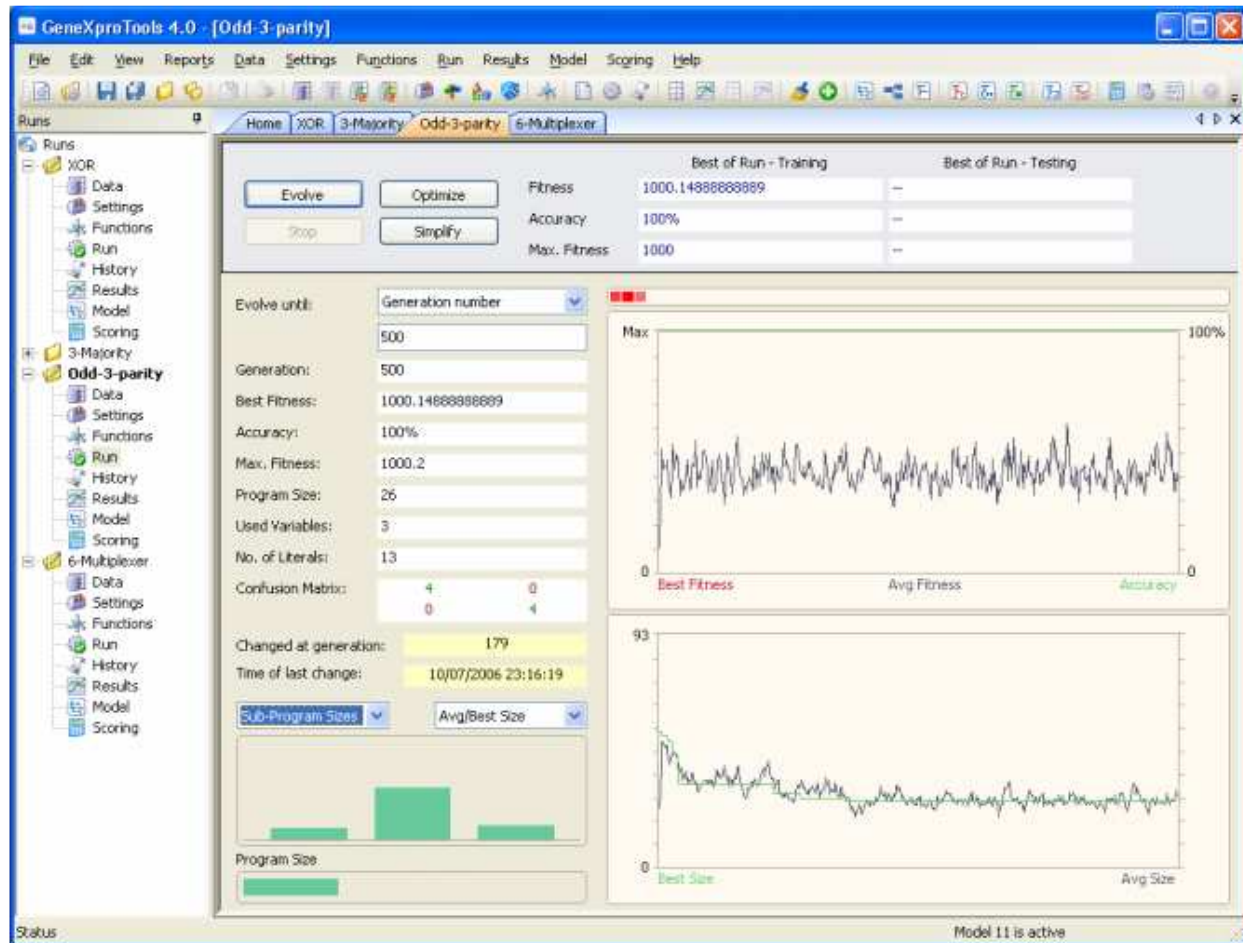
The **Exclusive Or** was chosen for its historical importance and also because it is quite simple and, therefore, ideal for you to learn your way around GeneXproTools modeling environment: try making a few runs, optimizing and simplifying a model, check the generated code and translate it into the programming language of your choice, try different fitness functions, different function sets, different architectures, and so on.



