NeurOn-Line Studio

User's Guide

Version 5.1 Rev. 0

NeurOn-Line Studio User's Guide, Version 5.1 Rev. 0
June 2016

Gensym Corporation
52 Second Avenue
Burlington, MA 01803 USA
Telephone: (781) 265-7100
Fax: (781) 265-7101

Part Number: DOC411-510

Contents Summary

Contents

Preface

Describes this guide and the conventions that it uses.

About this Guide

This guide describes NeurOn-Line Studio, a graphical environment for building neural network applications. It consists of these chapters:

This chapter...	Describes...
Overview	NeurOn-Line Studio, its capabilities, features, and computational environment
Importing and Managing Data	How to import, export, and perform other data management tasks in NeurOn-Line Studio.
Visualizing Data	How to visualize and explore data through charts, graphs, and tabular views.
Labeling Data	How to label data to identify the parts of the raw data you would like to use to train a model.

This chapter...	Describes...
Creating a Preprocessor	How to create a preprocessor that conditions the raw data used to build models.
Creating a Predictive Model	How you create a predictive model, using data you have prepared using a preprocessor.
Analyzing a Trained Model	How to inspect and validate the performance of a predictive model.
Creating a Backpropagation Net	How how to create a backpropagation net.
Creating an Autoassociative Net	How you create an autoassociative network.
Creating a Radial Basis Function Net	How you create a Radial Basis Function network
Creating a Rho Net	How you create a Rho network.
Creating a Partial Least Square Model	How to create a partial least square model.
Creating a Principal Component Analysis Model	How to create a principal component analysis model.
Optimization	How to use a model and user-defined criteria to determine correct setpoints for manipulated variables.
Model Deployment	How to export and deploy a predictive model in ActiveX and G2.
Optimization Deployment	How to export and deploy an optimization in ActiveX and G2
Appendix A, NOLPredictor Class	Data members and operations of the NOLPredictor class.
Appendix B, NOLOptimizer Class	Data members and operations of the NOLOptimization class.

Audience

This guide is for developers of neural network applications. Users should have some familiarity with neural networks. Depending on the deployment environment, users should also be familiar with G2, NeurOn-Line Classic, or G2 ActiveXLink.

Conventions

This guide uses the following typographic conventions and conventions for defining system procedures.

Typographic

Convention Examples	Description
g2-window, g2-window-1, ws-top-level, sys-mod	User-defined and system-defined G2 class names, instance names, workspace names, and module names
history-keeping-spec, temperature	User-defined and system-defined G2 attribute names
true, 1.234, ok, “Burlington, MA”	G2 attribute values and values specified or viewed through dialogs
Main Menu > Start KB Workspace > New Object create subworkspace Start Procedure	G2 menu choices and button labels
conclude that the x of y ...	Text of G2 procedures, methods, functions, formulas, and expressions
new-argument	User-specified values in syntax descriptions
<u>*text-string*</u>	Return values of G2 procedures and methods in syntax descriptions

Convention Examples	Description
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save Properties	GMS and native menu choices
workspace	Glossary terms
c:\Program Files\Gensym\	Windows pathnames
/usr/gensym/g2/kbs	UNIX pathnames
spreadsh.kb	File names
g2 -kb top.kb	Operating system commands
public void main() *gsi_start*	Java, C and all other external code

Note Syntax conventions are fully described in the *G2 Reference Manual*.

Procedure Signatures

A procedure signature is a complete syntactic summary of a procedure or method. A procedure signature shows values supplied by the user in *italics*, and the value (if any) returned by the procedure *underlined*. Each value is followed by its type:

g2-clone-and-transfer-objects
    (*list*: class item-list, *to-workspace*: class kb-workspace,
    delta-x: integer, *delta-y*: integer)
    -> *transferred-items*: g2-list

Related Documentation

NeurOn-Line

NeurOn-Line Release Notes

NeurOn-Line User's Guide

NeurOn-Line Reference Manual

NeurOn-Line Studio User's Guide

Gensym Neural Network Engine

G2 Core Technology

- *G2 Bundle Release Notes*
- *Getting Started with G2 Tutorials*
- *G2 Reference Manual*
- *G2 Language Reference Card*
- *G2 Developer's Guide*
- *G2 System Procedures Reference Manual*
- *G2 System Procedures Reference Card*
- *G2 Class Reference Manual*
- *Telewindows User's Guide*
- *G2 Gateway Bridge Developer's Guide*

G2 Utilities

- *G2 ProTools User's Guide*
- *G2 Foundation Resources User's Guide*
- *G2 Menu System User's Guide*
- *G2 XL Spreadsheet User's Guide*
- *G2 Dynamic Displays User's Guide*
- *G2 Developer's Interface User's Guide*
- *G2 OnLine Documentation Developer's Guide*
- *G2 OnLine Documentation User's Guide*
- *G2 GUIDE User's Guide*
- *G2 GUIDE/UIL Procedures Reference Manual*

G2 Developers' Utilities

- *Business Process Management System Users' Guide*
- *Business Rules Management System User's Guide*
- *G2 Reporting Engine User's Guide*
- *G2 Web User's Guide*
- *G2 Event and Data Processing User's Guide*
- *G2 Run-Time Library User's Guide*

- *G2 Event Manager User's Guide*
- *G2 Dialog Utility User's Guide*
- *G2 Data Source Manager User's Guide*
- *G2 Data Point Manager User's Guide*
- *G2 Engineering Unit Conversion User's Guide*
- *G2 Error Handling Foundation User's Guide*
- *G2 Relation Browser User's Guide*

Bridges and External Systems

- *G2 ActiveXLink User's Guide*
- *G2 CORBALink User's Guide*
- *G2 Database Bridge User's Guide*
- *G2-ODBC Bridge Release Notes*
- *G2-Oracle Bridge Release Notes*
- *G2-Sybase Bridge Release Notes*
- *G2 JMail Bridge User's Guide*
- *G2 Java Socket Manager User's Guide*
- *G2 JMSLink User's Guide*
- *G2 OPCLink User's Guide*
- *G2-PI Bridge User's Guide*
- *G2-SNMP Bridge User's Guide*
- *G2-HLA Bridge User's Guide*
- *G2 WebLink User's Guide*

G2 JavaLink

- *G2 JavaLink User's Guide*
- *G2 DownloadInterfaces User's Guide*
- *G2 Bean Builder User's Guide*

G2 Diagnostic Assistant

- *GDA User's Guide*
- *GDA Reference Manual*
- *GDA API Reference*

Customer Support Services

You can obtain help with this or any Gensym product from Gensym Customer Support. Help is available online, by telephone, by fax, and by email.

To obtain customer support online:

➔ Access G2 HelpLink at *www.gensym-support.com*.

You will be asked to log in to an existing account or create a new account if necessary. G2 HelpLink allows you to:

- Register your question with Customer Support by creating an Issue.
- Query, link to, and review existing issues.
- Share issues with other users in your group.
- Query for Bugs, Suggestions, and Resolutions.

To obtain customer support by telephone, fax, or email:

➔ Use the following numbers and addresses:

	Americas	**Europe, Middle-East, Africa (EMEA)**
Phone	(781) 265-7301	+31-71-5682622
Fax	(781) 265-7255	+31-71-5682621
Email	service@gensym.com	service-ema@gensym.com

Overview

Provides an overview of NeurOn-Line Studio, its capabilities, features, and computational environment

Introduction

NeurOn-Line Studio (NOL Studio) is a graphical, object-oriented software product for building neural network applications. Using NeurOn-Line Studio, you can model dynamic, nonlinear phenomena that are difficult to describe by analytical models, using historical data stored in databases, process data historians, or text files. Typical applications include quality assurance, sensor validation, diagnosis, and process modeling.

You don't have to be an expert in neural networks or statistics to use NeurOn-Line Studio. NOL Studio allows you to focus on your system and your data, while the software handles the technical details, behind the scene. You simply load your data, graphically select the portions you'd like to use for model development, and let NOL Studio do the rest. Advanced users can make use of additional, powerful options for data analysis and custom model building.

NOL Studio is specially designed to allow you to handle large, messy data sets that are typically produced from industrial operations. These data sets might have many defects, such as missing and bad data, incompatible formats deriving from different databases, historians, and lab records, and combine together different production runs into large files. Limited only by memory capacity, NOL Studio can readily handle data sets of upwards of 100,000 data points with 100+ variables.

NOL Studio supports three types of models: predictive models, statistical models, and optimization models. Predictive models are used for creating virtual analyzers (software sensors), fault detection, sensor validation, and forecasting. Statistical models are used to perform statistical analysis of your data and can be used as a statistical monitoring tool for your process. Optimization models are used for determining the best operational settings for a process to minimize an objective function you define.

Predictive models can be any of the following six types:

- Predictive model.
- Backpropagation Net model.
- Autoassociative Net model.
- Radial Basis Function Net model.
- Rho Net model.
- Partial Least Square model

The last four models are identical to the models in classic NeurOn-Line.

Once a tentative model is built, you can use a variety of powerful analysis tools to test and validate the fit. For example, you can apply the model to new data, that was not used in the training process. Or, you can plot response surfaces to analysis the input-output relationships learned by the model.

To deploy the resulting models, you can use a supplied ActiveX component that can be embedded in any COM-compliant container application (COM is Microsoft's standard component interface specification). Many databases, historians, DCSs and desktop applications are COM-compliant. Examples of suitable containers include Microsoft Office applications, OSI's Process Book, Aspen Technology's InfoPlus.21, and Honeywell's PHD. You can also use Gensym's flagship G2 real-time expert system with the Gensym Neural Network Engine (GNNE) and GEDP Graphical Language to configure the on-line deployment, acquire data, and control the context of applying the neural network.

Feature Summary

Here are some of the key features of NeurOn-Line Studio:

- Data Importing
  - Accepts a wide variety of ASCII text file formats
  - Accepts data sets from G2 online environment
  - Able to combine multiple files covering different data ranges
  - Search-and-replace capability
  - No explicit size limitation on data sets
- Data Preprocessing
  - Interactive graphical data labeling
  - User-defined label categories
  - Projection plots for outlier identification
  - User-defined mathematical formulas (transforms)
- Modeling
  - Steady-state and dynamic models, which can have recursive behavior
  - Automatic selection of relevant inputs
  - Automatic selection of time delays
  - Automatic feedback of output for auto-recursive models
  - Automatic determination of network architecture that optimizes future prediction accuracy
  - Efficient training with upwards of 100,000 samples and 100 variables
- Validation and Simulation
  - Predicted versus actual plots
  - Response surface plots
  - Input sensitivity analysis
- Optimization
  - Four-way variable categorization (manipulated, disturbance, state, and output)
  - Hard and soft upper and lower bounds
  - Cost-based objective functions

- Deployment
  - Embedded component deployment as ActiveX component in COM-compliant containers
  - Interface with G2 to leverage G2's data acquisition, expert system, and object-oriented environment
  - Configuration via graphical clone-and-connect block language

Platform Compatibility

NOL Studio runs on Windows 2000 and Windows XP. On Windows platforms, the user interface follows Microsoft user interface standards.

Running NeurOn-Line Studio

NOL Studio is now a component of NOL bundle software. To install NOL Studio, insert the NOL bundle CD into your CD drive, launch the setup.exe file, located in the root directory. Follow the instructions in the setup program.

To launch the application on Windows platforms:

➔ Do one of the following:

- Choose Start > Programs > Gensym G2 2011 > G2 NeurOn-Line > G2 NeurOn-Line Studio.
- Execute this command at a DOS command prompt:

  install-dir `\nolstudio`

  where *install-dir* is the directory you chose to install the software.

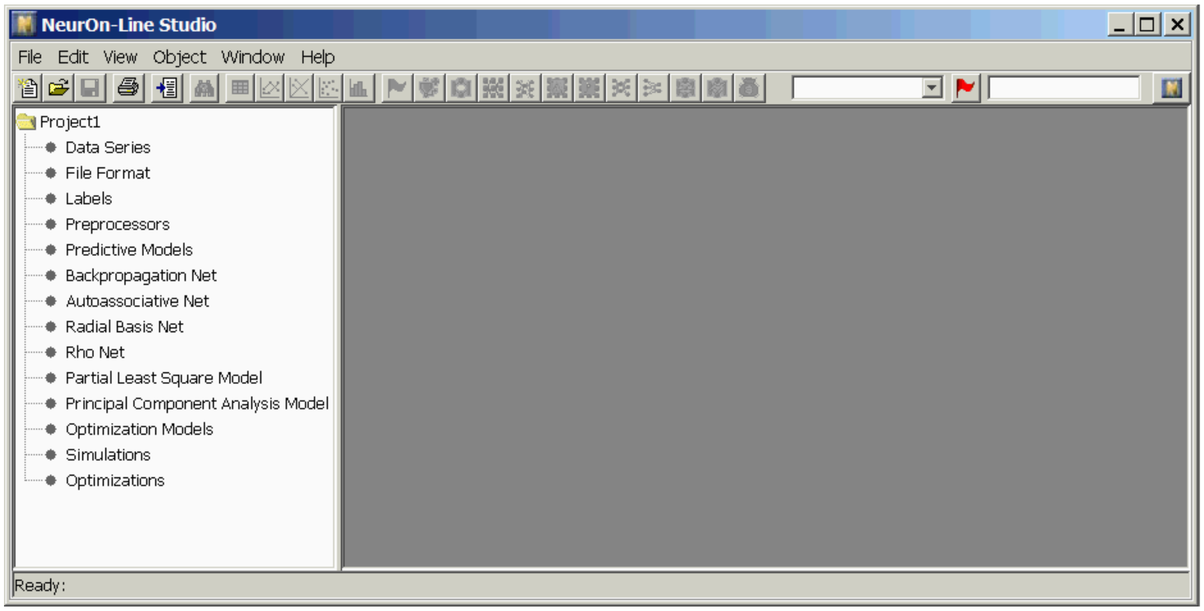
  If your path is properly set to include the directory where NOL Studio is installed, you can shorten this to just `nolstudio`.
- Execute this procedure in G2:

  `nols-launch-nolstudio-by-setting`

You might find it convenient to create a shortcut to allow you to launch the application. To do this, click the right mouse button on the `nolstudio.bat` file, and select Create Shortcut.

The Main Window

All activities in NOL Studio revolve around the application window, shown here:



The applications use familiar Windows user interface elements:

- Title bar at the top of the window, with tools to iconize, maximize, and close (quit) the application.
- Menu bar with pull-down menus.
- Toolbar, with common actions such as file operations, creating plots, building models, etc.
- A tree view, at the left side of the window, showing an inventory of the objects in the current project.
- A status bar, at the bottom of the window, showing additional information about the current action.
- A work area (the gray area above), where property tables, plots, and other sub-windows are displayed.

Mouse gestures, as a rule, follow Microsoft application standards:

To...	Do this...
Select an item	Click on it with the left mouse button.
Show a pop-up menu of actions for an item	Click the right mouse button.

To...	Do this...
Open an item	Double-click on the item, using the left mouse button.
Select a contiguous group of items, for example, successive entries in a scroll area	Select the first item in the group, hold down the shift key, and then select the last item in the group.
Toggle the selection of an individual item	Hold down the control key (ctrl) while selecting the item, using the left mouse button.
See a pop-up description (tool tip) of an item	Hold the mouse motionless over the item for one second, without depressing either mouse button. The tool tip disappears automatically after a brief period, or when the mouse is moved.

Navigating the Tree View

The tree view provides access to the objects you create while using NOL Studio. The tree view contains the following types of objects:

- Data series
- File Formats
- Labels
- Preprocessors
- Predictive Model
- Backpropagation Net
- Autoassociative Net
- Radial Basis Function Net
- Rho Net
- Partial Least Square Model
- Principal Component Analysis Model
- Optimization Model
- Simulations
- Optimizations

When you create an object, it is automatically added to the tree view. If a node contains items, the node will be prefaced with a plus (+) sign. To expand a node, left-click once on the plus sign. To close a node, click on the minus (-) sign. Initially, the tree view is empty.

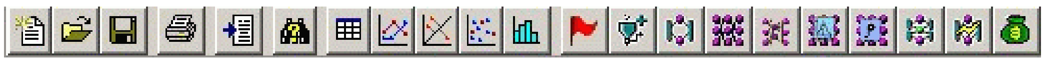
Every object in the tree view has a corresponding property table. The property table contains information about the object, and gives you access to certain actions on the object.

To navigate to any object from the tree view:

1. Open the appropriate node in the tree view by clicking on the plus (+) sign
2. Double click on the desired object
3. The property table for that object will appear in the main window

The Tool Bar

The tool bar, shown below, gives you easy access to frequently-used functionality.



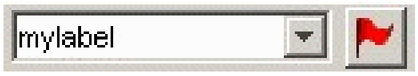
Option	Description
📄	Creates a new project.
📂	Opens an existing project file.
💾	Saves the current project to the disk.
🖨	Prints an internal window.
📥	Imports a text file containing raw data.
🔍	Search and replace (in a data series).
▦	Opens a spreadsheet view.
📈	Plots a line chart.

Option	Description
	Plots a projection chart.
	Plots a scatter (X-Y) chart.
	Plots a histogram.
	Creates a new label.
	Creates a new preprocessor.
	Creates a new predictive model.
	Creates a new backpropagation net.
	Creates a new autoassociative net.
	Creates a new radial basis function net.
	Creates a new rho net.
	Creates a new partial least square model.
	Creates a new principal component analysis model.
	Creates a new optimization model.
	Creates a new simulation.
	Creates a new optimization.

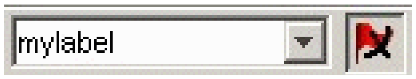
Like menu choices, the tool bar buttons may be disabled (grayed out) when they cannot be used. For example, you cannot plot a line chart until you have loaded data. Therefore, the line chart button is disabled until data is loaded.

Labeling Tools

Next to the toolbar, there is a selection box and buttons that determine the active label and labeling mode. These tools are used when you are categorizing (labeling) the raw data. The labeling tools are shown below:



To set the active label, use the down arrow to select one of the defined labels. This selection box will be empty until you have defined one or more labels. To toggle between label and unlabel mode, click the Label button.



When you are in label mode, selections in the spreadsheet, line chart, or scatter charts will apply the selected label to the data. In unlabel mode, the label will be removed when the data is selected on the spreadsheet, line chart or scatter charts.

More information on labeling can be found in the chapter on data selection.

Terminology

When you use NeurOn-Line Studio, you will create several types of objects. These include:

Object	Description
Data series	A **data series** represents a set of measurements on certain variables. Each row of a data series represents measurements taken at a certain time. Each row has a unique time stamp. Data series can be combined by appending or time-merging. Appending adds more rows to a data series. Time-merging creates a new data series by placing the variables in two or more data series under the same set of time stamps.
File Formats	A **file format** is a description of the layout of an ASCII text file, used to import data into NOL Studio. Whenever you import a text file, a file format is automatically created for that file. You can apply the format to load other, similar files.

Object	Description
Labels	**Labels** are used to mark the raw data, to indicate regions of special interest. You can define any label categories appropriate for your data. Examples of label categories are outlier, transient, steady state, product transition, or cut.
Preprocessors	A **preprocessor** defines the pretreatment of data, before it enters the neural network model. Each preprocessor contains two parts: a filter and an optional list of formulas. The filter defines which parts of the raw data you want to use in training a model. The filter is based on the labels you apply to the raw data. A simple filter might be "all data excluding data labeled cut". The formula list allows you to perform mathematical transformations on the filtered data, to fill in missing values, smooth noisy signals, calculate ratios, and the like.
Predictive Models	**Models** are generated by the training process. You designate input and output variables, optional time delays, and other training parameters. You can generate as many models as you wish, and compare their performance. Predictive models are used for creating virtual analyzers (software sensors), fault detection, sensor validation, and forecasting.
Backpropagation Nets	**BPNs** are generated by the training process. You designate input and output variables, model architecture, and training parameters. You can generate as many models as you want and compare their performance. BPNs are useful for creating virtual analyzers (software sensors), fault detection, sensor validation, and forecasting.

Object	Description
Autoassociative Nets	**AANs** are generated by the training process. You designate modeled variables, model architecture, and training parameters. You can generate as many models as you want and compare their performance. AANs are useful for sensor validation.
Radial Basis Function Nets	**RBFNs** are generated by the training process. You designate input and output variables, model architecture, and training parameters. You can generate as many models as you want and compare their performance. RBFNs are useful for fault detection, pattern recognition, and forecasting.
Rho Nets	**Rhos** are generated by the training process. You designate input and output variables, model architecture, and training parameters. You can generate as many models as you want and compare their performance. Rhos are useful for fault detection and classification.
Partial Least Square Models	**PLS** models are generated by the training process. You designate input and output variables, and model architecture. PLS models are useful for creating virtual analyzers (software sensors), fault detection, sensor validation, and forecasting.
Principal Component Analysis Models	**PCA** models are used to analyze the statistical properties of a given data set. PCA models are useful for fault detection, sensor validation, and process performance monitoring.
Optimization Models	**Optimization models** are used for determining the best operational settings for a process, to minimize an objective function you define.

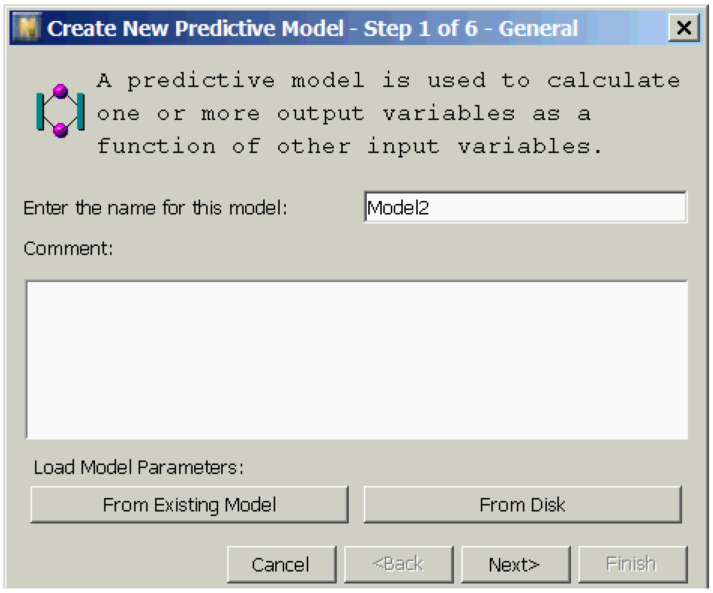
Object	Description
Simulations	**Simulations** are used to show the response of a model to user-defined inputs. Like other objects, simulations are automatically stored as part of your project, to allow you to return to scenarios, or apply the same scenarios to different models.
Optimizations	When you create an optimization model, you can define an objective function that represents a cost function for your operations. You can also define hard and soft constraints on the input and output variables. To run an scenario using your objective function, you create an **optimization** object. The optimization calculates values for inputs that minimize your objective function while satisfying given constraints.

Wizards, Data Views, and Property Tables

The user interface of NOL Studio is composed primarily of three types of items, wizards, data views, and property tables.

- Wizards are step-by-step dialogs that guide you through a task, usually the creation of a new item, such as a preprocessor, label, or model. Wizards also help you through multi-stage activities, such as importing data from text files. When you are working with a wizard, you move from step to step using the Back and Next buttons. After the last step, you complete the action by selecting the Finished button. You can quit the activity at any time using the Cancel or the close window button.

Here is a typical wizard dialog:



Create New Predictive Model - Step 1 of 6 - General

A predictive model is used to calculate one or more output variables as a function of other input variables.

Enter the name for this model: Model2

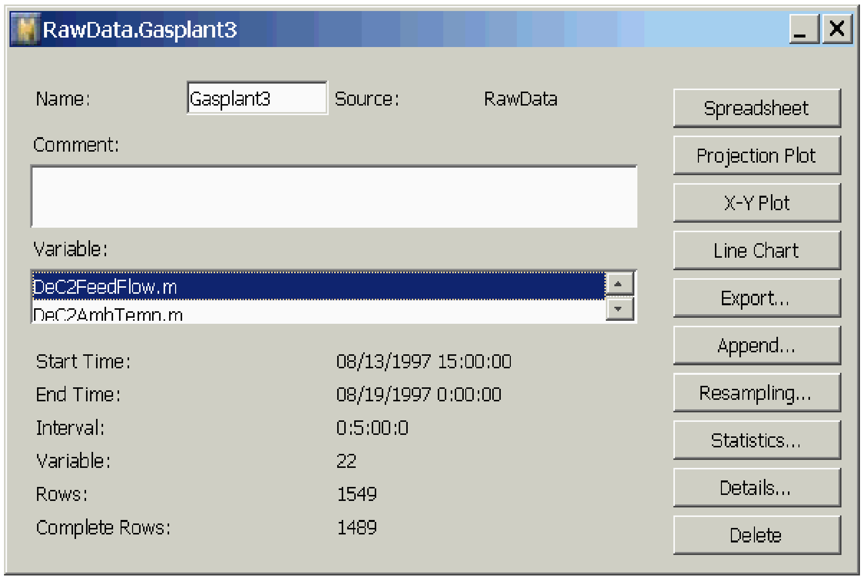
Comment:

Load Model Parameters:

- Data views are displays that show your data, or the properties of your data, in various graphical and tabular forms. The primary data views in NOL Studio are the spreadsheet, line charts, scatter charts, projection charts, and histograms. In many contexts, data views are interactive; the views both display information and receive input from mouse gestures. Views are also dynamically linked, so changes to the underlying data simultaneously updates all open views. For example, adding a label in a line chart view simultaneously adds a label in the spreadsheet view, represented by coloring a cell. See Visualizing Data, for more information.

- Property tables present information on the objects in a project. You access the property tables by double clicking on an object in the tree view, or by using the Go To choice on the Object menu. In some cases, the property table accepts input, allowing you to change some property, such as the object's name. In addition, there are buttons along the right side of the table for performing actions on the object. For example, the property table for a variable allows you to plot the variable in a line chart, or display a histogram.

  Here is an example of a property table for a data series:



RawData.Gasplant3

Name: Gasplant3 Source: RawData

Comment:

Variable:
DeC2FeedFlow.m
DeC2AmbTemp.m

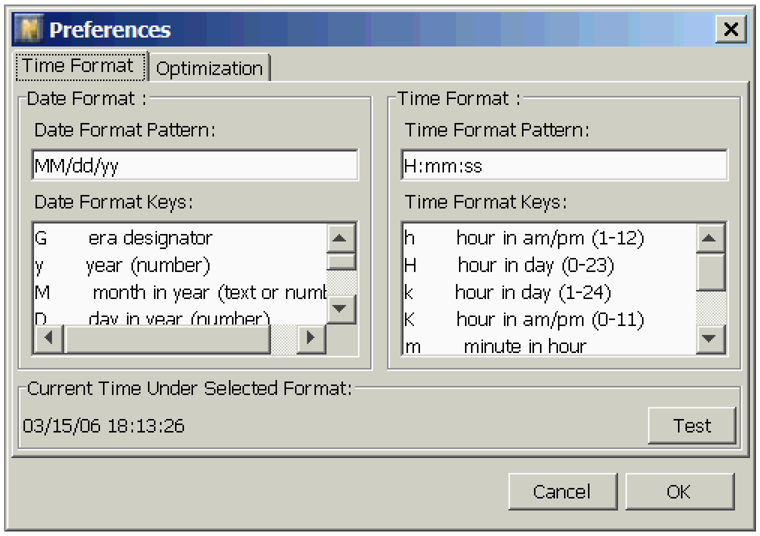
Start Time:	08/13/1997 15:00:00
End Time:	08/19/1997 0:00:00
Interval:	0:5:00:0
Variable:	22
Rows:	1549
Complete Rows:	1489

Global Preference Settings

From the File > Preferences menu, you can set global preferences for Time/Date format and Optimization maximum iteration.

Date/Time Format

You can define date and time formats that will be used as the default setting for spreadsheet display and text output of data series. The initial default settings are Date Format MM/dd/yy and Time Format H:mm:ss. When defining a new format you should test it with the current time by clicking the Test button before clicking OK. Invalid formats will be ignored.



Date Format :

Date Format Pattern: MM/dd/yy

Date Format Keys:

G	era designator
y	year (number)
M	month in year (text or numb
D	day in year (number)

Time Format :

Time Format Pattern: H:mm:ss

Time Format Keys:

h	hour in am/pm (1-12)
H	hour in day (0-23)
k	hour in day (1-24)
K	hour in am/pm (0-11)
m	minute in hour

Current Time Under Selected Format:

03/15/06 18:13:26

Optimization

You can set the number of maximum iterations that NOLStudio will use when running optimizations. The initial setting is 10000. The setting is used when creating a new optimization object. If you run an optimization and reach the maximum iteration, NOLStudio will give you a warning message and allow you to increase the maximum iteration for this object at that time.

A Methodology Roadmap

You use NeurOn-Line Studio by following a simple methodology, which include these basic steps:

1 Load the raw data.

2 Label (categorizing) the raw data.

3 Preprocess the data.

4 Train the model.

5 Validate the model.

6 Deploy the model.

You begin by loading raw data into NeurOn-Line Studio by importing text files, creating one or more data series in your project. Next, you add labels to the data, to indicate parts of the data you want use to build the model, or conversely, to mark bad data you want to exclude.

You next create a preprocessor, which conditions the raw data. The preprocessor has two parts: a filter and optional formulas. The filter specifies which parts of the raw data should be included in the training set, based on the labels you applied. The formulas can be used to fill in missing values, calculate new derived variables such as ratios, smooth noisy data, and much more.

Then, you are ready to train a model. After training, there are several ways to validate the results. Finally, you can deploy your model for on-line use. The remainder of this User's Guide explains each of these steps in detail.

Importing and Managing Data

Describes how to import, export, and perform other data management tasks in NeurOn-Line Studio.

Introduction

One of the biggest obstacles in modeling processes with neural nets is the ability to manage large sets of data easily and efficiently. NOL Studio allows you to perform common data manipulations quickly and easily. NOL Studio can read data from a variety of data sources, with flexible formatting. This chapter takes you through some basic data operations, such as importing data from outside sources, defining and managing file formats, creating data sets from multiple files, exporting data, and removing data from the NOL Studio. We preface this discussion with an overview of data representation in NOL Studio.

Data Series

A data series is a two-dimensional table of data, whose columns represent variables, and whose rows represent samples. There are two types of data series:

- Time-based
- Row-based.

Time-Based Data Series

In a time-based data series, each row represents observations or measurements at a certain time. The time for each row is referred to as the *timestamp* for the row. Timestamps are represented internally in a format-independent, year 2000 compliant manner, with millisecond precision. All time-based calculations are performed using this internal representation, and also are year 2000 compliant.

Time stamps must be in a strictly ascending order. However, the time intervals between rows do not have to be equally-spaced.

Row-Based Data Series

If rows of a data series lack time stamps, each row of the data series is assumed to represent a set of related observations on the same material, batch, or sample. The rows are assumed to be non-sequential, and the order of the rows is ignored in all data manipulations.

With row-based data, certain operations are not defined, in particular, delays and interpolations. Both delays and interpolations require the assumption that successive rows represent successive samples in time. However, because the rows of a row-based data series are not assumed to be sequential, a missing value in row *n* cannot be estimated from the values in rows *n-1* and *n+1*. You also cannot refer to the sample *d* rows before another sample, without raising the implication of sequence or time.

If the samples are sequential, then you should consider adding time stamps to the rows, even if you have to introduce artificial time stamps, starting at an arbitrary time and incrementing by an arbitrary fixed time interval. This will enable interpolation and delay calculations.

Importing Data

The first step in building a model in NeurOn-Line Studio is to import data from one or more external data sources. NeurOn-Line Studio provides wizards for importing data from several sources:

- Text files - NeurOn-Line Studio provides flexible file import facilities.
- G2 - NeurOn-Line Studio has access to G2's extensive connectivity solutions to databases, data historians, and raw process data.
- Network - NeurOn-Line Studio can fetch data through the Internet for a specified URL and a specified data format.

Importing Data Series From Files

To import data from a file into NOL Studio, you follow several predefined steps.

1. Select the file from which data will be imported.
2. Select a pre-existing format, or define a new format, for the data file.
3. Import the file.

This section will take you through all of the above steps.

Selecting a File for Data Import

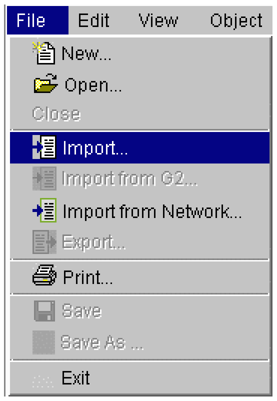
There are three methods to initiate data import into NOL Studio:

- A menu choice
- A toolbar button
- A tree-view selection.

All three methods launch the same file selection dialog.

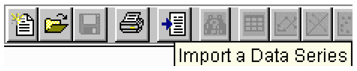
From the menu bar:

➔ Choose File > Import. For example:



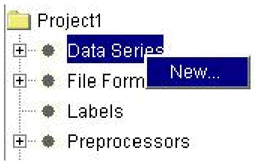
From the toolbar:

➔ Click Import. For example:

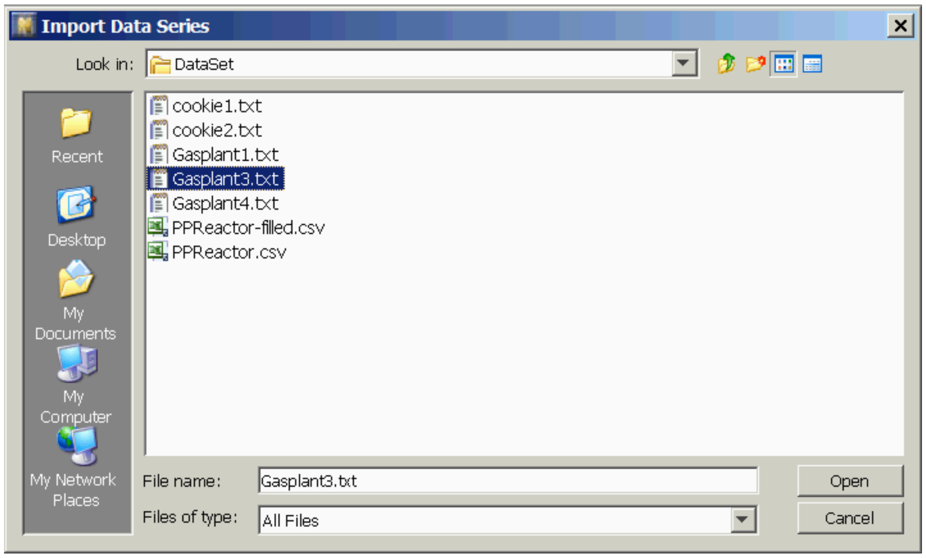


From the tree view:

➔ Right-click the Data Series category. For example:



Any of these actions displays the Import Data Series dialog. For example:

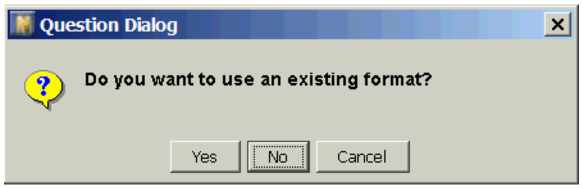


To select a file:

1 Navigate through the directories using standard techniques.

2 Left-click on the file, and choose Open.

If your file has a `.ds` or `.bds` extension, it is imported immediately into NOL Studio. If your file has a different extension, a dialog prompts you to specify the format of the file.



Do you want to use an existing format?

An existing format can be either a predefined NOL Studio format, or a format you defined previously, when loading an earlier data set. If there is no suitable format, you will create one using the File Import wizard. The file type option provides a drop-down list of predefined file types.

Specifying a File Format

There are two predefined file formats in NeurOn-Line Studio:

- An ASCII file format, which allows you to import or append an ASCII file that follows a standard formatting convention.
- A BINARY file format, which is used for saving and loading data after it was already imported into NOL Studio.

Files in the predefined ASCII format have the *.ds* extension. Files in the predefined BINARY format have the *.bds* extension.

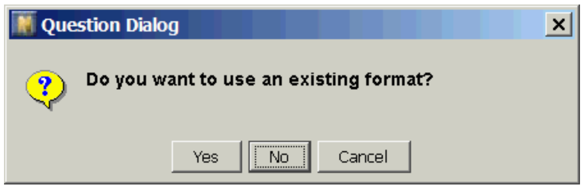
Files that do not have the extensions described above require a user-defined format. Defining a data file format is easy with NOL Studio - a Text Import Wizard takes you step by step through the process, allowing you to go back to any step you may wish to change. The Wizard is activated immediately if you answer No to the dialog above. So, let's imagine for a moment that this is the first data series being imported into NOL Studio, and that it is not formatted in either of the predefined formats. In this case, the Wizard will guide you through a new format creation.

Using the Text Import Wizard

You can open an ASCII file with an unknown format using the Text Import wizard. The basic assumption is that the data is arranged in rows and columns, where columns represent values of a certain variable at different times. NOL Studio provides several options for specifying the format of the delimiters, handling special symbols, and specifying the time and date information that may be found in the file. You save these specifications in a format file and associate this format with the current data series. You can also use this format for loading similarly formatted data series in the future.

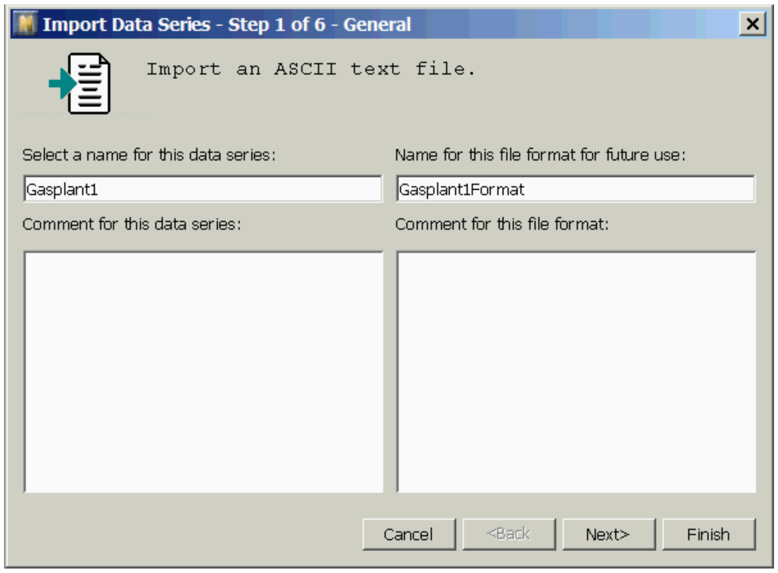
To use the Text Import Wizard to define a format for a specific data series:

1 Let's continue with the data series started in the previous section. The file name is *jpgdata.txt*, and you are asked if you wish to use an existing format.


Question Dialog

Do you want to use an existing format?

2 Click No. This invokes the Text Import wizard with the following dialog:


Import Data Series - Step 1 of 6 - General
Import an ASCII text file.
Cancel
<Back
Next>
Finish

Select a name for this data series: Gasplant1

Name for this file format for future use: Gasplant1Format

Comment for this data series:

Comment for this file format:

3 Enter a name for this data series.

Every import command creates a data series. The default name is based on the name of the file. Alternatively, you can name the data series to be imported by typing a string in the dialog.

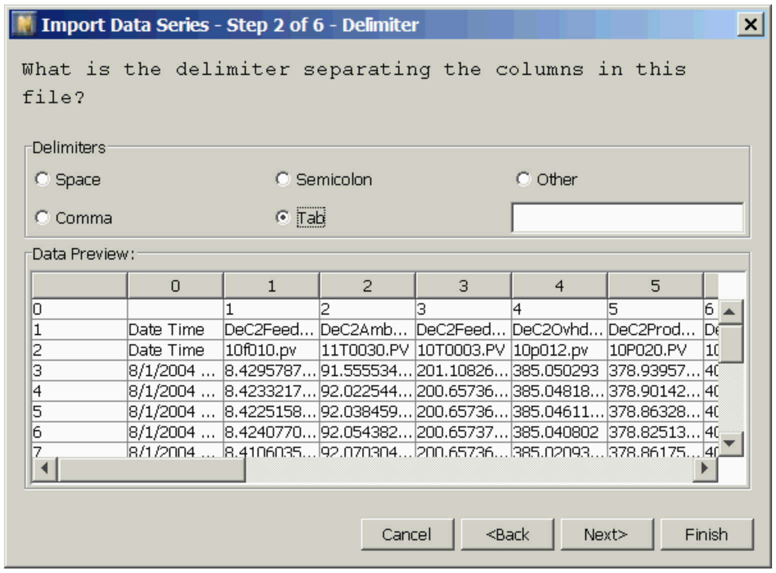
4 Enter a optional comment associated with this data series to remind yourself any information you may wish to remember in the future.

5 Enter the format name and comment for the format. The format name is essential if you wish to reuse it later for importing similar data series.

Tip The name of the data series should reflect a meaningful aspect of the imported data that lies at a higher level of abstraction than the cases to be extracted from the data series. For example, the name of the data series might be "JanFeb98Prod", denoting the entire production data for January and February. Following consistent naming conventions will help organize your NeurOn-Line Studio projects by indicating the type of data to be modeled.

---

6 Click Next to advance to the next screen, which asks you to specify the delimiter format of your data file.



Import Data Series - Step 2 of 6 - Delimiter

What is the delimiter separating the columns in this file?

Delimiters

Space [ ] Semicolon [ ] Other [ ]

Comma [ ] Tab [✓]

Data Preview:

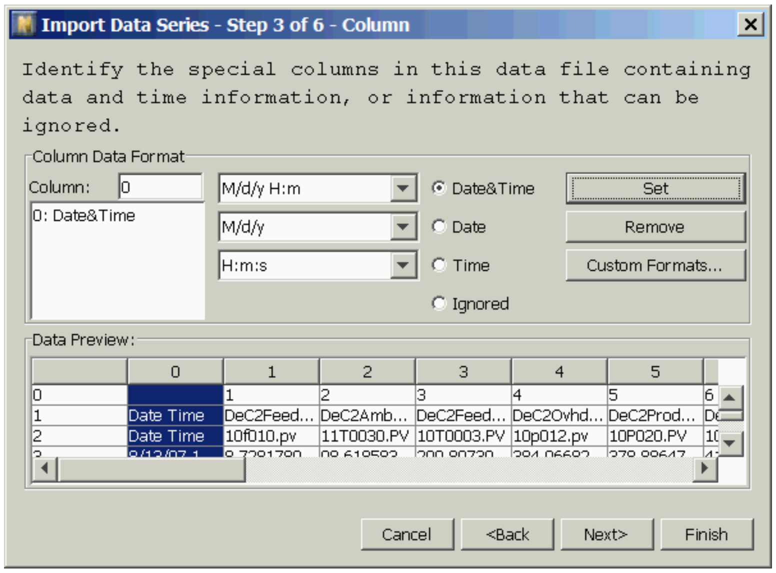
	0	1	2	3	4	5	
0		1	2	3	4	5	6
1	Date Time	DeC2Feed...	DeC2Amb...	DeC2Feed...	DeC2Ovhd...	DeC2Prod...	De
2	Date Time	10f010.pv	11T0030.PV	10T0003.PV	10p012.pv	10P020.PV	10
3	8/1/2004 ...	8.4295787...	91.555534...	201.10826...	385.050293	378.93957...	40
4	8/1/2004 ...	8.4233217...	92.022544...	200.65736...	385.04818...	378.90142...	40
5	8/1/2004 ...	8.4225158...	92.038459...	200.65736...	385.04611...	378.86328...	40
6	8/1/2004 ...	8.4240770...	92.054382...	200.65737...	385.040802	378.82513...	40
7	8/1/2004 ...	8.4106035...	92.070304...	200.65736...	385.02093...	378.86175...	40

NeurOn-Line Studio supports the reading of text delimited by white space characters, comma, semicolon and tab. An arbitrary string can also be specified as the separator.

7 Choose a delimiter by clicking on the appropriate radio button, or enter a string of your choice into the text box.

Once you choose a delimiter, the effects of the choice on how NeurOn-Line Studio will separate the data can be seen in the Data Preview window of the dialog. When the delimiter choice is changed, NeurOn-Line Studio applies the settings and displays the results in the Data Preview window immediately.

8 Click Next to advance to the next screen, which asks you to format the columns in the data file.



Import Data Series - Step 3 of 6 - Column

Identify the special columns in this data file containing data and time information, or information that can be ignored.

Column Data Format

Column: 0

0: Date&Time

M/d/y H:m — [✓ Date&Time]

M/d/y — Date

H:m:s — Time

Ignored

Data Preview:

	0	1	2	3	4	5	
0		1	2	3	4	5	6
1	Date Time	DeC2Feed...	DeC2Amb...	DeC2Feed...	DeC2Ovhd...	DeC2Prod...	De
2	Date Time	10f010.pv	11T0030.PV	10T0003.PV	10p012.pv	10P020.PV	10

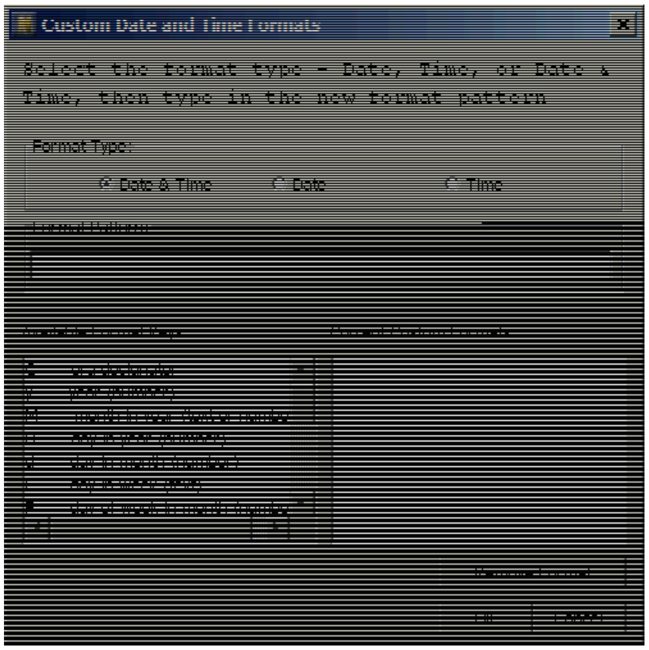
You need to select special formats for all columns that do not contain data, contain special data (such as date and time) or contain data that you wish to be ignored.