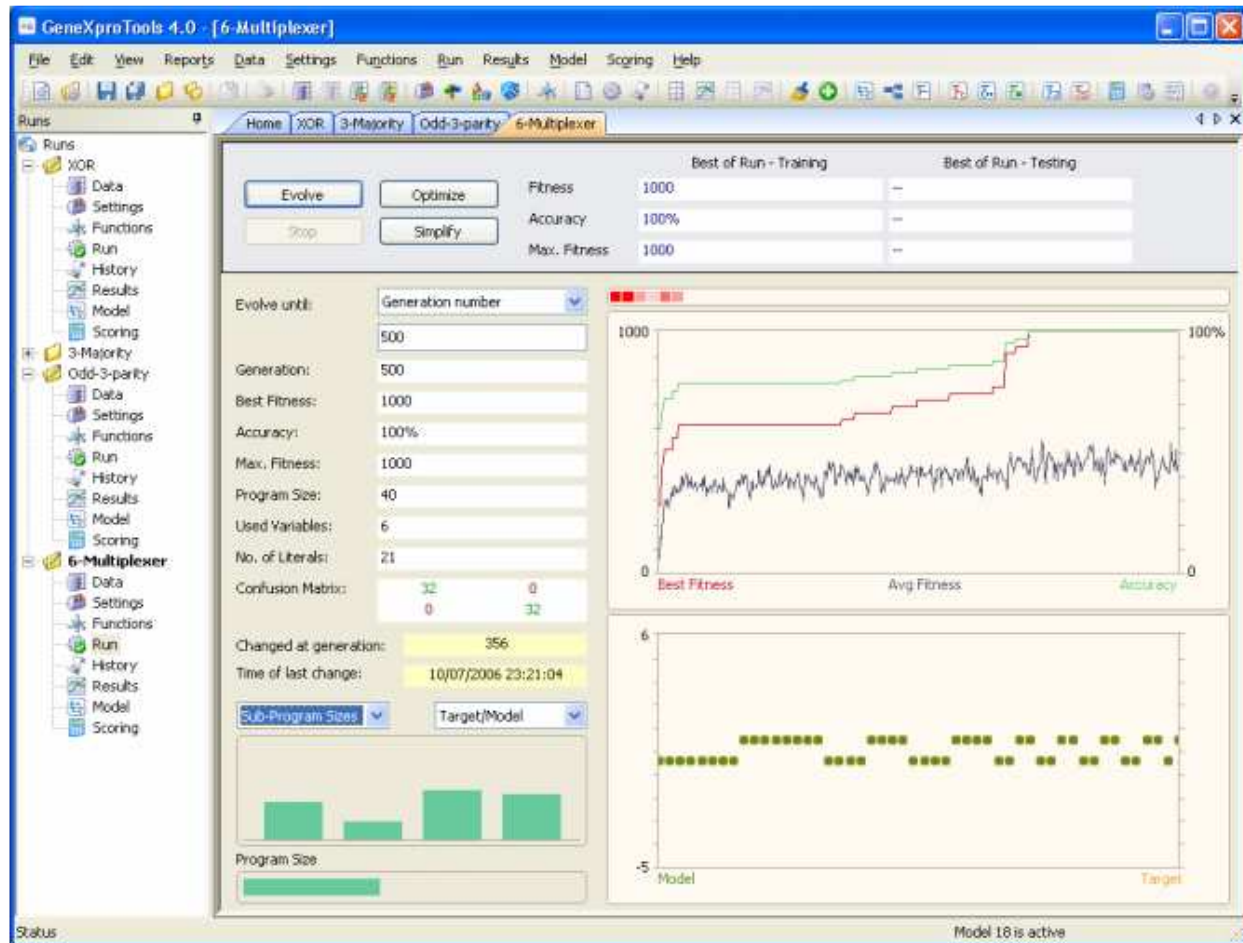
The **3-Majority** function was also chosen for its simplicity. With this sample run you will be able to explore and understand immediately all the capabilities of GeneXproTools including its more advanced features, such as linking functions, fitness functions with parsimony pressure, DDFs and UDFs, modeling with random constants, and so on.



The **Odd-3-Parity** is also a well-known function and, of the functions with three arguments, it is one of the hardest to design using adaptive algorithms such as Neural Networks and of course Gene Expression Programming. So, it might be interesting for you to see how GeneXproTools deals with such difficult functions, how it can be used to find their most parsimonious representations, how to tackle these more complex functions with a modular approach using [User Defined Functions](#) as smaller building blocks, and so forth.



The **6-Multiplexer** is also a well-known function and is much harder than even the [Odd-3-Parity function](#). You can explore this sample run to get a feeling of what can be achieved with GeneXproTools and try here its most advanced tools for logic circuit design, namely the use of [User Defined Functions](#) to design more complex circuits composed of simpler ones.



GeneXproTools Editions

This chapter describes the four GeneXproTools editions (**Standard**, **Advanced**, **Professional**, and **Enterprise**) and gives specific information about the requirements and installation of each one of them. In this chapter, you can find information about:

- The different editions of GeneXproTools 4.0.
- The minimum requirements for a successful installation of GeneXproTools.
- The installation procedure of GeneXproTools 4.0 in your computer.
- Converting APS 3.0 runs.
- The license agreement of GeneXproTools 4.0.
- The Demo of GeneXproTools 4.0.



GeneXproTools Editions

Editions

GeneXproTools 4.0 is available in four different editions: **Standard**, **Advanced**, **Professional**, and **Enterprise**. In addition, **Academic Versions** are available for both the Professional and Enterprise editions. The Academic Versions have all the features of their respective Professional and Enterprise editions and can only be used for academic purposes. The Academic license is sold in packages of three licenses for the **Academic Professional** and 30 Licenses for the **Academic Enterprise**.

All the four editions of GeneXproTools allow the creation of [Function Finding](#), [Classification](#), [Time Series Prediction](#), and [Logic Synthesis](#) models. For each category, the differences between the four GeneXproTools editions as well as their main features are outlined on the tables below (the asterisks *** indicate that Academic Versions are available at a reduced price):

General Features	Standard	Advanced	Professional***	Enterprise***
Independent Variables	16	64	250	Unlimited
Built-in Fitness Functions for Function Finding	36	36	36	36
Built-in Fitness Functions for Classification	30	30	30	30
Built-in Fitness Functions for Times Series	36	36	36	36
Built-in Fitness Functions for Logic Synthesis	30	30	30	30
User Defined Fitness Functions	Yes	Yes	Yes	Yes
Fitness Functions with Parsimony Pressure	Yes	Yes	Yes	Yes
Simplifying Algorithms	Yes	Yes	Yes	Yes
Data Screening Engine	Yes	Yes	Yes	Yes
Built-in Mathematical Functions and Comparison Rules	279	279	279	279
Built-in Logical Functions	258	258	258	258
Dynamic UDFs	Yes	Yes	Yes	Yes
Static UDFs	Yes	Yes	Yes	Yes
Mathematical Linking Functions	4	4	4	4
Logical Linking Functions	10	10	10	10
Random Numerical Constants	Yes	Yes	Yes	Yes
Evolve with Seed Models	Yes	Yes	Yes	Yes
Evolutionary History	Yes	Yes	Yes	Yes
Complexity Increase Engine	Yes	Yes	Yes	Yes
Built-in Mathematical Grammars	15	15	15	15
Built-in Logical Grammars	16x6	16x6	16x6	16x6
User Defined Grammars	Yes	Yes	Yes	Yes
Visualization of Parse Trees	Yes	Yes	Yes	Yes
Ada Code Generation	Yes	Yes	Yes	Yes
C Code Generation	Yes	Yes	Yes	Yes
C++ Code Generation	Yes	Yes	Yes	Yes
C# Code Generation	Yes	Yes	Yes	Yes
Fortran Code Generation	Yes	Yes	Yes	Yes
Java Code Generation	Yes	Yes	Yes	Yes
Java Script Code Generation	Yes	Yes	Yes	Yes

Matlab Code Generation	Yes	Yes	Yes	Yes
Pascal Generation	Yes	Yes	Yes	Yes
Perl Code Generation	Yes	Yes	Yes	Yes
PHP Generation	Yes	Yes	Yes	Yes
Python Code Generation	Yes	Yes	Yes	Yes
Visual Basic Code Generation	Yes	Yes	Yes	Yes
VB.Net Code Generation	Yes	Yes	Yes	Yes
Verilog Code Generation	Yes	Yes	Yes	Yes
VHDL Code Generation	Yes	Yes	Yes	Yes
Karva Code Generation	Yes	Yes	Yes	Yes
Scoring	Yes	Yes	Yes	Yes
Recursive Testing and Prediction for Time Series	Yes	Yes	Yes	Yes
Excel and Database Support	Yes	Yes	Yes	Yes
Flat Files Support	Yes	Yes	Yes	Yes
Statistical Evaluation of Models	Yes	Yes	Yes	Yes
Training Samples	Unlimited	Unlimited	Unlimited	Unlimited
Testing Samples	Unlimited	Unlimited	Unlimited	Unlimited
Scoring Samples	Unlimited	Unlimited	Unlimited	Unlimited
Chromosomes	4096	4096	4096	4096
Chromosome Size	2049	2049	2049	2049