9 Select the column to be formatted clicking on the column header.

10 Choose the format setting by selecting one of the provided radio buttons. For example, if you select Date&Time, you can then examine the options of the associated drop-down list box to find the correct format for your data representation.

11 If you do not find a format to fit your date/time representation, click on the Custom Formats... button, which will bring up a dialog allowing you to design your own format.

For example:


Custom Date and Time Formats

Select the format type - Date, Time, or Date & Time, then type in the new format pattern

Format Type:

(● Date & Time) ○ Date ○ Time

a Select the Format Type: Date&Time, Date, or Time

b Click in the Format Pattern type-in box. You can either type in the format, using the Available Format Keys in the list box below, and any separators you wish, or you can double-click on any key in the Available Format Keys to insert it into the Format Pattern.

c With the cursor in the Format Pattern type-in box, press Return to insert the format into the Current Custom Format list.

d If you wish to delete a specific custom format, select it from the Current Custom Formats list, and click on the Remove Format button.

e Click on the OK button when you finish adding the necessary custom formats.

f The custom formats will appear at the top of the appropriate format lists in the Import Wizard, step 3. Select and apply custom formats to chosen columns.

Caution Date, Time, and Date&Time are handled as three separate choices. If a column contains both date and time information, you must choose Date&Time.

If a time-based process model is the goal of your modeling efforts, NeurOn-Line Studio requires accurate accounting of the time associated with each data sample. It is common for time information to be provided in a devoted column or columns by the file generation system in some consistent format.

For each of the main date and time options (Date&Time, Date, Time), there are several possible date and time formats.

When you enter select a Date, Time, or Date&Time format, NeurOn-Line Studio will attempt to confirm your selection by reading a small portion of the file.

- If there is an error parsing the date or time with the format you selected, you will be informed, and your choice will not be allowed.
- If parsing is successful, you will be asked to confirm that dates and times have been correctly interpreted. Selecting "yes" will confirm your choice.

Caution You must select a date/time format that exactly matches the date/time representation in the file, or the file information will not be interpreted correctly.

---

12 Click Ignored if the column you selected contains irrelevant data.

13 Click Set to save the format of the current column.The column number and its format is added to the formatted columns list, below the Current Column text box.

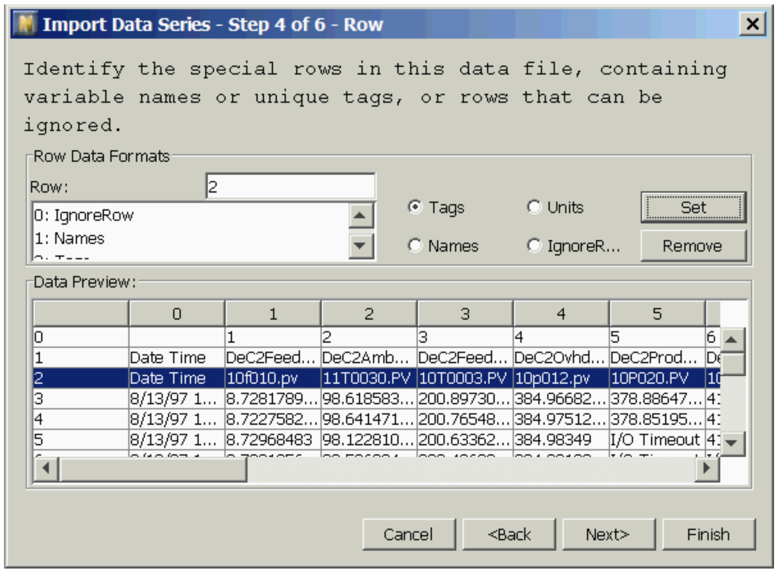
Caution If you forget to click Set, your specification will not be recorded.

---

14 To format another column, follow the same steps, starting with selecting the column number, and ending with clicking Set.

15 To remove a column format, select the column number and its associated format from the list box, and click Remove.

16 Click Next to advance to the next screen, where you are asked to format rows.


Import Data Series - Step 4 of 6 - Row

```
Identify the special rows in this data file, containing
variable names or unique tags, or rows that can be
ignored.
```

Row Data Formats

Row: 2

0: IgnoreRow
1: Names

[✓ Tags] Units Names IgnoreR...

Data Preview:

	0	1	2	3	4	5
0		1	2	3	4	5
1	Date Time	DeC2Feed...	DeC2Amb...	DeC2Feed...	DeC2Ovhd...	DeC2Prod...
2	Date Time	10f010.pv	11T0030.PV	10T0003.PV	10p012.pv	10P020.PV
3	8/13/97 1...	8.7281789...	98.618583...	200.89730...	384.96682...	378.88647...
4	8/13/97 1...	8.7227582...	98.641471...	200.76548...	384.97512...	378.85195...
5	8/13/97 1...	8.72968483	98.122810...	200.63362...	384.98349	I/O Timeout

Row formatting options are generally provided to deal with header information that is usually found at the beginning of a file, and often contains information about the contents and size of a file.

17 Select the row to be formatted, by either typing the row number into the dialog, or by clicking on the corresponding row header in the data preview window. Selecting the row by clicking on its header updates the row number displayed in the dialog.

18 Choose the format setting by selecting one of the radio buttons: Tags, Units, Names, or Ignore. Format settings allow you to specify that the row entries are to be interpreted as:

- **Tags**: unique alphanumeric strings that originate in a DCS (distributed control system) or data historian. If there are no tags, tag names will be created.
- **Units**: strings that indicate the measurement units of a particular variable, or column of data. Units are not used by NOL Studio, but are included for clarity.
- **Names**: a short string naming the variable. Names are usually more descriptive than tags.
- **Ignored**: an entire row can be ignored, based solely on its position within the data file. For example, the first row from a particular data historian may contain information related to the size of the file, which you will want NOL Studio to ignore.

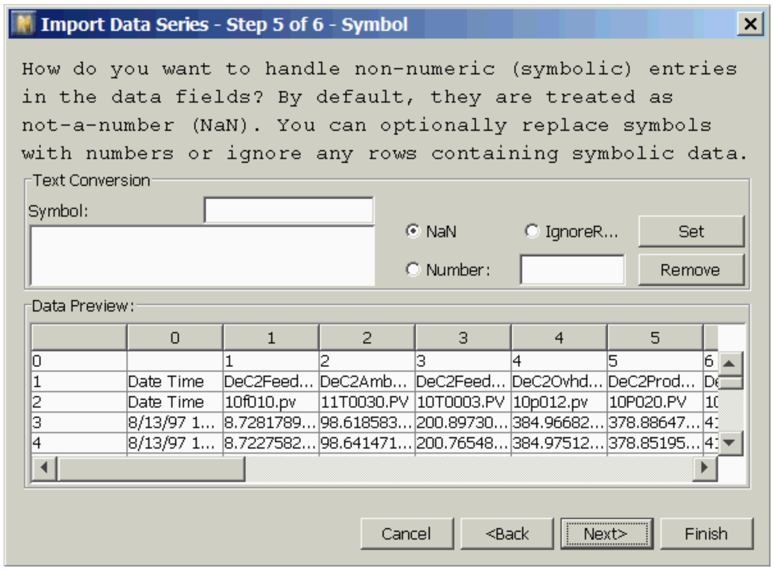
Only one setting can be selected for any given row.

19 When you are done formatting a row, click Set to associate the format with the current row.

20 To format another row, follow the steps described above.

21 To remove a row setting, select the row with its associated setting from the list box, and click the Remove button.

22 Click Next to advance to the next dialog, which asks you to format symbols.



How do you want to handle non-numeric (symbolic) entries in the data fields? By default, they are treated as not-a-number (NaN). You can optionally replace symbols with numbers or ignore any rows containing symbolic data.

[✓ NaN] IgnoreR... Number:

	0	1	2	3	4	5	
0		1	2	3	4	5	6
1	Date Time	DeC2Feed...	DeC2Amb...	DeC2Feed...	DeC2Ovhd...	DeC2Prod...	De
2	Date Time	10f010.pv	11T0030.PV	10T0003.PV	10p012.pv	10P020.PV	10
3	8/13/97 1...	8.7281789...	98.618583...	200.89730...	384.96682...	378.88647...	41
4	8/13/97 1...	8.7227582...	98.641471...	200.76548...	384.97512...	378.85195...	41

You can specify NeurOn-Line Studio's interpretation of special character strings, termed symbols, that may be found in the imported file. The system that originally produced the file may use a certain string to denote an event or failure in data collection. For example, the string "Error" may have been written in the data file to denote that a value was not read in correctly.

23 To interpret a symbol (such as "Error") in NOL Studio there are several choices:

- **Number**: a symbol can be interpreted as a fixed value, entered into the text box next to the Set button.
- **NaN** (default): a symbol can be interpreted as "not a number" or NaN. NaN is a common special value that is often used in computer systems to represent an illogical or unreliable result of some operation, such as the result of dividing by zero. In NeurOn-Line Studio, NaN is used in this sense, as well as when there is no value recorded for a data point.
- **Ignore Row**: ignore any row containing the specified symbol, which means that any row containing "Error" would not be imported, and NeurOn-Line Studio would resume importing on the following line.

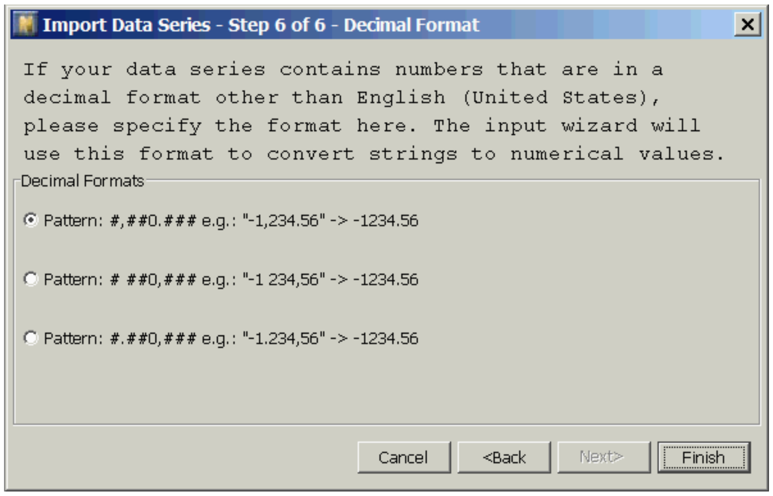
24 When you have specified the desired format for the first symbol, click Set.

25 To add another symbol format, repeat the steps above.

26 To remove a symbol format from the list, select the symbol with its associated format from the list box, and click Remove.

You can format an unlimited number of symbols.

27 Click Next to advance to the next dialog, which asks you to format the decimal numbers.


Import Data Series - Step 6 of 6 - Decimal Format

```
If your data series contains numbers that are in a
decimal format other than English (United States),
please specify the format here. The input wizard will
use this format to convert strings to numerical values.
```

Decimal Formats

[✓ Pattern: #,##0.### e.g.: "-1,234.56" -> -1234.56]

Pattern: # ##0,### e.g.: "-1 234,56" -> -1234.56

Pattern: #.##0,### e.g.: "-1.234,56" -> -1234.56

If your data series contains numbers that are in a decimal format other than English (United States), please specify the format here. The input wizard will use this format to convert strings to numerical values.

28 Click Finish to import the data series using your new format into NeurOn-Line Studio.

You have just created a user-defined format, which you can use later to import similar data series into NOL Studio. Now let's examine the predefined formats that NOL studio provides.

Predefined Formats

There are two predefined formats supported by the NOL Studio: ASCII and BINARY. The ASCII files must have a *.ds* extension and the BINARY files must have *.bds* extension. If your data series is saved as an ASCII file, it will import automatically into NOL Studio, when you select Open from the Import Data Series dialog.

The ASCII format for importing and exporting data series from files consists of the following lines:

1 The tag information line: contains variables' tags as strings separated with tab, commas, or space.

2 The name information line: contains variables' names as strings separated with the same separating character as in the tag line.

3 The unit information line: contains variables' unit names as strings separated with the same separating character as in the tag line.

4 Lines of data, one line for each data pair in the data series: contains the following items separated with the same separating character as in the tag line

   a The time stamp for the data pair as either a long or an integer.

   b The data value of the data pair.

Here is an example of a data series stored as text.

Time	var1	var2	var3	var4	var5
Time	var1	var2	var3	var4	var5
Millisecon	None	None	None	None	None
80000000	220	2600	8521.621	82.76367	45.65
80000001	220	2600	8521.621	83.1543	45.65
80000002	223	2443	8957.707	83.05664	46
80000003	221	2705	8989.459	82.91016	46
80000004	221	2705	8989.459	82.51953	46
80000005	221	2647	6255.802	82.66602	45.2
80000006	223	2422	7613.56	81.00586	45.35
80000007	222	2551	7212.631	83.34961	45.2
80000008	221	2737	9679.736	83.74023	45.55
80000009	242	2686	9072.731	83.39844	45.75
80000010	242	2686	9072.731	82.71484	45.75
80000011	241	2621	10145.72	84.17969	45.65
80000012	241	2621	10145.72	84.32617	45.65
80000013	242	2427	9447.666	83.93555	45.1
80000014	241	2230	7284.694	83.83789	45.35
80000015	241	2361	9223.239	84.27734	45.25
80000016	241	2235	5118.755	82.61719	45.25

To export data from NOL Studio in the ASCII format:

➔ Export the data series giving the file a `.ds` extension. The data is automatically saved in the NOL Studio predefined format described above.

The predefined BINARY file format is used by NOL Studio for saving and loading data series between NOL Studios.

To export data from NOL Studio in the BINARY format:

➔ Export the data series giving the file a `.bds` extension. The data series is automatically saved as a BINARY file with a predefined format.

Importing Data from G2

G2 Gateway supports two-way communication between dynamic external processes and G2 applications. Through a G2 Gateway bridge to an external system, you can quickly obtain real-time data that a G2 application needs to make intelligent control decisions in a time-critical processing environment. G2 Gateway bridges enable G2 KBs to communicate with a wide variety of external system, such as:

- Database management systems (DBMSs)
- Programmable logic controllers (PLCs)
- Supervisory control and data-acquisition (SCADA) systems
- Distributed control systems (DCSs)
- C/C++ programs, Non-G2 operator consoles or displays
- External simulation software

G2 Gateway bridges can communicate across the network that uses the TCP/IP or DECnet protocols. Gensym's Intelligent Communications Protocol (ICP), which is built into G2 Gateway, handles the details of network communication automatically, enabling you to develop distributed systems among heterogeneous platforms without having detailed knowledge of protocols or of network software in general.

Once you have imported data into G2 using the options mentioned above, you can import the data into NOL Studio in two different ways:

- Save data in G2 into ASCII files, and import data files into NOL Studio.
- Import data through G2 Gateway.

To import data through the G2 Gateway, you must have a G2 Gateway link between this NOL Studio and a G2 application. Also, the G2 application should have compatible dataset objects for NOL Studio to load.

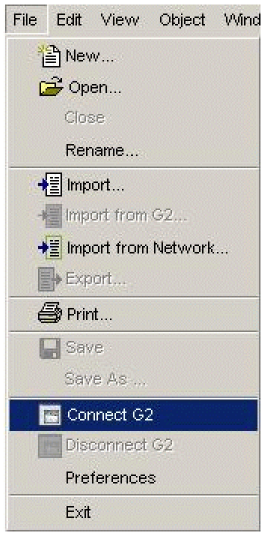
To load a data series from G2:

1 Connect NOL Studio and G2.

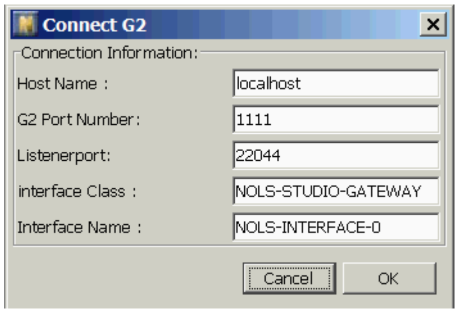
  There are two ways to connect NOL Studio with G2. You can connect from NOL Studio to an existing G2 process, or you can launch NOL Studio from G2 to connect that NOL Studio automatically to G2. Please refer to the *Gensym Neural Network Engine* for the steps to launch NOL Studio from G2.

To connect to G2 from NOL Studio:

➔ Choose File > Connect G2:



The Connect G2 dialog appears:



Connect G2

Connection Information:

Host Name :	localhost
G2 Port Number:	1111
Listenerport:	22044
interface Class :	NOLS-STUDIO-GATEWAY
Interface Name :	NOLS-INTERFACE-0

If the connection is successful, the connection information appears in the toolbar, for example:

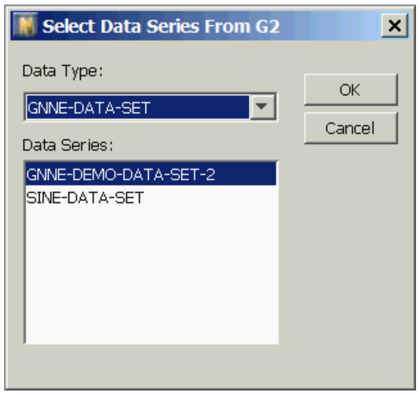
localhost:1111

You can now import data from G2.

Note The NOL Studio and G2 gateway is associated with each G2 window. Each G2 window can have only one NOL Studio connection. If you connect G2 from NOL Studio, the gateway is associated with the G2 server, even if the NOL Studio console and the G2 server are not on the same machine. To create a gateway between NOL Studio and a particular Telewindows, launch NOL Studio from that Telewindows.

2 Choose File > Import from G2 from the menu bar.

The following dialog appears:



Every import command creates a data series. The default name is based on the item name in G2. If the item does not have a name, the UUID of that item is the default name for the data series in NOL Studio. You can change the name of a data series from its property dialog.

Importing Data through Networks

NOL Studio can communicate across Intranet/Internet if it is based on the TCP/IP protocol. You don't need to have detailed knowledge of protocols or of network software in general as long as your machine is connected to the network. You can load the remote data series files by providing the URL link for the data files.

URL Format

It's often easiest, although not entirely accurate, to think of a URL as the name of a file on the World Wide Web. Most URLs refer to a file on some machine on the network, however, a URL also can point to other resources on the network, such as database queries and command output.

Note URL is an acronym for Uniform Resource Locator and is a reference (an address) to a resource on the Internet.

The following URL example addresses a remote file.

```
file://www.gensym.com/SW/NeurOnLine/com/gensym/nols/
docs/introduction.html
```

This URL has two main components:

Protocol identifier	`file`
Resource name	The rest of the line, starting with `www.gensym`

Note that the protocol identifier and the resource name are separated by a colon and two forward slashes. The protocol identifier indicates the name of the protocol to be used to fetch the resource. This example uses the File, which is typically used to point to a remote file. File is just one of many different protocols used to access different types of resources on the net. Other protocols include:

- Hypertext Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- Gopher
- News

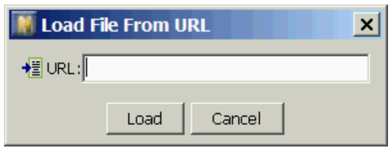
The resource name is the complete address to the resource. The format of the resource name depends entirely on the protocol used, but for many protocols, including File and HTTP, the resource name contains one or more of the components listed in the following table:

- **Host Name**: The name of the machine on which the resource lives.
- **Filename**: The pathname to the file on the machine.
- **Port Number**: The port number to which to connect (typically optional).
- **Reference**: A reference to a named anchor within a resource that usually identifies a specific location within a file

To load a data series from a remote file:

1 Select File > Import from Network from the menu bar.

The following dialog appears:

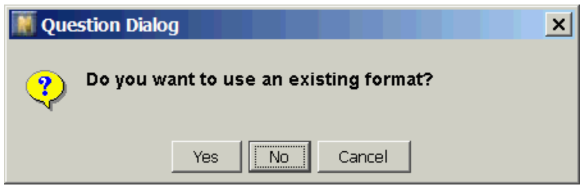


2 Enter the URL link of the file from which to load the data.

3 Click Load.

If the file is in one of the predefined formats, the data series will be loaded immediately into the NOL Studio.

4 In case of irregular file format, you will be asked if you wish to use an existing format or to create a new one.


Question Dialog

Do you want to use an existing format?

5 Follow the steps defined in Importing Data Series From Files, starting on , to create a new format, or to use an existing one.

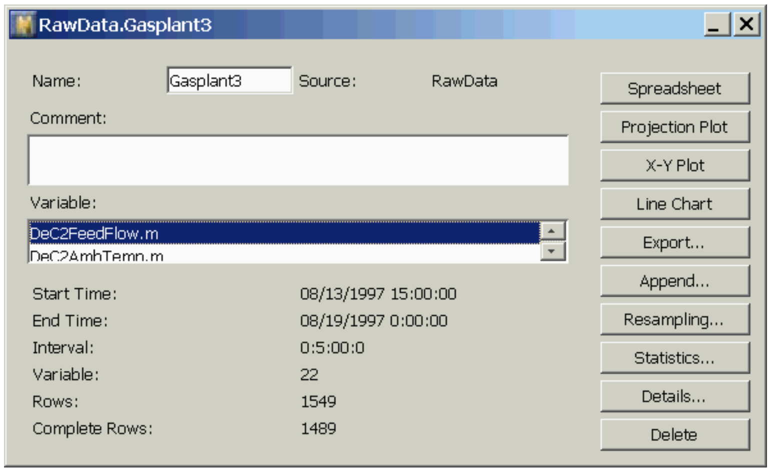
Viewing Data Series

Once you have imported data successfully into NOL Studio, you can begin to examine the properties of your data. You do this by accessing the properties table for the data series, and then drilling down to view individual variables.

To open a property table for any data series:

➔ In the tree view, double-click on any data series to display its properties table.

For example:


RawData.Gasplant3

Name: Gasplant3 Source: RawData

Comment:

Variable:

DeC2FeedFlow.m
DeC2AmbTemp.m

Start Time:	08/13/1997 15:00:00
End Time:	08/19/1997 0:00:00
Interval:	0:5:00:0
Variable:	22
Rows:	1549
Complete Rows:	1489

The property table for a data series displays the name and source of the data series. The source can be Raw Data, any preprocessor, a simulation, or optimization. You will also see the list of variables in the data series, the start and end times, number of rows of data, and the average time interval between data points. Several actions are available from the property table, such as launching spreadsheet or projection plot views, x-y plots, exporting the data, appending to the data series, and deleting the data series.

Hint After importing a data series, you should verify that the information has been loaded successfully by examining the data series property table.

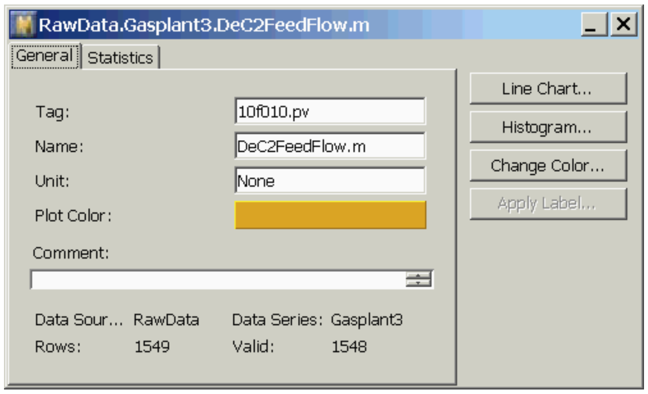
You can also view the properties of individual variables by opening their property tables. Individual variables are not listed in the tree view, because they are not represented as fully-fledged objects. Variables are subcomponents of a data series, corresponding to the columns of the data series.

To open a property table for any individual variable contained in a data series:

1. Open the property table of the data series containing the variable.
2. Do one of the following:
   - Select the variable you want in the scroll area, which shows all the variables in the data series, and then click the Details button.

     or
   - Double-click the variable you want in the scroll area.

For example, selecting the DeC2FeedFlow variable displays the following properties dialog:

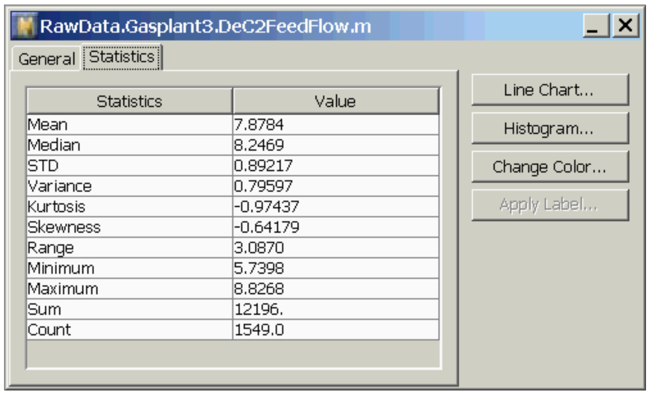


Tag:	10f010.pv
Name:	DeC2FeedFlow.m
Unit:	None
Plot Color:	🟧
Comment:	

Data Sour... RawData Data Series: Gasplant3
Rows: 1549 Valid: 1548

This dialog shows you the name, tag, and units for the variable. You can change the name and units, and enter a comment by editing the appropriate fields. Information on the data series, data source, total number of samples, and number of valid samples is also displayed. Using the buttons on the right side of the dialog, you can plot a line chart of the variable, and show a histogram of the variable. For more information on the line chart and histogram views, see the following chapter.

The second tab on this dialog, Statistics, shows summary statistics for the variable. These statistics include the maximum and minimum values, mean, standard deviation, variance, and more. This is what the Statistics tab looks like:



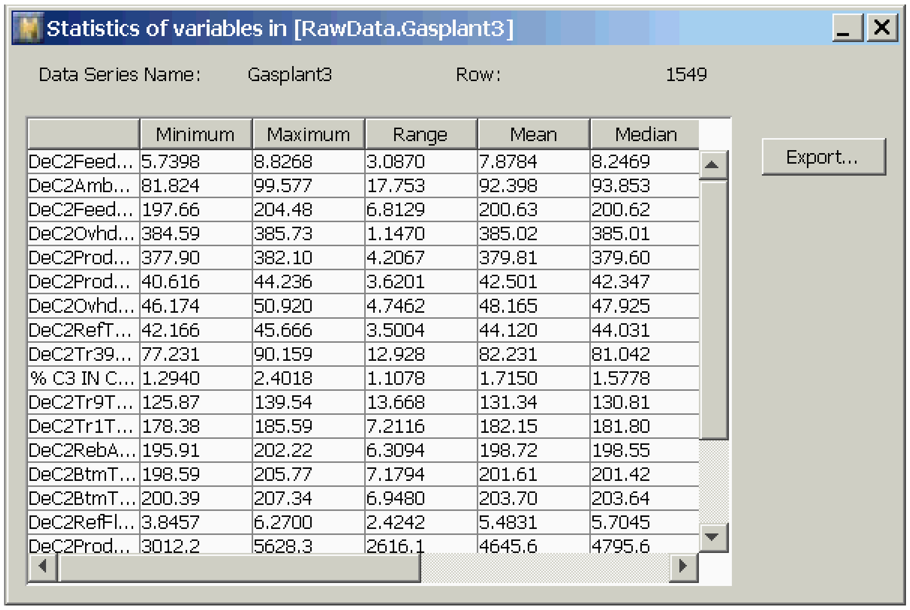
Statistics	Value
Mean	7.8784
Median	8.2469
STD	0.89217
Variance	0.79597
Kurtosis	-0.97437
Skewness	-0.64179
Range	3.0870
Minimum	5.7398
Maximum	8.8268
Sum	12196.
Count	1549.0

To view the statistical summary of all variables in a data series:

1 Open the property table of the data series, as described previously.

2 Click the Statistics button.

A table similar to the following example appears:



Data Series Name: Gasplant3 Row: 1549

	Minimum	Maximum	Range	Mean	Median
DeC2Feed...	5.7398	8.8268	3.0870	7.8784	8.2469
DeC2Amb...	81.824	99.577	17.753	92.398	93.853
DeC2Feed...	197.66	204.48	6.8129	200.63	200.62
DeC2Ovhd...	384.59	385.73	1.1470	385.02	385.01
DeC2Prod...	377.90	382.10	4.2067	379.81	379.60
DeC2Prod...	40.616	44.236	3.6201	42.501	42.347
DeC2Ovhd...	46.174	50.920	4.7462	48.165	47.925
DeC2RefT...	42.166	45.666	3.5004	44.120	44.031
DeC2Tr39...	77.231	90.159	12.928	82.231	81.042
% C3 IN C...	1.2940	2.4018	1.1078	1.7150	1.5778
DeC2Tr9T...	125.87	139.54	13.668	131.34	130.81
DeC2Tr1T...	178.38	185.59	7.2116	182.15	181.80
DeC2RebA...	195.91	202.22	6.3094	198.72	198.55
DeC2BtmT...	198.59	205.77	7.1794	201.61	201.42
DeC2BtmT...	200.39	207.34	6.9480	203.70	203.64
DeC2RefFl...	3.8457	6.2700	2.4242	5.4831	5.7045
DeC2Prod...	3012.2	5628.3	2616.1	4645.6	4795.6

The variable statistics are calculated from the data series used to develop models. This information is also useful in the online environment for data preproccessing. A set of APIs exist for the `OnlinePredictor` and `OnlineOptimizer` classes. The detail of these APIs can be found in the changes to both the G2 and ActiveX API sections.

Exporting Data

NOL Studio allows you to export raw data series and processed data into data files. You can save a data series to a binary file or to an ASCII file. NOL Studio supports several predefined formats. If the current NOL Studio console is connected to a G2 process, you can also export data series directly into a data set object defined in the Gensym Neural Network Engine.

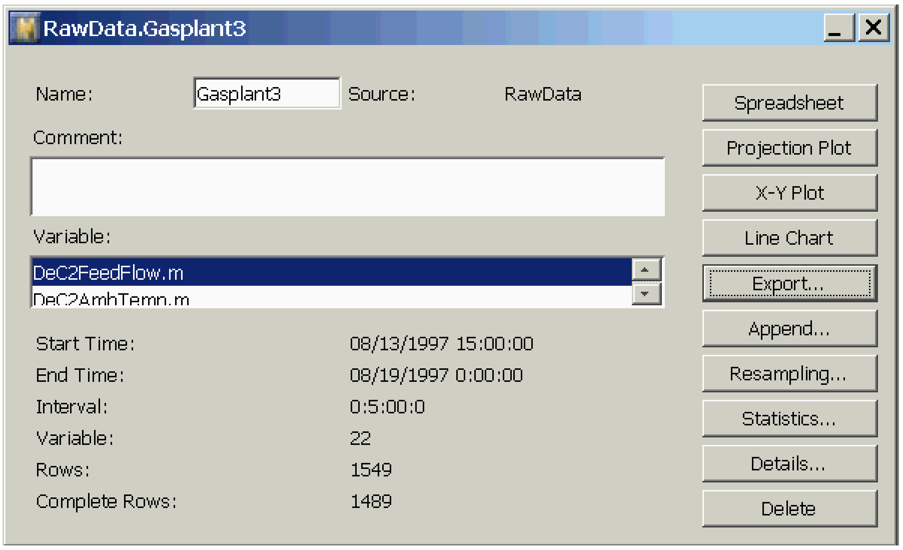
To export a data series from NOL Studio, you use a menu choice or a button on the data series properties workspace. Either technique displays the Export Data Series dialog. The steps for exporting the data series depend on whether NOL Studio is connected to G2.

To export a data series:

➔ Select File > Export from the menu bar.

or

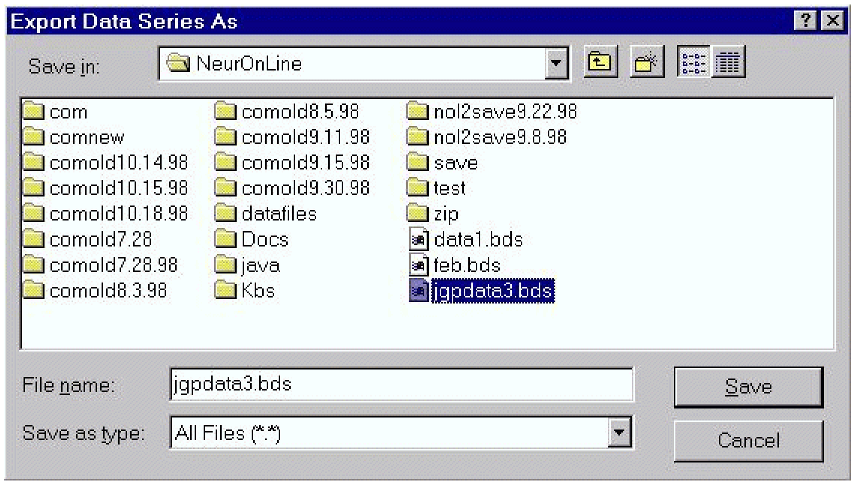
➔ Click the Export button in the data series properties dialog:


RawData.Gasplant3

Name:	Gasplant3
Source:	RawData
Comment:	
Variable:	DeC2FeedFlow.m, DeC2AmbTemp.m
Start Time:	08/13/1997 15:00:00
End Time:	08/19/1997 0:00:00
Interval:	0:5:00:0
Variable:	22
Rows:	1549
Complete Rows:	1489

To export data when NOL Studio is not connected to G2:

1 Choosing the Export menu choice or clicking the Export button displays the Export Data Series dialog Series As dialog:



File name: jgpdata3.bds

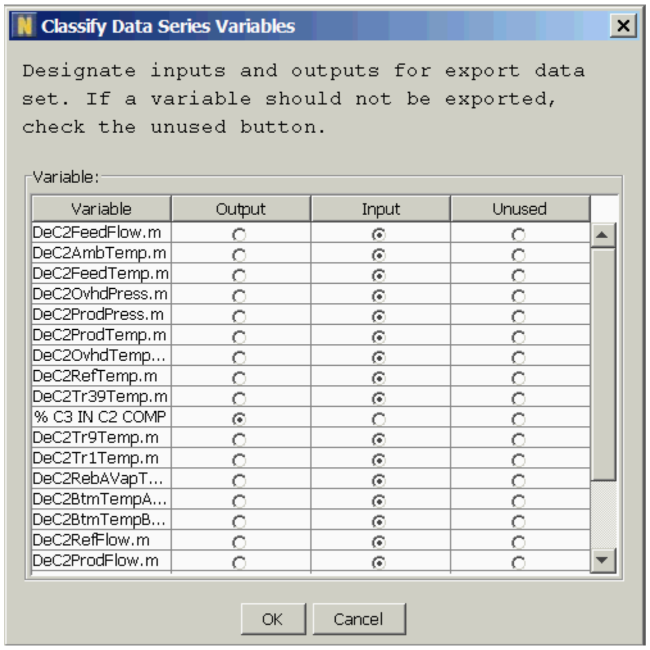
Save as type: All Files (*.*)

2 Select the file format by specifying one of these file type extensions:

- `.ds` for predefined ASCII files.
- `.bds` for BINARY files.
- `.csv` for comma separated ASCII file with the time format as specified in the global preferences.
- `.txt` for tab separated ASCII file with the time format as specified in the global preferences.
- `.set` for Data Set format in NOL Classic and GNNE. This format requires you specify the input and output variables from the data series.

3 Click the Save button.

4 If you select `.set` Data Set format, a variable classification dialog appears for you to specify the input and output:



Classify Data Series Variables

Designate inputs and outputs for export data set. If a variable should not be exported, check the unused button.

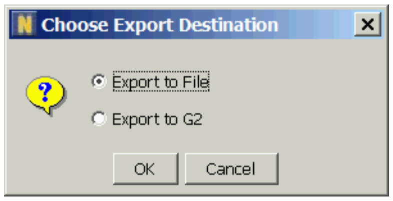
Variable:

Variable	Output	Input	Unused
DeC2FeedFlow.m	○	●	○
DeC2AmbTemp.m	○	●	○
DeC2FeedTemp.m	○	●	○
DeC2OvhdPress.m	○	●	○
DeC2ProdPress.m	○	●	○
DeC2ProdTemp.m	○	●	○
DeC2OvhdTemp...	○	●	○
DeC2RefTemp.m	○	●	○
DeC2Tr39Temp.m	○	●	○
% C3 IN C2 COMP	●	○	○
DeC2Tr9Temp.m	○	●	○
DeC2Tr1Temp.m	○	●	○
DeC2RebAVapT...	○	●	○
DeC2BtmTempA...	○	●	○
DeC2BtmTempB...	○	●	○
DeC2RefFlow.m	○	●	○
DeC2ProdFlow.m	○	●	○

OK Cancel

To export data when NOL Studio is connected to a G2 process:

1 Choosing the Export menu choice or clicking the Export button displays the following dialog for choosing the export destination:



Choose Export Destination

[✓ Export to File]

Export to G2

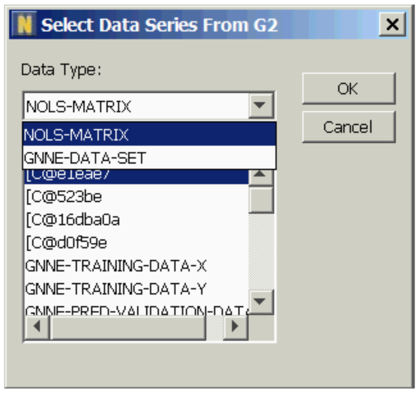
OK Cancel

2 Choose the export destination:

➔ If you choose export the data series to a file, the Export Data Series dialog appears and you can follow above steps to save the data series to a file.

or

a If you choose export the data series into a G2 data object, a selection dialog appears:

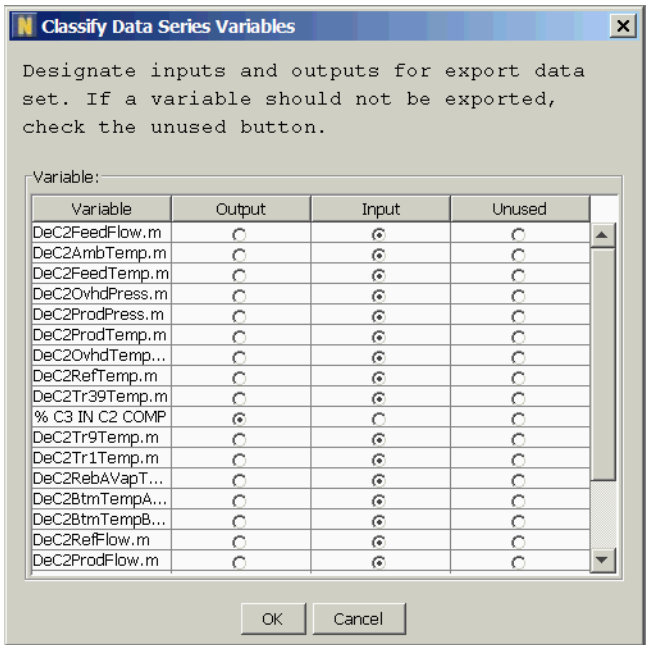


b Choose the data object into which to export the data in G2.

You can export data into one of two types of data objects: nols-matrix and gnne-data-set:

- nols-matrix object — Stores the data series as a matrix.
- gnne-data-set — Stores the data series as a data set, which allows you to classify the input and output for the data series.

c If you choose gnne-data-set, the following dialog appears for classifying the input and output data:


Classify Data Series Variables

Designate inputs and outputs for export data set. If a variable should not be exported, check the unused button.

Variable:

Variable	Output	Input	Unused
DeC2FeedFlow.m	○	●	○
DeC2AmbTemp.m	○	●	○
DeC2FeedTemp.m	○	●	○
DeC2OvhdPress.m	○	●	○
DeC2ProdPress.m	○	●	○
DeC2ProdTemp.m	○	●	○
DeC2OvhdTemp...	○	●	○
DeC2RefTemp.m	○	●	○
DeC2Tr39Temp.m	○	●	○
% C3 IN C2 COMP	●	○	○
DeC2Tr9Temp.m	○	●	○
DeC2Tr1Temp.m	○	●	○
DeC2RebAVapT...	○	●	○
DeC2BtmTempA...	○	●	○
DeC2BtmTempB...	○	●	○
DeC2RefFlow.m	○	●	○
DeC2ProdFlow.m	○	●	○

d Click OK to export the data into selected G2 data object.

Appending Data

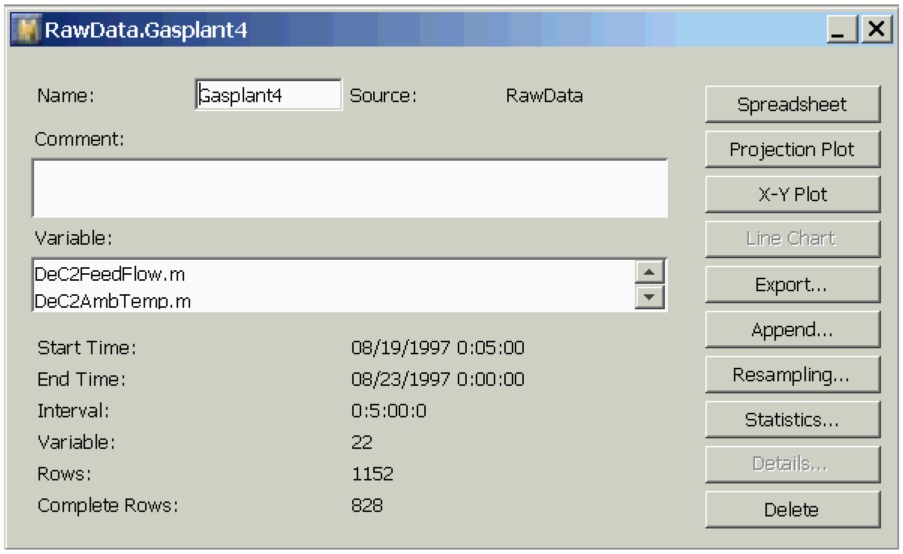
The append action is used to read additional data into new rows appended to an existing data series containing the same variables. When you append a new data series to the existing one, the following checks and actions may be taken:

- The earliest time stamp in the new data series has to be after the latest time stamp in the existing data series, or
- The latest time stamp in the new data series has to be before the earliest time stamp in the existing data series
- Variables from two data series with the same Tag and Name have their values appended row by row according to the time stamp of each row
- A variable from one data series without a matched Tag and Name from other data series will occupy a column in the appended data series with NaNs filled in corresponding rows of the other data series

You can append a data file into an existing data series by clicking the Append button in that data series property workspace. The Append Data Series dialog is invoked to tell NeurOn-Line Studio where to find the data.

To append a data series to an existing data series:

1 Click Append in the data series property workspace:



RawData.Gasplant4

Name:	Gasplant4
Source:	RawData
Comment:	
Variable:	DeC2FeedFlow.m DeC2AmbTemp.m
Start Time:	08/19/1997 0:05:00
End Time:	08/23/1997 0:00:00
Interval:	0:5:00:0
Variable:	22
Rows:	1152
Complete Rows:	828

2 You import the data file to append in the same way as you import a new data file from the file system.

Removing a Data Series

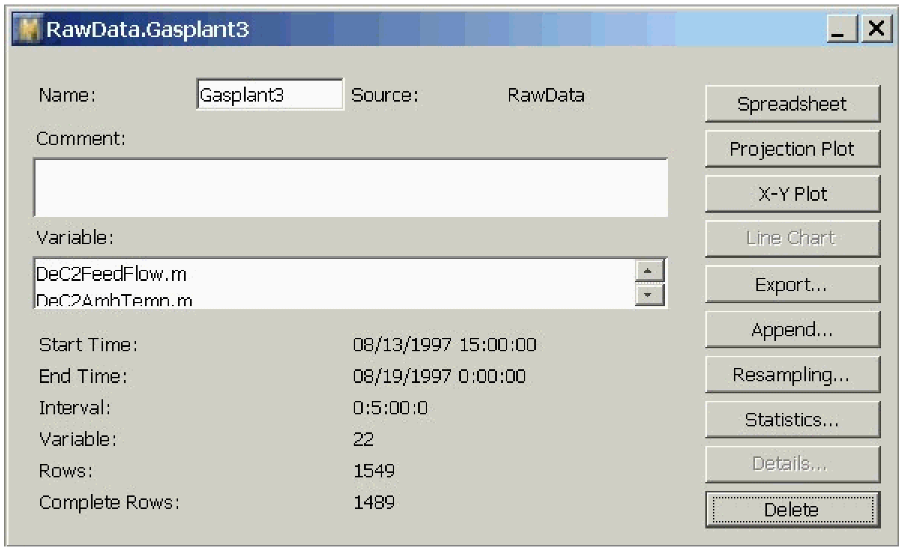
You can remove a data series from the project as long as it is not being used by views or preprocessors. The requirements for removing a data series are as follows:

- There is no view, except its own property workspace, opened for this data series or any variable in this data series. The views include spreadsheet, line chart, projection chart, X-Y chart, variable property workspace, and variable histogram.
- There is no processed data derived from this data series by going through preprocessors.

There are two methods to delete a data series: by clicking Delete in that data series property workspace, or by using right mouse click on the data series object in the tree view and choosing Delete.

To delete a data series using a button on the data series properties workspace:

➔ Click Delete on the data series properties workspace:



Name: Gasplant3 Source: RawData

Comment:

Variable:

DeC2FeedFlow.m
DeC2AmbTemp.m

Start Time:	08/13/1997 15:00:00
End Time:	08/19/1997 0:00:00
Interval:	0:5:00:0
Variable:	22
Rows:	1549
Complete Rows:	1489

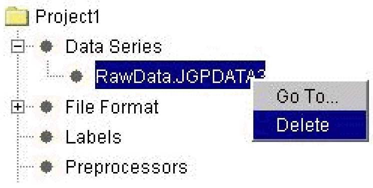
To delete a data series from a tree view:

1 Navigate in the tree view to find the data series you wish to remove.

2 Right-click on the data series to display its pop-up menu.

3 Choose Delete from the menu.

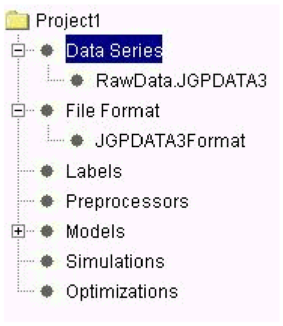
For example:



Managing Data Formats in NOL Studio

When you import a data series from a text files, the format you define is automatically saved by NOL Studio. When a format is saved, a file format object is created automatically and appears in the tree view.