GeneXproTools Editions

Requirements

All Editions:

- Windows 2000 SP2, Windows XP or Windows Server 2003.
- Pentium 4 class processor with 512 Mb RAM.

GeneXproTools can also be installed in Windows 98, Windows Me and Windows NT 4.0 although these operating systems are not supported and need to be updated to run GeneXproTools:

- Windows 98 - install SP1, Internet Explorer 6.0 and MDAC 2.5 (this may already be present if Office 2000 or later is installed).
- Windows Me - no further requirements.
- Windows NT4 - install SP6a, Internet Explorer 6.0 and MDAC 2.5 (this may already be present if Office 2000 or later is installed).

Internet Explorer 6.0 can be downloaded from: <http://www.microsoft.com/ie>

MDAC 2.5 can be downloaded from: <http://msdn.microsoft.com/data>



GeneXproTools Editions

Installation

To install GeneXproTools, run the file GeneXproToolsSetup.exe and follow the instructions of the setup program. Do not forget to close other applications, especially antivirus software, otherwise the setup program may not be able to install GeneXproTools properly. To install GeneXproTools on Windows NT, Windows 2000, Windows XP or Windows 2003 you may be required to login as Administrator for the installation.

GeneXproTools 4.0 can be installed without uninstalling APS 2.0 or APS 3.0.

To uninstall GeneXproTools use the Control Panel Add and Remove Programs applet. All the files in the GeneXproTools directory (except files created by the user) will be removed.



GeneXproTools Editions

Converting APS 3.0 Runs

Converting APS 3.0 runs is easy: you just have to open them with GeneXproTools 4.0 and they are automatically converted. There are, however, a few points you must be aware of:

First, there are new parameters in GeneXproTools 4.0, such as program size, number of literals, and used variables, that were not available in APS 3.0. You will notice that these parameters are set to zero in the [Run Panel](#), but they have all been evaluated for all the models in the run history and are correctly shown in the History of the [Report Panel](#).

Second, for Classification problems GeneXproTools 4.0 shows and keeps the [classification accuracy](#) of all the models in the [History Panel](#), whereas in APS 3.0 the values of [R-square](#) were shown instead. So, you will have to update the classification accuracies both on the training and testing sets in the [History Panel](#) by pressing the Refresh All button.

Third, GeneXproTools 4.0 has several new [fitness functions](#) for Classification problems ([Hits with Penalty](#), [Accuracy](#), [Squared Accuracy](#), [SSPN](#), and [Correlation Coefficient](#)), but the old Number of Hits fitness function has now a slightly different implementation. In the old version, there was a minimum requirement for viability that required the number of hits to be greater than the number of samples in the predominant class, whereas in the new [Number of Hits](#) there is no penalty and the fitness corresponds exactly to the number of hits (as you have already guessed, the new [Hits with Penalty](#) fitness function, although slightly different from the old Number of Hits, imposes the limit for viability).

Fourth, GeneXproTools 4.0 has a total of 279 [built-in functions and comparison rules](#) compared to just 70 in APS 3.0. So, new naming conventions were required to accommodate all the new [comparison rules](#). GeneXproTools takes care of all the conversions but you just must be aware that their representation in [Karva notation](#) has changed.