This is an example of a tree view with one user-defined format:



Later you can use this file format object to load similarly formatted files.

Tip In the Data Import wizard, give your file format a name you can remember and easily associate with the type of file it describes.

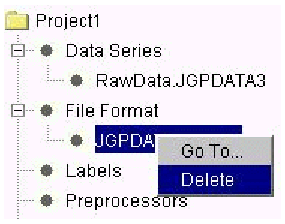
Deleting File Formats

There are two methods to remove a file format object from NOL Studio: clicking on the Delete button in the file format properties workspace, or from a pop-up menu on the tree view.

To delete a file format from the tree view:

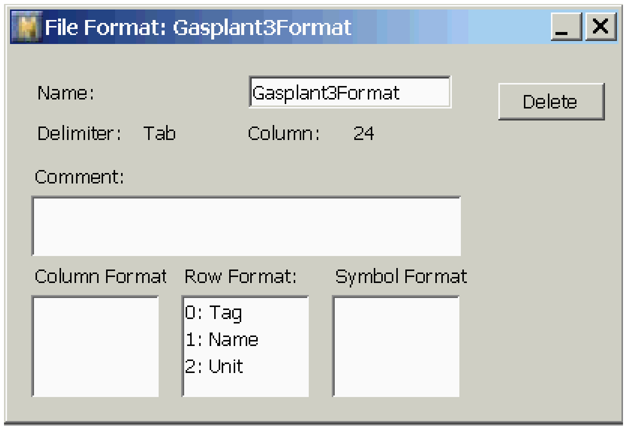
1 Right-click on the selected file format in the tree view and choose Delete from its pop-up menu.

For example:



To delete a file format from the file format properties workspace:

➔ Click Delete.



Name: Gasplant3Format

Delimiter: Tab

Column: 24

0: Tag
1: Name
2: Unit

Visualizing Data

Describes how to visualize and explore data through charts, graphs, and tabular views.

Introduction

NOL Studio allows you visualize data in many different views. Each view presents a different aspect of your data, and helps you gain additional insight into the underlying process. Having different visualizations of your data also helps you locate anomalies in your data, so you can remove them before training a model.

Specifically, these views are:

- Spreadsheet view

  This view allows you to view a data series in a tabular, column/row format.

- Line chart view

  This view allows you to plot one or more variables versus time or row index.

- X-Y scatter chart view

  This view allows you to plot one variable versus another variable, with time implicit. The number of rows of both variables viewed must be of equal length.

- Projection chart view

  This view depicts a projection of selected variables from a single data series, using Principal Component Analysis (PCA). Projection plots are powerful ways to examine the multivariate distribution of your data.

- Histogram view

  This view depicts a bar chart showing the distribution of a specified variable.

You access any of these views either from the NOL Studio View menu or the toolbar. You can also open these views from G2 through a procedure call. For details, see the *Gensym Neural Network Engine*. All of these views are read-only in that you cannot modify the data contained within the view. However, the views are also interactive, allowing you to select and label data using mouse gestures. Zooming is supported in all chart types, except the histogram view.

The various views are optimized to handle large data sets, and can efficiently display upwards of 100,000 values. Limitations are based upon the amount of available memory.

Views are dynamically linked to the source data, so that if source values or labels change, they are consequently reflected in all open views. Thus, if you label a range of data on one view, the label simultaneously appears on other views of the same variable.

Viewing Data in a Spreadsheet

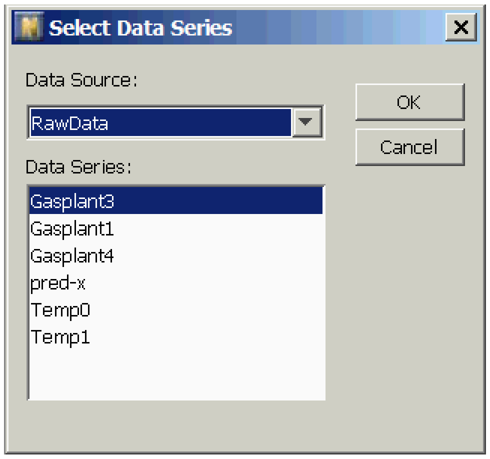
Spreadsheets allow you to visualize data in a tabular, row/column format. Each spreadsheet allows you to view the data in exactly one data series. Variables from different data series cannot be displayed as columns of a single spreadsheet, since the time stamps of each row may not correspond.

To open a spreadsheet view:

➔ Do one of the following:

Menu Bar:	Choose View > Spreadsheet.
Toolbar:	Click the Open Spreadsheet button.

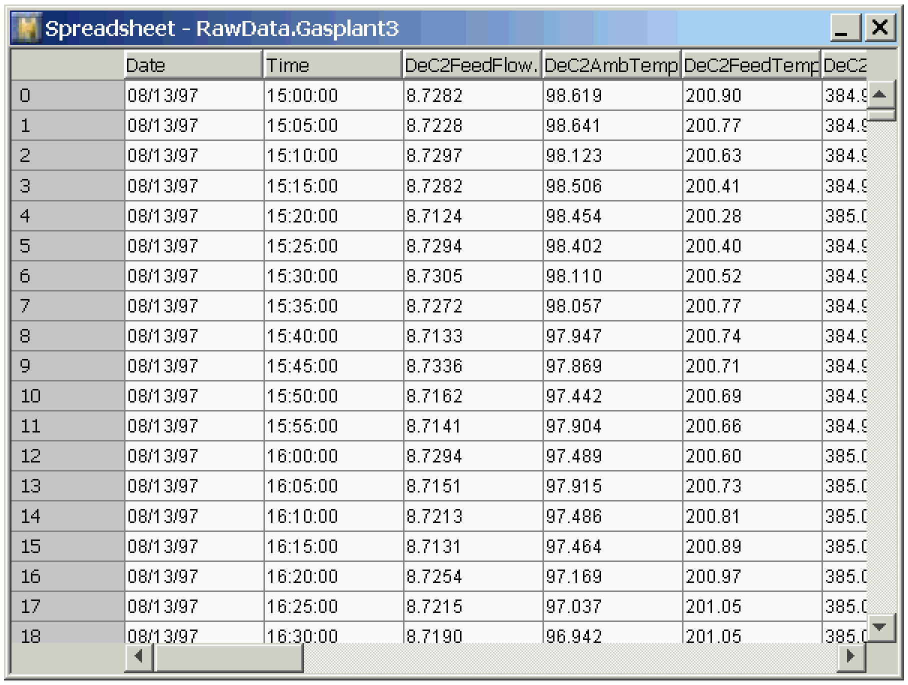
The Select Data Series dialog appears, for example:



Data Source: RawData

Data Series:
Gasplant3
Gasplant1
Gasplant4
pred-x
Temp0
Temp1

This dialog allows you to specify the data source and the data series to be displayed in the spreadsheet. After selecting the data source and data series that you want to view, click OK to display a spreadsheet view of the specified data series. For example:



	Date	Time	DeC2FeedFlow.	DeC2AmbTemp	DeC2FeedTemp	DeC2
0	08/13/97	15:00:00	8.7282	98.619	200.90	384.9
1	08/13/97	15:05:00	8.7228	98.641	200.77	384.9
2	08/13/97	15:10:00	8.7297	98.123	200.63	384.9
3	08/13/97	15:15:00	8.7282	98.506	200.41	384.9
4	08/13/97	15:20:00	8.7124	98.454	200.28	385.0
5	08/13/97	15:25:00	8.7294	98.402	200.40	384.9
6	08/13/97	15:30:00	8.7305	98.110	200.52	384.9
7	08/13/97	15:35:00	8.7272	98.057	200.77	384.9
8	08/13/97	15:40:00	8.7133	97.947	200.74	384.9
9	08/13/97	15:45:00	8.7336	97.869	200.71	384.9
10	08/13/97	15:50:00	8.7162	97.442	200.69	384.9
11	08/13/97	15:55:00	8.7141	97.904	200.66	384.9
12	08/13/97	16:00:00	8.7294	97.489	200.60	385.0
13	08/13/97	16:05:00	8.7151	97.915	200.73	385.0
14	08/13/97	16:10:00	8.7213	97.486	200.81	385.0
15	08/13/97	16:15:00	8.7131	97.464	200.89	385.0
16	08/13/97	16:20:00	8.7254	97.169	200.97	385.0
17	08/13/97	16:25:00	8.7215	97.037	201.05	385.0
18	08/13/97	16:30:00	8.7190	96.942	201.05	385.0

Notice that the first two columns show the date and time of the corresponding row of the data series. The time stamps apply to the entire row.

Viewing Data in a Line Chart

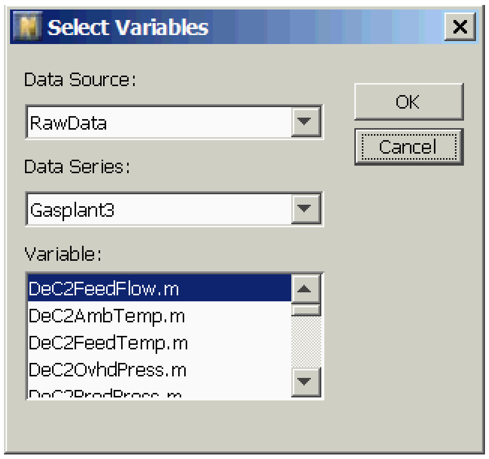
Line charts allow you to view one or more variables, plotted versus time or row. Line charts are useful for showing the trends of variables. Variables from different data series can be plotted in the same line chart. Because of this ability, line charts are useful for comparing data from different data sources, such as raw versus preprocessed data, or model predicted values versus actual values.

To open a line chart view:

➔ Do one of the following:

Menu Bar:	Choose View > Line Chart.
Toolbar:	Click the Open Line Chart button.

The following dialog appears:



Select Variables

Data Source: RawData

Data Series: Gasplant3

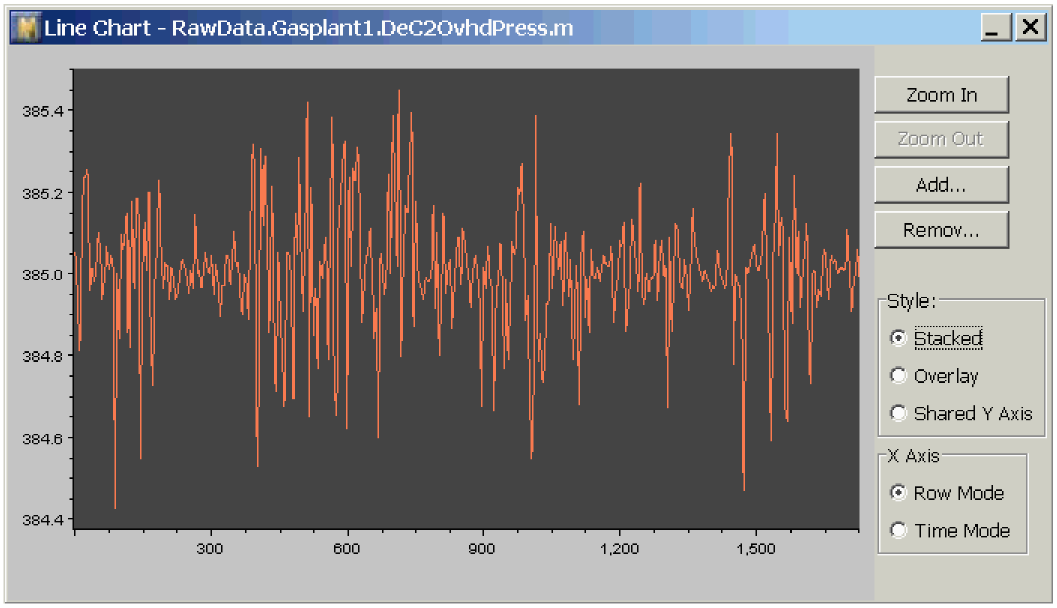
Variable:

- DeC2FeedFlow.m
- DeC2AmbTemp.m
- DeC2FeedTemp.m
- DeC2OvhdPress.m

You use this dialog to select the data source, data series, and variable or variables you wish to plot in the line chart:

If you want to plot more than one variable from a data series, select the desired variables in the scroll area. Then click OK to open the line chart with the specified variable(s).

If you select a single variable to plot, the line chart view might look something like this:



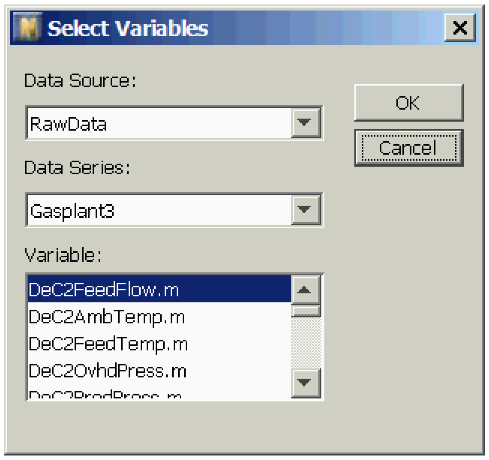
Style: [✓ Stacked] Overlay Shared Y Axis

X Axis: [✓ Row Mode] Time Mode

The buttons on the right side of the view control zooming, adding and removing data series, and changing the handling of the x and y axes. These options are explained in the following sections.

Adding and Removing Variables from the Line Chart

You may add more variables to the plot by clicking Add, which opens the following dialog, where you specify the variable(s) to be added to the chart:



Data Source: RawData

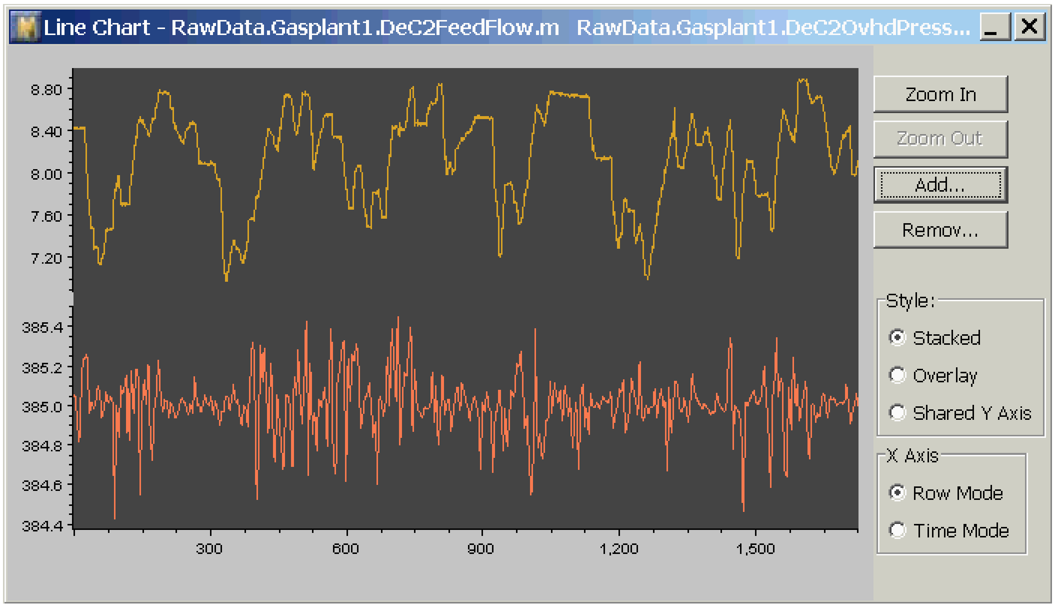
Data Series: Gasplant3

Variable:
DeC2FeedFlow.m
DeC2AmbTemp.m
DeC2FeedTemp.m
DeC2OvhdPress.m

Note that each variable is described by three attributes:

- Data source
- Data series
- Variable name

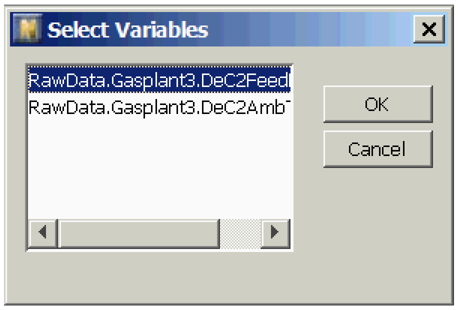
Click OK after you have selected the variables you want to add to the chart. If you added a second variable, the result might look something like this:



Style: [✓ Stacked] Overlay Shared Y Axis

X Axis: [✓ Row Mode] Time Mode

To remove a variable (or variables) from the Line Chart, click Remove. A dialog appears to allow you to select a variable (or variables) to remove. The dialog looks like this:



RawData.Gasplant3.DeC2Feed
RawData.Gasplant3.DeC2Amb

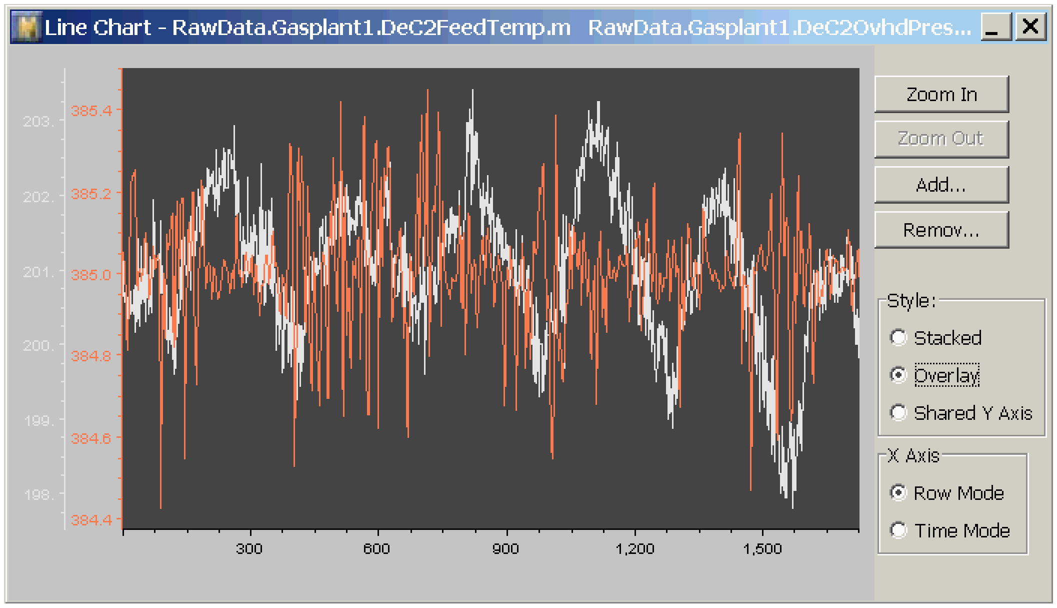
Setting Axis Styles

The line chart offers three styles for the y (vertical) axis, and two styles for the x (horizontal) axis. The y-axis styles are stacked, overlay, and shared. The x-axis styles are row or time.

Y-Axis Styles

When multiple data series are displayed on a line chart, by default, they are shown as separate plots, spanning the same horizontal range, but with separate y axes. This is called the stacked style. The stacked style is illustrated by the plot in the previous section.

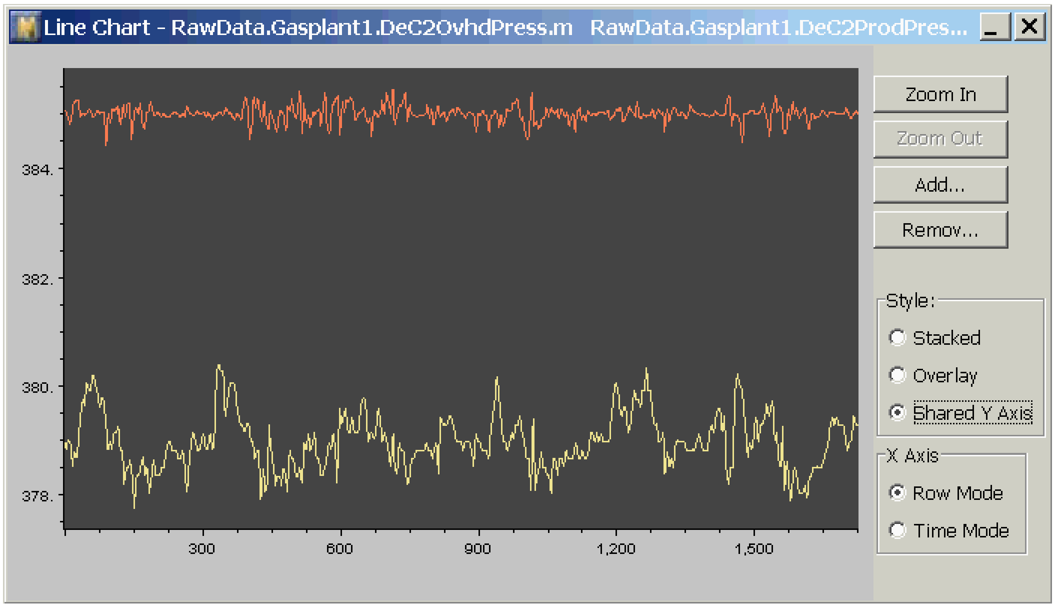
To overlay variables on top of each other, check the Overlay radio button in the Style box to the right of the chart. In the overlay style, the plots share the same area, but each variable has its own y axis, as shown below:



Style: Stacked [✓ Overlay] Shared Y Axis

X Axis: [✓ Row Mode] Time Mode

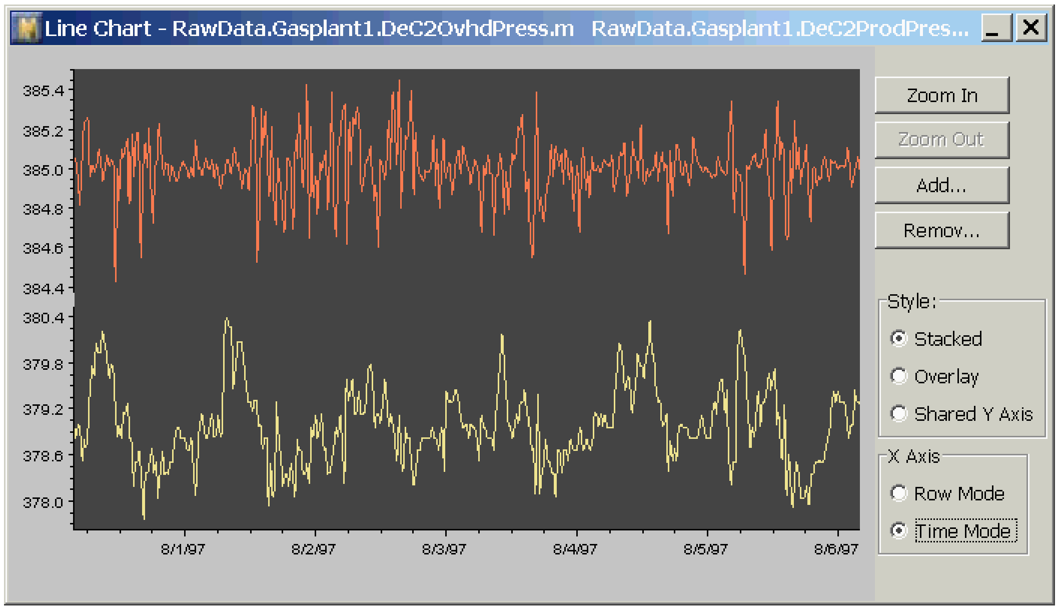
The final y axis style is shared y axis. In this style, all variables are shown on the same scale. This option is most appropriate when the range of all the variables is approximately the same. To view variables on a shared y axis, check Shared Y Axis on the Style menu to the right of the chart. The result looks like this:



Style: Stacked [ ] Overlay [ ] [✓ Shared Y Axis]
X Axis: [✓ Row Mode] Time Mode [ ]

X Axis Styles

There are two styles for the x axis, row and time. The default view option for the x axis is by row. To view by time, check Time in the x axis menu to the right of the chart. Shown in time mode, the line chart looks like this:

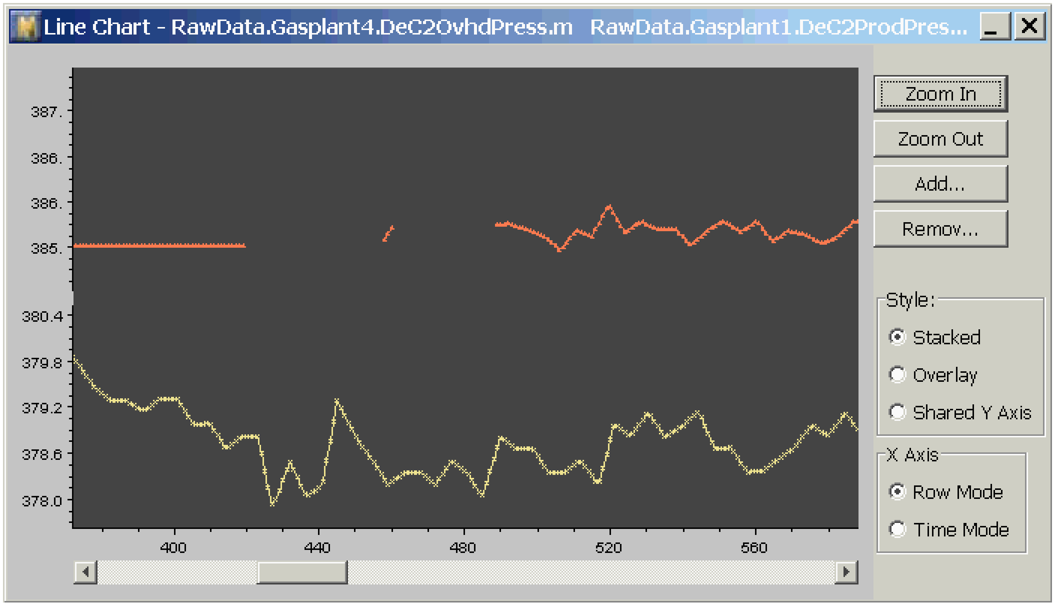


Style: [✓ Stacked] Overlay [ ] Shared Y Axis [ ]
X Axis: Row Mode [ ] [✓ Time Mode]

Zooming

The line chart allows you to interactively zoom in/out and scroll through your data. When you zoom in, the location of the point in the center of the plot is invariant. Thus, if you want to zoom in on a particular point, scroll the plot horizontally until the point is shown in the center of the screen. Then select the zoom in button.

When you zoom in, plot symbols representing the individual data points will appear. In the line chart view below, the chart has been zoomed in to the point where individual data points are visible:



In a line chart, you cannot magnify the y axis by zooming.

Display of Missing Values

Missing or invalid values, represented by the symbol NaN (not a number) are indicated by a gap in the plot. A gap indicates that at least one point between the visible points has the value NaN. Note a section of missing values in the figure directly above.

Tool Tips

When you place the mouse over a data point in a line chart, a tool tip (a small pop-up text window) appears, displaying the following information:

- The fully-qualified variable name, which is a concatenation of the data source, data series name, and variable name.
- The time or row.
- The value.

You can use the tool tip to identify the exact row number, time and value of the point.

Viewing Data in a X-Y Scatter Chart

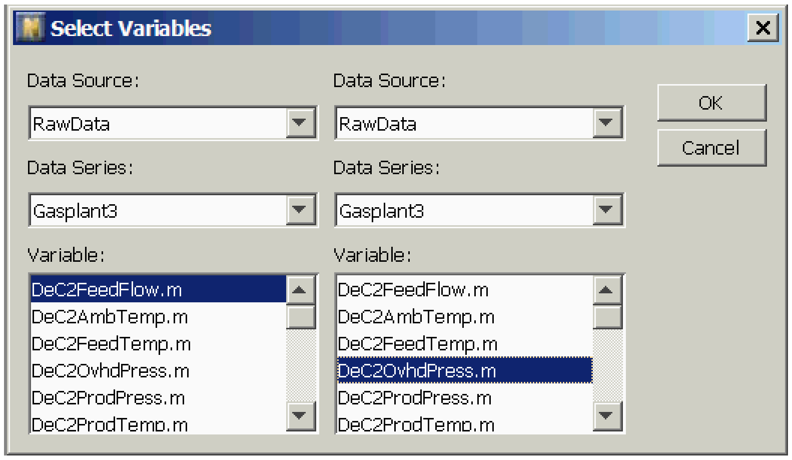
X-Y Scatter Charts allow you to plot one variable versus another variable, with time implicit. The number of rows of both variables viewed must be of equal length. The variables can be from different data series.

To view an X-Y scatter chart of two variables:

➔ Do one of the following:

Menu Bar:	Choose View > X-Y Chart.
Toolbar:	Click the Open Scatter Chart button.

The Select Variables dialog appears, for example:

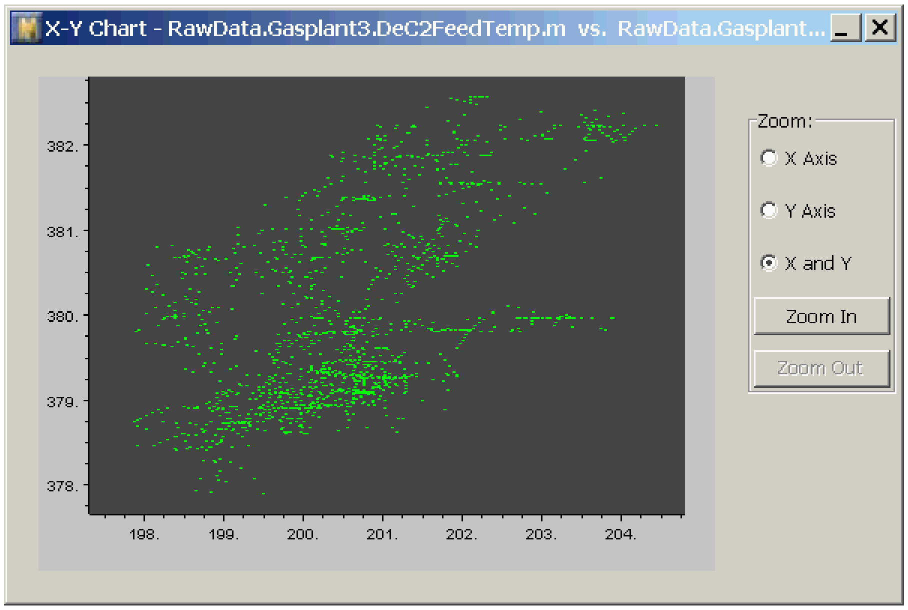


Data Source: RawData
Data Series: Gasplant3
Variable: DeC2FeedFlow.m, DeC2AmbTemp.m, DeC2FeedTemp.m, DeC2OvhdPress.m, DeC2ProdPress.m, DeC2ProdTemp.m

Data Source: RawData
Data Series: Gasplant3
Variable: DeC2FeedFlow.m, DeC2AmbTemp.m, DeC2FeedTemp.m, DeC2OvhdPress.m, DeC2ProdPress.m, DeC2ProdTemp.m

This dialog allows you to specify two variables from their corresponding data series and data sources that you wish to plot in the X-Y chart.

Once you have selected the variables you wish to view in the X-Y chart, click OK to display an X-Y chart, similar to the following:



Zooming

In X-Y scatter charts, you can zoom in and out on either the X or Y axis, or both. The zooming mode is controlled by the buttons on the right side of the chart.

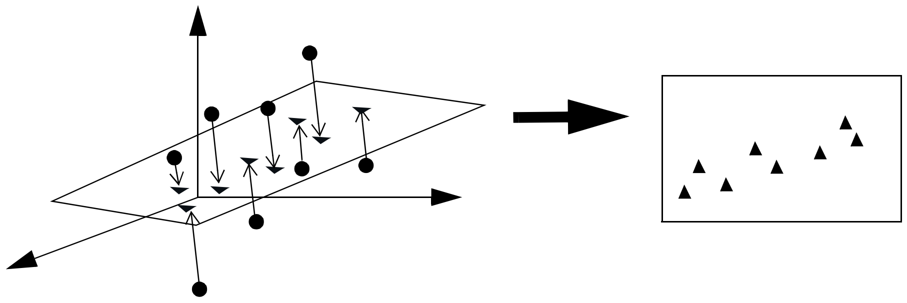
Viewing Data in Projection Charts

Projection charts show a “shadow portrait” of your data. Geometrically, your data set can be represented as a scatter of points in *m*-dimensional space, where *m* is the number of variables. Just as you can shine a light at a three-dimensional object to cast a two-dimensional shadow, projection charts mathematically reduce your multivariate data set to two dimensions, so it can be displayed on the screen.

Projection plots are valuable because they capture the distribution of your data in a single graphic. Since data sets are often large and complex, they can be difficult to fully assimilate and comprehend using multiple X-Y plots or line charts. As a result, projection charts let you see the “forest”, not just the “trees”. Projection charts help you interpret overall patterns in your data, including determining if your data is distributed uniformly, and whether there are clumps, outliers, or void areas. In addition, you can use projection charts to compare two or more data sets, to determine if they cover the same or different parts of the input space.

Projection plots use a technique called Principal Component Analysis (PCA) to project your data onto a planar surface.

Geometrically, this process is equivalent to finding a plane that minimizes the sum of squared distances between the plane and the data points, as shown below



Original data

(3 or more dimensions)

Planar projection of data

(2 dimensions)

Because of the mathematical transformation involved, the axes of a projection chart are not physical variables, but rather, weighted linear combinations of the original variables. The coordinates are normalized to indicate the number of standard deviations from the center of gravity of the data. For a brief mathematical explanation of PCA, see What is PCA?.

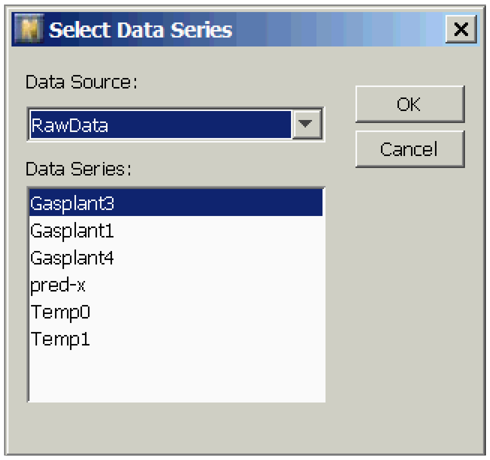
Using Projection Charts

To view a data set in a projection chart:

➔ Do one of the following:

Menu Bar:	Choose View > Projection Plot.
Toolbar:	Click the Open Projection Chart button.

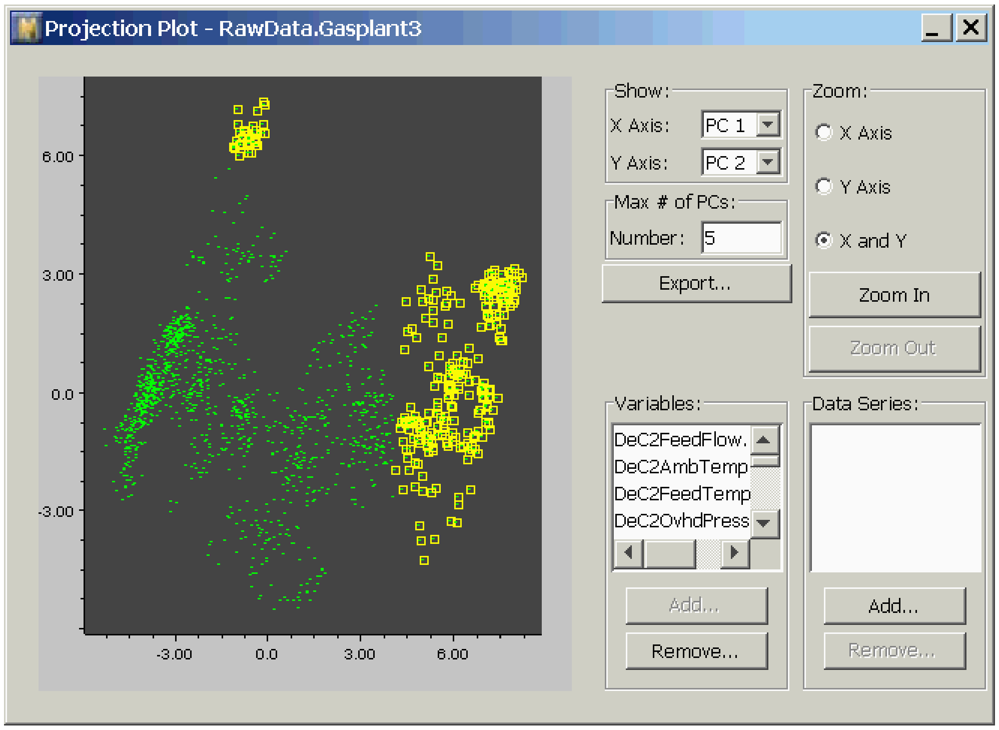
The Select Data Series dialog appears, for example:



Data Source: RawData

Data Series:
- Gasplant3
- Gasplant1
- Gasplant4
- pred-x
- Temp0
- Temp1

After you specify the data series you wish to plot in the projection chart, click OK to display a projection chart similar to the following:



Show:
X Axis: PC 1
Y Axis: PC 2

Max # of PCs:
Number: 5

Zoom: ( ) X Axis ( ) Y Axis [✓ X and Y]

Variables:
DeC2FeedFlow.
DeC2AmbTemp
DeC2FeedTemp
DeC2OvhdPress

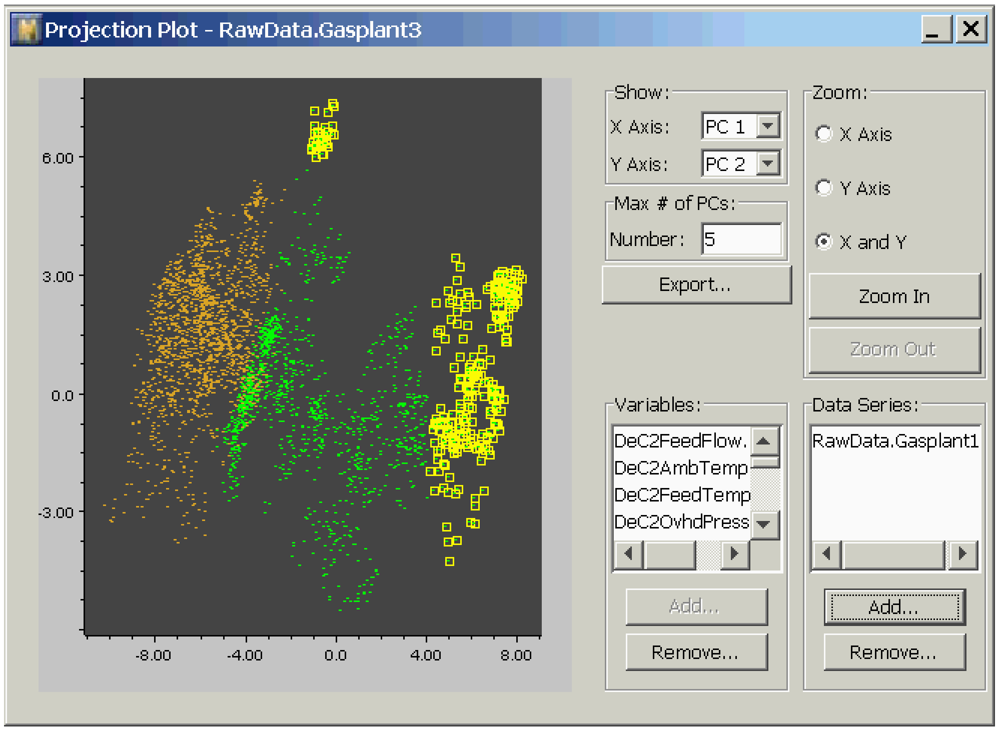
Data Series:

By default, all variables in the selected data series are included in the PCA analysis. If you want to exclude any variables from the PCA analysis, select the variables you want to exclude in the Variables scroll area to the right of the chart, and select Remove. When you do this, the PCA will be recalculated, excluding the variables you selected. To add variables you previously removed, use Add. This feature is useful if you want to see the distribution of the input variables, excluding the output variables, or any subset thereof.

By default, the first two principal components, PC1 and PC2, are displayed in the plot, since these variables contain most of the information on the variation in your data set. However, you can also view less significant principal components (PC3 - PC5), using the Show selection boxes in the upper right of the projection chart view. To view higher-order principal components, enter the preferred number in the Maximum # of PCs edit box.

To compare the distribution of two data sets, use the Add button in the lower right corner of the window, in the area labelled Data Series. The Select Data Series dialog appears, listing only the data series with the same variables as plotted on the current projection plot. If there are no compatible data series, you will not be allowed to add a data series to the current plot.

When you select a data series to add to the plot, the additional data series will be superimposed over the original plot, as illustrated below:



Show:
X Axis: PC 1
Y Axis: PC 2
Max # of PCs:
Number: 5
Zoom:
X Axis
Y Axis
[✓ X and Y]
Variables:
DeC2FeedFlow.
DeC2AmbTemp
DeC2FeedTemp
DeC2OvhdPress
Data Series:
RawData.Gasplant1

This plot shows you that the two data sets cover/do not cover the same area. Therefore, a model trained on one of these data sets might not predict the second data set very well. You can add as many data series as you like by repeating this process, or remove them using the Remove button.

You can export the parameters of the displayed PCA model to a text file by clicking the Export button, then load it into the G2 environment for online statistical monitoring of your process. For information on PCA deployment, see Model Deployment.

What is PCA?

In-depth descriptions of PCA are available in many statistics and chemometrics texts, so only a brief description is given here. This section is only relevant to users who want to understand the mathematical basis of projection plots.

The starting point is the data matrix, X, whose *m* columns represent variables, and whose *n* rows represent observations. Prior to analysis, X is normalized such that each column has a mean value of 0, and a standard deviation of 1. Normalization makes the PCA results independent of units.

In projection plots, PCA is used to reduce X ($n \times m$) to an *f*-dimensional matrix T ($n \times f$), where $f < n$. The scores matrix T is produced by multiplying X by a projection matrix P ($m \times f$), as follows:

$$T = XP$$

The P matrix is chosen so that T is an optimum projection, in the sense of minimizing the "lost information" that results from reducing the number of columns of X. Mathematically, the lost information can be quantified by a matrix E ($n \times m$), calculated as follows:

$$E = X - TP^t = X(I - PP^t)$$

PCA chooses the projection matrix P so the 2-norm (equivalent to the mean of the squares of the elements) of E is minimized. In addition, P is constrained such that the rows of P are orthogonal and unit length (this constraint ensures a unique solution to the minimization problem). It is known that the solution to this problem is that the *j*th column of P is equal to the *j*th eigenvector of X^tX (the covariance of X).

The columns of T are ordered according to their importance. The first column of T is the optimal solution for $f = 1$. The first two columns of T are the optimal solution for $f = 2$, and so on. When NOL Studio generates a projection plot, it calculates the top five principal components. By default, the top two principal components (the first two columns of T) are plotted. You can also plot higher principal components.

To superimpose an additional data set (X_2) on the same projection chart, the columns of the second data set are permuted, if necessary, to correspond to the order in the first data set. The data is then normalized using the same normalization constants as the original data. The projection is calculated by multiplication by the original projection matrix, as follows:

$$T_2 = X_2P$$

Viewing Data in a Histogram View

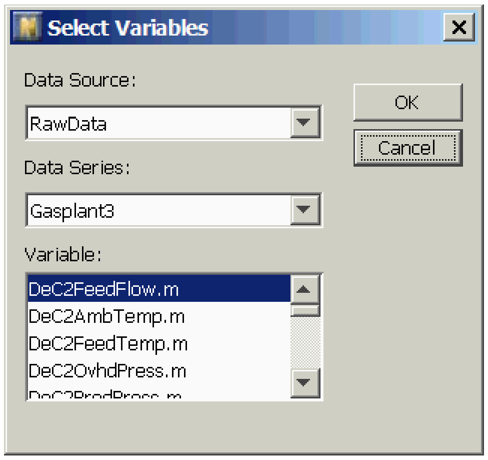
The histogram view is used to show the statistical distribution of individual variables.

To view a histogram of a specific variable:

➔ Do one of the following:

Menu Bar:	Choose View > Histogram.
Toolbar:	Click the Open Histogram button.

The Select Variables dialog appears, for example:

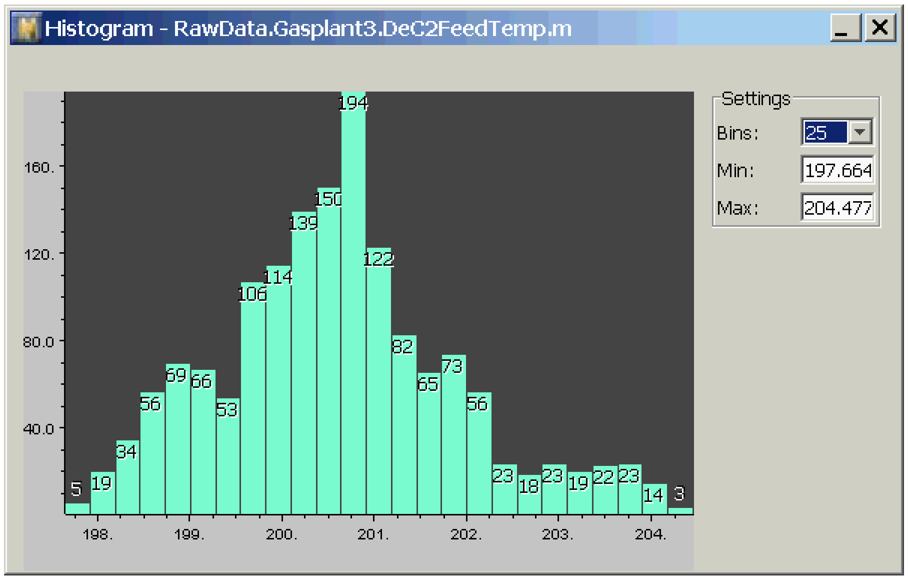


Data Source: RawData

Data Series: Gasplant3

Variable:
DeC2FeedFlow.m
DeC2AmbTemp.m
DeC2FeedTemp.m
DeC2OvhdPress.m

After specifying a data source, data series, and variable name, click OK to display the histogram with specified variable. A typical histogram view looks something like this:



Settings

Bins: 25

Min: 197.664

Max: 204.477

The number of bins that defines the histogram can be changed by increasing or decreasing the value in the Bins settings text field. The range over which the histogram is defined can be set by overriding the minimum and maximum values automatically placed in the text fields.