And finally, functions Acsc, Asec, and Acot are implemented differently in version 4.0. If any of these functions is part of your old models you must implement it as a [DDF](#) and then, in the [Change Seed](#)

[Window](#), replace its symbol by the corresponding [DDF](#). You can use the JavaScript code below to implement the old Acsc, Asec, and Acot functions as [DDFs](#):

```
function aps3Acsc(x)
{
    var varSign = 0;
    if (x < 0)
        varSign = -1;
    else
    {
        if (x > 0)
            varSign = 1;
        else
            varSign = 0;
    }
    return Math.atan(x/Math.sqrt(x*x-1))+(varSign-1)*(2*Math.atan(1));
}

function aps3Asec(x)
{
    var varSign = 0;
    if (x < 0)
        varSign = -1;
    else
    {
        if (x > 0)
            varSign = 1;
        else
            varSign = 0;
    }
    return Math.atan(x/Math.sqrt(x*x-1))+varSign*(2*Math.atan(1));
}

function aps3Acot(x)
{
    return Math.atan(x)+2*Math.atan(1);
}
```



GeneXproTools Editions

License

1. Important Notice

This is a legal agreement between Gepsoft Limited and you, the Licensee.

By installing this software, you are agreeing to be bound by the terms of this agreement.

If you do not agree to the terms of this agreement, do not use the software and return it.

2. One Computer

Gepsoft licenses this product for use on a single CPU only. You must obtain a licensed copy for each workstation, standalone or on a network, on which the product is used.

3. Ownership, Copyright, and Transfer

All copies of GeneXproTools are owned by Gepsoft or its suppliers and are protected by the copyright laws and international treaty provisions.

You must treat Gepsoft software products as any other copyrighted material, except that you may make backups for archival purposes, provided that you have only one working copy at a time for each copy of the product licensed to you. You may not transfer, sublicense, rent, lease or assign any of the above license or any Gepsoft software.

The source code generated by GeneXproTools belongs to the owner of the purchased license of GeneXproTools and can be freely distributed and/or transferred.

4. Term and Termination

You may terminate the above license at any time by returning or destroying all copies of GeneXproTools in your possession and notifying Gepsoft. The above license will terminate immediately if you infringe upon Gepsoft's copyrights or breach this agreement.

5. No Liability for Consequential Damages

Neither Gepsoft nor its suppliers shall be liable to you or any third party for any indirect, special, incidental or consequential damages (including damages for loss of business, loss of profits, business interruption or the like), arising out of the use or inability to use GeneXproTools based on any theory of liability including breach of contract, breach of warranty, tort (including negligence), product liability or otherwise, even if Gepsoft or its representatives have been advised of the possibility of such damages.

GeneXproTools is not fault tolerant and is not designed, manufactured, or intended for use or resale as online control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of GeneXproTools could lead directly to death, personal injury, or severe physical or environment damage.

6. Academic License

The Academic License of GeneXproTools cannot be used for commercial purposes and can only be acquired by educational institutions or equivalent. You cannot install and use the application outside the institution or distribute applications outside of it without purchasing a Commercial License of GeneXproTools. The Academic license is sold in packages of three licenses for the Academic Professional and 30 Licenses for the Academic Enterprise.