Labeling Data

Describes how to label data to identify the parts of the raw data you would like to use to train a model.

Introduction

At this point, you have loaded data into NOL Studio, and you have used various graphical views to examine the data. During this process, you may have noticed some flaws in your data: outliers, shutdown periods, operational transients, changeovers, and the like. It is necessary to cut out the bad or inapplicable portions of the data, to get a “clean” data set suitable for training.

Your data may also contain regions that relate to production of different products, grades of material, or modes of operation. You may want to model one or more of these products, grades, or modes. To do so, you must create separate data sets relating to single products, grades, or production modes.

To support the extraction of one or more training sets from the raw data, NOL Studio gives you the capability of classifying the raw data into categories you define, through a graphical labeling process.

You can define as many label categories as necessary. If you simply want to eliminate some bad data, you can create just one category, such as “bad”. If you want to create several training sets from a single raw data source, you will need several categories.

Graphically, data points you label are highlighted in a particular color in both the chart and spreadsheet views. NOL Studio allows you to label data in any view, excluding the histogram view. The views are dynamically linked, so when you label a point in one view, all views are simultaneously updated.

Note Labels can only be applied to raw data.

After you have labeled the raw data, you create a preprocessor that includes or excludes particular types of labels. For example, once you have found all the outliers and labeled them with an “outlier” label, a preprocessor can be made out of all the points that are NOT labeled as outliers with the following queries: “include ALL points”, and “exclude ANY point labeled 'Outlier'”. Some data points may carry more than one label. For example, a data point may simultaneously be labeled as an outlier, and by the product or grade of material being produced at that time. When you create a training data set from the labeled raw data, you can choose to include or exclude points with specific label combinations, via the joined label facility. For more information on the query facility, see the following chapter.

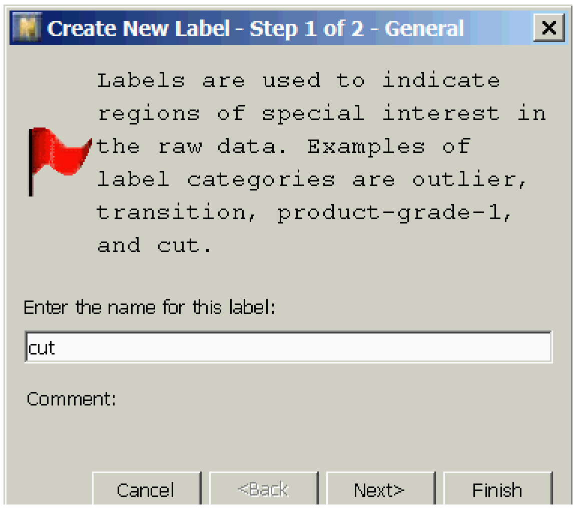
Defining Label Categories

To create labels:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Label.
Toolbar:	Click the New Label button.
Tree View:	Right-click the Labels Category and choose New from its menu.

This brings up the following dialog:



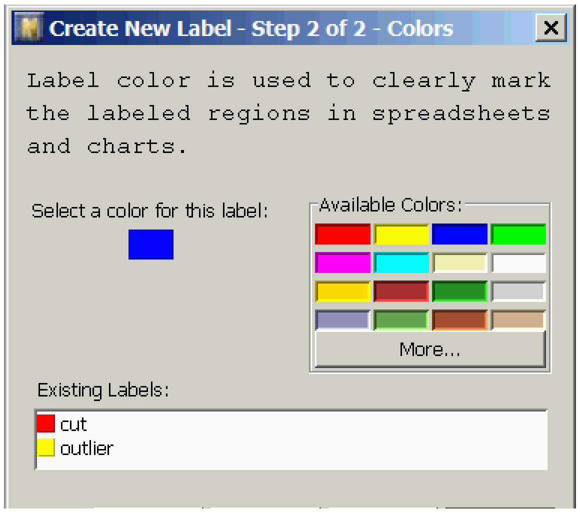
Labels are used to indicate regions of special interest in the raw data. Examples of label categories are outlier, transition, product-grade-1, and cut.

Enter the name for this label:

cut

Comment:

After you type in a name for the label and an optional comment, you can click Finish to complete the creation of a label with a default color. However, if you wish to specify a color for this label, click Next. The following dialog appears:



Label color is used to clearly mark the labeled regions in spreadsheets and charts.

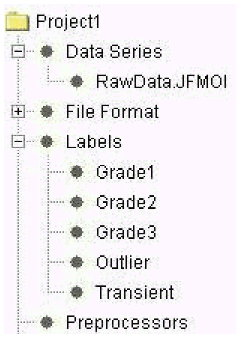
Select a color for this label: 🟦

Existing Labels:

🟥 cut
🟨 outlier

Specify a color and click Finish to dismiss the dialog and create the label. Follow the steps above to create as many labels as you wish.

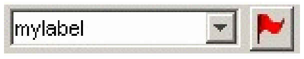
Once you have created a number of labels, you can open the Labels item in the tree view. For example:

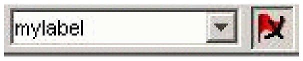


The name of a label or the color can be changed at any time using the label's property table. Access the property table by right-clicking on the desired label in the tree view.

Setting the Active Label

Once you have created labels, you can label data. At any time, only one label is active. The toolbar contains a labeling control, which specifies the active label and the action currently enabled, which is either Label or Unlabel. To select the labeling mode, click the Label or Unlabel toggle button. To select a different label, click on the pulldown menu and select the label you wish to be active. You will see the following in the upper-right toolbar:





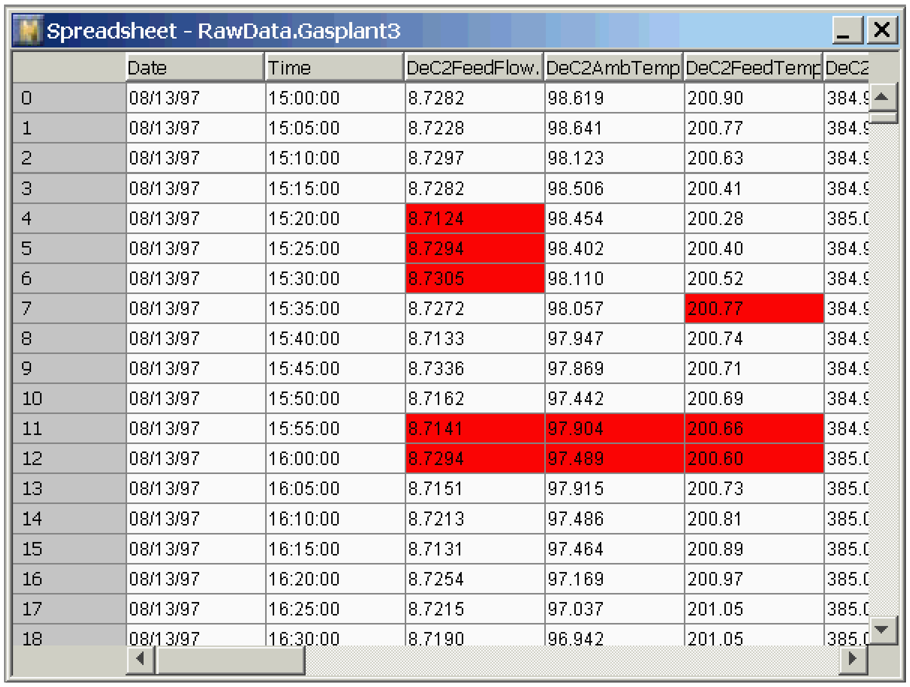
Labeling Data in the Spreadsheet View

The first step in labeling from the spreadsheet is to select the active label in the toolbar, and make sure you are in labeling mode by selecting Label, as explained above.

The following gestures label data in the spreadsheet view:

- Clicking with the mouse on individual cells labels individual points.
- Dragging over multiple cells labels a region.
- Clicking on the first cell in a range and then shift-clicking on the final cell results in the selection of the rectangular region of cells with the first and last cells as the corners.
- Entire rows can be selected by clicking on the row number.

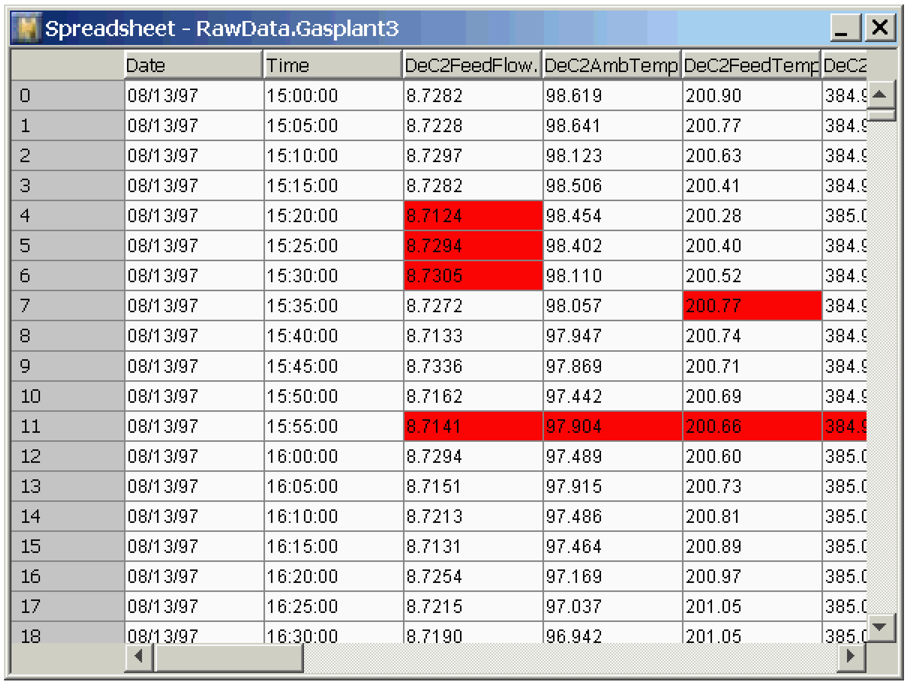
Unlabeling is done in the same way, with selections applying only to those cells labeled with the currently chosen label. To label individuals cells, click on the cell that you wish to label while in the Label mode. After labeling several cells, your spreadsheet view will look something like this:



Spreadsheet - RawData.Gasplant3

	Date	Time	DeC2FeedFlow.	DeC2AmbTemp	DeC2FeedTemp	DeC2
0	08/13/97	15:00:00	8.7282	98.619	200.90	384.9
1	08/13/97	15:05:00	8.7228	98.641	200.77	384.9
2	08/13/97	15:10:00	8.7297	98.123	200.63	384.9
3	08/13/97	15:15:00	8.7282	98.506	200.41	384.9
4	08/13/97	15:20:00	🟥 8.7124	98.454	200.28	385.0
5	08/13/97	15:25:00	🟥 8.7294	98.402	200.40	384.9
6	08/13/97	15:30:00	🟥 8.7305	98.110	200.52	384.9
7	08/13/97	15:35:00	8.7272	98.057	🟥 200.77	384.9
8	08/13/97	15:40:00	8.7133	97.947	200.74	384.9
9	08/13/97	15:45:00	8.7336	97.869	200.71	384.9
10	08/13/97	15:50:00	8.7162	97.442	200.69	384.9
11	08/13/97	15:55:00	🟥 8.7141	🟥 97.904	🟥 200.66	384.9
12	08/13/97	16:00:00	🟥 8.7294	🟥 97.489	🟥 200.60	385.0
13	08/13/97	16:05:00	8.7151	97.915	200.73	385.0
14	08/13/97	16:10:00	8.7213	97.486	200.81	385.0
15	08/13/97	16:15:00	8.7131	97.464	200.89	385.0
16	08/13/97	16:20:00	8.7254	97.169	200.97	385.0
17	08/13/97	16:25:00	8.7215	97.037	201.05	385.0
18	08/13/97	16:30:00	8.7190	96.942	201.05	385.0

Labeling an entire row by clicking a row number will result in the following:


Spreadsheet - RawData.Gasplant3

	Date	Time	DeC2FeedFlow.	DeC2AmbTemp	DeC2FeedTemp	DeC2
0	08/13/97	15:00:00	8.7282	98.619	200.90	384.
1	08/13/97	15:05:00	8.7228	98.641	200.77	384.
2	08/13/97	15:10:00	8.7297	98.123	200.63	384.
3	08/13/97	15:15:00	8.7282	98.506	200.41	384.
4	08/13/97	15:20:00	8.7124	98.454	200.28	385.
5	08/13/97	15:25:00	8.7294	98.402	200.40	384.
6	08/13/97	15:30:00	8.7305	98.110	200.52	384.
7	08/13/97	15:35:00	8.7272	98.057	200.77	384.
8	08/13/97	15:40:00	8.7133	97.947	200.74	384.
9	08/13/97	15:45:00	8.7336	97.869	200.71	384.
10	08/13/97	15:50:00	8.7162	97.442	200.69	384.
11	08/13/97	15:55:00	8.7141	97.904	200.66	384.
12	08/13/97	16:00:00	8.7294	97.489	200.60	385.
13	08/13/97	16:05:00	8.7151	97.915	200.73	385.
14	08/13/97	16:10:00	8.7213	97.486	200.81	385.
15	08/13/97	16:15:00	8.7131	97.464	200.89	385.
16	08/13/97	16:20:00	8.7254	97.169	200.97	385.
17	08/13/97	16:25:00	8.7215	97.037	201.05	385.
18	08/13/97	16:30:00	8.7190	96.942	201.05	385.

If you have accidently labeled a data element or series, you can undo labeling by choosing Edit > Undo Labeling from the menu bar. This unlabels the last label action only.

Labeling Data in the Line Chart View

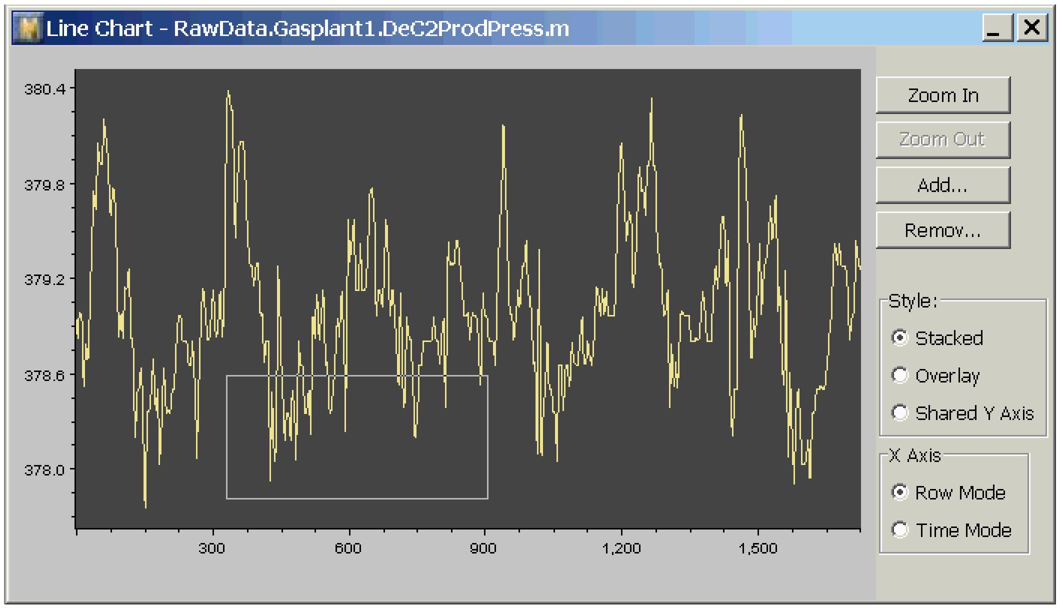
The following gestures can be used to label points in a line chart:

- Clicking on a single point
- Dragging over an area of the chart
- Dragging over a region of the x or y axis

Note Before labeling, you must first set the active label using the selection box on the tool bar, and you must be in labeling mode.

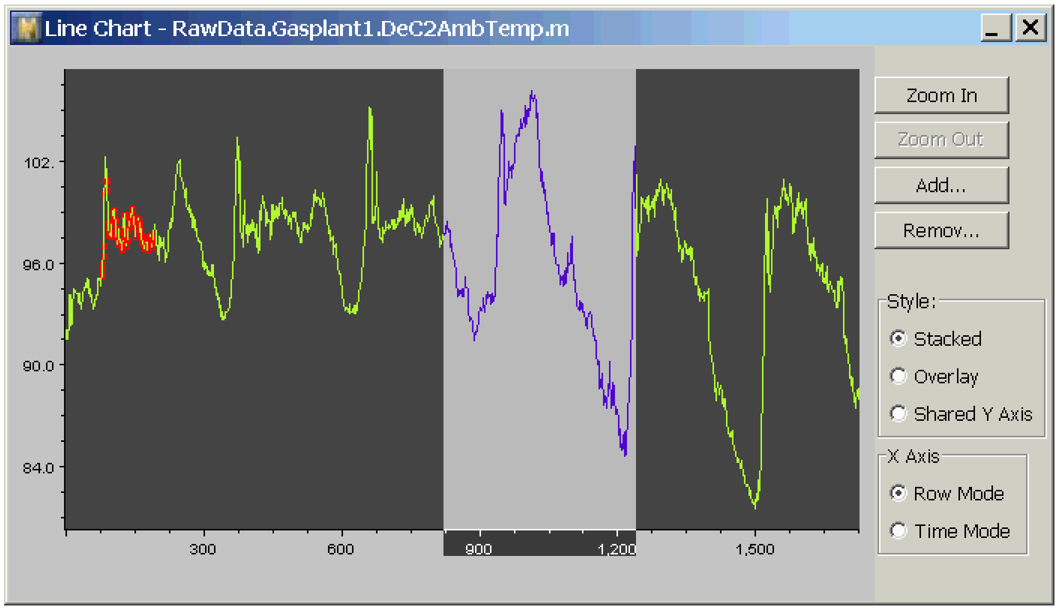
To label an individual point, click on the point.

To label a group of points, drag a bounding box over a region. Releasing the mouse applies the labeling action to all points within the box. The selection box can extend over more than one variable. The action would look something like this:



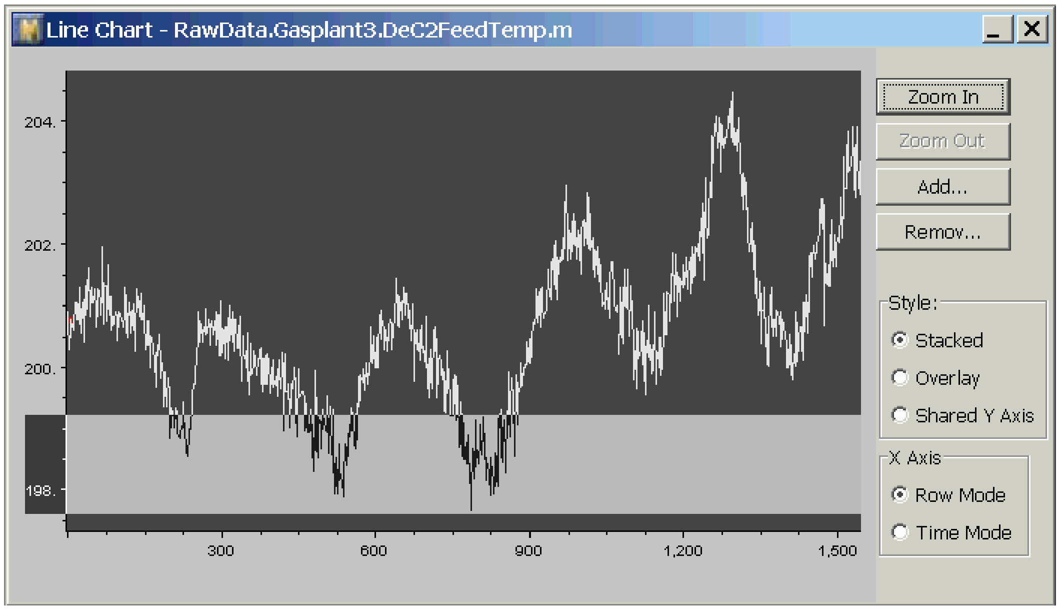
Another useful gesture is to drag along the x-axis, below or beside the tick marks. If you drag along the x-axis, a selection region will extend the entire height of the plot, allowing you to select all displayed points that fall in a certain time or row range.

This gesture works in all plot modes (single, overlayed, and stacked). An example of dragging on the X-axis to label a specific series of values (either by row or by time) might be the following:



Note Selections on the x-axis apply to all variables in the data series, regardless of what particular variables are shown in the chart. This gives you the ability to label all variables in a data series with a single gesture.

Similarly, you can drag on the Y-axis to label all values within a range of Y values. Even if you are zoomed in to show only a portion of the x axis, the label applies across all time/rows. Here is an example of that gesture:



If you have accidently labeled a data element or series, you can undo labeling by choosing Edit > Undo Labeling from the menu bar. This unlabels the last label action only.

Labeling Data in the Scatter Chart Views

Projection Chart View

In the projection chart view, the labeling gestures are the same as in the line chart view. Any labeling actions are applied to all the variables used to calculate the principal components, found in the list of variables on the chart. If all variables are included in the PCA calculation, selecting a point in the projection chart view is the same as selecting a row of the spreadsheet. A label is only shown on the projection chart if all the variables involved in the PCA are selected, for that point.

X-Y Scatter Chart View

In the X-Y scatter chart view the labeling gestures are the same as in the line chart view. Any labeling actions are applied only to the variables from the Raw data source.

Labeling Data

In some cases, you might need to label the data precisely according to the range of data values or sample indexes. You can use the custom labeling dialog to label the data, based on the range you defined through the dialog.

To label data:

1 Choose Edit > Labeling.

2 Select the raw data series from the DataSeries combo box.

3 Select the Variable you want to label.

You can only label one variable at a time.

4 Select the type of label you want to use for labeling the data.

5 Click the range definition button to define the type of range to use.

6 Configure the Y value range or X sample index value.

7 Click the Label button.

Creating a Preprocessor

Describes how to create a preprocessor that conditions the raw data used to build models.

Introduction

In the previous chapter, you learned how labels are applied to data in the graphical views, designating different data categories. To use these labels, and to extract data sets suitable for training models, you must create a data preprocessor.

In creating the preprocessor, you need to consider several preprocessing steps. These steps include:

- Extracting a subset of data from raw data through specific label queries
- Creating new preprocessors for the extracted data
- Reorganizing and conditioning the data by using a list of formulas

This chapter describes how to:

- Create a new preprocessor
- Construct a list of formulas for preprocessing your data

Creating a New Preprocessor

A preprocessor is a tool that processes a subset of the raw data used to build models. You define any model in relation to its preprocessor, i.e. a model derives from one and only one preprocessor. Preprocessors are constructed from one or several data series. You create a preprocessor, using the Create New Preprocessor wizard.

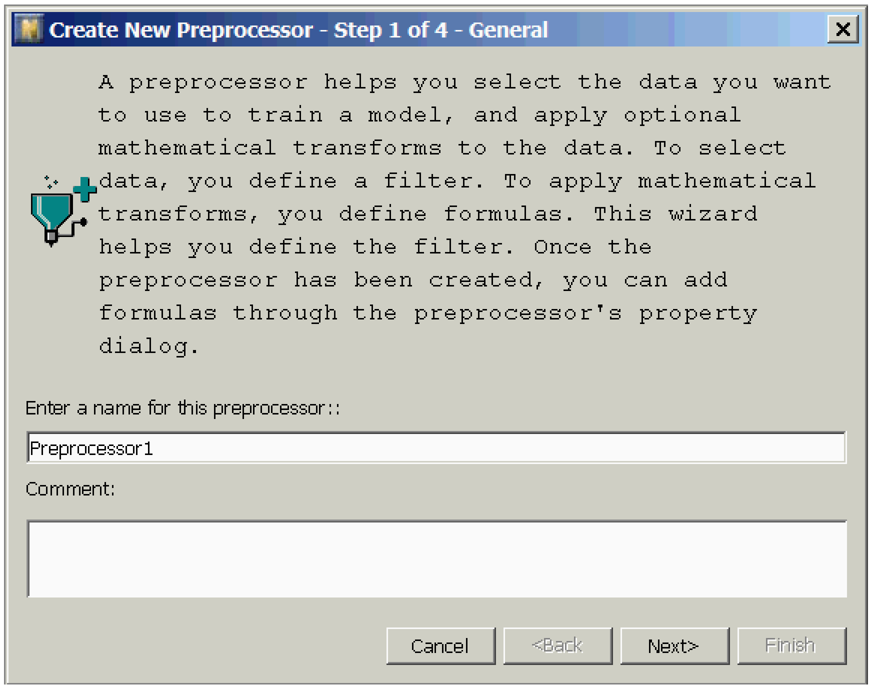
Using the Create New Preprocessor Wizard

To open the Create New Preprocessor wizard:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Preprocessor.
Toolbar:	Click the New Preprocessor button.
Tree View:	Right-click the Preprocessor category and choose New from its menu.

The Create New Preprocessor wizard appears, for example:



A preprocessor helps you select the data you want to use to train a model, and apply optional mathematical transforms to the data. To select data, you define a filter. To apply mathematical transforms, you define formulas. This wizard helps you define the filter. Once the preprocessor has been created, you can add formulas through the preprocessor's property dialog.

Enter a name for this preprocessor::

Preprocessor1

Comment:

The Create New Preprocessor wizard guides you through the necessary steps, as follows.

To create a new preprocessor:

1 Enter a new name for the preprocessor.

The default name for a new preprocessor takes the form:

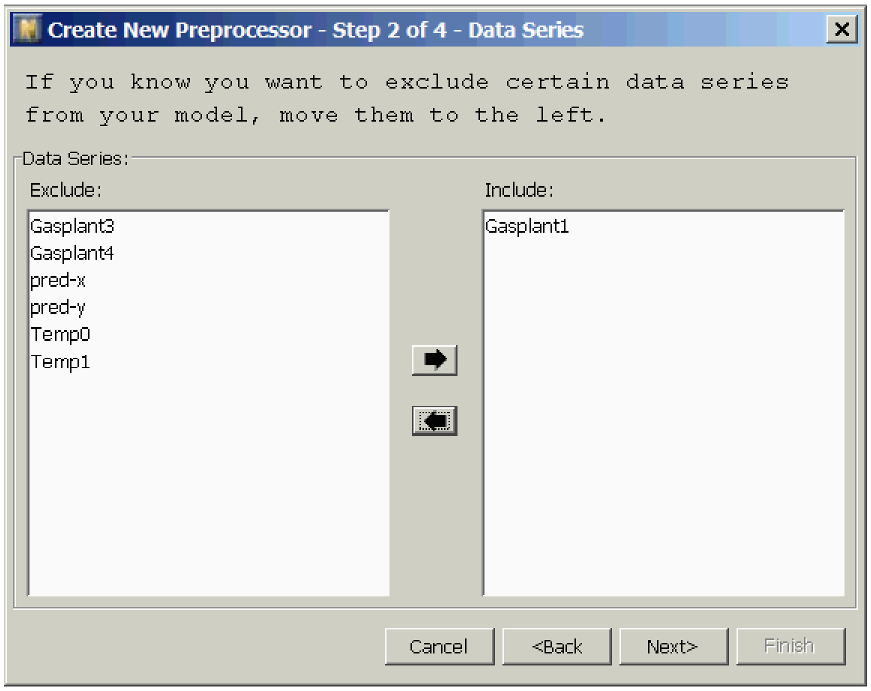
preprocess#

where # specifies the index of default preprocess or names.

2 Enter any comment or information associated with this preprocessor that you may wish to remember in the future.

3 Click Next to advance to the next screen, to select the data series to use. You can select any subset of the data series of the raw data for preprocessing.

You select the data series by moving selections to the right or left, using the arrow buttons. Your ultimate goal is to select variables from within these data series for building a model. For example:


Create New Preprocessor - Step 2 of 4 - Data Series

If you know you want to exclude certain data series from your model, move them to the left.

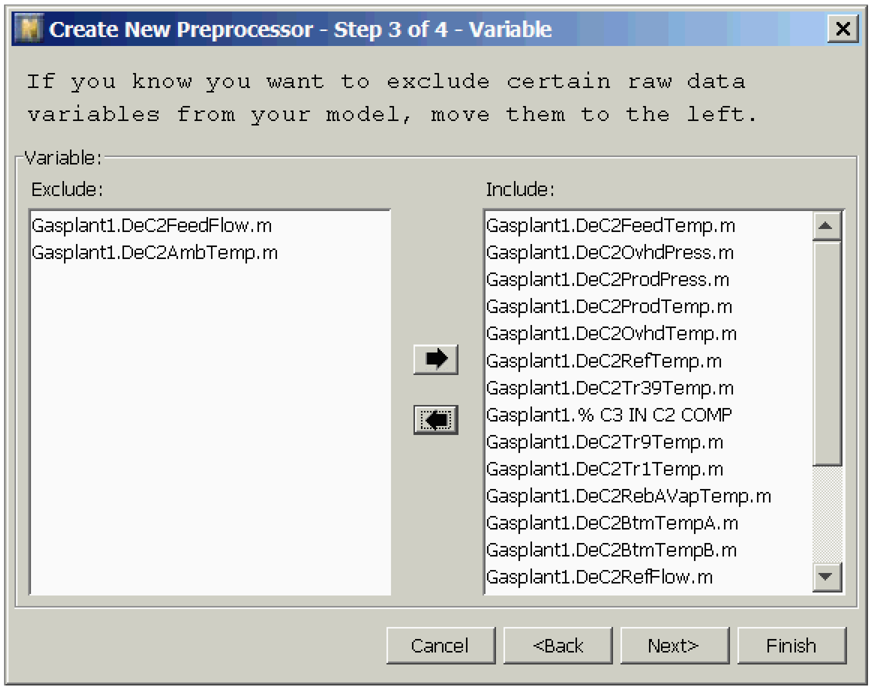
Data Series:

Exclude:
Gasplant3
Gasplant4
pred-x
pred-y
Temp0
Temp1

Include:
Gasplant1

4 Click Next to advance to the next screen, to select the variables to use. You can include any subset of the variables of a data series for preprocessing.

For example:


Create New Preprocessor - Step 3 of 4 - Variable

```
If you know you want to exclude certain raw data
variables from your model, move them to the left.
```

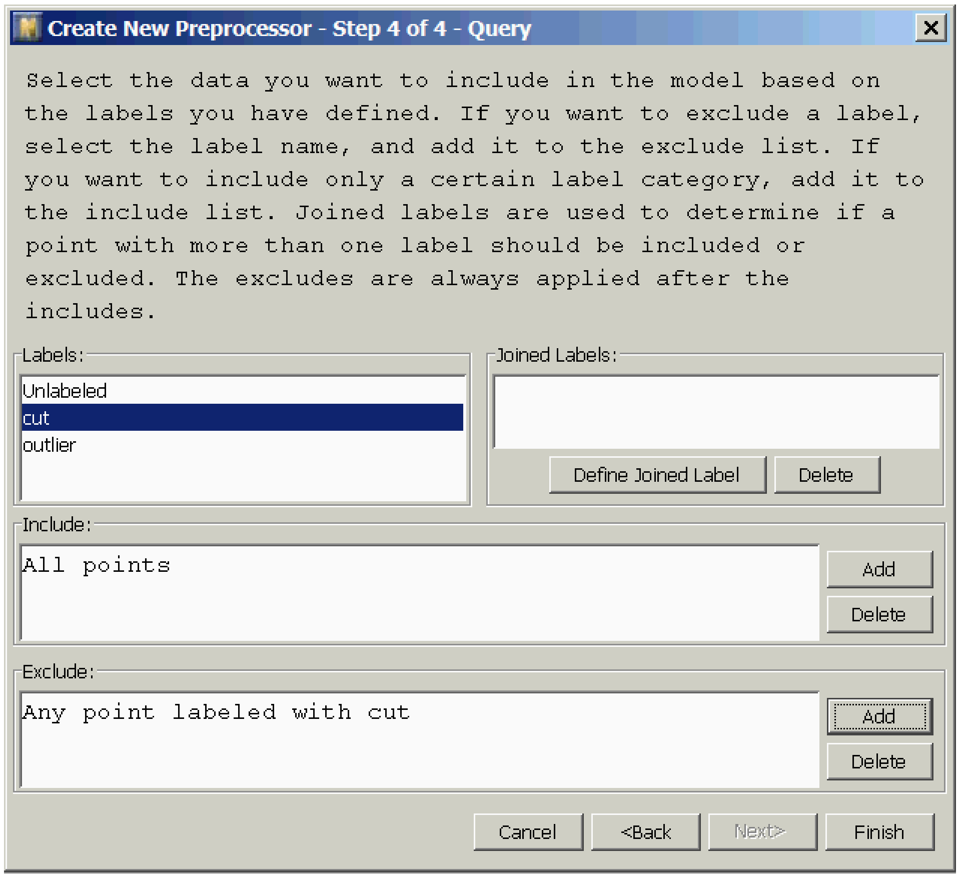
Variable:

Exclude:	Include:
Gasplant1.DeC2FeedFlow.m	Gasplant1.DeC2FeedTemp.m
Gasplant1.DeC2AmbTemp.m	Gasplant1.DeC2OvhdPress.m
	Gasplant1.DeC2ProdPress.m
	Gasplant1.DeC2ProdTemp.m
	Gasplant1.DeC2OvhdTemp.m
	Gasplant1.DeC2RefTemp.m
	Gasplant1.DeC2Tr39Temp.m
	Gasplant1.% C3 IN C2 COMP
	Gasplant1.DeC2Tr9Temp.m
	Gasplant1.DeC2Tr1Temp.m
	Gasplant1.DeC2RebAVapTemp.m
	Gasplant1.DeC2BtmTempA.m
	Gasplant1.DeC2BtmTempB.m
	Gasplant1.DeC2RefFlow.m

5 Click Next to advance to the screen which lets you define queries on labels.

Once the name and the variables are chosen, you must decide which category of the data to include in the preprocessor. This is done by filtering with queries. Queries define searches through the data for points that either are unlabeled or are assigned to a particular label. Those points matching a query will be included in (or excluded from) the preprocessor.

Here is the page for defining the query in the preprocessor wizard:



Create New Preprocessor - Step 4 of 4 - Query

```
Select the data you want to include in the model based on
the labels you have defined. If you want to exclude a label,
select the label name, and add it to the exclude list. If
you want to include only a certain label category, add it to
the include list. Joined labels are used to determine if a
point with more than one label should be included or
excluded. The excludes are always applied after the
includes.
```

Labels: Unlabeled, cut, outlier

Joined Labels:

Include: All points

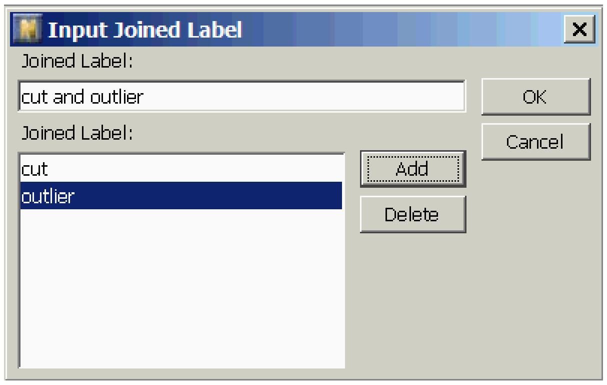
Exclude: Any point labeled with cut

Queries are constructed by defining labels to include and labels to exclude. By default, all data is included, and none is excluded. If you want to exclude a single label category, you select the label, and add it to the excluded labels by selecting Add, in the Exclude section. To exclude another label, select that label, and add it to the excluded labels. If you add a label to the Include section, the default (include all points) will be automatically removed.

Note The Includes are always applied before the Excludes. Thus, if you include label A and exclude label B, the points labeled B are removed from the set of points labeled A.

Joined labels allow you to create detailed queries that specify the handling of data points with multiple labels. Joined labels are defined in a separate dialog. Once you define a joined label, it is treated as a new, compound label. Therefore, you can include or exclude the points with your joined label.

Here is a example of defining a joined label “cut and transient:”



cut and outlier

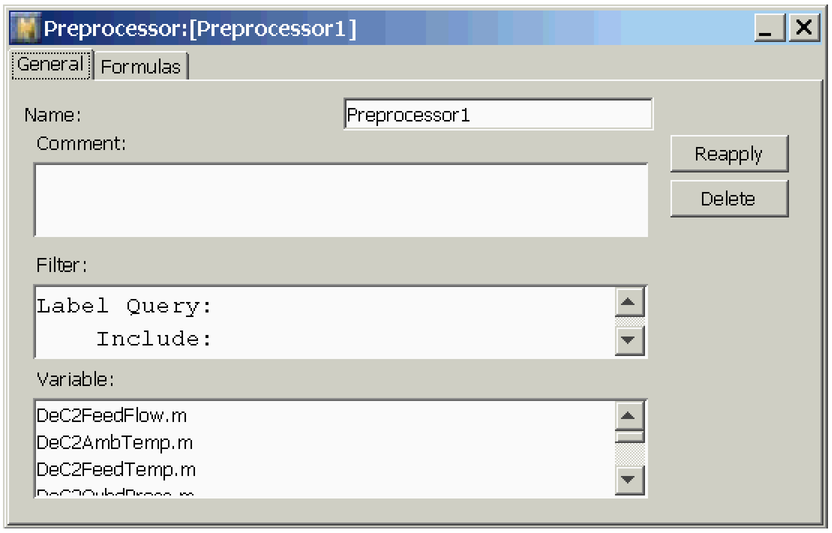
cut
outlier

6 Click Finish in the wizard to create a new preprocessor.

You can view and change the information of this preprocessor through its property workspace.

Working With an Existing Preprocessor

Once a preprocessor has been defined, you work with the preprocessor using the preprocessor property workspace, accessed through the Object menu or tree view. Here is an example of a preprocessor property workspace:



Name: Preprocessor1

Comment:

Filter:

```
Label Query:
    Include:
```

Variable:

DeC2FeedFlow.m
DeC2AmbTemp.m
DeC2FeedTemp.m

You can change the name and comment of this preprocessor by typing in the corresponding display field.

Caution Press Return when you finish typing the name. Otherwise, the name won't changed.

Accessing the Formula List

To view or edit the formulas associated with the preprocessor, use the Formula tab on the preprocessor property workspace. For details on how to use preprocessor formulas, see Using Formulas to Preprocess Data.

Reapplying the Preprocessor

If you change the labels on the raw data after creating the preprocessor, these changes are not automatically propagated to the preprocessed data series. To capture the effect of changing labels in an existing preprocessor, select the Reapply Preprocessor button on the preprocessor properties workspace. This will apply the filter to the current set of labels, and recalculate all formulas.

Deleting the Preprocessor

To delete a preprocessor, use the Delete button on the preprocessor property workspace. If there is a model that gets its data from the preprocessor, you must delete the model before you will be allowed to delete the preprocessor.

Using Formulas to Preprocess Data

NeurOn-Line Studio provides the capability to define formulas to further condition your data. Formulas are functions that operate on variables. NeurOn-Line Studio provides many built-in formulas—from simple mathematical functions, such as absolute value and logarithm, to complex model-based formulas, such as neural network models. You can use formulas to do sensor validation, to replace missing data, and to smooth noisy signals. Formulas are also useful for calculating derived variables by algebraically combining measured quantities. An example of a derived variable is heat flux, calculated as the product of a heat transfer coefficient and a temperature driving force. The goal of defining derived variables is to find new variables that are closely correlated with the target output variable you wish to predict.

For example, suppose you want to predict the taste of a batch of lemonade, made from three ingredients: water, lemons, and sugar. You know the taste depends only on the ratio of lemons to water, and sugar to lemons, not the absolute amounts of the ingredients, which constitute the raw measurements. A model based on the ratios of ingredients will apply regardless of the size of the batch of lemonade, and would also require less data, since there will be fewer parameters in a two-input versus a three-input model. Therefore, you use the formula facility to create the required ratios, and base your model on the derived ratio variables.

When you apply a formula, the results are written into either existing or new (derived) variables. Formula operations are carried out on a copy of the raw data, so the raw data is never changed. This allows you to “undo” the action of any formula, by removing it from the formula list.

The formulas you create are exported as an integral part of the deployment model, and can be automatically applied in real time through a G2 application or through an external application, using an ActiveX control.

Showing Variables Before and After Formulas

Charting facilities in NOL Studio have the capability to show, on a single plot, variables that derive from different data sources. Using this capability, you can display plots showing raw data variables and the corresponding preprocessed variables, to assess the effectiveness of the preprocessor. For more details on how to plot variables from different data sources in various graphical views, see the chapter on Visualizing Data.

You can also view derived variables, created through formulas, which are not in the raw data set. When you define a formula that creates a new derived variable, the variable becomes a first-class member of the preprocessor’s data series. You can view a property table for the derived variable, examine its statistics, see a histogram, plot it, etc.

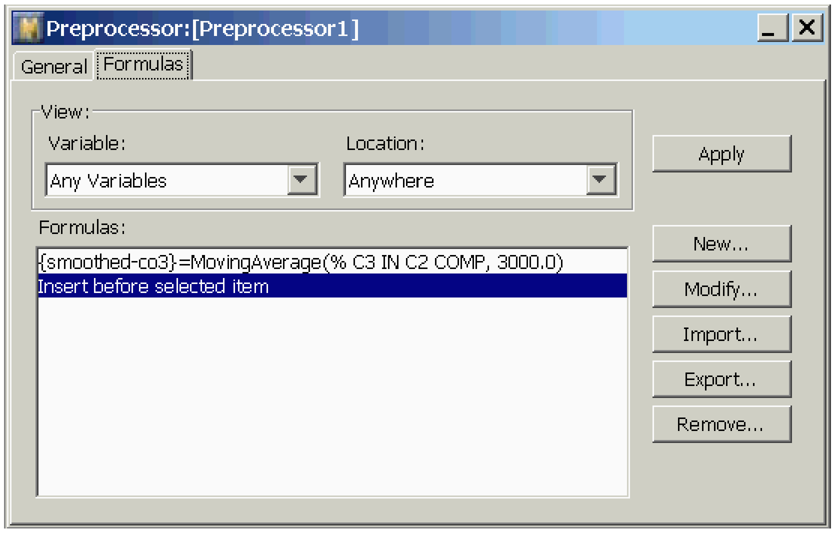
Time Merging Data Series

Sometimes the data series in your preprocessor are sampled on different frequencies, or the data in a single data series might not be equally spaced in time. For example, laboratory data might be available approximately each 30 minutes, while on-line measurements are available every minute. In the application of formulas, and during training, NOL Studio automatically accounts for different or irregular frequencies, through a process of smoothing and interpolation. It is not necessary to explicitly merge these data series using a special time merge formula.

For more details on the internal handling of irregularly sampled data and data sampled on different frequencies, see Preparing the Training Set.

The Formula List

The current set of formulas for a preprocessor can be viewed by selecting the Formula tab on the preprocessor workspace. This action shows the Formula List:



Variable: Any Variables
Location: Anywhere

{smoothed-co3}=MovingAverage(% C3 IN C2 COMP, 3000.0)
Insert before selected item

This window lists all the formulas you have defined for this preprocessor. The formulas are applied in sequential top-down order to the data selected by the preprocessor's query. The order of formulas can affect the numerical results. For example, cutting high values and taking the moving average of the result is not the same as taking the moving average and then cutting high values.

Changing the Order of the Formula List

You can change the order of fomulas in the formula list by dragging a formula to the desired position in the list. All moves are validated before they are allowed. A legal move is one in which all the variables used in the input arguments have already been defined. Therefore, a formula that defines a new (derived) variable must appear higher in the list than other formulas that use the variable.

Selectively Viewing the Formula List

After you have defined a large number of formulas, it may become difficult to find the formulas referring to a specific variable. NeurOn-Line Studio provides a filtering mechanism that allows you to view only formulas containing a specific variable. Do this by selecting a variable and a location in the drop-down boxes at the top of the Formula List window. Here you specify:

- Variable — to display only those formulas referring to a particular variable
- Location — to select the formulas where the target variable appears in either the input or output of the formulas. "Anywhere" denotes both input and output.

Applying Formulas

When you create, modify, or import a formula, the formula is not immediately applied to the preprocessed data. This enables you to enter several formulas, without waiting for the numerical calculation to complete after each entry. Normally, the formulas are not applied until you close the Formula List dialog. However, to force the calculation to occur at any time, you can apply the formulas by clicking Apply in the Formula List dialog.

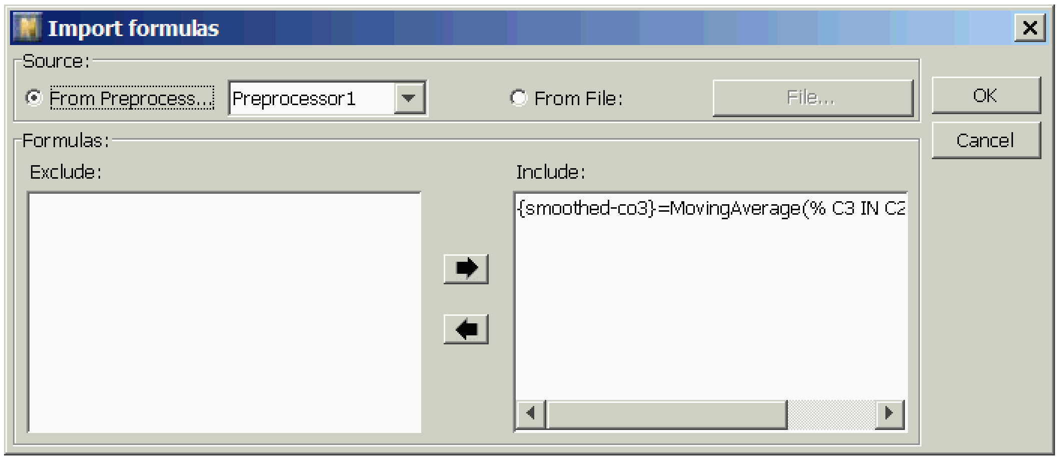
Exporting Formulas

You can save a list of formulas to a file using the Export button on the Formula List dialog. Together with the Import command, you can copy a set of formulas from one project to another, independent of any other objects. The formulas are saved in a file with the suffix .tfm.

Importing Formulas

The Import button allows you to load formulas from a .tfm file, or copy formulas from another preprocessor. This is an extremely useful feature when you want to apply the same set of formulas to two different sets of data. For example, you might want to divide a set of data into two parts, for training and validation. To divide the data, you will need two preprocessors, with different queries. But both preprocessors should use the same set of preprocessor formulas. Instead of entering the formulas twice, you can use the formula import facility to copy the formulas you need from one preprocessor to the other.

When you select Import, you will see the following dialog:



Preprocessor1

{smoothed-co3}=MovingAverage(% C3 IN C2

If you want to copy formulas from an existing preprocessor, select From Preprocessor, and specify a preprocessor in the list box. If you want to import formulas from a file, select From File, and then using the File button and resulting dialog to select a file containing formulas. In either case, when you have selected a formula source, you will see a list of formulas on the right-hand side of the Import Formula dialog.

If you want to import all formulas in the list, select OK. If you want to exclude any formulas, select the formulas you want to exclude, and move them using the arrow button to the left side of this dialog. Then, select OK.

The formulas you import are validated before they are added to the list of formulas for the preprocessor. If they refer to undefined variables, the import action will not be permitted.

Inserting a New Formula

You can add a new formula by clicking New in the Formula List window. The formula is inserted above the currently-selected formula. To add a formula to the end of the formula list, select the entry labeled "Insert before selected item". A Define Formula dialog will appear for you to define a new formula. For information on using the Define Formula dialog, see Editing a Formula.

Modifying an Existing Formula

To modify an existing formula, select the formula and click Modify. This launches the Define Formula dialog, where you can modify the formula. For information on using the Define Formula dialog, see Editing a Formula.

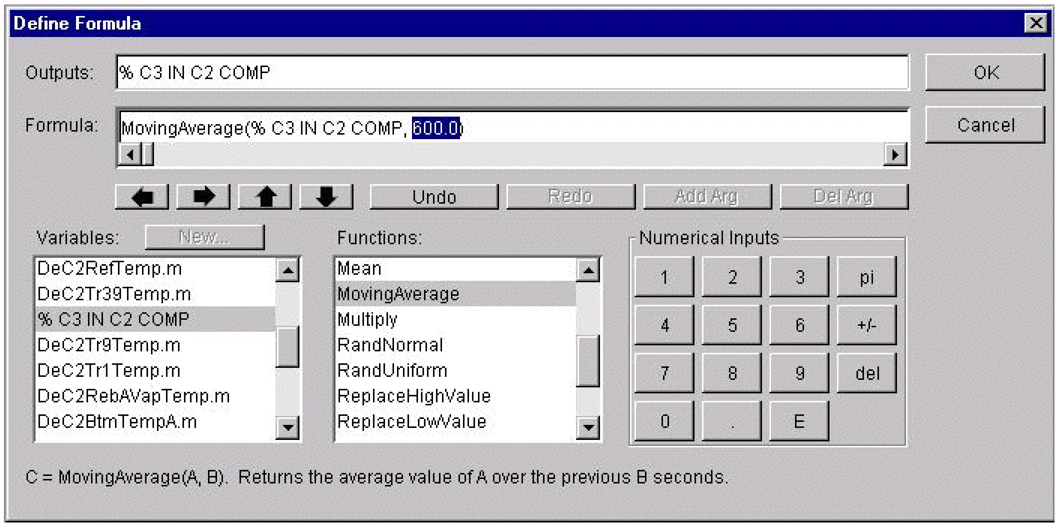
Removing a Formula

To delete a formula, select the formula you want to remove and choose Remove. Before the formula is deleted, the reduced list of formula is verified. If removing the formula results in an illegal formula list, the action is not allowed. An example of an illegal remove action is to remove a formula that defines derived variable used in subsequent formulas.

Editing a Formula

The Define Formula Dialog

To create or modify a formula, you use the Define Formula dialog, accessed from the Formula List dialog. Here is a example of a formula defined with the Define Formula dialog.


Define Formula

Outputs: % C3 IN C2 COMP

Formula: MovingAverage(% C3 IN C2 COMP, 600.0)

Variables: DeC2RefTemp.m, DeC2Tr39Temp.m, % C3 IN C2 COMP, DeC2Tr9Temp.m, DeC2Tr1Temp.m, DeC2RebAVapTemp.m, DeC2BtmTempA.m

Functions: Mean, MovingAverage, Multiply, RandNormal, RandUniform, ReplaceHighValue, ReplaceLowValue

C = MovingAverage(A, B). Returns the average value of A over the previous B seconds.

Here is a description of the various areas of this dialog:

Outputs	this is where you specify the outputs of the formula, the variable or variables whose values are calculated by this formula.
Formula	this is where you specify a function name and input arguments, using the syntax described in Formula Syntax.
Variables	The Entries in this list are the variables available to use in this formula. When you click on an entry in this list, the variable is added to the formula at the currently-selected location in the formula.
New Variable Button	Use this button to define a new derived variable name. New variables can only be defined in the output of a formula.
Functions	These are the functions available to use in this formula. To add a function to the formula, select where you want the function to be substituted, and click on a function.

Numerical Inputs	Use these buttons to enter numerical values as function arguments. These buttons are equivalent to number keys on your keyboard.
Left/Right Arrows ← →	These navigation buttons move the focus from argument to the next argument, at the same level in the formula. The left and right arrow keys on the keyboard perform the same function. For more information on these buttons, see Navigating a Formula.
Up/Down Arrows ↑ ↓	These navigation buttons move the focus up and down levels of nesting. The up and down arrows on the keyboard perform the same function. For more information on these buttons, see Navigating a Formula.
Undo/Redo Buttons	These buttons undo or redo previous actions.
Add Arg Button **Del Arg** Button	Use these buttons to add or remove an input argument. For more information on adding and removing arguments, see Adding and Removing Arguments.
Help Line	This line of text provides information on the function you have selected.

Several keys on your keyboard are also active when editing formulas:

Arrow Keys	These keys perform the same actions as the Left/Right and Up/Down Navigation Arrow buttons.
Backspace key Delete key	Use these keys to delete the highlighted entry in the formula.
Number keys	Use these keys to enter numerical values. The +, -, and e (for scientific notation) keys are also recognized.

Note You cannot use your keyboard to type variable or function names. To enter variables or function names, select the variable or function names from the lists shown on the dialog.

Formula Syntax

To define a formula, you specify the output variables, a function, and input arguments. Functions use prefix notation, so, if you want to multiply two variables, you express this as:

```
output = Multiply(input1, input2)
```

where output, and input1 and input2 are names of variables.

Functions can be nested, so an input argument can be another function with its own input arguments. An example of a nested function is:

```
output = Divide(Multiply(input1, input2), 2.0)
```

Navigating a Formula

When you edit a formula, part of it is highlighted in blue. This is called the *current focus*. The current focus indicates the part of the formula you are currently editing.

To edit a particular argument, you must put that argument into focus. There are two ways to move focus to a particular argument in a formula:

1 Click on the argument you want to edit.

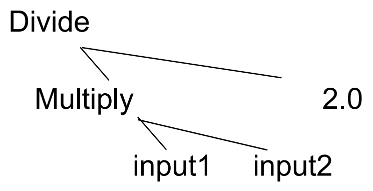
2 Use the navigation arrows to move the focus to the argument you want to edit.

To move the focus from the Output field to the Function field or visa-versa, you must use the mouse and click on the field you wish to edit. Once you are on the field you want to edit, you can shift the focus to other arguments by clicking on that arguments, or by using the arrow keys to move around the formula.

To understand how to navigate a formula using the arrow keys, consider the formula

```
output = Divide(Multiply(input1, input2), 2.0)
```

The structure of this formula can be diagrammed as follows:



If the current focus is on input1, and you click the up arrow button, the focus shifts to Multiply. Clicking the right arrow shifts the focus to 2.0. Clicking the up arrow button again shifts the focus to Divide. Conversely, if the focus is on Divide, and you click the down arrow, the focus shifts to Multiply. Clicking the down arrow again shifts the focus to input1. Clicking the right arrow then shifts the focus to input2.

Entering Variables and Functions

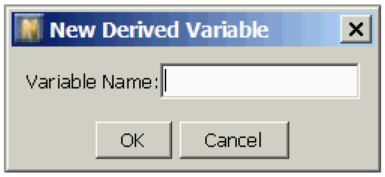
To enter a variable or function, move the focus to the desired position in the formula's outputs or function. If you have not yet specified a variable or function at the target location, three question marks (???) are shown as a placeholder. For example, when you first enter the function Multiply, the arguments will appear as:

Multiply(???, ???)

You must replace each ??? with a variable or function, before you click OK.

Once the focus is in the desired position, select a function of variable from one of the two lists in the lower left of the dialog. Depending on the location of the current focus, the list of variables or functions may be disabled, appearing in light gray. For example, if you are focused on the formula outputs, the function list will be disabled, because functions cannot appear in the outputs of a formula.

When you are focused on the formula's outputs, you can create a derived variable, using the New Variable button. The New Derived Variable dialog appears, prompting you to enter the new variable name. For example:



Once you have defined a derived variable, and entered the formula that defines it, you can use the derived variable in subsequent formulas, like any other variable. However, you must observe the rule that the formula defining the derived variable must appear in the formula list before any formulas that use the derived variable as an input.

Adding and Removing Arguments

Certain functions, such as Multiply and Sum allow a variable number of arguments. The Add Arg and Remove Arg buttons may be enabled when the focus is on the arguments of a variable-argument function. For example, if you have entered:

Multiply(???, ???)

and the focus is on either argument, you will be able to add a third argument by selecting Add Arg. The result is:

Multiply(???, ???, ???)

To return to two arguments, use the Del Arg button.

Creating a Predictive Model

Describes how you create a predictive model, using data you have prepared using a preprocessor.

Introduction

In previous chapters, you have seen how to import data, select the data you want to use for training through labeling and filtering, and create formulas that condition the data in a preprocessor. Once you have completed these steps, you are ready to start building models. This chapter describes how you set up and train a predictive model.

You do not have to be an expert in statistics, parameter estimation, or neural network to get an excellent model, using NOL Studio. Most technical decisions, such as best combination of inputs, time delays, model type, number of hidden nodes, etc. are determined automatically. This saves you the repetitious, time-consuming task of training multiple models to determine the best settings for these parameters. In most cases, a single training run is sufficient to get an optimal model.

If you desire, you can override the automatic features and create a model with customized features. However, most users will want to take full advantage of the built-in, time-saving features of NOL Studio.

One NOL Studio project can contain any number of models. This allows you to train models for different phases of your operation, gives you the freedom to experiment with different preprocessing strategies, and enables you to compare models, using the validation tools described in Analyzing a Trained Model until you are completely satisfied with the performance of your model or models. Then you save out your best model or models for on-line deployment.

Creating a Predictive Model

To create a predictive model, follow the steps in the modeling wizard. The wizard guides you through the steps to create a model:

1. Name the model.
2. Select whether to use old model parameters.
3. Specify the preprocessor for the model.
4. Specify the output data series to be used in the model.
5. If the output data series is time-based, specify a recursive model.
6. Classify the variables as input, output, or unused.
7. Specify time delays, if any, for the model inputs.
8. Automatically selecting inputs and delays.

These steps are detailed in the following sections.

To launch the wizard:

➔ Do one of the following:

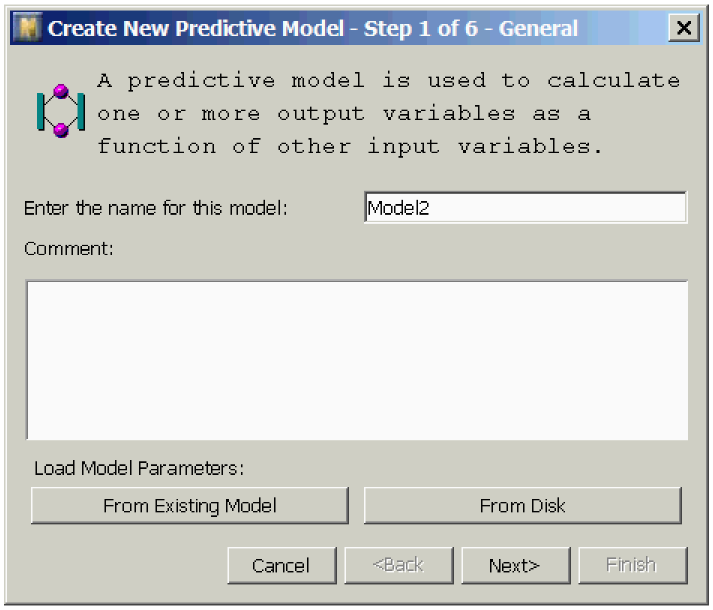
Menu Bar:	Choose Object > New > Predictive Model.
Toolbar:	Click the New Predictive Model button.
Tree View:	Right-click the Predictive Models node and choose New from its menu.

All of these actions display the model wizard.

Naming the Model

The first panel in the wizard prompts you to enter a name for your model, as well as a free-form comment.Use the comment to help you remember what data was used for training, how the data was preprocessed, and other special characteristics of the model.

For example:


Create New Predictive Model - Step 1 of 6 - General

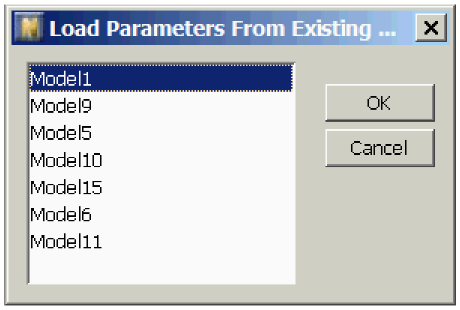
A predictive model is used to calculate one or more output variables as a function of other input variables.

Enter the name for this model: Model2

Comment:

Selecting to Use Old Model Parameters

When you build a new model, you can load the parameters from one of the existing models, or a model parameter file `*.mp` and start from there. This way you can save the works of setting the same parameters. Two buttons on the new predictive model wizard let you specify to load parameters from an existing model or a model parameter file. A dialog will let you to select the model parameters from existing models in current project. Please refer to the sensitivity workspace discussion to find how to save model parameters as an `*.mp` file.

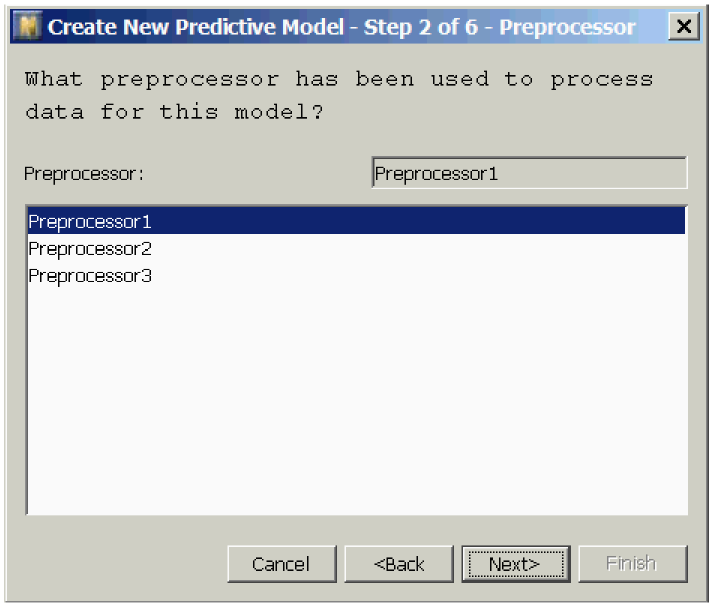

Load Parameters From Existing ...

Model1
Model9
Model5
Model10
Model15
Model6
Model11

Selecting the Preprocessor

In the second step of the wizard, specify the preprocessor that provides the training data: Each model requires a specific preprocessor as the source of training data. The preprocessor must contain all variables used in your model. You cannot take data from more than one preprocessor, and you cannot train a model on raw data. If you do not want to preprocess the raw data, simply create a preprocessor with no filter, and no formulas, and use this preprocessor as the data source for your model.

For example:



What preprocessor has been used to process data for this model?

Preprocessor: Preprocessor1

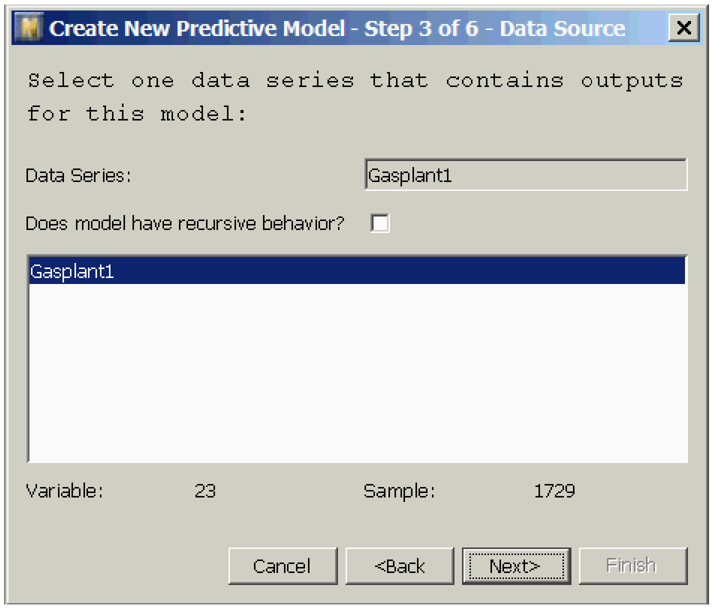
Preprocessor1
Preprocessor2
Preprocessor3

Selecting the Output Data Series

Next, you select the data series containing the output variable or variables. Only variables associated with the selected data series can be outputs of your model. However, both inputs and outputs can be contained in the selected data series. If you want to model variables from different data series, you must create different models.

If the output data series is time-based, you can specify whether you want to use delayed output feedback for input variables by clicking the check box in the wizard. This type of model is called a recursive model.

To select data series, click on the desired data series, for example:


Create New Predictive Model - Step 3 of 6 - Data Source

Select one data series that contains outputs for this model:

Data Series: Gasplant1

Does model have recursive behavior? ☐

Gasplant1

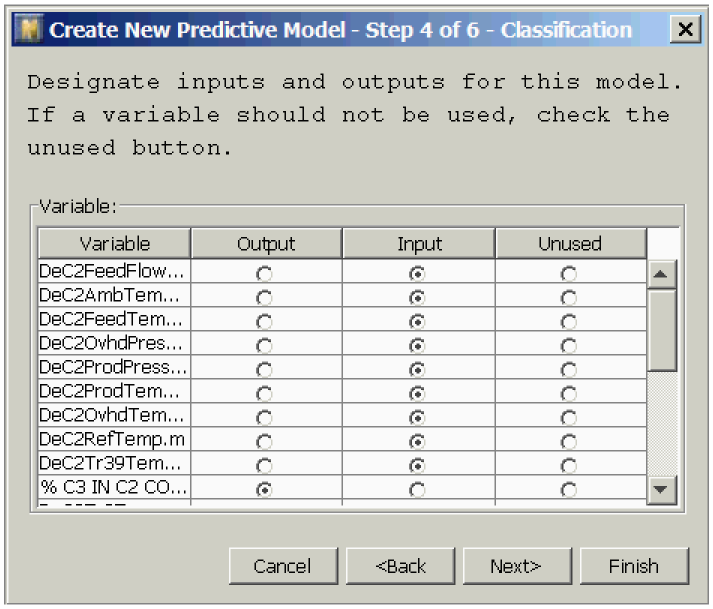
Variable: 23 Sample: 1729

Classifying Variables

In the fourth step of the wizard, you specify which variables will be inputs and outputs, and which variables are not used in the model. The list of variables include all variables in the selected data series, including derived variables defined by the preprocessor formulas.

Your selection of input variables at this stage should be considered tentative. A final decision will be made at a later stage. If you are unsure whether a variable should be included in the inputs, leave it in at this step. Later, each proposed input will be tested statistically to determine whether it should really be included in the model. At that point, the selection of inputs will be finalized.

To classify the variables, select the appropriate radio buttons, for example:



Designate inputs and outputs for this model. If a variable should not be used, check the unused button.

Variable:

Variable	Output	Input	Unused
DeC2FeedFlow...	○	●	○
DeC2AmbTem...	○	●	○
DeC2FeedTem...	○	●	○
DeC2OvhdPres...	○	●	○
DeC2ProdPress...	○	●	○
DeC2ProdTem...	○	●	○
DeC2OvhdTem...	○	●	○
DeC2RefTemp.m	○	●	○
DeC2Tr39Tem...	○	●	○
% C3 IN C2 CO...	●	○	○

Specifying Time Delays

By default, it is assumed that there are no significant time delays between inputs and outputs. In terms of dynamic systems, this implies that the cause and effect relationship between inputs and output acts rapidly, compared to the time interval between measurements. Therefore, the model output at any time is a function of the inputs at the same time. This is a good assumption for processes operated at steady state, and processes that reach steady state at a time scale faster than the time scale of measurements.

A process whose equilibration time scale is fast compared to observation time scale of the output variable is sometimes referred to as *quasi-steady state*. A quasi-steady state process might display dynamics, but these dynamics reflect the external forcing, and a state of “moving equilibrium” is always maintained. You can tell if your process operates at quasi-steady state by observing the response of the process to a step change in external (control or disturbance) parameters. If the output measurements move to new values at the next measurement time, and stay at those values, delays are not significant in your process.

In some processes, there are significant lags or delays between input and output variables. Significant, in this context, means on the same time scale as the interval between output measurements. Typically, lags are associated with the time it takes for material or energy to be transported from the input to the output of the process. To get the best results, you must account for these lags.

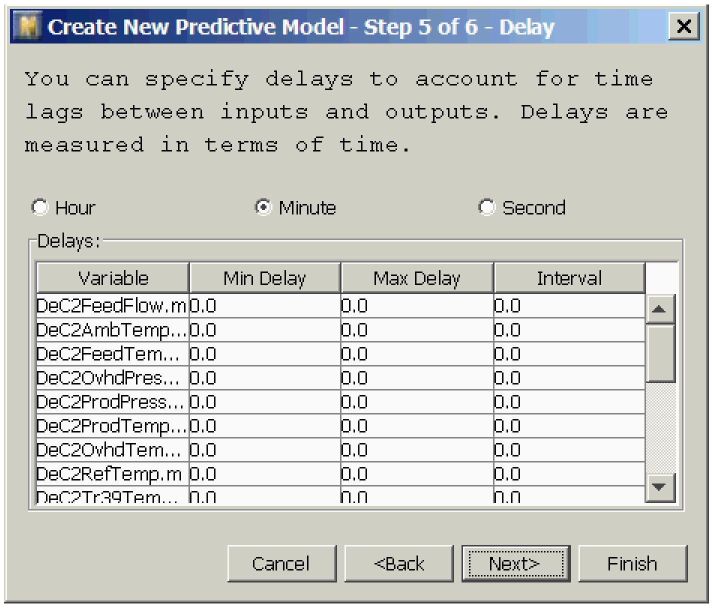
Output Delays

You can specify delays on output variables, which means the variable is both an input and output of the model. This model structure is suitable when you have online measurements, where there can be significant lags, or delays, between cause and effect of the process changes. At the same time, there can be some unmeasurable disturbances that enter the process. In this context, using delayed outputs as inputs can account for the effects caused by these undefined disturbances, which enables you to get the best results.

Note Time delays are only relevant for time-based data. Row-based data (data without time stamps) cannot have delays. If you want to add delays to row-based data, you must first specify time stamps.

If you know the delay time or approximate delay between the input and output variables, enter the delay time as the minimum and maximum delay. This replaces the input with a lagged value of the same variable. If you only know an approximate range of delay times, enter a range of values that bracket the actual delay. Then, enter an interval defining a grid of time points between the minimum and maximum delay. This creates a “window” of past values that are inputs to the model. At the next step, you can determine which, if any, of these delays are most important to actually include in the model. You can specify a different set of delays for each input variable.

For example, if the minimum delay is 0 hours, the maximum delay is 6 hours, and the interval is one hour, then your model will tentatively include delayed inputs of 0, 1, 2, 3, 4, 5, and 6 hours. This is illustrated in the following dialog:


Create New Predictive Model - Step 5 of 6 - Delay

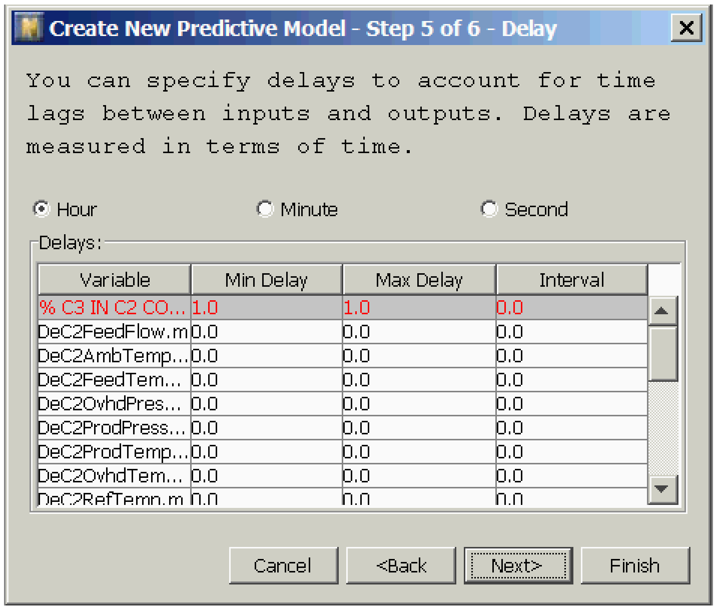
You can specify delays to account for time lags between inputs and outputs. Delays are measured in terms of time.

Hour [✓ Minute] Second

Delays:

Variable	Min Delay	Max Delay	Interval
DeC2FeedFlow.m	0.0	0.0	0.0
DeC2AmbTemp...	0.0	0.0	0.0
DeC2FeedTem...	0.0	0.0	0.0
DeC2OvhdPres...	0.0	0.0	0.0
DeC2ProdPress...	0.0	0.0	0.0
DeC2ProdTemp...	0.0	0.0	0.0
DeC2OvhdTem...	0.0	0.0	0.0
DeC2RefTemp.m	0.0	0.0	0.0
DeC2Tr39Tem...	0.0	0.0	0.0

If you have selected a recursive model, the outputs with delays appear at the top of the variable list, as this figure shows. Follow the steps described above to set the output delays. Note that cannot specify zero delays for any output.


Create New Predictive Model - Step 5 of 6 - Delay

You can specify delays to account for time lags between inputs and outputs. Delays are measured in terms of time.

(●) Hour ( ) Minute ( ) Second

Delays:

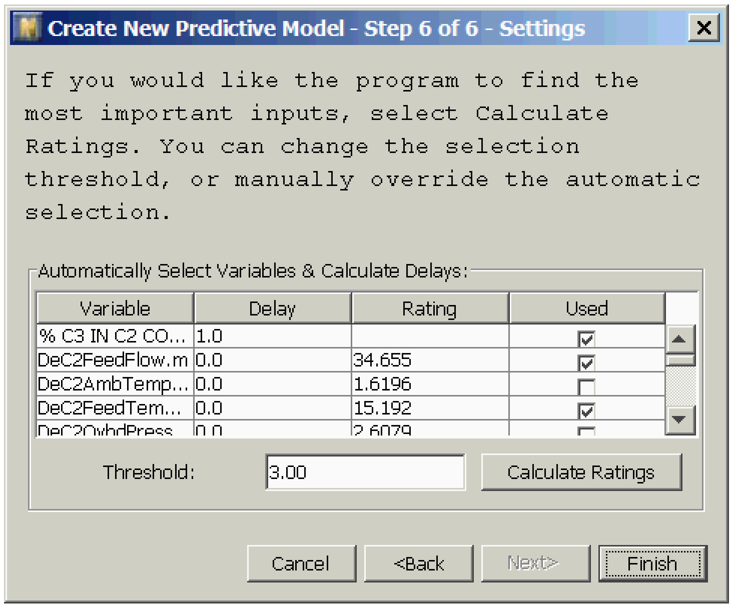
Variable	Min Delay	Max Delay	Interval
% C3 IN C2 CO...	1.0	1.0	0.0
DeC2FeedFlow.m	0.0	0.0	0.0
DeC2AmbTemp...	0.0	0.0	0.0
DeC2FeedTem...	0.0	0.0	0.0
DeC2OvhdPres...	0.0	0.0	0.0
DeC2ProdPress...	0.0	0.0	0.0
DeC2ProdTemp...	0.0	0.0	0.0
DeC2OvhdTem...	0.0	0.0	0.0
DeC2RefTemp.m	0.0	0.0	0.0

Automatic Selection of Inputs and Delays

At this stage, you have selected a set of tentative input variables, possibly including some delayed variables. However, not all these variables should necessarily be included in your model. Some proposed inputs may have no effect, or very weak effect, on the outputs. In this step of the wizard, NOL Studio will automatically select the model inputs and optimal delays from your tentative input set.

Your model should include only those variables significantly correlated with an output variable. NOL Studio uses a nonparametric (nonlinear) correlation analysis to determine the correlation of each proposed input variable. Variables identified as significantly correlated are retained in the set of input variables. Those variables not significantly correlated are not selected. This rating is expressed in terms of a number of standard deviations, or sigmas, similar to statistical quality control. By default, NOL Studio uses the value of 3.0 (meaning 3 sigmas) as the level of significance for inclusion of a variable in the model. At this level, the input variable is almost certainly correlated with the output, and should be include in the input set.

To trigger the calculation of inputs, click Calculate Ratings, as shown below. This figure shows a non-default threshold of 2.0:



Create New Predictive Model - Step 6 of 6 - Settings

If you would like the program to find the most important inputs, select Calculate Ratings. You can change the selection threshold, or manually override the automatic selection.

Automatically Select Variables & Calculate Delays:

Variable	Delay	Rating	Used
% C3 IN C2 CO...	1.0		✓
DeC2FeedFlow.m	0.0	34.655	✓
DeC2AmbTemp...	0.0	1.6196	
DeC2FeedTem...	0.0	15.192	✓
DeC2OvhdPress	0.0	2.6079	

Threshold: 3.00

The automatic selection of inputs is shown in the column of check boxes next to each variable. If a variable is not checked, it will not be included in the model. If a variable is checked, it is included. You can override these choices by manually checking or unchecking the boxes.

Hint You can sort the input variables from most to least important by clicking on the Ratings column header. To reverse the order of the sort (from least to most important) click on the column header again. To sort the variables alphabetically, click on the first column header, labelled Variables.

In the case of a delayed input, the program determines the significance rating for each delay between the minimum and maximum you specified. You may want to select only the delay that is most highly correlated with the target output value (with the highest rating).

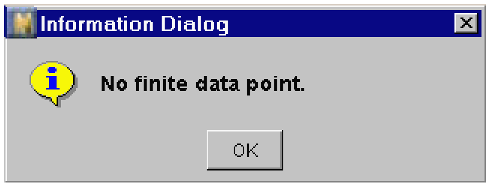
Even when the output is a multivariate function of the inputs, individual input variables will register correlations with the outputs, in most cases. For example, if an output y is the product of two independent, random input variables x_1 and x_2, correlations between x_1 and y and x_2 and y will be detectable, since y generally increases when either variable increases. However, for small data sets, it may be difficult to detect the correlation between x_1 and y because of the noise injected by the independent variation of x_2. If you have a small data set and few variable correlations exceed the default threshold, you may have to lower the threshold, or choose inputs on a physical, causal basis.

Tip If you feel an important variable has been excluded from the model, the problem might be insufficient variation of that variable in the training data set. Variables with insufficient excitation may be excluded from the model on statistical grounds, even if there is a real causal effect between the input and output. If this is the case, consider gathering more data, intentionally varying this input.

Tip Including more input variables in your model than indicated by the program can degrade the performance of your model.

After ratings have been calculated and the final choice of inputs have been determined, click Finish.

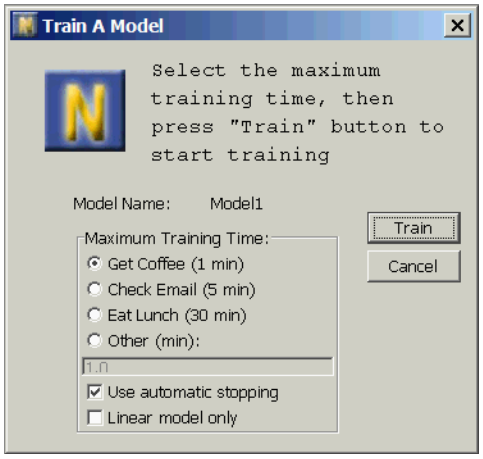
If the data series that you selected for building a model does not have valid data point, a message dialog will be display when you click Calculate Ratings button. If you see this message, you will need to modify the data series to add more valid points.



No finite data point.

The Training Console

When you exit the wizard, the following dialog appears, prompting you to select a maximum training time, stopping strategy and whether you want to train a linear model only.



Select the maximum training time, then press "Train" button to start training

Model Name: Model1

Maximum Training Time:

- [✓ Get Coffee (1 min)]
- Check Email (5 min)
- Eat Lunch (30 min)
- Other (min):

1.0

- [✓ Use automatic stopping]
- Linear model only

In most cases, you will want to employ the Use Automatic Stopping option. When you select this option, NOL Studio will continue to create and train models until no further improvements are experienced. Setting a maximum training time limits the amount of time NOL Studio will use to train the model. If you do not use automatic stopping, training will continue to the maximum training time, whether or not the training process is making progress. However, you can manually stop the training at any time.

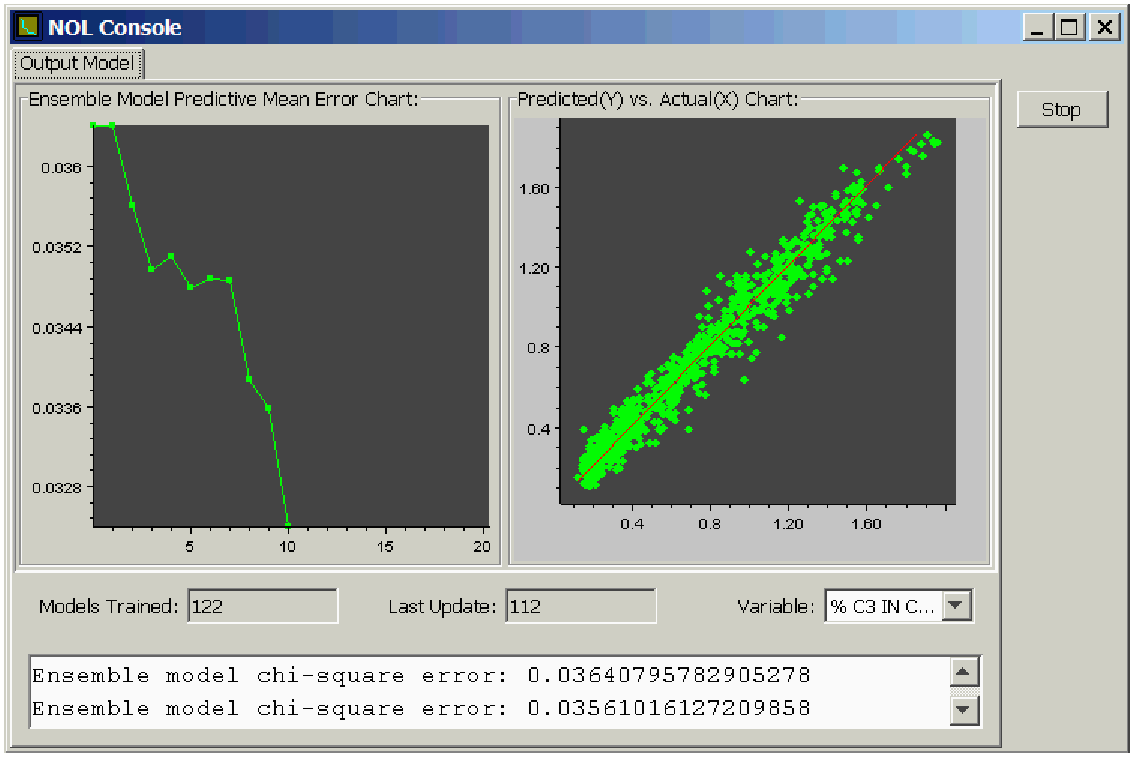
Linear models have different characteristics from nonlinear models. In some cases, if you know that your system is likely to have linear relationship or can be well approximated by a linear relationship, the best choice is to limit the training algorithm to train a best linear model only. In such cases, linear models intend to be more robust than nonlinear models and may have better extrapolation property, because nonlinear models are more likely over-fit with sample noise. The ways to validate and deploy the linear models are the same as the unspecified models. A check box on the training dialog lets you specify that you want to train a linear model only.

Note A linear model can be exported to a text file and loaded into a G2 environment as either an ensemble model or as a Partial Least Squares (PLS) model. For details, see Model Deployment.

Initially, you may opt for a short training time, so you can view the preliminary results quickly, and if they are promising, train for a longer time. You can continue the training as many times as needed. Because of the special way that NOL Studio trains and selects models, extending the training time cannot lead to overtraining the model. For each model, training is automatically stopped at the optimal point where the model is neither overtrained nor undertrained.

Once a training time has been selected, the training process begins. Training is monitored through a special window called the training console. Unlike other dialogs in NOL Studio, the training console exists outside the main NOL Studio window. Therefore, the training console can be separately closed or iconized, so you can continue to use NOL Studio application while the training is underway. However, during training, performance might be more sluggish because of increased computational demands.

The console is shown below:



```
Ensemble model chi-square error: 0.03640795782905278
Ensemble model chi-square error: 0.03561016127209858
```

The chart on the left shows the current fitting error on test data, averaged over the best five models trained to date. The top five models together constitute the ensemble model, as discussed in the following section. The chart on the right shows a plot of predicted versus actual fit, for the output variable designated in the selection box in the lower right. To view predicted versus actual for another variable, change the selection in the selection box. The red line in the predicted versus actual plot is represents the perfect model where the predicted value generated by the neural network model exactly equals the training target value. Because of measurement noise, points are usually scattered around the target line.

Below the plots, the two edit boxes show the number of models trained thus far, and the iteration when last model that was added to the ensemble. As explained below, when in automatic mode (the default), the program trains multiple models as it searches for the best architecture.

Training will terminate when the allotted time has expired. Alternatively, you can terminate training at any time by selecting the Stop button on the console. To continue training, use the Continue Training button on the Model Properties dialog. You navigate to the Model Properties dialog via the tree view.

Training and Model Selection Algorithms

During the training process, NOL Studio prepares a training set from the preprocessed data series, trains candidate models, and automatically optimizes the model architecture. This section describes the internal process that results in the optimal model. This section is technical and assumes a background in modeling and statistics. If you do not have a technical background in modeling and statistics, you can skip this section.

Preparing the Training Set

When you begin training, NOL Studio prepares a training set according to your specified variables and delays. First, the appropriate columns of data are selected. Next, the rows of the training set are prepared. The details of this process are different for row-based and time-based data.

- For row-based data, each sample corresponds to a single row of a one data series. It is not possible to interpolate missing values, because of the fundamental assumption that rows are independent of one another. Therefore, rows containing missing values are discarded.
- For time-based data, the rows are prepared by a process of smoothing and interpolation. The timestamps of the rows of the training set are taken from the timestamps of the data series containing the output variable or variables. Timestamps are discarded if any of the output variables are missing. Once the output times have been determined, the inputs corresponding to the output times are determined. For each input variable, the corresponding input times are determined.
    - If there are no delays, the input times are the same as the output times.
    - If there is a time delay, the input times are the output times less the delay time.

  The values of the input variables at the input times are determined by interpolation. The default method is linear interpolation. Other available methods include polynomial and cubic spline interpolation.

  Depending on the interpolation option, an input variable can be smoothed, by filtering high-frequency variations. Because of smoothing, the interpolated value at a time might not be equal to the measured value at the same time. Also, missing values can be filled in by interpolating across gaps in the data.

Note Polynomial and cubic spline interpolation and smoothing options are not available in this version of NOL Studio.

Model Types

NeurOn-Line can create both linear and nonlinear models.

- If your data follows linear trends, NOL Studio will create a linear model.

  NOL Studio uses a linear modeling technique known as partial least squares (PLS) to model linear processes. In contrast to normal linear regression, which can be ill-conditioned when inputs are correlated, PLS models are very effective when there are many inputs, even when the inputs are linear combinations of other inputs. PLS automatically determines input correlations and creates a new input space with reduced dimension, containing uncorrelated latent variables (factors). In PLS models, there is one parameter, the number of latent factors, that must be optimized. Detailed information on PLS is widely available in the statistics and chemometrics literature.

- To model nonlinear processes, NOL Studio uses two types of nonlinear models. Both types combine PLS and backpropagation neural networks. In both models, PLS automatically determines input correlations and creates latent variables to reduce the effective dimensionality of the model. The latent variables are linear combinations of the input variables which maximize correlation with the output variable.

  - In the first type of nonlinear model, the latent factors are developed one at a time, and a small neural network is used to map each latent variable to the output variables. The process is repeated until the desired number of factors is reached.
  - In the second type of nonlinear model, all PLS factors are determined at once, and a multiple-input, multiple-output neural network is used to relate the factor space and the corresponding reduced variable space calculated for the outputs.

  The first nonlinear model type is quicker to train, but may be less accurate than the second type.

Model Structure Determination

NOL Studio automatically selects the best architecture through a process of guided evolution. This evolution starts with simple models, and leads to more complex structures. Both the basic type (linear versus nonlinear), and model parameters (e.g. number of hidden nodes) are evolved toward optimality. Many models may be trained during a short training run when this evolutionary process is used.

When you initiate training, NOL Studio will train a set of linear models. The quality of fit of these models will be evaluated to determine whether the data has been underfit or overfit. A model is underfit if there are systematic variations in the target data that are not accounted for, by the model. A model is overfit if the model fits random variations in the target data.

- When the model is determined to be underfit, the new model is created that adds some degree of complexity above and beyond that of the earlier model. This complexity may be to move from linear to nonlinear behavior, add a latent variable, add more nonlinear elements to the network, or a combination of these elements.
- If, on the other hand, the model is overfit, complexity will be removed, by changing nonlinear elements to linear elements, reducing the number of latent variables, or pruning the network.

When seeking a model to evolve, NOL Studio draws from a pool containing the most successful models. Source models are selected in random manner, weighted by the success of the model. Maintaining a pool of models prevents early termination at suboptimal model structures.

Model Selection

When comparing the performance of multiple models, it is necessary to have an accurate basis for comparison. It is well known that the accuracy of fit on the training data is not an accurate predictor of performance. For example, you can fit 5 points perfectly with a fourth order polynomial, thus achieving zero error, but the polynomial may have wild oscillations and thus be a very poor predictor of the actual shape of the function. A straight line may be a more reasonable fit, even though the fitting error is non-zero. A more relevant performance indicator is the performance of the model on future data, drawn from the same population as the training data, but unseen during the training process. The error on unseen data is called the prediction error.

Cross-validation is the best general-purpose technique for determining the prediction error. The cross-validation technique establishes a testing subset, extracted from the training data, which is held back from the training process. Once the model has been trained, the model is run using the inputs in the testing subset. The result is compared to the targets in the testing subset, yielding an estimate of the prediction error. The prediction error thus calculated represents the model performance on “unseen” data, and is a fair predictor of the model performance on future data, under the assumption that the training set is representative of future conditions.

NOL Studio always uses the prediction error in assessing model performance and selecting the most successful models. Because of this strategy, you do not have to worry about overtraining (continuing training to the point of “memorization” of the training examples). NOL Studio will always give you a model optimized for performance on future data.

Ensemble Models

Several research studies have shown that combining multiple neural networks can result in a more robust predictive model. In NOL Studio, we take the approach of retaining a group (ensemble) of the most successful models, and having them vote to determine the final prediction. If, for a particular input, one of the models makes a poor prediction, the voting process will make sure the poor prediction is not allowed to influence the final result. Mathematically, the voting process involves taking the median of the individual model predictions. The median is a more robust statistic than the mean, since the mean can be severely impacted by an outlier.

The ensemble model approach also provides an indication of the certainty of the prediction. If all models agree on a prediction, the level of confidence is high. If there is a large disagreement between the predictions, the confidence is low. This feature of the ensemble model may be exploited in future releases of NOL Studio.

Analyzing a Trained Model

Describes how to inspect and validate the performance of a predictive model.

Introduction

Once a model has been trained, you can inspect the model to determine its suitability, robustness, and performance. Typical activities include:

- Analyzing statistical measures of performance on the original training and testing data series.
- Viewing scatter plots and line charts of predicted versus actual output on the training and testing data.
- Viewing plots and statistics of model performance on new data never seen in the training process. For example, if you have built a model using plant data from January and February, you might want to validate the model using data from several subsequent months. The performance of the model on the new data will indicate how well the model has generalized.
- Creating a simulation that can either run *what-if* scenarios on existing data series, or on fictitious data specified by the user to analyze how the model will perform on a wide range of data values.

These exercises will help you determine if the model has been trained on sufficient data, whether it has generalized well, and within what ranges of input values it is expected to do well. NOL Studio provides a rapid model development environment to *build-train-validate* models. You can easily evaluate the performance of a model, continue training, or create an entirely new model.