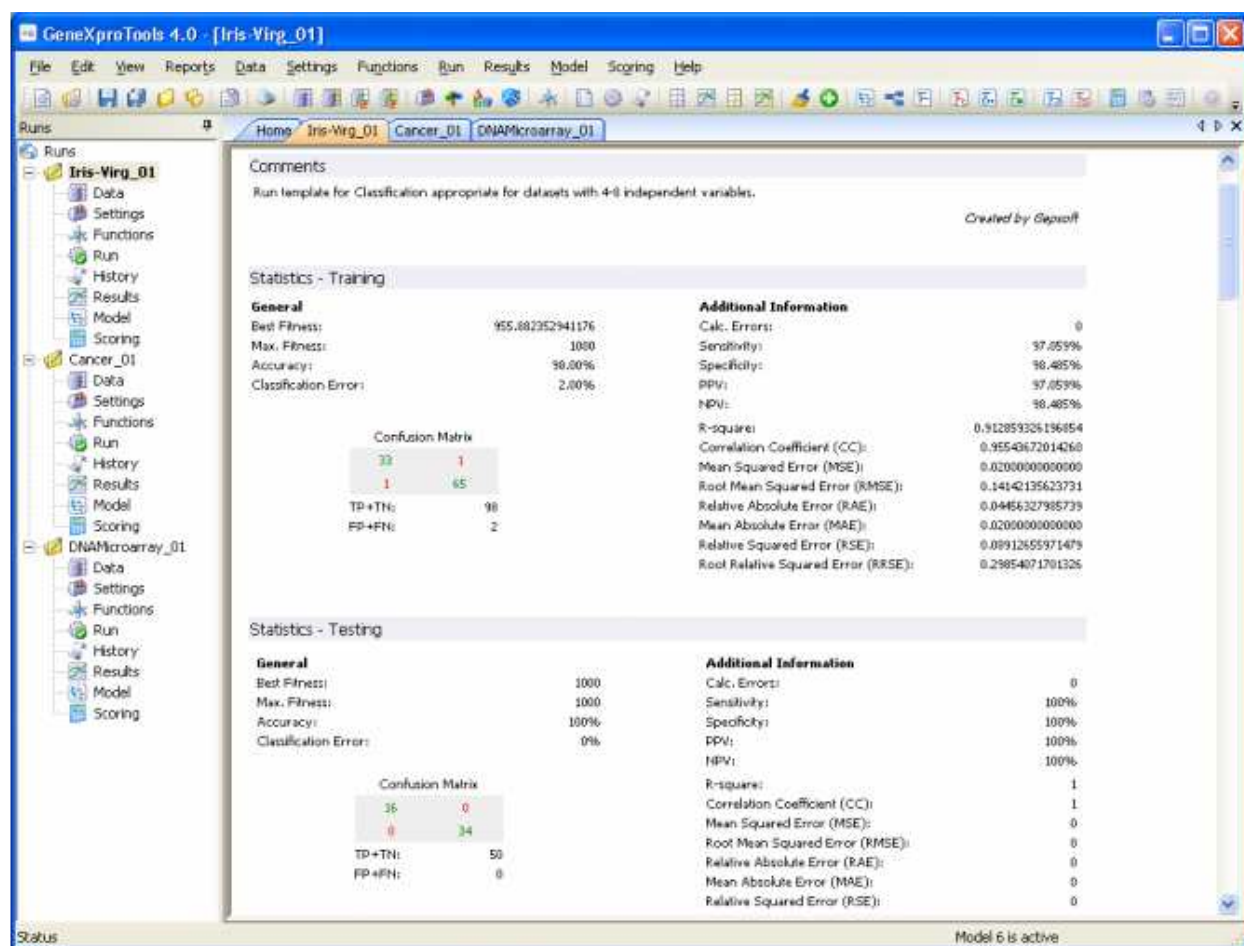


GeneXproTools Editions

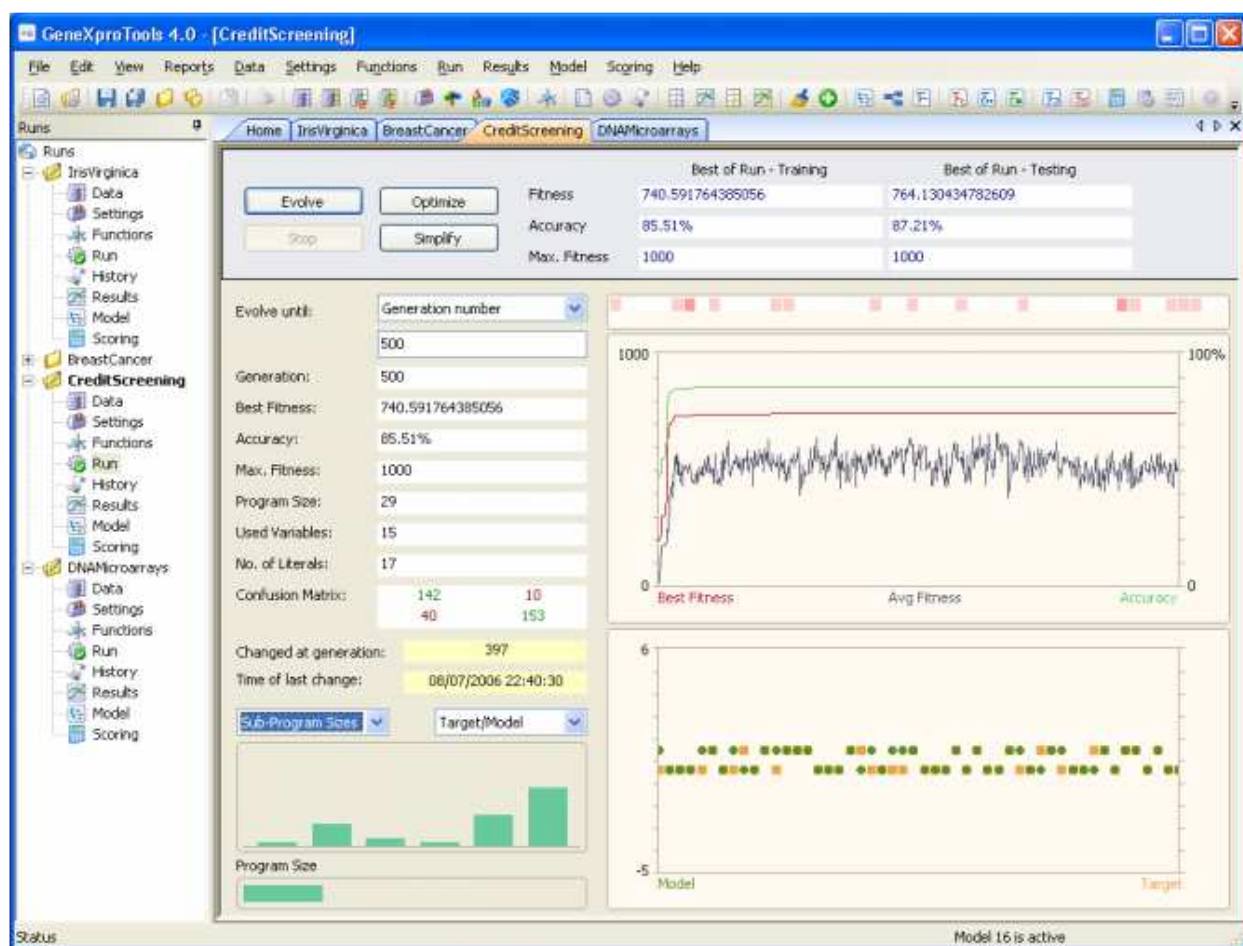
Demo

The demo of GeneXproTools 4.0 allows you to experiment with your own data, and, therefore, evaluate the modeling capabilities of the software. However, with the Demo version you are neither able to see

the code of the evolved models nor scoring them to make predictions. Furthermore, despite allowing the use of a testing set to evaluate the generalizing capabilities of the evolved models, the Demo does not show the tables and charts with the outputs on the testing set. However, all the [statistical functions](#) used by GeneXproTools to evaluate the performance of the evolved models are available and fully functional.



In addition, the Demo of GeneXproTools comes with a gallery of Sample Runs for [Function Finding](#), [Classification](#), [Time Series Prediction](#), and [Logic Synthesis](#) for which all the features of GeneXproTools (code generation, visualization of parse trees, scoring, etc.) are accessible as long as the original training set has not changed.



Download the Demo at:

<http://www.gepssoft.com/downloads.htm>

To uninstall GeneXproTools use the Control Panel Add and Remove Programs applet. All the files in the GeneXproTools directory will be removed except the files created by the user.



Learning Resources

The [modeling algorithms](#) of GeneXproTools are based on [Gene Expression Programming](#), one of the most powerful learning algorithms available.

GeneXproTools can be used without understanding [Gene Expression Programming](#), but if you do you will be able to better explore all of its features. The Help file provides all the necessary information, both theoretical and practical, for exploring all GeneXproTools features successfully.

For complementary information we recommend:

- Ferreira, C., 2001. Gene Expression Programming: A New Adaptive Algorithm for Solving Problems, *Complex Systems*, 13 (2): 87-129 (this paper is freely available in pdf format at the [Author's Home Page](#)).
- Ferreira, C., *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, 2nd Edition, Springer-Verlag, Germany, 2006 (you can learn more about this book at the [Gene Expression Programming website](#)).
- And other papers and books available at the [GEP Online Bibliography](#).



Support

Online: <http://www.gepsoft.com/support/request.asp>

Email: Write us at support@gepsoft.com. Don't forget to state your problem clearly in the subject line and provide complete information about the problem.



Contacts

Support

- **Online:** <http://www.gepsoft.com/support/request.asp>
- **Email:** support@gepsoft.com

Sales

- **Email:** sales@gepsoft.com

Registration

- **Online:** <http://www.gepsoft.com/register.asp>
- **Email:** register@gepsoft.com (send the name of the product, the date of purchase, your full name, email address, company name, and address).