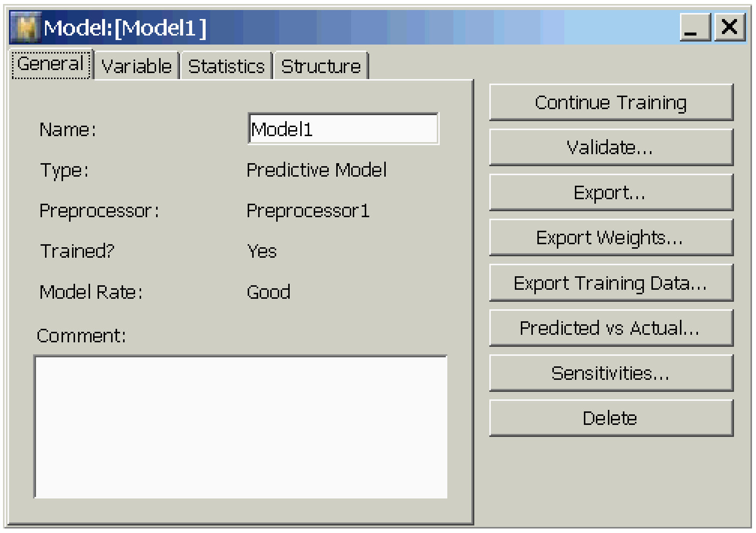
Viewing the Model Properties

When you finish the model wizard, a predictive model object is created and added to the tree view. You can view the Properties dialog for any model by double clicking the corresponding tree view node.

General Properties

The General Properties tab on the model properties dialog gives general information about the model, such as the name of the model, the type of the model, the name of the preprocessor associated with the model, whether the model has been trained, a brief description of the model performance, and any comments you entered about the model.

For example:


Model:[Model1]

Name:	Model1
Type:	Predictive Model
Preprocessor:	Preprocessor1
Trained?	Yes
Model Rate:	Good
Comment:	

Brief Information of Model Performance

In the Statistics section, you can find the specification of available statistics and the rules of thumb of interpreting them. However, sometimes these statistics are hard to interpret. Based on the correlation coefficients, which represent the predictive ability of models, the predictive models are rated as “Good”, “Ok”, and "Need Improvement". This rating is based on the average the correlation coefficients of all output variables.

1 If 0.75< AVE_COEF, the model is rated "Good".

2 If 0.55< AVE_COEF < 0.75, the model is rated "Ok".

3 If AVE_COEF < 0.55, the model is rated "Need Improvement".

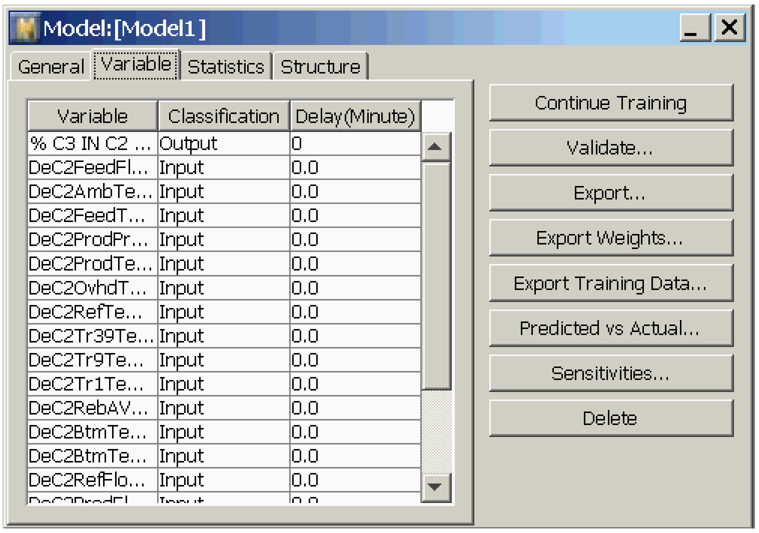
This rating can only be used as a primary indicator of the model performance. Please refer to the Statistics section for detailed performance indexes.

Model Variables

The Variables tab on the model properties dialog summarizes the classification of variables and delays for each input and output for this model. Information on this tab is read only.

The variables in the model may be a subset of the variables that are available in the data series, since you are allowed to choose which variables are relevant for predicting the output variables. This tab also shows you the classification of each variable. For predictive models, this can be either Input or Output. Input variables of optimization models are categorized as Exogenous, Manipulated, or State. The list does not display unused variables.

For example:



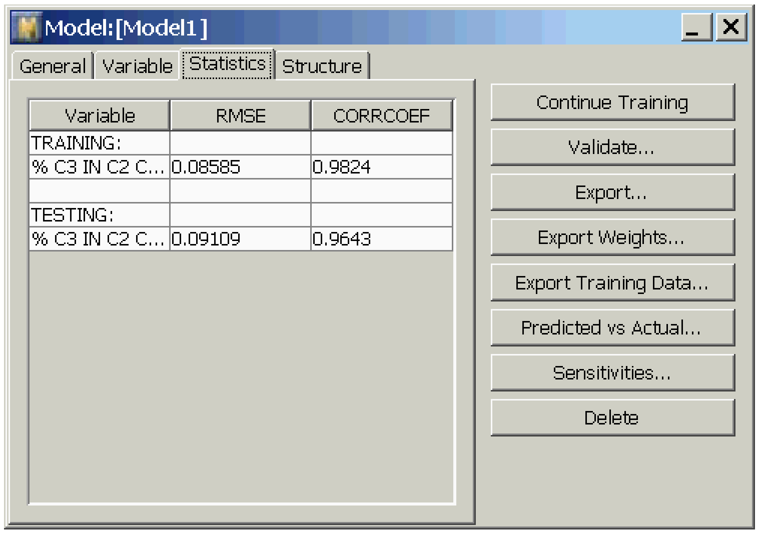
Variable	Classification	Delay(Minute)
% C3 IN C2 ...	Output	0
DeC2FeedFl...	Input	0.0
DeC2AmbTe...	Input	0.0
DeC2FeedT...	Input	0.0
DeC2ProdPr...	Input	0.0
DeC2ProdTe...	Input	0.0
DeC2OvhdT...	Input	0.0
DeC2RefTe...	Input	0.0
DeC2Tr39Te...	Input	0.0
DeC2Tr9Te...	Input	0.0
DeC2Tr1Te...	Input	0.0
DeC2RebAV...	Input	0.0
DeC2BtmTe...	Input	0.0
DeC2BtmTe...	Input	0.0
DeC2RefFlo...	Input	0.0
DeC2ProdFl	Input	0.0

Statistics

The Statistics tab provides performance statistics for the trained model. During training, data is divided into training and testing subsets, whose size and content is automatically determined. The statistics show how well the model fits the training and testing subsets.

Note that the future performance of the model is predicted only by the testing statistics, since these statistics represents the model performance on data not directly used for determining the model parameters. The training statistics are usually slightly better than the testing statistics, since the model is tuned specifically to this data. If the training statistics are much better than the test statistics, you probably do not have enough data for the model to have captured the underlying functionality.

The specific statistics shown for each output are the root mean squared error (RMSE) and the correlation coefficient (CORRCOEF). For example:



Variable	RMSE	CORRCOEF
TRAINING:		
% C3 IN C2 C...	0.08585	0.9824
TESTING:		
% C3 IN C2 C...	0.09109	0.9643

The RMSE is calculated in the following manner:

1 For each training or testing example, calculate the difference between the model's prediction and the training target.

2 Square the differences.

3 Determine the mean of the squared differences, over the entire training or testing set.

4 Calculate the square root of the mean.

The RMSE is not normalized; therefore, its units are the same as the units of the output.

The correlation coefficient (often given the symbol *r* in statistics texts) measures the degree of scatter in the predicted versus actual (target value) plot. It is the covariance of the predicted and actual values, divided by the square root of the product of the variance of the predicted values and the variance of the actual values. If the predicted versus actuals lie perfectly on a line (which is unlikely to

happen because of measurement noise), the correlation coefficient is 1.0. If the data is randomly scattered, the correlation coefficient
is 0.0.

As shown in many statistics texts, the square of the correlation coefficient (r^2) can be interpreted as the ratio of the variation explained by the model, to the total variation in the data. Thus, a value of $r = 0.8$ indicates that the model accounts for 64% of the variation in the data. The remaining variation has two possible sources: measurement noise and model mismatch.

It is important to bear in mind that, due to measurement error of the output variables, a “perfect” model -- one that predicts the true value at times -- does not necessarily correspond to an RMSE of 0 and a correlation coefficient of 1. In fact, if there is measurement error on an output variable, the RMSE should always be greater than or equal to the standard deviation of that error. This is because the measurement error is essentially unpredictable, and cannot be captured by the model.

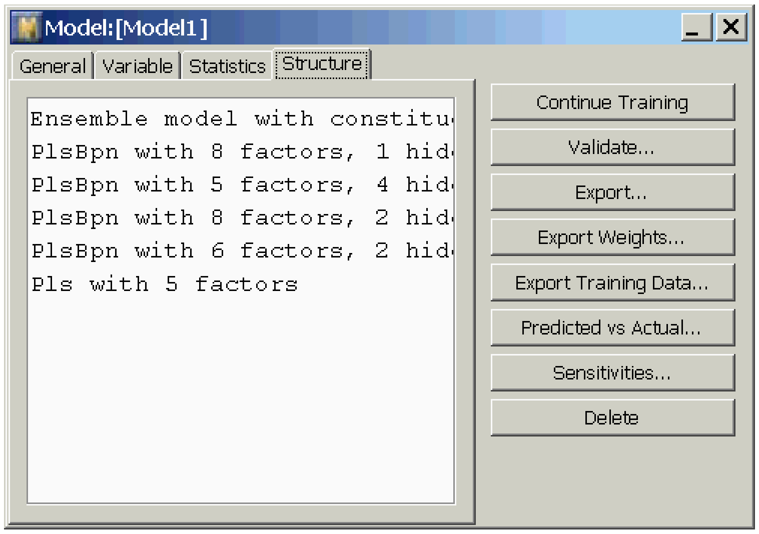
The measurement error associated with an instrument or lab test is sometimes given in terms of a plus/minus accuracy (a 95% confidence value). In this case, the measurement standard deviation (MSD) is 1/3 of the accuracy. Otherwise, you can determine the MSD experimentally by repeatedly analyzing a control sample, and taking the standard deviation of the resulting values.

In interpreting the RMSE, use the following rules of thumb:

- If RMSE < MSD, the model is fitting the noise, and the model is “too good to be true.”
- If RMSE = MSD, the neural network model is exact.
- If RMSE = 1.2*MSD, the model has about half the error of the physical measurement.
- If RMSE = 1.4*MSD, the model is about as accurate as the physical measurement.
- If RMSE = 1.7*MSD, the model has about twice the error of the physical measurement.

Model Structure

The Structure tab shows the structure of the ensemble model. This gives you a chance to review and verify the models created by NOL Studio. For example:



```
Ensemble model with constitu
PlsBpn with 8 factors, 1 hid
PlsBpn with 5 factors, 4 hid
PlsBpn with 8 factors, 2 hid
PlsBpn with 6 factors, 2 hid
Pls with 5 factors
```

To see the internal parameters and other details, you must export the model as a text file by using the Export Weights button. For details, see Saving a Model. For more information on the various types of models, see Model Types.

Performing Operations on the Model

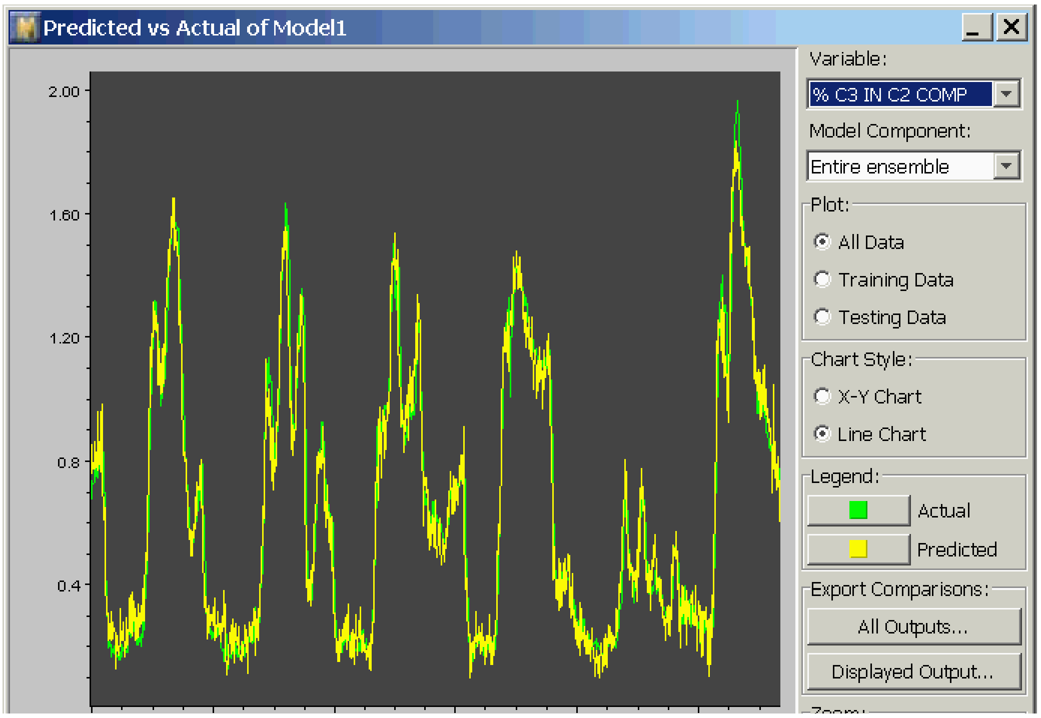
This section describes the operations you can perform on the model, using the buttons along the right side of the model properties dialog.

Continuing Training

To continue training this model, use the Continue Training button. This will launch the model training time dialog, as described in The Training Console, and allow you to initiate further training.

Showing the Predicted Versus Actual Plot

To show a plot of predicted versus actual values attained during training, click Predicted vs. Actual. This brings up the following dialog:



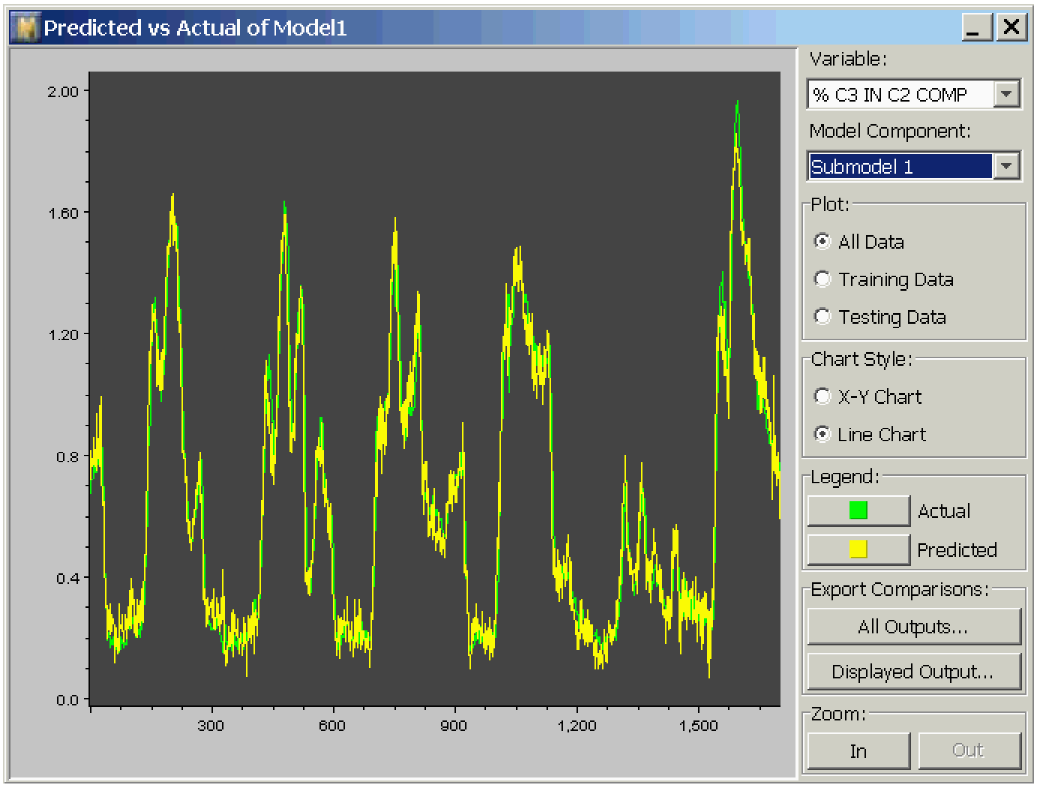
In this dialog, you can show the prediction of any output versus the training target values of that output, as a line chart (shown above), or as an x-y (scatter) chart. You change outputs by selecting from the Variable drop-down list. You change chart formats by selecting the Chart Style radio buttons. To change colors on the plot, select the appropriate button in the Legend section.

Viewing Parts of the Ensemble Model

NOL Studio models are actually a group of models working together to make a prediction (see Ensemble Models). In this dialog, you can see how each model in the ensemble was trained, and how each performs, independent of the rest of the ensemble. To focus on a single model in the ensemble, select a submodel from the Model Component drop-down box. The plot is updated to reflect only the contribution of that component of the ensemble.

Caution If your models were originally trained using Version 2.0 Rev. 0 or earlier versions of NOL Studio, you will not be able to use the features described in this section.

In addition, you can see how NOL Studio divided the data into training and test subsets during the training of the selected submodel, by selecting All Data, Training Data, or Testing Data from the Plot section. The following figure shows the predicted versus actual plot for submodel 1, on the test data:



In this case, only a fraction of the data was used for testing (up to 95% of the data can be used for training). The discontinuities in the predicted (yellow) line indicate data used for training this submodel. When you compare the various submodels, you will see that different divisions of the data are used for each submodel. The greater coverage of the data by the ensemble is one reason that the performance of the ensemble model typically exceeds that of the individual submodels.

Exporting Predictions

You can save the model predictions to a file, using the Export Comparisons buttons. The data that is saved reflects the predictions as seen on the screen when you perform this action. For example, if you are displaying the predictions for the test data subset when you export the predictions, only selected rows of the data will be written. The “gaps” -- corresponding to samples not chosen for the test set -- will be written as not-a-number (NaN). When you select the All Outputs button, you save as many columns of data as there are outputs in the model. The Displayed Outputs button saves only the currently-displayed output. Time stamps are included in the output, regardless of the option selected. The actual output values corresponding to the specified predictions are also included in the exported file. In this way, it is easy than before to make a comparison when using the exported file.

When writing the file, you can choose either binary (`.bds`) or text (`.ds`) formats. You specify which option by specifying the file extension in the Save File dialog. An example format of the text file is shown below:

```
Time           10A100C3.PV_pred          10A100C3.PV
Time           % C3 IN C2 COMP_pred      % C3 IN C2 COMP
Millisecond    None                      None
     8.71964E+11           1.225174967            1.385540366
     8.71964E+11           1.286989442            1.385540247
     8.71964E+11           1.366936914            1.385540247
     8.71964E+11           1.374557197            1.382564783
     8.71965E+11           1.393119682            1.382488489
     8.71965E+11           1.409462661            1.367229223
     8.71965E+11           1.417492178            1.412197351
     8.71966E+11           1.439030908             1.42521441
     8.71966E+11           1.466799704             1.42521441
```

Zooming

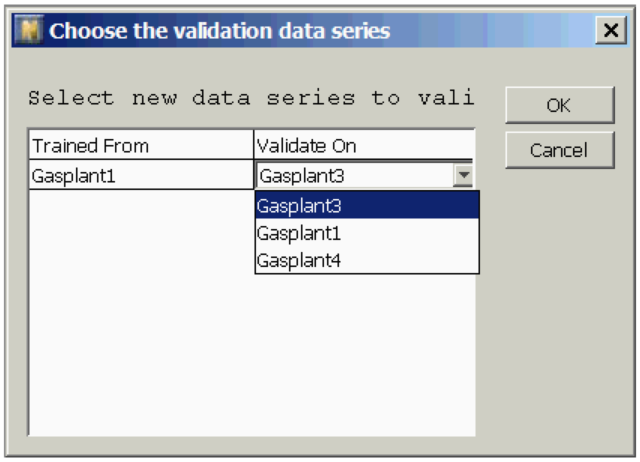
All of the charts in the predicted vs. actual dialog allow you to interactively zoom in/out and scroll through your data. When you zoom in, the location of the point in the center of the plot does not change. Thus, if you want to zoom in on a particular point, scroll the plot horizontally until the point is shown in the center of the screen, then click the Zoom In button. When you zoom in, plot symbols representing the individual data points appear. In a line chart, you cannot magnify the y axis by zooming, whereas in a scatter chart, you can.

Validating a Model Against Another Data Series

If you are satisfied with the performance of the model on the present data series, then you may want to test the model on another data series. You can use any data series other than the one used for training the model, as long as it contains variables with the same tags as the original training data series. This is because validation is meant to be performed with a data series from the same underlying data sources as during training, but from a different time range or set of operating conditions.

To start a validation session from the main model dialog:

1 Click the Validate button to display this dialog:



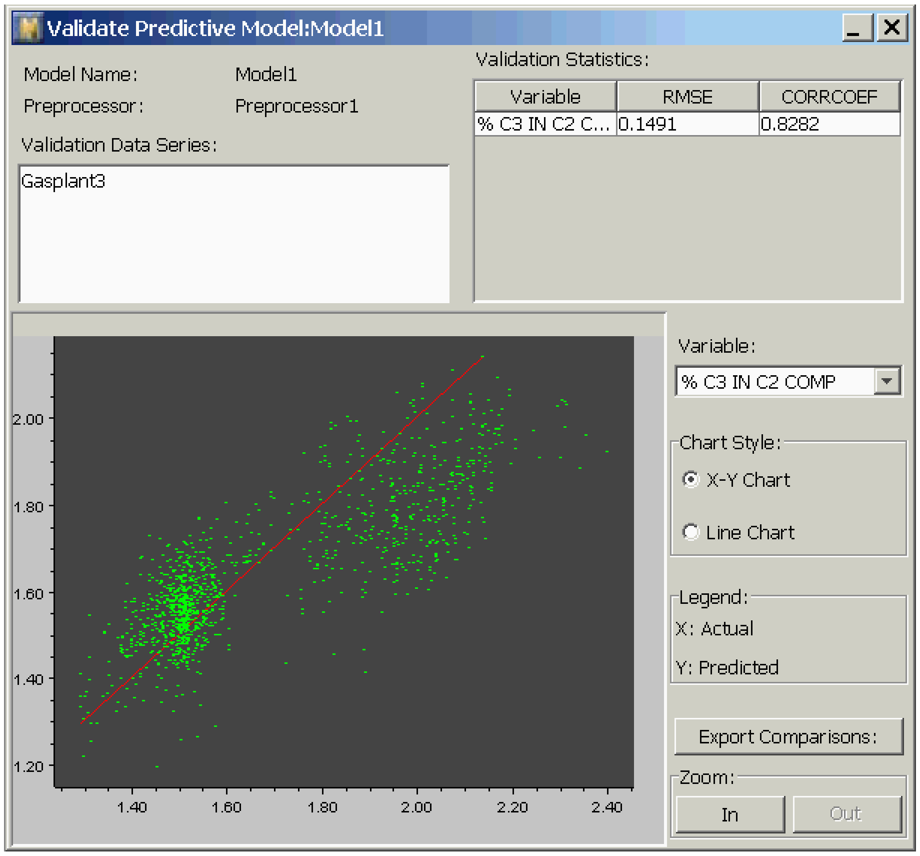
Select new data series to vali

Trained From	Validate On
Gasplant1	Gasplant3

Gasplant3
Gasplant1
Gasplant4

Only valid data series will be displayed in the drop-down list on the dialog. Click OK once you have chosen a data series for the validation.

2 The next step is to view the performance of the model on the new data series. There are several items of information displayed in the following dialog:


Validate Predictive Model:Model1

Model Name: Model1

Preprocessor: Preprocessor1

Validation Data Series:

Gasplant3

Validation Statistics:

Variable	RMSE	CORRCOEF
% C3 IN C2 C...	0.1491	0.8282

Variable: % C3 IN C2 COMP

Chart Style: [✓ X-Y Chart] Line Chart

Legend:
X: Actual
Y: Predicted

This dialog shows the name of the model, its preprocessor and the name of the validation data series. The RMSE and CORRCOEF statistics are shown in the table at the upper right. These statistics are calculated based on how well the model fits the validation data set. (For more discussion of the meaning of these statistics, see Statistics).

The bottom of the dialog contains the plot, with its controls arranged on the right hand side. For multiple-output predictive models, you can choose which output variable is plotted. You can view the plot as a line chart or scatter chart.

The predictions can be saved by selecting Export Comparison. If you want to save the predictions as a text file, change the file extension to *.ds*. If you want to save the predictions as a binary file, which is faster to reload into NOL Studio, use the extension *.bds*. The file will save all outputs, along with the time stamps, associated with the validation data series. The actual values are saved along with predictions the of the validation data. The file format is the same as the data file exported from the Predicted vs. Actual dialog.

Zooming

All of the charts in the validation dialog allow you to interactively zoom in/out and scroll through your data. When you zoom in, the location of the point in the center of the plot does not change. Thus, if you want to zoom in on a particular point, scroll the plot horizontally until the point is shown in the center of the screen, then click the Zoom In button. When you zoom in, plot symbols representing the individual data points appear. In a line chart, you cannot magnify the y axis by zooming, whereas in a scatter chart you can.

Input-Output Sensitivities

Background

Sensitivities help you identify which input variables have the most effect on individual output variables. Sensitivities are useful for understanding the correlations in your data, which may lead to a greater understanding of the physical causality of the process. Using sensitivities, you can identify inputs that have a strong influence on an output variable, or inputs that have little or no influence on the output.

If there are one or more variables that display very small influences on a variable, you may want to train another model with these inputs removed. This will reduce the number of adjustable parameters in the model, and may improve the resulting model.

Sensitivities are averages of local derivative information. They are calculated via the following process:

1 Select a random data point from the data series.

2 Generate the outputs for the data point using the model.

3 Perturb the *j*th input by a small amount, and recalculate the output.

4 For each output, form the ratio S_{ij} = ($output_i'$ - $output_i$)/($input_j'$ - $input_j$), where the prime indicates the perturbed input and output. This is the estimate of the local derivative at the selected data point.

5 Repeat from step 3, for each input.

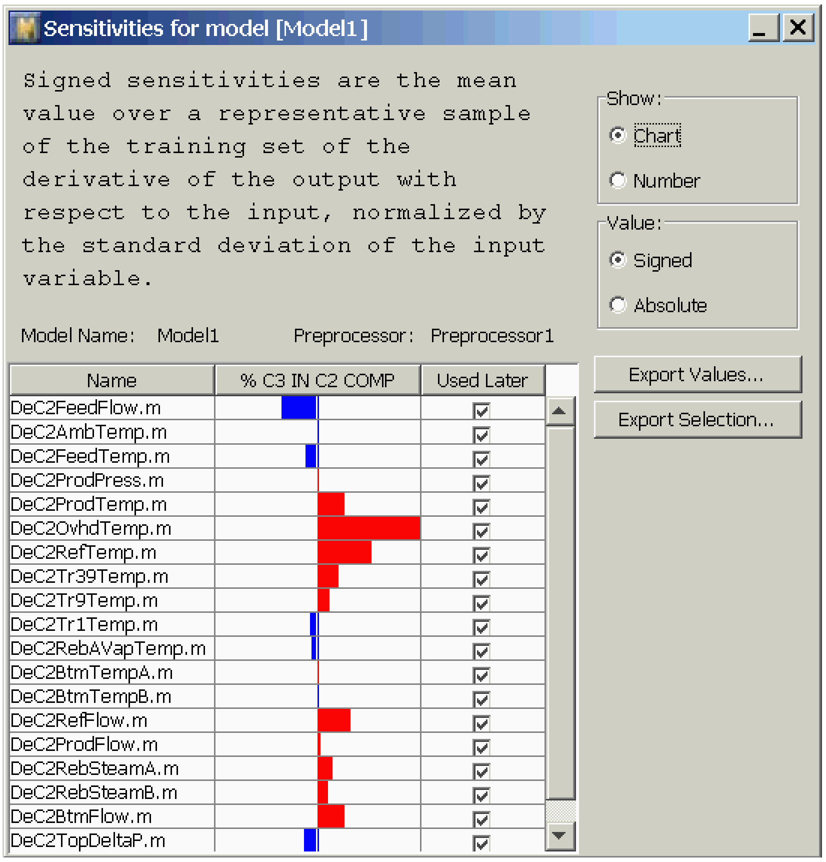
6 Repeat for another random data point.

The sensitivity value of output *i* with respect to input *j* is then calculated by taking the average of the S_{ij} values, over the sample of randomly-selected data points. To calculate absolute sensitivities, the absolute values of S_{ij} are averaged. Finally, each sensitivity is normalized by dividing the sensitivity by the standard deviation of the respective input variable. Without normalization, it would be impossible to compare sensitivities with respect to different inputs, because the sensitivities would have different units. With normalization, the sensitivities have the same units, specifically, the units of the output variable.

For a particular input-output pairing, the absolute sensitivities and the signed sensitivities have the same magnitude if all the local sensitivities have the same sign. However, if there are sign changes in the local sensitivities (indicating that the function passes through a maximum or minimum), the average signed sensitivity might be misleadingly small, because positive and negative sensitivities tend to cancel out in the averaging process. Therefore, it is a good idea to check the absolute sensitivities when assessing the importance of an input.

Displaying Sensitivities

To see the sensitivities, select the Sensitivities button on the model properties dialog. This will bring up the following dialog:



```
Signed sensitivities are the mean
value over a representative sample
of the training set of the
derivative of the output with
respect to the input, normalized by
the standard deviation of the input
variable.
```

Model Name: Model1 Preprocessor: Preprocessor1

Show: [✓ Chart] Number
Value: [✓ Signed] Absolute

Name	% C3 IN C2 COMP	Used Later
DeC2FeedFlow.m		✓
DeC2AmbTemp.m		✓
DeC2FeedTemp.m		✓
DeC2ProdPress.m		✓
DeC2ProdTemp.m		✓
DeC2OvhdTemp.m		✓
DeC2RefTemp.m		✓
DeC2Tr39Temp.m		✓
DeC2Tr9Temp.m		✓
DeC2Tr1Temp.m		✓
DeC2RebAVapTemp.m		✓
DeC2BtmTempA.m		✓
DeC2BtmTempB.m		✓
DeC2RefFlow.m		✓
DeC2ProdFlow.m		✓
DeC2RebSteamA.m		✓
DeC2RebSteamB.m		✓
DeC2BtmFlow.m		✓
DeC2TopDeltaP.m		✓

This dialog shows the sensitivity for each variable, with respect to each output, as a red or blue bar whose length reflects the magnitude of the sensitivity. A blue bar that extends to the left of center indicates that, on average, increasing the input tends to decrease the output. A red bar extending to the right of the center line indicates that increasing the input tends to increase the output. Larger bars indicate stronger effects. You can also view the numeric results instead of the graphical bars, by selecting the Number button at the top right.

To switch to absolute sensitivities, select the Absolute radio button on the right side of the dialog. When you display absolute sensitivities, the zero point is the left edge of the cell, rather than the midpoint. All values are positive and are displayed in red.

To help you find the most and least sensitive inputs, you can sort the sensitivities by clicking on the top of any column. To reverse the sorting order, click again. To sort the variables alphabetically, click at the top of the variables column.

Saving Sensitivity Values

Click on the Export Values button to save the sensitivity values into a txt file. You can use this txt file for external analysis.

Use Sensitivity Values to Help Select Model Inputs

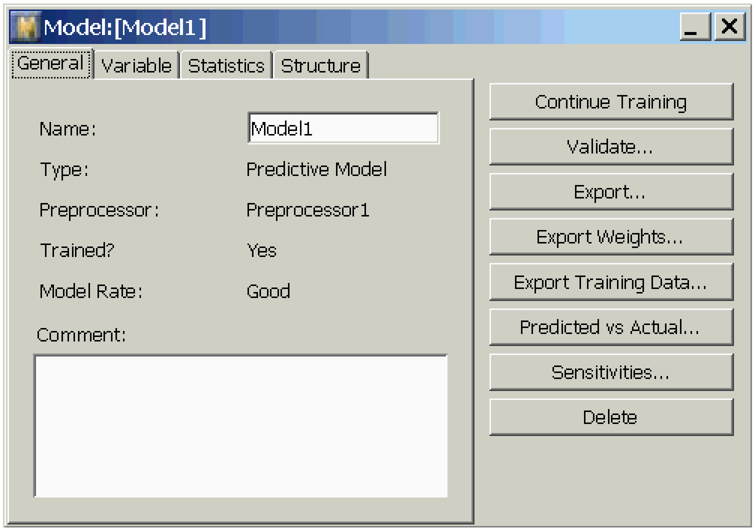
Using sensitivities, you can identify inputs that have a strong influence on an output variable, or inputs that have little or no influence on the output. If there are one or more variables that display very small influences on a variable, you may want to train another model with these inputs removed. This will reduce the number of adjustable parameters in the model, and may improve the resulting model. You can now specify whether a particular input will be used in later models. A set of check boxes is included in the sensitivity work space to mark the input selection based on sensitivity values. An Export Selection button at the bottom of the sensitivity work space can be used to export the input selection as model parameters into a file of type *.mp. This *.mp file can be used when you create a new predictive model using the new model wizard.

Saving a Model

To export a model and its preprocessor for on-line use as an ActiveX control or with the G2 API, click Export on the right side of the Properties dialog. This will export the model and its associated preprocessor. You can also use the Export button to export the model and its input/output information into an XML file, which can be uploaded into a GNNE Predictive Model object. To export a model for use with NeurOn-Line Classic, select Export Weights. This will save the model in a text file containing the weights only, in a format that can be uploaded into a NOL Classic block. For more information on exporting and deploying a trained model, see Model Deployment.

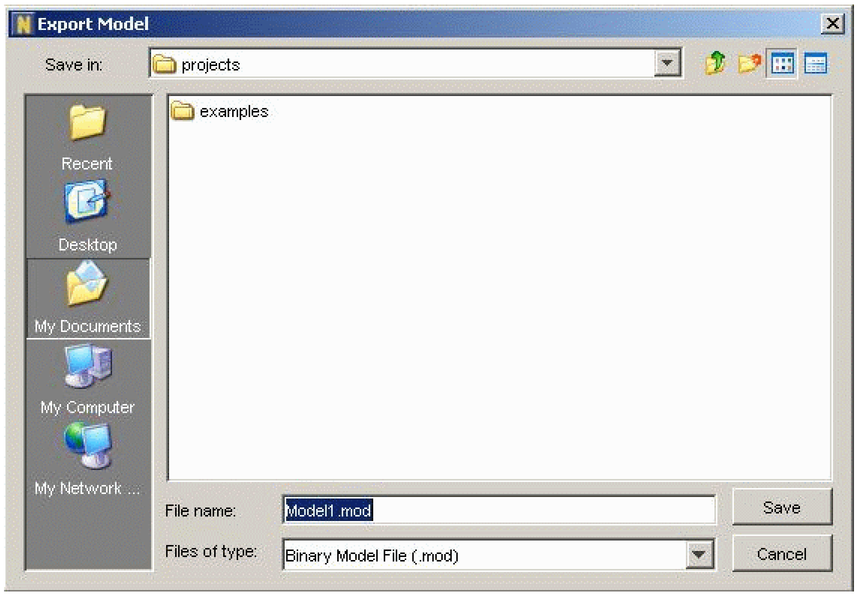
To export the model:

1 Click the Export button on the Model Properties workspace:



Name: Model1
Type: Predictive Model
Preprocessor: Preprocessor1
Trained? Yes
Model Rate: Good

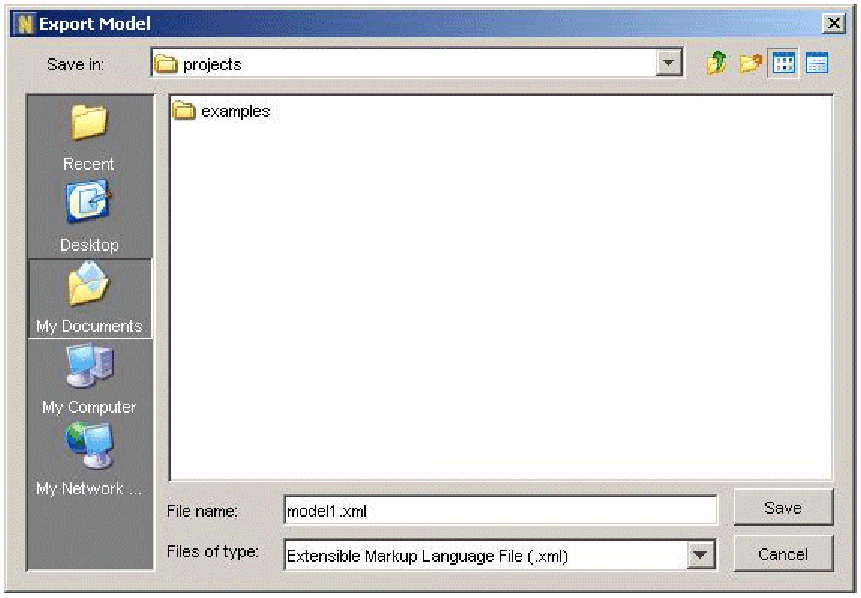
2 Specify a path and file name in the dialog:



Save in: projects
File name: Model1.mod
Files of type: Binary Model File (.mod)

Specify the file extension as *.mod* to save the model in a *.mod* file.

3 To export the model and its variable information into an XML file, specify the file extension as `.xml`:



File name: model1.xml

Files of type: Extensible Markup Language File (.xml)

For more information on using XML files for GNNE Predictive Models, see the *Gensym Neural Network Engine*.

Your model is now ready for deployment.

To export the model weight:

1 Click Export Weight on the Model Properties workspace.

2 Specify a path and file name in the dialog. The model weight will be saved in a `.txt` file.

Your model weight is now ready for deployment in NOL Classic as the weight for an ensemble model.

Note You can export to a text file a single linear model weight, then load it into the NOL Classic environment as an ensemble model, or into the NOL Studio deployment environment as a Partial Least Squares (PLS) model. For details, see Model Deployment.

Performing Simulations with a Trained Model

There are two primary ways to validate a model within NOL Studio:

- Using an existing data series. See Validating a Model Against Another Data Series.

- Using a simulation.

  Simulations allow you to specify some data to input to a model, inspect the output generated from that data, and then save the results. To generate a range of values for use by the simulation, you specify the range (min to max) of values and x number of increments for a given variable. NOL Studio will create a series of x+1 inputs with the variable incremented by (max - min)/x. The other variables will remain constant. For example, if you specified the minimum value to be 5, the maximum to be 10, and the number of increments to be 5, NOL Studio would generate the following range of values: 5, 6, 7, 8, 9, 10.

NOL Studio can display the output data from a simulation in two different formats:

- A spreadsheet view.
- A line chart.

Creating a New Simulation

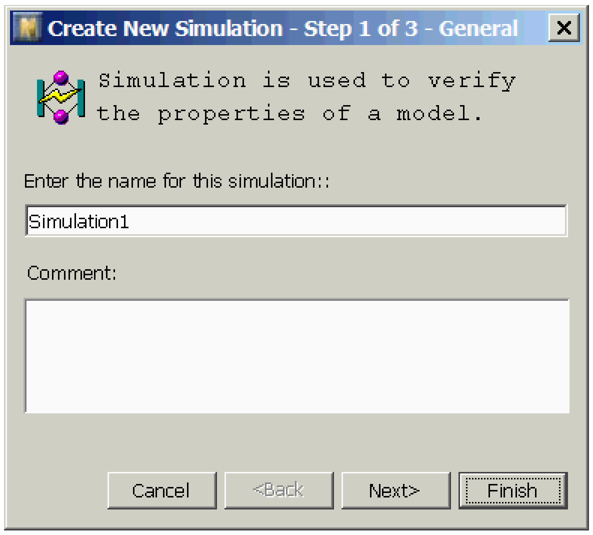
To create a new simulation:

1 Do one of the following:

Menu Bar:	Choose Object > New > Simulation.
Toolbar:	Click the Simulate button.
Tree View:	Right-click Simulation and choose New from its menu.

The Simulation wizard appears to guide you through the process of creating a new simulation.

2 Choose a name for the new simulation, for example:

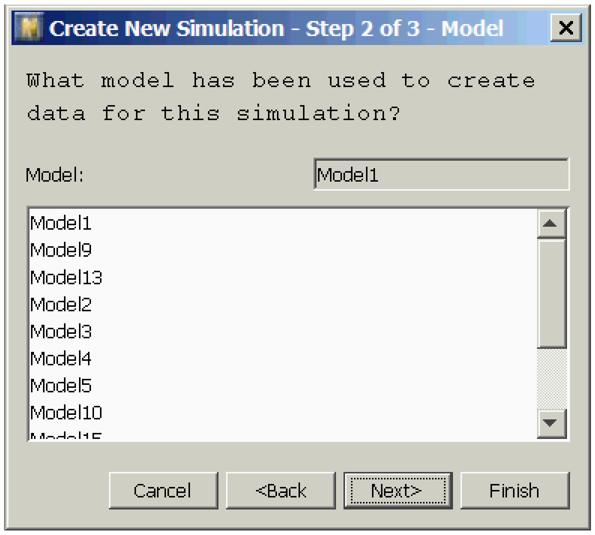

Create New Simulation - Step 1 of 3 - General

Simulation is used to verify the properties of a model.

Enter the name for this simulation:: Simulation1

Comment:

3 Select the model from the list that you want to test, for example:


Create New Simulation - Step 2 of 3 - Model

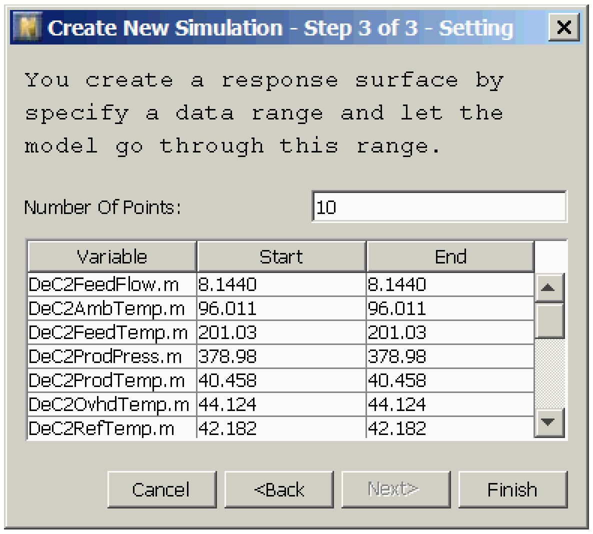
What model has been used to create data for this simulation?

Model: Model1

Model1
Model9
Model13
Model2
Model3
Model4
Model5
Model10

4 Tell NOL Studio how many points you want it to create for the surface calculation by typing in the edit box. Set the Start and End values by typing values in the spreadsheet.

For example:



Create New Simulation - Step 3 of 3 - Setting

You create a response surface by specify a data range and let the model go through this range.

Number Of Points: 10

Variable	Start	End
DeC2FeedFlow.m	8.1440	8.1440
DeC2AmbTemp.m	96.011	96.011
DeC2FeedTemp.m	201.03	201.03
DeC2ProdPress.m	378.98	378.98
DeC2ProdTemp.m	40.458	40.458
DeC2OvhdTemp.m	44.124	44.124
DeC2RefTemp.m	42.182	42.182

5 Click Finish to create the simulation.

The simulation is saved with the Project.

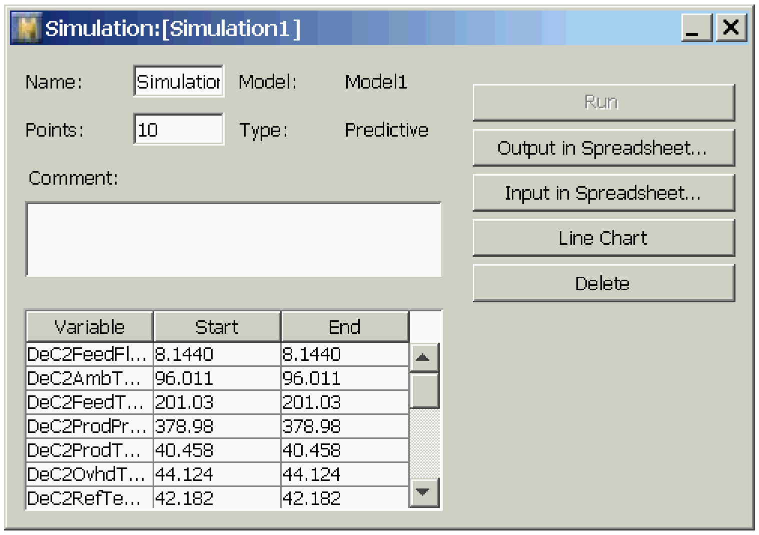
You can now view the results of the simulation in a spreadsheet or plots of the response surface and/or the Predicted vs. Actual by displaying the main simulation dialog as follows:

- Double-click on its entry in the tree view.

  or

- Choose Object > Go to > Simulation from the menu bar.

Here is the property table for the Simulation object:



Simulation:[Simulation1]

Name: Simulatior
Model: Model1
Points: 10
Type: Predictive
Comment:

Variable	Start	End
DeC2FeedFl...	8.1440	8.1440
DeC2AmbT...	96.011	96.011
DeC2FeedT...	201.03	201.03
DeC2ProdPr...	378.98	378.98
DeC2ProdT...	40.458	40.458
DeC2OvhdT...	44.124	44.124
DeC2RefTe...	42.182	42.182

Displaying Simulation Results

When you run a simulation, two new data series are created, representing the input and output of the simulation. These data series can be visualized in a chart or spreadsheet, using the techniques described in Chapter 3   Visualizing Data.

You can also display the results, with slightly less navigation, using the action buttons on the property table for the simulation, shown above.

Creating a Backpropagation Net

Describes how to create a backpropagation net.

Introduction

The Backpropagation Net, or BPN, is another type of predictive model. It is a feed-forward, layered network. Each node in a layer is connected to all other nodes in the layer before it and in the layer after it. It is especially useful for modeling multivariate functions.

Similar to predictive models, you can start building BPN models after importing data, labeling and filtering data by using a preprocessor, and creating formulas that condition the data in the same preprocessor. This chapter describes how you set up and train a BPN.

NOL Studio helps you make modeling decisions, such as the best combination of inputs and time delays. You need to manually select the architecture of a Backpropagation Network model before training.

One NOL Studio project can contain any number of BPN models. This allows you to train models with different architectures for the same problem, then to compare models, using the validation tools until you are completely satisfied with the performance of your model or models. You can then save out your best model or models for online deployment.

Creating Backpropagation Net Models

To create a BPN model, you follow the steps in the modeling wizard. The wizard guides you through these steps to create a model:

1. Name the model.
2. Select whether to use old model parameters.
3. Specify the preprocessor for the model.
4. Specify the output data series to be used in the model.
5. Classify the variables as input, output, or unused.
6. Specify time delays, if any, for the model inputs,
7. Automatically select inputs and delays.
8. Specify the model architecture.

These steps are detailed in the following sections.

To launch the wizard:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Backpropagation Net.
Toolbar:	Click the New Backpropagation Net button.
Tree View:	Right-click the Backpropagation Nets node and choose New from its menu.

All of these actions display the model wizard.

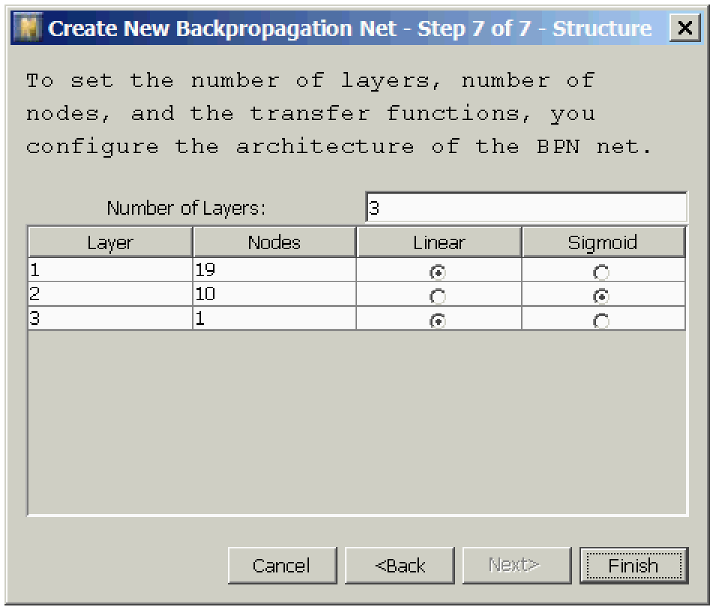
Note After the wizard is displayed, follow the first seven steps in the procedure summarized in Creating a Predictive Model.

Specifying the Model Architecture

After you finish defining the input and output structure, the last step is to specify the internal architecture of the BPN model. The number of nodes in the first layer and the number of input variables must be the same. The number of nodes in the last layer and the number of output variables must also be the same. The hidden or intermediate layers (layers between the first and last layers) can be any size. You can have up to three hidden layers, for a total of up to five layers. In general, a network has one hidden layer. The number of nodes depends on the complexity of the function that the network has to model. The more complex the function, the more nodes needed.

You can choose whether a layer uses the sigmoidal or linear function for its nodes. In general, the input and output layers use the linear function, and at least one of the hidden layers use a sigmoidal function.

When you get to this step, the wizard shows following dialog:



To set the number of layers, number of nodes, and the transfer functions, you configure the architecture of the BPN net.

Number of Layers: 3

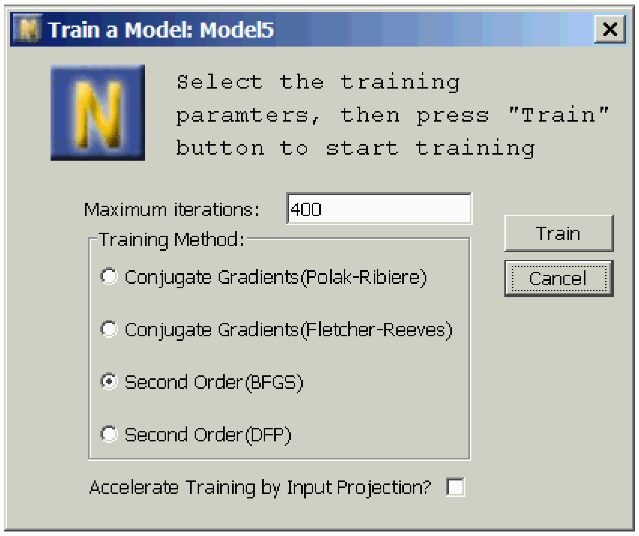
Layer	Nodes	Linear	Sigmoid
1	19	●	○
2	10	○	●
3	1	●	○

First, specify the number of layers, then press Return. You can change the number of nodes for each hidden layer. The numbers of nodes for the first and last layer are read only and are defined by previous steps. You can select the layer function by choosing the sigmoidal or linear function button for each line.

After specifying the details of the architecture, click Finish.

The Training Console

When you exit the wizard, the following dialog appears, prompting you to select a maximum number of iterations, training method and whether you want to accelerate training by input projection.


Train a Model: Model5

Select the training paramters, then press "Train" button to start training

Maximum iterations: 400

Training Method:

( ) Conjugate Gradients(Polak-Ribiere)

( ) Conjugate Gradients(Fletcher-Reeves)

(●) Second Order(BFGS)

( ) Second Order(DFP)

Accelerate Training by Input Projection? ☐

Train

Cancel

Choosing the Maximum Number of Iterations

The training algorithm improves the weights in a number of individual steps. To specify the maximum number of steps, enter a number in the Maximum Iterations attribute. Typically, values range from 50 to as much as 1000. If you are not satisfied with the trained model, you can Continue Training button in the model property dialog, the training algorithm continues the training from where it stopped. Continuing Training.

Choosing the Training Method

To choose how to train the network, select one of the options below Training Method. The options fall into two main categories:

- Conjugate Gradient options: Conjugate Gradients (Fletcher-Reeves) and Conjugate Gradients (Polak-Ribiere).
- Second Order options: BFGS (Broyden-Fletcher-Goldfarb-Shanno) and DFP (Davidon-Fletcher-Powell).

In general, the Second Order options are more powerful and use significantly more memory, and the Conjugate Gradient options use less memory and take less time per step. The Conjugate Gradient options are generally better for larger networks (over 100 weights), and the Second Order options are better for smaller networks.

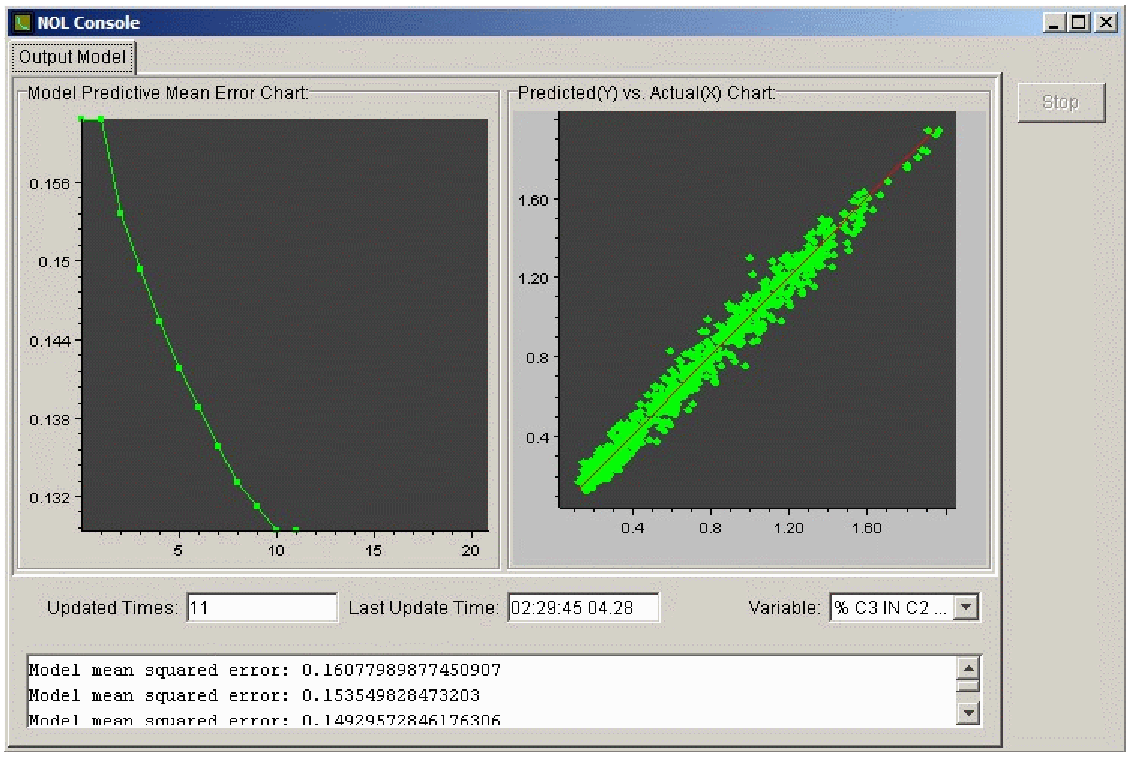
Once you choose which category of training methods to use, you may want to experiment with the different methods in each category to find out which is best for your network.

Note Some neural network packages allow you to choose fixed or scheduled “learning rates” and “momentum factors.” NOL Studio automatically optimizes the training parameters and does not require you to configure these option. The parameters that NOL Studio chooses are always better than those chosen by hand.

Choosing Whether to Accelerate Training

The Accelerate Training by Input Projection option can speed up the training time. Select it if your input data has columns that might be correlated. To speed up training, the block projects your input data vector to a vector with fewer dimensions, trains with the smaller vector, and projects the smaller vector and its training results backwards to obtain results useful for the original vector. In general, this option is recommended if you have more than ten inputs.

Once the training parameters have been selected, the training process begin. Training is monitored through a special window, called the training console.



Updated Times: 11 Last Update Time: 02:29:45 04.28 Variable: % C3 IN C2 ...

```
Model mean squared error: 0.16077989877450907
Model mean squared error: 0.153549828473203
Model mean squared error: 0.14929572846176306
```

The chart on the left shows the current fitting error on training data. The chart on the right shows a plot of predicted versus actual fit for the output variable specified in the selection box in the lower right. To view predicted versus actual for another variable, change the selection in the selection box. The red line in the predicted versus actual plot represents the perfect model, where the predicted value generated by the neural network model exactly equals the training target value. Because of measurement noise, points are usually scattered around the target line.

Below the plots, the two edit boxes show how many times the model is updated, based on the minimum training cycle, and the time when the update occurs. The training cycle is defined internally, based on the model architecture.

Training will terminate when the maximum number of iterations is reached. Alternatively, you can terminate training at any time by clicking the Stop button on the console. To continue training, click the Continue Training button on the Model Properties dialog. You navigate to the Model Properties dialog via the tree view.

Preparing the Training Set

When you begin training, NOL Studio prepares a training set according to your specified variables and delays. The method to prepare the training set for all models is the same. See Preparing the Training Set.

Viewing the Model Properties

When you finish the new model wizard, a backpropagation network object is created and added to the tree view. You can view the properties dialog for any model by double-clicking the corresponding tree view node, or by choosing Object > Go To > Models > Backpropagation Net.

General Properties

The General Properties tab on the model properties dialog gives general information about the model. This is the same as for the predictive model. See General Properties.

Brief Information of Model Performance

The ratings of "Good", "OK", and "Need Improvement" for a backpropagation network are based on the same criteria as for the predictive model. See Brief Information of Model Performance.

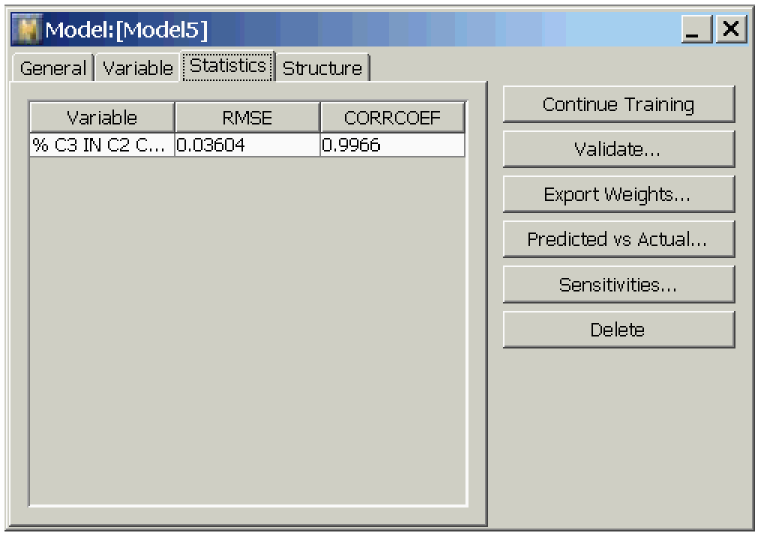
Model Variables

The Variables tab on the model properties dialog summarizes the classification of variables and delays for each input and output for this model. See Model Variables.

Statistics

The Statistics tab provides performance statistics for the trained BPN model. During BPN training, all data is used for training. The statistics show how well the model fits the training data set. You can use other data series to perform the validation of this BPN model to see the model performance on new data.

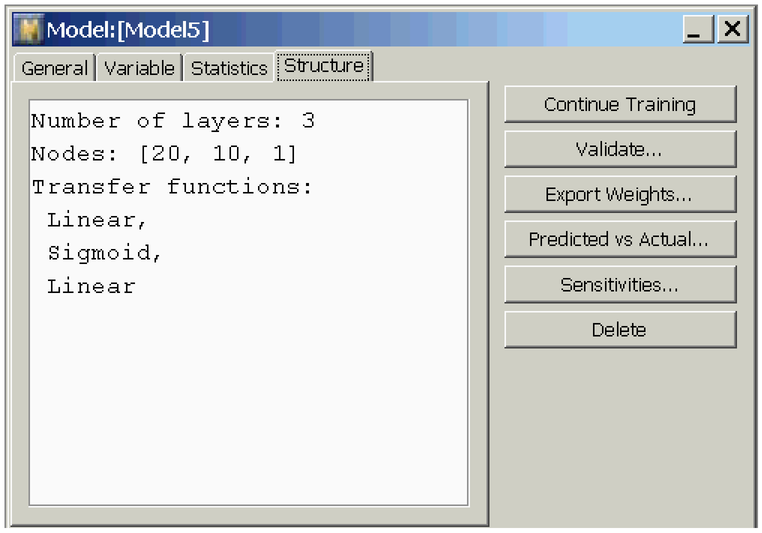
Here is the statistics panel:



Variable	RMSE	CORRCOEF
% C3 IN C2 C...	0.03604	0.9966

Model Structure

The Structure tab shows the structure of the backpropagation network model. This gives you a chance to review and verify the models created by NOL Studio. For example:



```
Number of layers: 3
Nodes: [20, 10, 1]
Transfer functions:
 Linear,
 Sigmoid,
 Linear
```

To see the internal weights and other details, you must export the model as a text file, using the Export Weights button.

Performing Operations on the Model

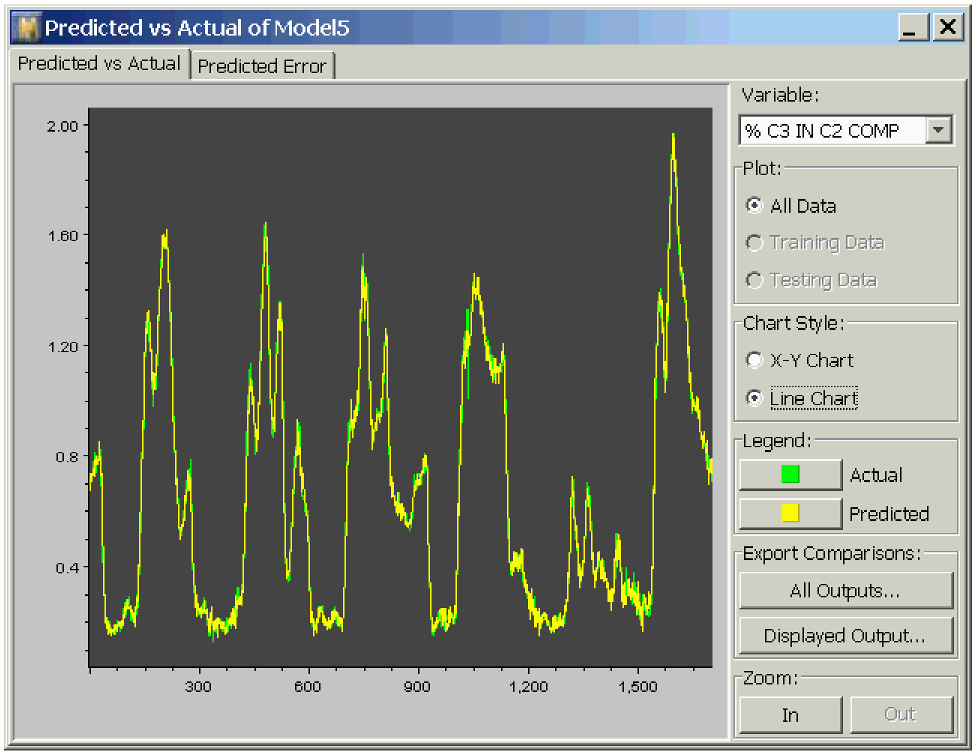
This section describes the operations you can perform on the model, using the buttons along the right side of the model properties dialog.

Continuing Training

To continue training this model, click the Continue Training button. This will launch the model training parameter dialog, as described in The Training Console, and allow you to initiate further training. The training algorithm will start the training at the place where it is stopped last time.

Showing the Predicted Versus Actual Plot

To show a plot of predicted versus actual values attained during training, click the Predicted vs. Actual button to show this dialog:



In this dialog, you can show the prediction of any output versus the training target values of that output, as a line chart (shown above), or as an x-y (scatter) chart. You change outputs by selecting from the Variable drop-down list. You change chart formats by selecting the Chart Style radio buttons. To change colors on the plot, select the appropriate button in the Legend section.

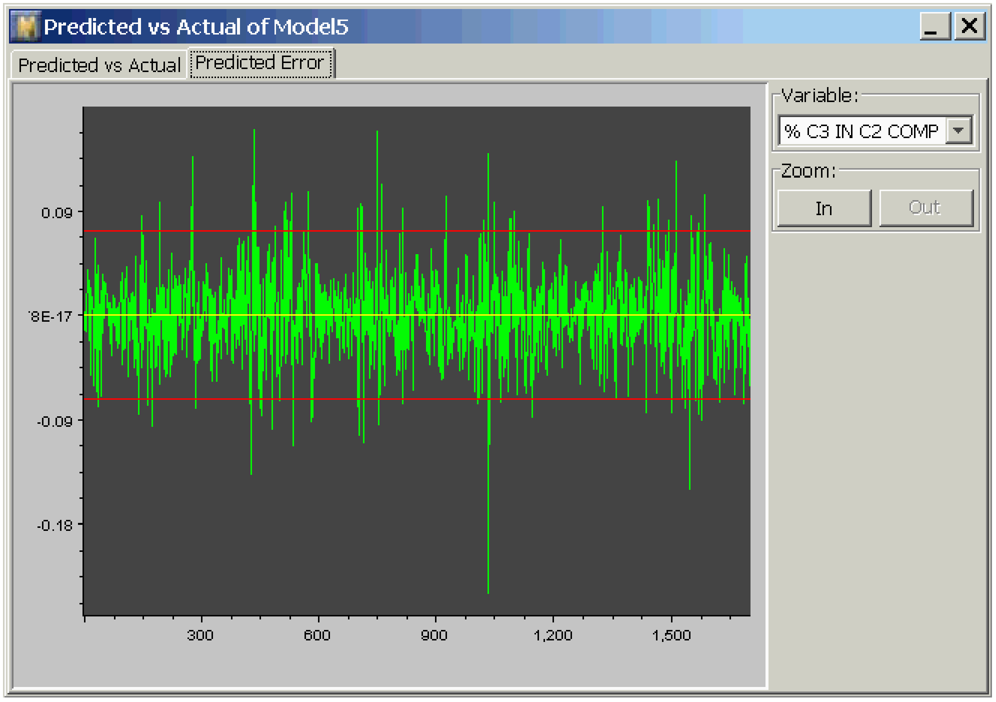
Exporting Predictions

You can save the model predictions to a file, using the Export Comparisons button. The data that is saved reflects the predictions as seen on the screen when you perform this action. For example, if you are displaying the predictions for the test data subset when you export the predictions, only selected rows of the data will be written. The "gaps"—corresponding to samples not chosen for the test set—will be written as not-a-number (NaN). When you click the All Outputs button, you save as many columns of data as there are outputs in the model. The Displayed Outputs button saves only the currently displayed output. Timestamps are included in the output, regardless of the option selected. The actual output values corresponding to the specified predictions are also included in the exported file. In this way, you can make comparisons between exported files.

When writing the file, you can choose either binary (`.bds`) or text (`.ds`) formats. You specify which option by specifying the file extension in the Save File dialog. A example format of the text file is shown below:

```
Time              10A100C3.PV_pred        10A100C3.PV
Time              % C3 IN C2 COMP_pred    % C3 IN C2 COMP
Millisecond       None                    None
      8.71964E+11             1.225174967             1.385540366
      8.71964E+11             1.286989442             1.385540247
      8.71964E+11             1.366936914             1.385540247
      8.71964E+11             1.374557197             1.382564783
      8.71965E+11             1.393119682             1.382488489
      8.71965E+11             1.409462661             1.367229223
      8.71965E+11             1.417492178             1.412197351
      8.71966E+11             1.439030908              1.42521441
      8.71966E+11             1.466799704              1.42521441
```

Viewing the Predicted Error



In the Predicted Error dialog, you can show the prediction error of any output as a line chart, as shown above. You change outputs by selecting from the Variable dropdown list. In the chart, the 95% upper and lower limits for the predicted error are shown in red.

Zooming

All of the charts in this predicted vs. actual dialog allow you to interactively zoom in/out and scroll through your data. When you zoom in, the location of the point in the center of the plot does not change. Thus, if you want to zoom in on a particular point, scroll the plot horizontally until the point is shown in the center of the screen, then click the Zoom In button. When you zoom in, plot symbols representing the individual data points appear. In a line chart, you cannot magnify the y axis by zooming, whereas in a scatter chart you can.

Validating a Model Against Another Data Series

You can validate a BPN model against a new data series. The validation process is the same as with a predictive model. For details, see Validating a Model Against Another Data Series.

Viewing Input-Output Sensitivities

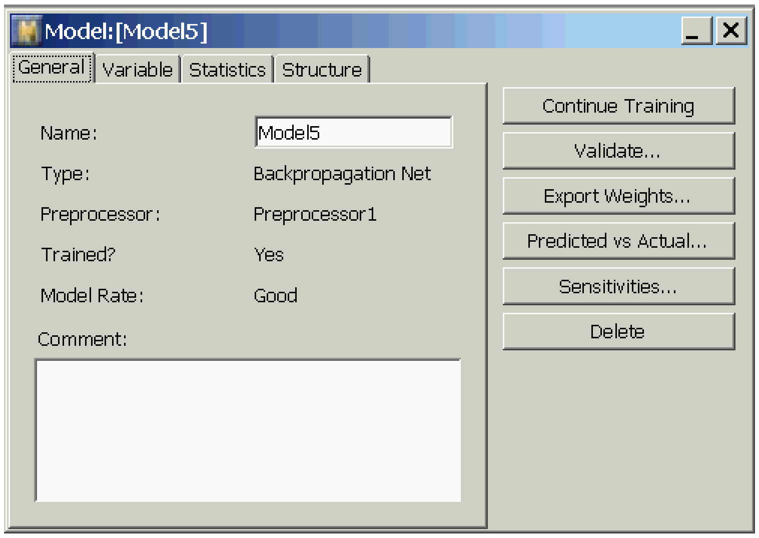
To see the sensitivities, click the Sensitivities button on the model properties dialog. You can analyze the sensitivities by clicking the buttons in the sensitivity dialog. See Input-Output Sensitivities.

Exporting a Model

To deploy a backpropagation network in Gensym Neural Net Engine (GNNE) or NeurOn-Line (NOL) Classic, you must save the weights of the backpropagation network to a text file. You can also directly export the model parameters to a BPN object in GNNE. For more information on deploying a trained BPN model, see Model Deployment.

To export the model weights:

1 Click the Export Weight button in the Model Properties dialog:



Name: Model5
Type: Backpropagation Net
Preprocessor: Preprocessor1
Trained? Yes
Model Rate: Good

2 Specify a path and file name in the dialog. The model weight will be saved in a `.txt` file.

Your model weight is now ready for deployment in GNNE and NOL Classic.

Performing Simulations with a Trained Model

You can validate the backpropagation network model by using simulations in NOL Studio. Simulations allow you to specify some data as inputs to a model, inspect the output generated from that data, then save and analyze the results. For details, see Performing Simulations with a Trained Model.

Creating an Autoassociative Net

Describes how you create an autoassociative network.

Introduction

The Autoassociative Network model, or ANN, is a type of Backpropagation network that uses autoassociative mappings. It is a feed-forward, layered network. Each node in a layer is connected to all other nodes in the layer before it and in the layer after it. In general, the input and output vectors are targeted to the same process variables. It is especially useful for sensor validation problems.

Similar to predictive models, you can start building ANN models after importing data, labeling and filtering data by using a preprocessor, and creating formulas that condition the data in the same preprocessor. This chapter describes how you set up and train an ANN model.

NOL Studio helps make modeling decisions, such as selecting variables. You need to define the architecture of an Autoassociative Network model.

One NOL Studio project can contain any number of ANN models. This allows you to train models with different architectures for the same problem, then to compare models, using the validation tools until you are completely satisfied with the performance of your model or models. You can then save out your best model or models for online deployment.

Creating Autoassociative Net Models

To create an ANN model, you follow the steps in the modeling wizard. The wizard guides you through these steps to create a model:

1. Name the model.
2. Select whether to use old model parameters.
3. Specify the preprocessor for the model.
4. Specify the output data series to be used in the model.
5. Classify the variables as input, output, or unused.
6. Specify the model architecture.

These steps are detailed in the following sections.

To launch the wizard:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Autoassociative Net.
Toolbar:	Click the New Autoassociative Net button.
Tree View:	Right-click the Autoassociative Nets node and choose New from its menu.

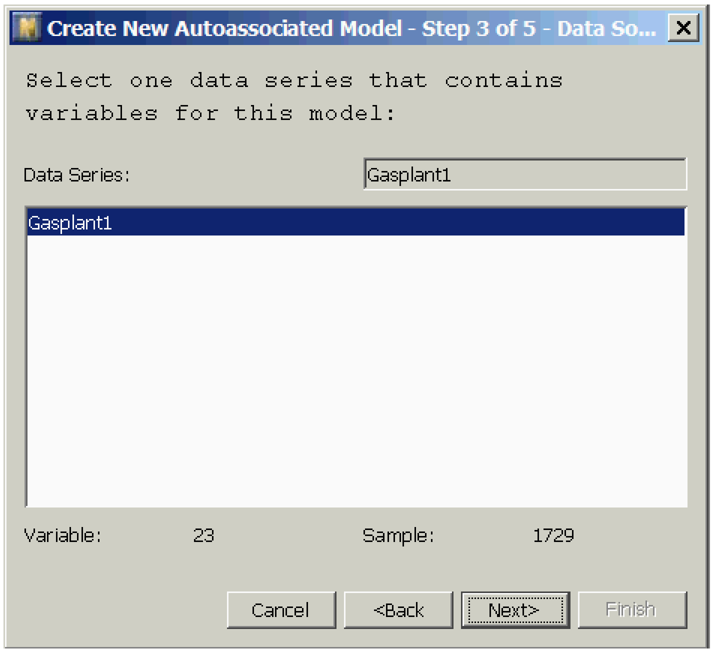
All of these actions display the model wizard.

Note After the wizard is displayed, follow the first three steps in the procedure summarized in Creating a Predictive Model.

Selecting the Data Series

After you select the preprocessor for your model, you select the data series containing the variables you want to do the mapping. Only variables inside the selected data series can be used in your model. That also means you can only use one selected data series to build an ANN model. If you want to model variables from different data series, you must create different models.

To select data series, click the desired data series, for example:



Select one data series that contains variables for this model:

Data Series: Gasplant1

Gasplant1

Variable: 23 Sample: 1729

Classifying Variables

In the fourth step of the wizard, you specify which variables are to be used in the model, and which variables are not used. The list of variables include all variables in the selected data series, including derived variables defined by the preprocessor formulas. For more information, see Classifying Variables.

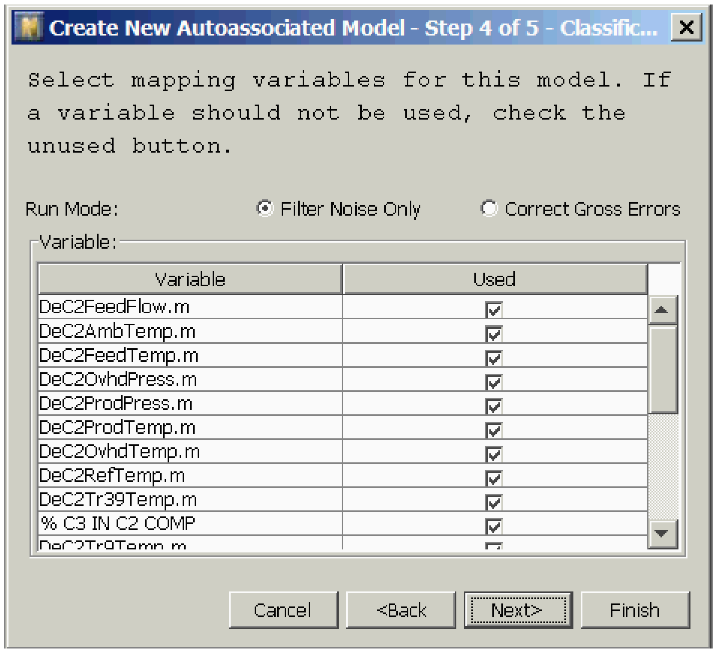
Defining the Run Mode

You also define the running mode of the ANN model in the fourth step. The Run Mode attribute lets you choose whether the network replaces faulty input values.

If you choose the Filter Noise Only option, the network does not perform the replacement. When you run the network, it performs a single forward pass, which filters random errors from the inputs but not systematic errors (or biases).

If you choose the Correct Gross Errors option, the network does perform the replacement. When you run the network, it performs n+1 passes, where n is the number of elements in the input vector. The first pass is the same as the pass used for the Filter Noise Only option. In the rest of the passes, one of the input values is ignored, and the network computes the best replacement value. Using the standard deviation for the input value, the network computes how far off the input value is from its replacement value. The network then replaces the input value that is furthest from its replacement value.

To select the variables, check the appropriate radio buttons, for example:


Create New Autoassociated Model - Step 4 of 5 - Classific...

Select mapping variables for this model. If a variable should not be used, check the unused button.

Run Mode: [✓ Filter Noise Only] Correct Gross Errors

Variable:

Variable	Used
DeC2FeedFlow.m	✓
DeC2AmbTemp.m	✓
DeC2FeedTemp.m	✓
DeC2OvhdPress.m	✓
DeC2ProdPress.m	✓
DeC2ProdTemp.m	✓
DeC2OvhdTemp.m	✓
DeC2RefTemp.m	✓
DeC2Tr39Temp.m	✓
% C3 IN C2 COMP	✓

Cancel
<Back
Next>
Finish

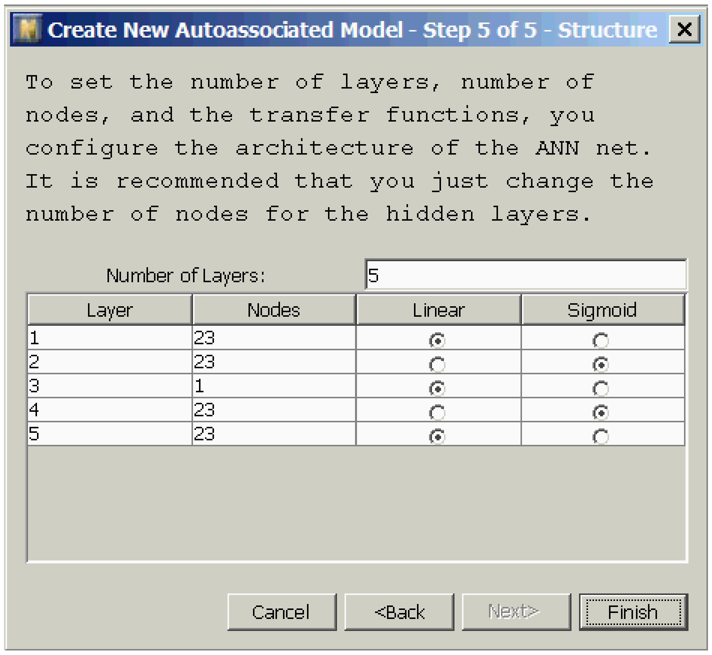
Note There is no time delay for Autoassociative Net models. The time-based data series and row-based data series are treated the same when preparing the training data set for ANN model training.

Specifying the Model Architecture

After you finish selecting the variables, the next step is to specify the internal architecture of the ANN model. Normally, an Autoassociative network contains five layers. The first layer and the last layer must be the same size. Both have the same number of nodes as the number of the selected variables. The hidden or intermediate layers (layers between the first and last layers) can be any size. Usually, an Autoassociative Network has three hidden layers. The middle layer, or bottleneck layer, must have fewer nodes than any other.

You can choose whether a layer uses the sigmoidal or linear function for its nodes. In general, the input, output, and bottleneck layers use the linear function, and the rest use the sigmoid function.

When you get to this step, the wizard shows following dialog:



```
To set the number of layers, number of
nodes, and the transfer functions, you
configure the architecture of the ANN net.
It is recommended that you just change the
number of nodes for the hidden layers.
```

Number of Layers: 5

Layer	Nodes	Linear	Sigmoid
1	23	●	○
2	23	○	●
3	1	●	○
4	23	○	●
5	23	●	○

You first specify the number of layers, then press Return. You can change the number of nodes for each hidden layer. The numbers of nodes for the first and last layer are read only and are defined by the previous steps. You can select the layer function by choosing the sigmoidal or linear function button for an individual line.

After specifying the details of the architecture, click Finish.

The Training Console

The training console for ANN models is the same console as for BPN models. The strategies to select maximum number of iterations, training method, and whether you want to use acceleration for training are all the same as with BPN models. For details, see The Training Console.

Viewing the Model Properties

When you finish the new model wizard, an autoassociative network object is created and added to the tree view. You can view the properties dialog for any model by double-clicking the corresponding tree view node, or by choosing Object > Go To > Models > Autoassociative Net.

General Properties

The General Properties tab on the model properties dialog gives general information about the model. This is the same as for the predictive model. See General Properties.

Brief Information of Model Performance

The ratings of “Good”, “OK”, and “Need Improvement” for an autoassociative network are based on the same criteria as for the predictive model. See Brief Information of Model Performance.

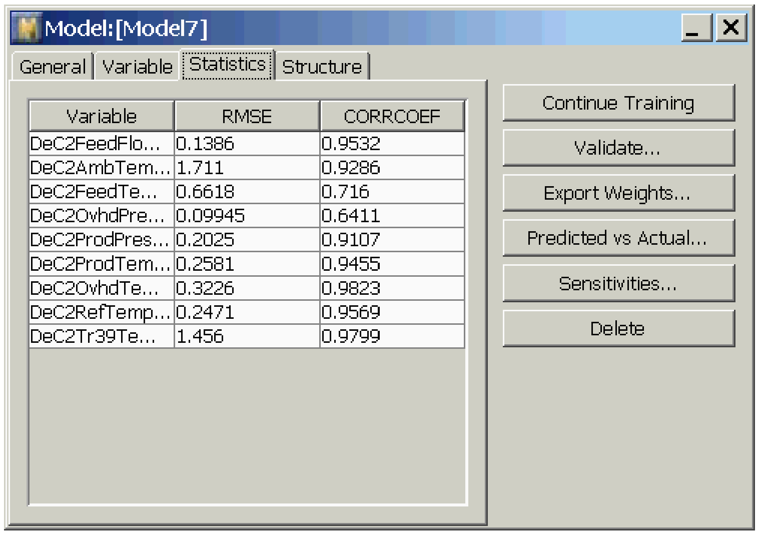
Model Variables

The Variables tab on the model properties dialog summarizes the classification of variables and delays for each input and output for this model. See Model Variables.

Statistics

The Statistics tab provides performance statistics for the trained ANN model. During ANN training, all data is used for training. The statistics show how well the model fits the training data set. You can use other data series to perform the validation of this ANN model to see the model performance on new data.

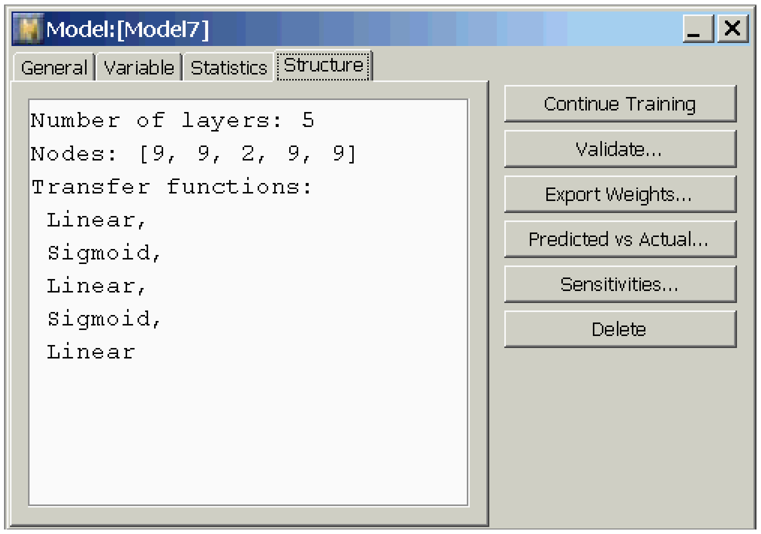
The statistics panel is shown below:



Variable	RMSE	CORRCOEF
DeC2FeedFlo...	0.1386	0.9532
DeC2AmbTem...	1.711	0.9286
DeC2FeedTe...	0.6618	0.716
DeC2OvhdPre...	0.09945	0.6411
DeC2ProdPres...	0.2025	0.9107
DeC2ProdTem...	0.2581	0.9455
DeC2OvhdTe...	0.3226	0.9823
DeC2RefTemp...	0.2471	0.9569
DeC2Tr39Te...	1.456	0.9799

Model Structure

The Structure tab shows the structure of the autoassociative network model. This gives you a chance to review and verify the models created by NOL Studio. For example:



```
Number of layers: 5
Nodes: [9, 9, 2, 9, 9]
Transfer functions:
 Linear,
 Sigmoid,
 Linear,
 Sigmoid,
 Linear
```

To see the internal weights and other details, you must export the model as a text file, using the Export Weights button.

Performing Operations on the Model

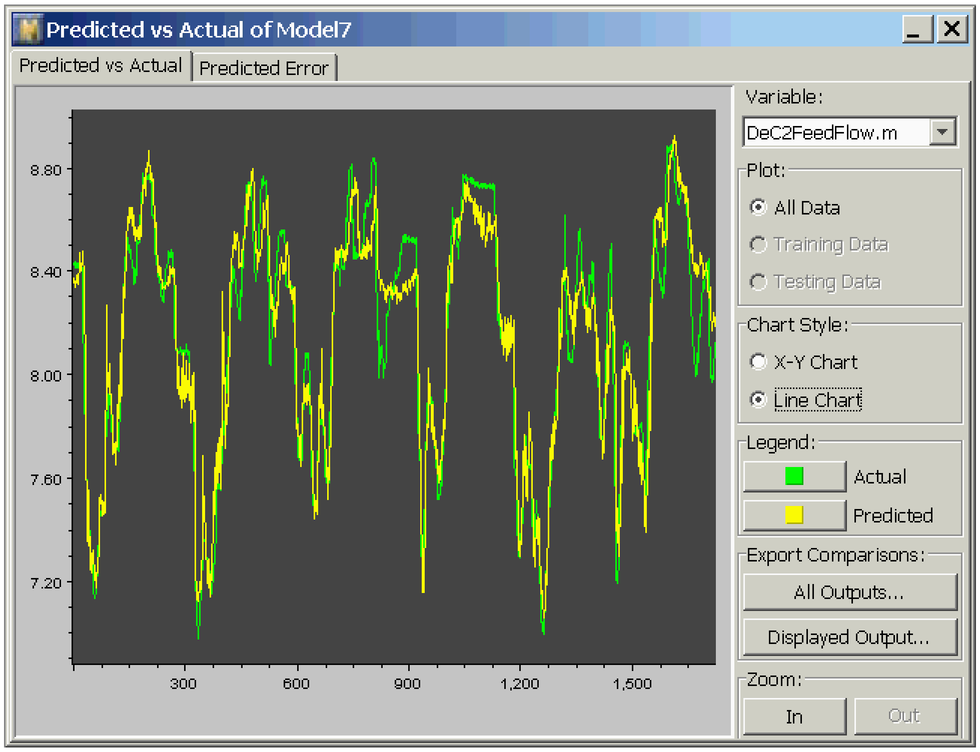
This section describes the operations you can perform on the model, using the buttons along the right side of the model properties dialog.

Continuing Training

To continue training this model, click the Continue Training button. This will launch the model training parameter dialog, as described in The Training Console, and allow you to initiate further training. The training algorithm will start the training at the place where it is stopped last time.

Showing the Predicted Versus Actual Plot

To show a plot of predicted versus actual values attained during training, click the Predicted vs. Actual button to show this dialog:

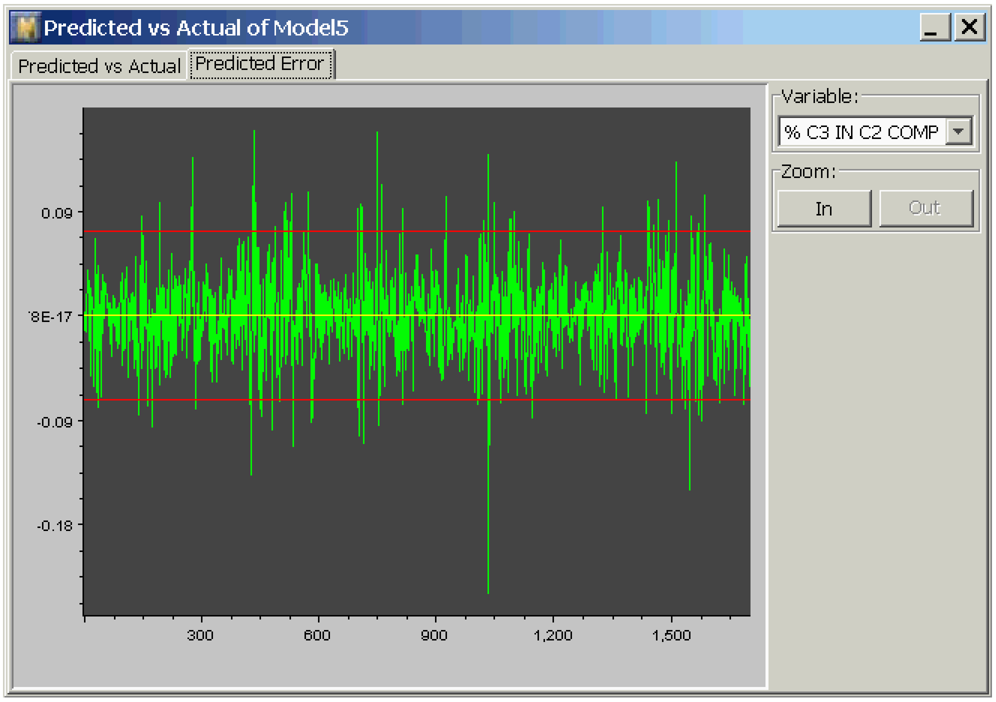


In this dialog, you can show the prediction of any output versus the training target values of that output, as a line chart (shown above), or as an x-y (scatter) chart. You change outputs by selecting from the Variable drop-down list. You change chart formats by selecting the Chart Style radio buttons. To change colors on the plot, select the appropriate button in the Legend section.

Exporting Predictions

You can save the model predictions to a file, using the Export Comparisons button, which is the same as for a backpropagation network. For details, see Exporting Predictions.

Viewing the Predicted Error



In the Predicted Error dialog, you can show the prediction error of any output as a line chart, as shown above. You change outputs by selecting from the Variable dropdown list. In the chart, the 95% upper and lower limits for the predicted error are also shown in red.

Zooming

All of the charts in the predicted vs. actual dialog allow you to interactively zoom in/out and scroll through your data. When you zoom in, the location of the point in the center of the plot does not change. Thus, if you want to zoom in on a particular point, scroll the plot horizontally until the point is shown in the center of the screen, then select the zoom in button. When you zoom in, plot symbols representing the individual data points appear. In a line chart, you cannot magnify the y axis by zooming, whereas in a scatter chart you can.

Validating a Model Against Another Data Series

You can validate an ANN model against a new data series. The validation process is the same as with a predictive model. For details, see Validating a Model Against Another Data Series.

Viewing Input-Output Sensitivities

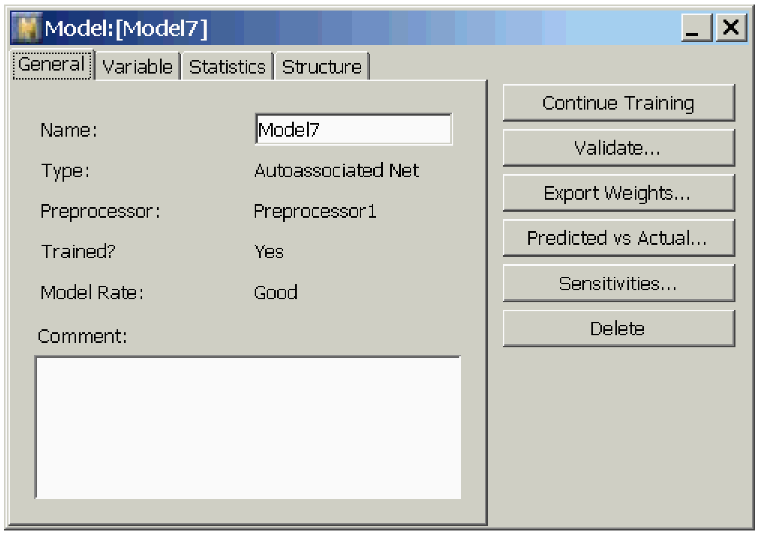
To see the sensitivities, click the Sensitivities button on the model properties dialog. You can analyze the sensitivities by clicking the buttons in the sensitivity dialog. See Input-Output Sensitivities.

Exporting a Model

To deploy an autoassociative network in Gensym Neural Net Engine (GNNE) or NeurOn-Line (NOL) Classic, you must save the weights of the autoassociative network to a text file. You can also directly export the model parameters to an autoassociative network object in GNNE. For more information on deploying a trained AAN model, see Model Deployment.

To export the model weights:

1 Click Export Weight on the Model Properties dialog:



Name: Model7
Type: Autoassociated Net
Preprocessor: Preprocessor1
Trained? Yes
Model Rate: Good
Comment:

2 Specify a path and file name in the dialog. The model weight will be saved in a `.txt` file.

Your model weight is now ready for deployment in GNNE and NOL Classic.

Performing Simulations with a Trained Model

You can validate the autoassociative network model by using simulations in NOL Studio. Simulations allow you to specify some data as inputs to a model, inspect the output generated from that data, then save and analyze the results. For details, see Performing Simulations with a Trained Model.

Creating a Radial Basis Function Net

Describes how you create a Radial Basis Function network

Introduction

The Radial Basis Function Network, or RBFN, is a 3-layer, feed-forward network, whose middle layer uses a multi-variate Gaussian function. It is especially useful for classification problems. The RBFN is best for choosing which class out of many classes an item belongs to.

Similar to predictive models, you can start building RBFN models after importing data, labeling and filtering data by using a preprocessor, and creating formulas that condition the data in the same preprocessor. This chapter describes how you set up and train an RBFN model.

NOL Studio helps make modeling decisions, such as selecting variables. You need to define the architecture of an Autoassociative Network model.

One NOL Studio project can contain any number of RBFN models. This allows you to train models with different architectures for the same problem, then to compare models, using the validation tools until you are completely satisfied with the performance of your model or models. You can then save out your best model or models for online deployment.

Creating Radial Basis Function Net Models

To create a RBFN model, you follow the steps in the modeling wizard. The wizard guides you through these steps to create a model:

1. Name the model.
2. Select whether to use old model parameters.
3. Specify the preprocessor for the model.
4. Specify the output data series to be used in the model.
5. Classify the variables as input, output, or unused.
6. Specify time delays, if any, for the model inputs,
7. Automatically select inputs and delays.
8. Specify the model architecture.

These steps are detailed in the following sections.

To launch the wizard:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Radial Basis Function Net
Toolbar:	Click the New Radial Basis Function Net button.
Tree View:	Right-click the Radial Basis Function Nets node and choose New from its menu.

All of these actions display the model wizard.

Note After the wizard is displayed, follow the first seven steps in the procedure summarized in Creating a Predictive Model.

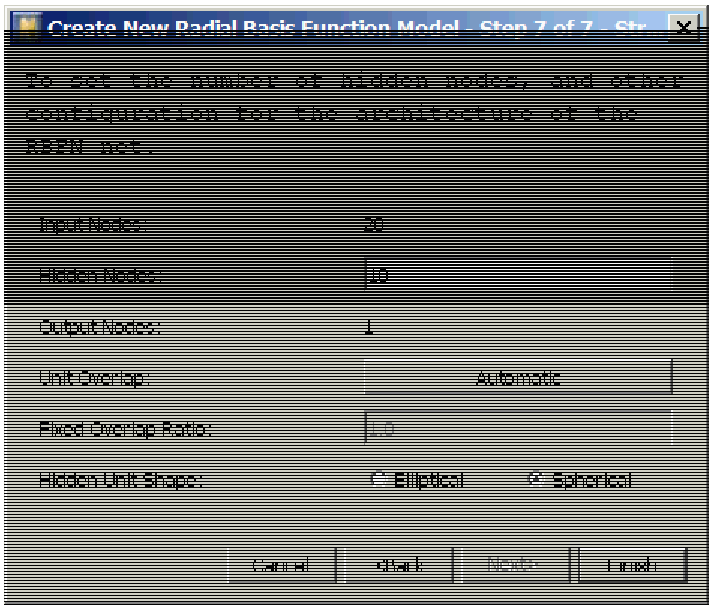
Specifying the Model Architecture

After you finish defining the input and output structure, the eighth step is to specify the internal architecture of the RBFN model. A RBFN model contains exactly three layers. The number of nodes in the first layer is the same as the number of input variables. The number of nodes of the last layer is the same as the number of output variables. The middle or hidden layer can have any number of nodes.

Each node in a layer is connected to all other nodes in the layers before it and after it. The connections between the input and hidden layers are unweighted. However, RBFNs weight the connections between the hidden layer and output layer normally, like a BPN or an Autoassociative network.

The transfer function of the input and output layer is linear. You can choose whether the transfer functions of the hidden layer are spherical or elliptical Gaussians.

When you get to this step, the wizard shows following dialog:



Create New Radial Basis Function Model - Step 7 of 7 - Str...

To set the number of hidden nodes, and other configuration for the architecture of the RBFN net.

Input Nodes: 20

Hidden Nodes: 10

Output Nodes: 1

Unit Overlap: Automatic

Fixed Overlap Ratio: 1.0

Hidden Unit Shape: Elliptical [✓ Spherical]

You can change the number of nodes for the hidden layer. The numbers of nodes for the first and last layer are read only and are defined by the previous steps.

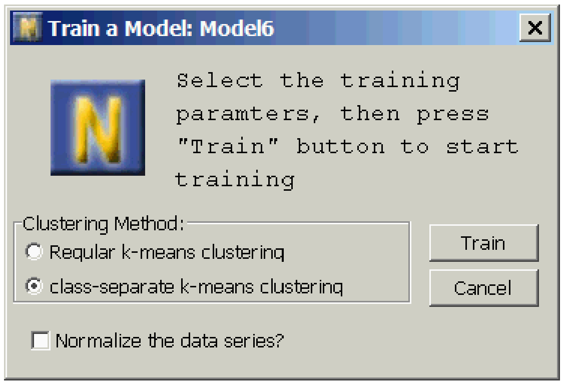
To set the unit overlap, choose whether the overlap is automatic or fixed by clicking the toggle button. If the overlap is automatic, the network chooses the best unit overlap for you automatically. Generally, you will use an automatic overlap. If the overlap is fixed, enter a positive value in the Unit Overlap attribute edit box. The unit overlap affects how smoothly the trainer fits the function to the data. A larger unit overlap creates a smooth, slowly changing fit. A smaller unit overlap allows rapid changes in the fit.

To choose the function shape for the hidden layer, select one of the options under Hidden Unit Shapes: Spherical or Elliptical. When data is sparse or the input values are not correlated to each other, spherical units may perform better. When more data is available or the input values are correlated to each other, elliptical units may perform better. If the input dimension is 1, there is no difference between spherical and elliptical nodes, and the network selects Spherical by default.

After specifying the details of the architecture, click Finish.

The Training Console

When you exit the wizard, the following dialog appears, prompting you to select the training method.



Select the training paramters, then press "Train" button to start training

Clustering Method:
( ) Regular k-means clustering
(•) class-separate k-means clustering

[ ] Normalize the data series?

Choosing the Training Method

Choose an option for Clustering Method. The option you choose depends on what kind of problem you are trying to solve:

- If you are fitting the network to a function, choose Regular K-Means Clustering.
- If you are solving a classification problem, choose Class-Separate K-Means Clustering.

These methods differ in how they assign locations for unit centers. When regular clustering assigns locations, it uses all the data in the data set simultaneously. When class-separate clustering assigns locations, it goes through all the members of one class before going through the members of another. This method prevents centers from being placed near the boundaries between classes, where they do not help to discriminate between the classes.

Once the training parameters have been selected, the training process begin. The final training result is displayed in the training console.

Training will terminate when the parameters cannot be further improved. Normally, you can not stop the training process.

Preparing the Training Set

When you begin training, NOL Studio prepares a training set according to your specified variables and delays. The method to prepare the training set for all models is the same. Preparing the Training Set.

Viewing the Model Properties

When you finish the new model wizard, a radial basis function network object is created and added to the tree view. You can view the properties dialog for any model by double-clicking the corresponding tree view node, or by choosing Object > Go To > Models > Radial Basis Function Net.

General Properties

The General Properties tab on the model properties dialog gives general information about the model. This is the same as for the predictive model. See General Properties.

Brief Information of Model Performance

The ratings of “Good”, “OK”, and “Need Improvement” for a radial basis function network are based on the same criteria as for the predictive model. See Brief Information of Model Performance.

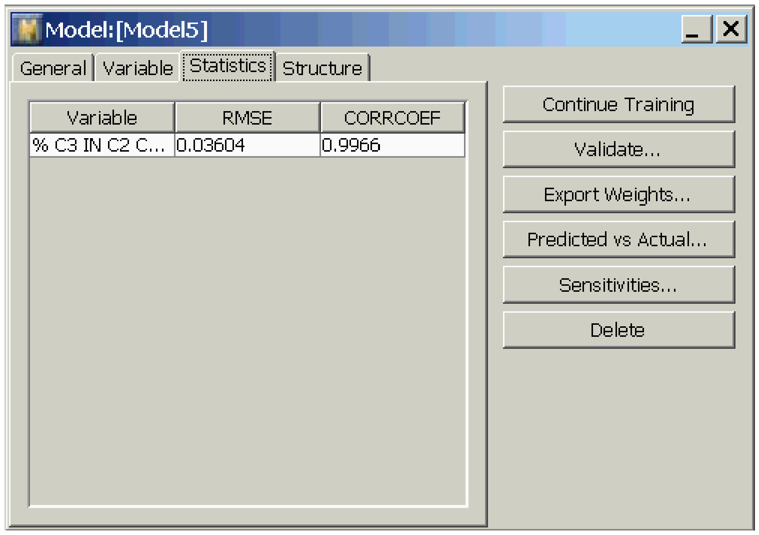
Model Variables

The Variables tab on the model properties dialog summarizes the classification of variables and delays for each input and output for this model. See Model Variables.

Statistics

The Statistics tab provides performance statistics for the trained RBFN model. During RBFN training, all data is used for training. The statistics show how well the model fits the training data set. You can use other data series to perform the validation of this RBFN model to see the model performance on new data.

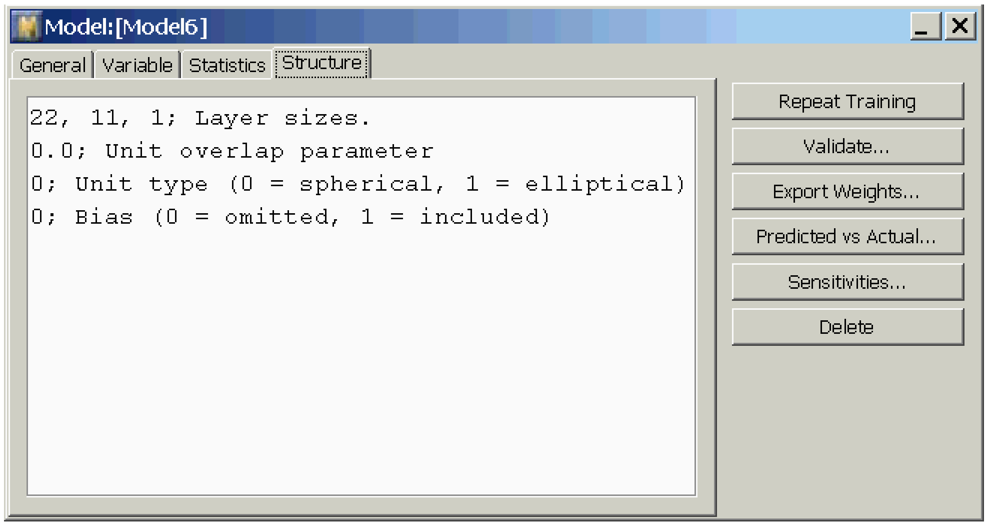
The statistics panel is shown below:



Variable	RMSE	CORRCOEF
% C3 IN C2 C...	0.03604	0.9966

Model Structure

The Structure tab shows the structure of the backpropagation network model. This gives you a chance to review and verify the models created by NOL Studio. For example:



```
22, 11, 1; Layer sizes.
0.0; Unit overlap parameter
0; Unit type (0 = spherical, 1 = elliptical)
0; Bias (0 = omitted, 1 = included)
```

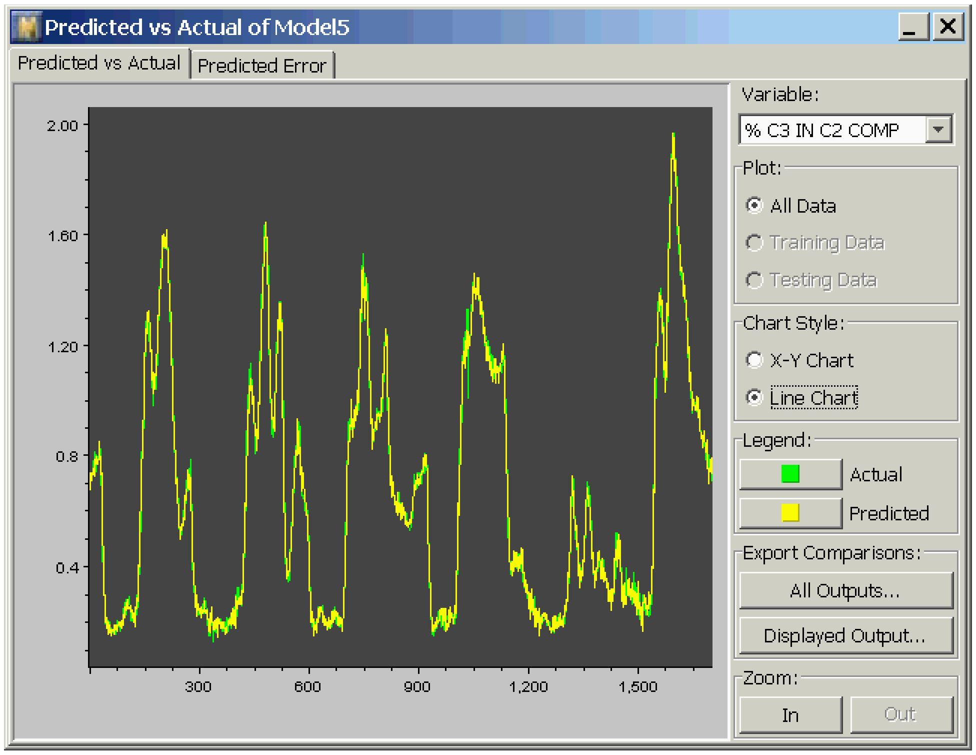
To see the internal weights and other details, you must export the model as a text file, using the Export Weights button.

Performing Operations on the Model

This section describes the operations you can perform on the model, using the buttons along the right side of the model properties dialog.

Showing the Predicted Versus Actual Plot

To show a plot of predicted versus actual values attained during training, click the Predicted vs. Actual button to show this dialog:

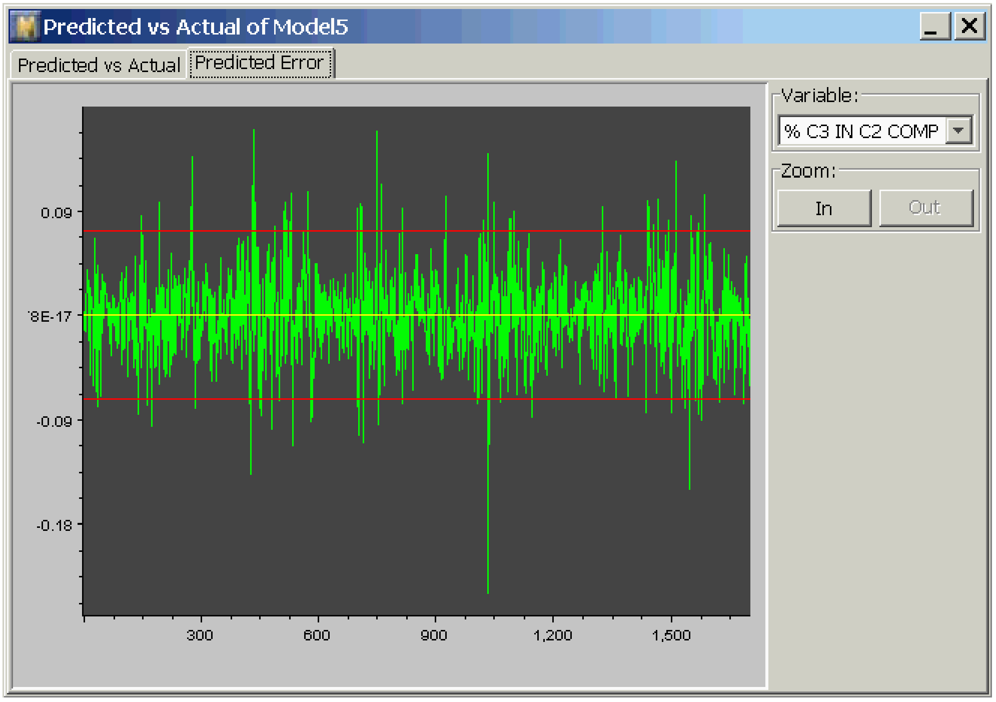


In this dialog, you can show the prediction of any output versus the training target values of that output, as a line chart (shown above), or as an x-y (scatter) chart. You change outputs by selecting from the Variable drop-down list. You change chart formats by selecting the Chart Style radio buttons. To change colors on the plot, select the appropriate button in the Legend section.

Exporting Predictions

You can save the model predictions to a file, using the Export Comparisons button, which is the same as for a backpropagation network. For details, see Exporting Predictions.

Viewing the Predicted Error



In the Predicted Error dialog, you can show the prediction error of any output as a line chart, as shown above. You change outputs by selecting from the Variable dropdown list. In the chart, the 95% upper and lower limits for the predicted error are also shown in red.

Zooming

All of the charts in the predicted vs. actual dialog allow you to interactively zoom in/out and scroll through your data. When you zoom in, the location of the point in the center of the plot does not change. Thus, if you want to zoom in on a particular point, scroll the plot horizontally until the point is shown in the center of the screen, then select the zoom in button. When you zoom in, plot symbols representing the individual data points appear. In a line chart, you cannot magnify the y axis by zooming, whereas you can in a scatter chart.

Validating a Model Against Another Data Series

You can validate an RBFN model against a new data series. The validation process is the same as with a predictive model. For details, see Validating a Model Against Another Data Series.

Viewing Input-Output Sensitivities

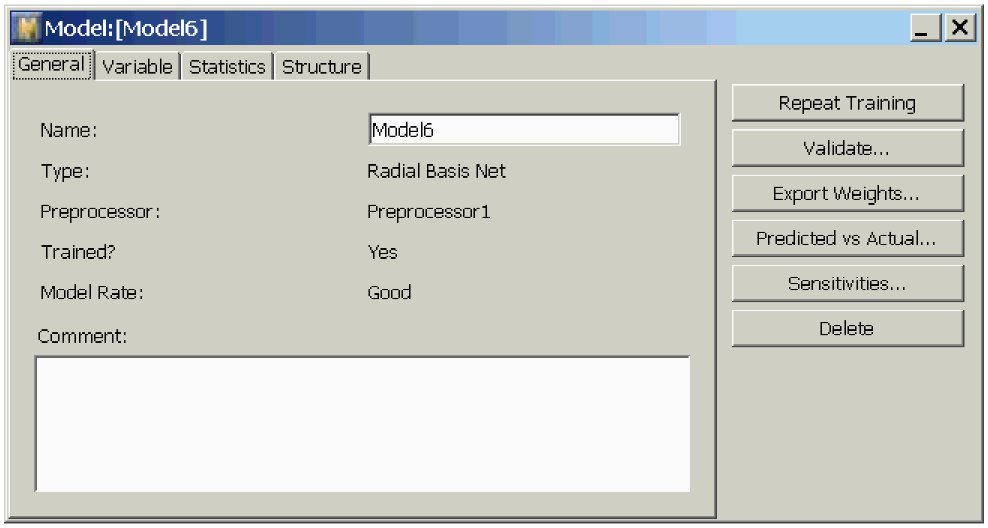
To see the sensitivities, click the Sensitivities button on the model properties dialog. You can analyze the sensitivities by clicking the buttons in the sensitivity dialog. See Input-Output Sensitivities.

Exporting a Model

To deploy an radial Basis Function network in Gensym Neural Net Engine (GNNE) or NeurOn-Line (NOL) Classic, you must save the weights of the radial basis function network to a text file. You can also directly export the model parameters to a radial basis function network object in GNNE. For more information on deploying a trained RBFN model, see Model Deployment.

To export the model weights:

1 Click Export Weight on the Model Properties dialog:



Name: Model6
Type: Radial Basis Net
Preprocessor: Preprocessor1
Trained? Yes
Model Rate: Good
Comment:

2 Specify a path and file name in the dialog. The model weight will be saved in a `.txt` file.

Your model weight is now ready for deployment in NOL Classic.

Performing Simulations with a Trained Model

You can validate the radial basis function network model by using simulations in NOL Studio. Simulations allow you to specify some data as inputs to a model, inspect the output generated from that data, then save and analyze the results. For details, see Performing Simulations with a Trained Model.

Creating a Rho Net

Describes how you create a Rho network.

Introduction

The Rho Network is based on the Radial Basis Function Network. It is a 3-layer feed-forward, layered network, whose middle layer uses a multi-variate Guassian function. It is useful for classification problems, especially when deciding whether an item belongs to one particular class or not, which is generally called single-class membership problem.

Similar to predictive models, you can start building Rho Net models after importing data, labeling and filtering data by using a preprocessor, and creating formulas that condition the data in the same preprocessor. This chapter describes how you set up and train an RBFN model.

NOL Studio helps make modeling decisions, such as selecting variables. You need to define the architecture of an Rho Net model.

One NOL Studio project can contain any number of Rho Net models. This allows you to train models with different architectures for the same problem, then to compare models, using the validation tools until you are completely satisfied with the performance of your model or models. You can then save out your best model or models for online deployment.

Creating Rho Net Models

To create a Rho Network model, you follow the steps in the modeling wizard. The wizard guides you through these steps to create a model:

1. Name the model.
2. Select whether to use old model parameters.
3. Specify the preprocessor for the model.
4. Specify the output data series to be used in the model.
5. Classify the variables as input, output, or unused.
6. Automatically select inputs.
7. Specify the model architecture.

These steps are detailed in the following sections.

To launch the wizard:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Rho Net.
Toolbar:	Click the New Rho Net button.
Tree View:	Right-click the Rho Nets node and choose New from its menu.

All of these actions display the model wizard.

Note After the wizard is displayed, follow the first six steps in the procedure summarized in Creating a Predictive Model.

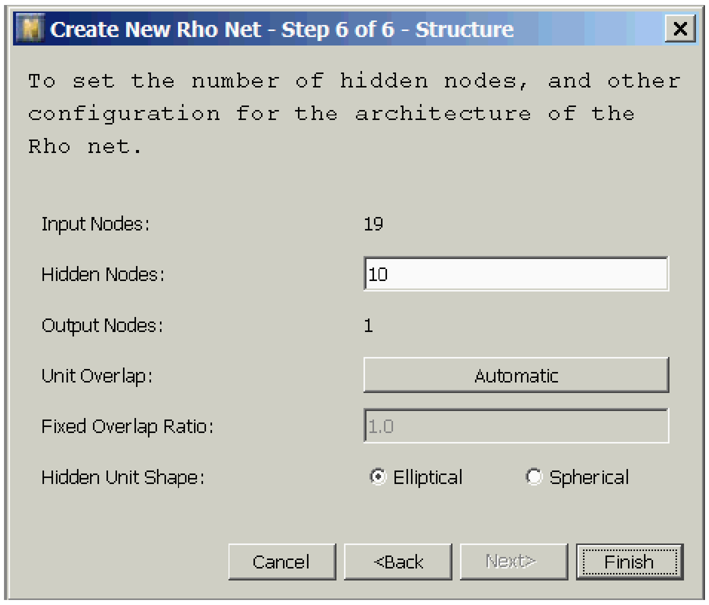
Specifying the Model Architecture

After you finish defining the input and output structure, the last step is to specify the internal architecture of the Rho Net model. A Rho Net model contains exactly three layers. The number of nodes in the first layer is the same as the number of input variables. The number of nodes of the last layer is the same as the number of output variables. The middle or hidden layer can have any number of nodes.

Each node in a layer is connected to all other nodes in the layers before it and after it. The connections between the input and hidden layers are unweighted. However, Rho Nets weight the connections between the hidden layer and output layer normally, like a BPN or an Autoassociative network.

The transfer function of the input and output layer is linear. You can choose whether the transfer functions of the hidden layer are spherical or elliptical Gaussians.

When you get to this step, the wizard shows following dialog:


Create New Rho Net - Step 6 of 6 - Structure

To set the number of hidden nodes, and other configuration for the architecture of the Rho net.

Input Nodes: 19

Hidden Nodes: 10

Output Nodes: 1

Unit Overlap: Automatic

Fixed Overlap Ratio: 1.0

Hidden Unit Shape: [✓ Elliptical] Spherical

You can change the number of nodes for the hidden layer. The numbers of nodes for the first and last layer are read only and are defined by previous steps.

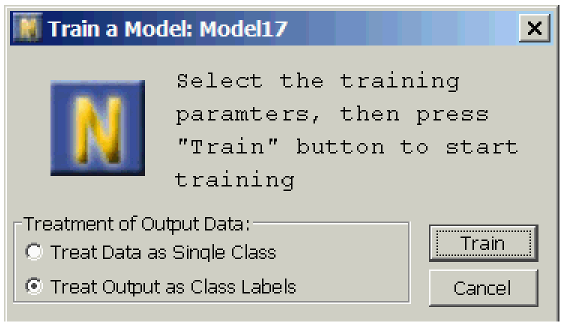
To set the unit overlap, choose whether the overlap is automatic or fixed by clicking the toggle button. If the overlap is automatic, the network chooses the best unit overlap for you automatically. Generally, you will use an automatic overlap. If the overlap is fixed, enter a positive value in the Unit Overlap attribute edit box. If the Unit Overlap is 1.0, each hidden unit's width is the distance to the nearest hidden unit. If the overlap parameter is 2.0, for example, the unit's width is twice the nearest neighbor distance. The unit overlap affects how smoothly the trainer fits the function to the data. A larger unit overlap creates a smooth, slowly changing fit. A smaller unit overlap allows rapid changes in the fit. The Unit Overlap should usually be between 0.5 and 5.0.

To choose the function shape for the hidden layer, select one of the options under Hidden Unit Shapes: Spherical or Elliptical. When data is sparse or the input values are not correlated to each other, spherical units may perform better. When more data is available or the input values are correlated to each other, elliptical units may perform better. If the input dimension is 1, there is no difference between spherical and elliptical nodes, and the network selects Spherical by default.

After specifying the details of the architecture, click Finish.

The Training Console

When you exit the wizard, the following dialog appears, prompting you to select the training method:


Train a Model: Model17

Select the training paramters, then press "Train" button to start training

Treatment of Output Data:

( ) Treat Data as Single Class

(•) Treat Output as Class Labels

Choosing the Training Method

Which option you choose depends on what kind of problem you are trying to solve:

- If the data set contains data that belongs to a single class, choose Treat Data as Single Class. The training algorithm ignores any output values in the data set. When you later evaluate the Rho Network, its output will be the probability that the input data is part of the distribution defined by the training data series.
- If the data set contains data that belongs to several classes, choose Treat Output Data as Class label. The number of output values in the data series corresponds to the number of possible classes. In each data pair, the output that corresponds to the element's class should be one, and the rest of the outputs should be zero. When you later evaluate the Rho Network, its output will be a vector with one element for each class, and each element will be the probability that the input belongs to that class.

Once the training parameters have been selected, the training process begin. Training terminates when the parameters cannot be further improved. Normally, you cannot stop the training process. An information dialog appears when training is complete.

Preparing the Training Set

When you begin training, NOL Studio prepares a training set according to your specified variables and delays. The method to prepare the training set for all models is the same. See Preparing the Training Set.

Viewing the Model Properties

When you finish the new model wizard, a rho network object is created and added to the tree view. You can view the properties dialog for any model by double-clicking the corresponding tree view node, or by choosing Object > Go To > Models > Rho Net.

General Properties

The General Properties tab on the model properties dialog gives general information about the model. This is the same as for the predictive model. See General Properties.

Brief Information of Model Performance

The ratings of "Good", "OK", and "Need Improvement" for a rho network with more than one output are based on the same criteria as for the predictive model. However, if the rho network has output for a single class, the rating is not valid. See Brief Information of Model Performance.

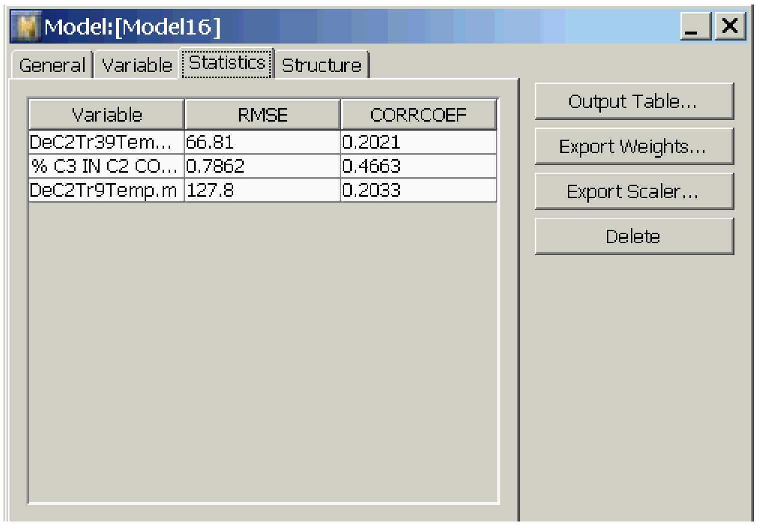
Model Variables

The Variables tab on the model properties dialog summarizes the classification of variables and delays for each input and output for this model. See Model Variables.

Statistics

The Statistics tab provides performance statistics for the trained Rho Net model. The statistics are only valid for models with more than one output. During Rho Net training, all data is used for training. The statistics show how well the model fits the training data set. You can use other data series to perform the validation of this Rho Net model to see the model performance on new data.

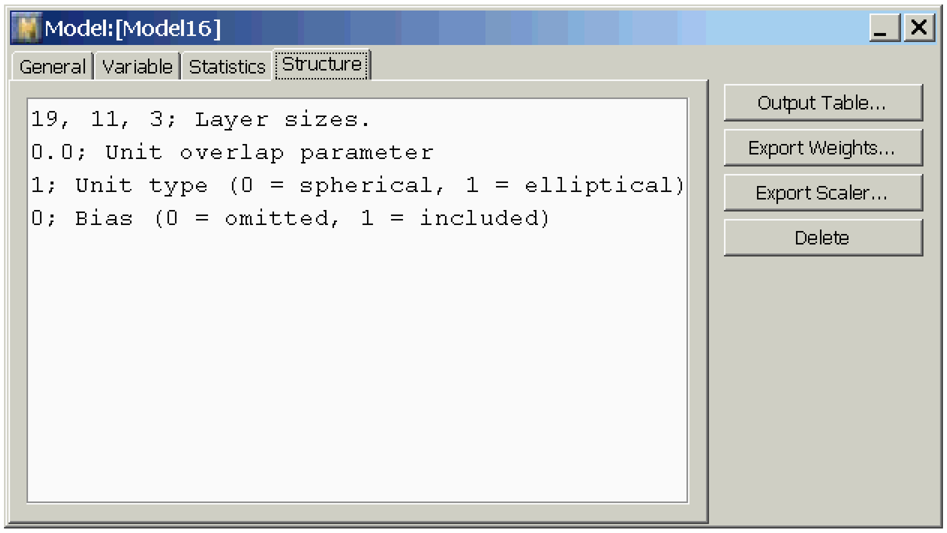
The statistics panel is shown below:



Variable	RMSE	CORRCOEF
DeC2Tr39Tem...	66.81	0.2021
% C3 IN C2 CO...	0.7862	0.4663
DeC2Tr9Temp.m	127.8	0.2033

Model Structure

The Structure tab shows the structure of the backpropagation network model. This gives you a chance to review and verify the models created by NOL Studio. For example:



```
19, 11, 3; Layer sizes.
0.0; Unit overlap parameter
1; Unit type (0 = spherical, 1 = elliptical)
0; Bias (0 = omitted, 1 = included)
```

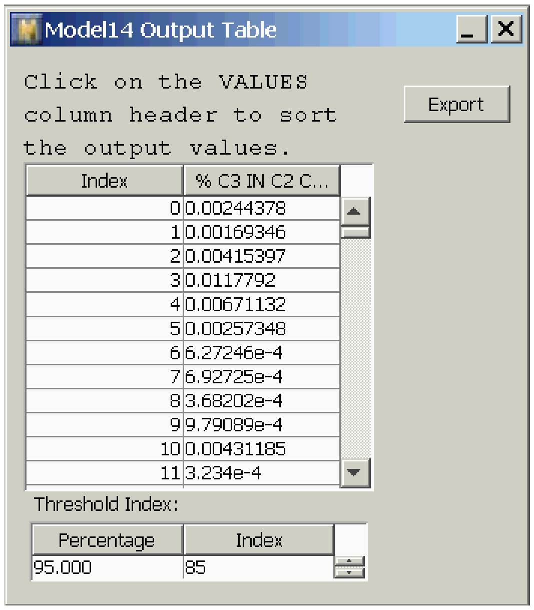
To see the internal weights and other details, you must export the model as a text file, using the Export Weights button.

Performing Operations on the Model

This section describes the operations you can perform on the model, using the buttons along the right side of the model properties dialog. Some actions, such as Predicted vs. Actual, Validation, and Sensitivities, are only available if the model has more than one output variable.

Showing the Output Table

For single-class membership problem, NOL Studio provides an output table dialog to sort and display the density function values in a table format. To show this dialog, click the Output Table button:



Click on the VALUES column header to sort the output values.

Index	% C3 IN C2 C...
0	0.00244378
1	0.00169346
2	0.00415397
3	0.0117792
4	0.00671132
5	0.00257348
6	6.27246e-4
7	6.92725e-4
8	3.68202e-4
9	9.79089e-4
10	0.00431185
11	3.234e-4

Threshold Index:

Percentage	Index
95.000	85

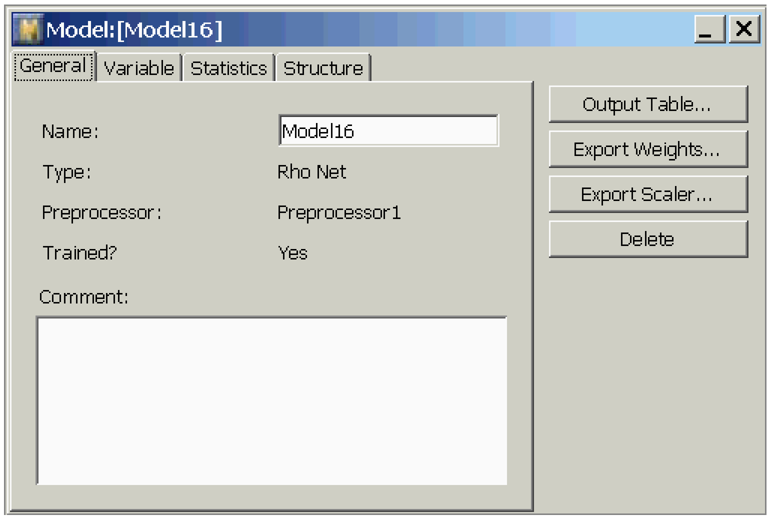
To find the threshold for your single-class membership problem, sort the probability density value first and enter the percentage in the lower table and press Return to determine the threshold index. The probability density value at the given index in the upper table is the threshold for your problem.

Exporting a Model

To deploy a rho network in Gensym Neural Net Engine (GNNE) or NeurOn-Line (NOL) Classic, you must save the weights of the rho network to a text file. You can also directly export the model parameters to a rho network object in GNNE. For more information on deploying a trained RHO model, see Model Deployment.

To export the model weights:

1 Click Export Weight on the Model Properties dialog:



Name: Model16
Type: Rho Net
Preprocessor: Preprocessor1
Trained? Yes
Comment:

2 Specify a path and file name in the dialog. The model weight will be saved in a `.txt` file.

Your model weight is now ready for deployment in GNNE and NOL Classic.

Creating a Partial Least Square Model

Describes how to create a partial least square model.

Introduction

Partial least squares (PLS) is a technique for constructing predictive models when both input variables and output variables are many and highly co-linear. PLS simultaneously reduces the dimensions of the input and output space and finds latent vectors for input and output which maximize the correlation of these two set of variables. Note that the emphasis is on predicting the responses and not necessarily on trying to understand the underlying relationship between the variables. PLS is a useful tool to model multivariate process where linear correlation dominates the relationship when prediction is the goal and there is no practical need to limit the number of measured variable space. PLS has been applied to monitoring and controlling industrial processes; a large process can easily have hundreds of controllable variables and dozens of outputs.

Similar to predictive models, you can start building PLS models after importing data, labeling and filtering data by using a preprocessor, and creating formulas that condition the data in the same preprocessor. This chapter describes how you set up and build a PLS.

NOL Studio helps you make modeling decisions, such as the best combination of inputs and time delays. You can set the model architecture of a PLS by setting the number of the latent variable, or you can let NOL Studio find the best architecture based on the minimum prediction error.

One NOL Studio project can contain any number of PLS models. This allows you to train models with different architectures for the same problem, then to compare models, using the validation tools until you are completely satisfied with the performance of your model or models. You can then save out your best model or models for online deployment.

Creating Partial Least Square Models

To create a PLS model, you follow the steps in the modeling wizard. The wizard guides you through these steps to create a model:

1. Name the model.
2. Select whether to use old model parameters.
3. Specify the preprocessor for the model.
4. Specify the output data series to be used in the model.
5. Classify the variables as input, output, or unused.
6. Specify time delays, if any, for the model inputs,
7. Automatically select inputs and delays.
8. Specify the number of factors.

These steps are detailed in the following sections.

To launch the wizard:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Partial Least Square Model.
Toolbar:	Click the New Partial Least Square Model button.
Tree View:	Right-click the Partial Least Square Model node and choose New from its menu.

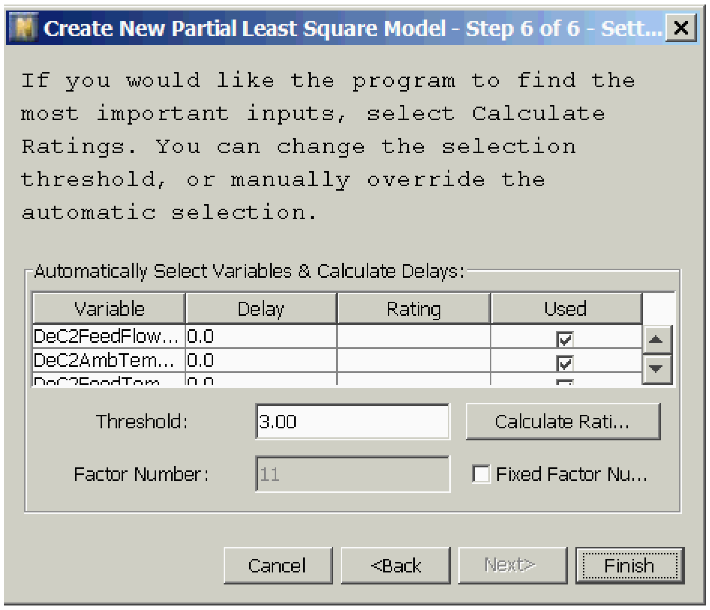
All of these actions display the model wizard.

Note After the wizard is displayed, follow the first seven steps in the procedure summarized in Creating a Predictive Model.

Specifying the Model Architecture

After you finish defining the input and output structure, the last step is to specify the internal architecture of the PLS model. The only parameter to set is the number of factors as latent variables. You set the number of factors by checking the Fixed Factor Number check box and entering the number in the Factor Number text field. In general, the number of factors depends on the complexity of the function the network has to model. The number of factors should define the number of degrees of freedom of the linear relationship between the joined input and output space.

When you get to this step, the wizard shows following dialog:


Create New Partial Least Square Model - Step 6 of 6 - Sett...

If you would like the program to find the most important inputs, select Calculate Ratings. You can change the selection threshold, or manually override the automatic selection.

Automatically Select Variables & Calculate Delays:

Variable	Delay	Rating	Used
DeC2FeedFlow...	0.0		✓
DeC2AmbTem...	0.0		✓
DeC2FeedTem	0.0		

Threshold: 3.00

Factor Number: 11

Fixed Factor Nu... ☐

After specifying the number of factors, click Finish.

Note The Partial Least Square model is a special multivariate regression model. It does not require a multistep training process. There is no training console to show the status of PLS model training.

Preparing the Training Set

When you begin building a PLS model, NOL Studio prepares a data set according to your specified variables and delays. The method to prepare the data set for all models is the same. See Preparing the Training Set.

Viewing the Model Properties

When you click Finish in the new model wizard, a partial least square model object is created and added to the tree view. You can view the properties dialog for any model by double-clicking the corresponding tree view node, or by choosing Object > Go To > Models > Partial Least Square Model.

General Properties

The General Properties tab on the model properties dialog gives general information about the model. This is the same as for the predictive model. See General Properties.

Brief Information of Model Performance

The ratings of “Good”, “OK”, and “Need Improvement” for a partial least square model are based on the same criteria as for the predictive model. See Brief Information of Model Performance.

Model Variables

The Variables tab on the model properties dialog summarizes the classification of variables and delays for each input and output for this model. See Model Variables.

Statistics

The Statistics tab provides performance statistics for the PLS model. The statistics show how well the model fits the training and testing data set. See Statistics.

Model Structure

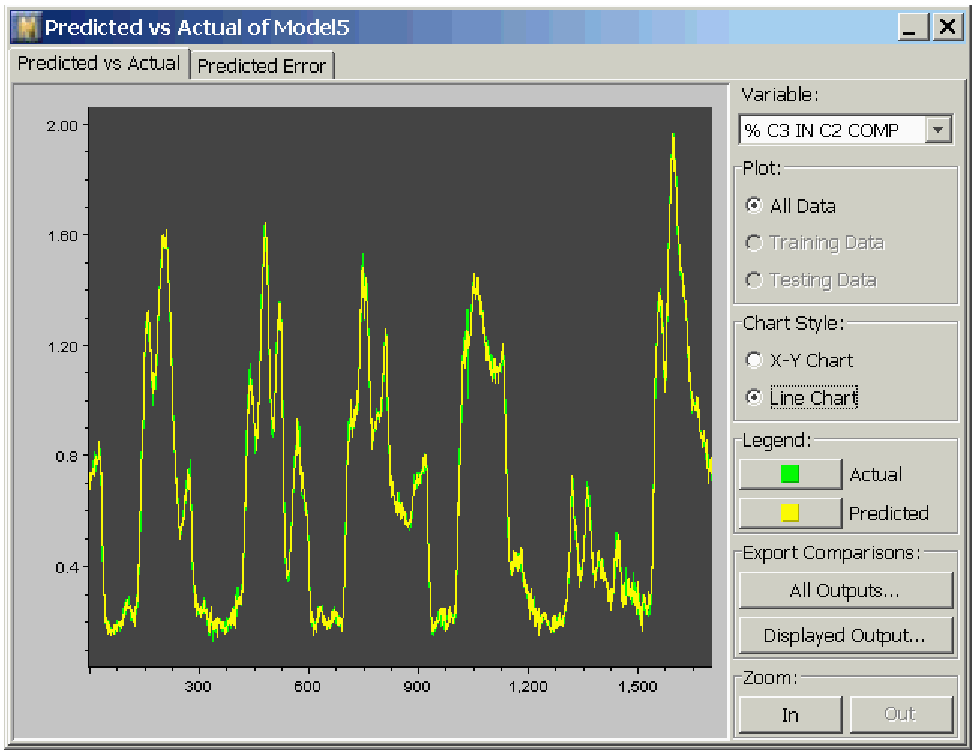
The Structure tab shows the number of factors for the partial least square model. To see the internal weights and other details, you must export the model as a text file, using the Export Weights button.

Performing Operations on the Model

This section describes the operations you can perform on the model, using the buttons along the right side of the model properties dialog.

Showing the Predicted Versus Actual Plot

To show a plot of predicted versus actual values attained during training, click the Predicted vs. Actual button to show this dialog:



Variable: % C3 IN C2 COMP

Plot: [✓ All Data] Training Data Testing Data

Chart Style: X-Y Chart [✓ Line Chart]

Legend: 🟩 Actual 🟨 Predicted

Export Comparisons: All Outputs... Displayed Output...

In this dialog, you can show the prediction of any output versus the training target values of that output, as a line chart (shown above), or as an x-y (scatter) chart. You change outputs by selecting from the Variable drop-down list. You change chart formats by selecting the Chart Style radio buttons. To change colors on the plot, select the appropriate button in the Legend section.

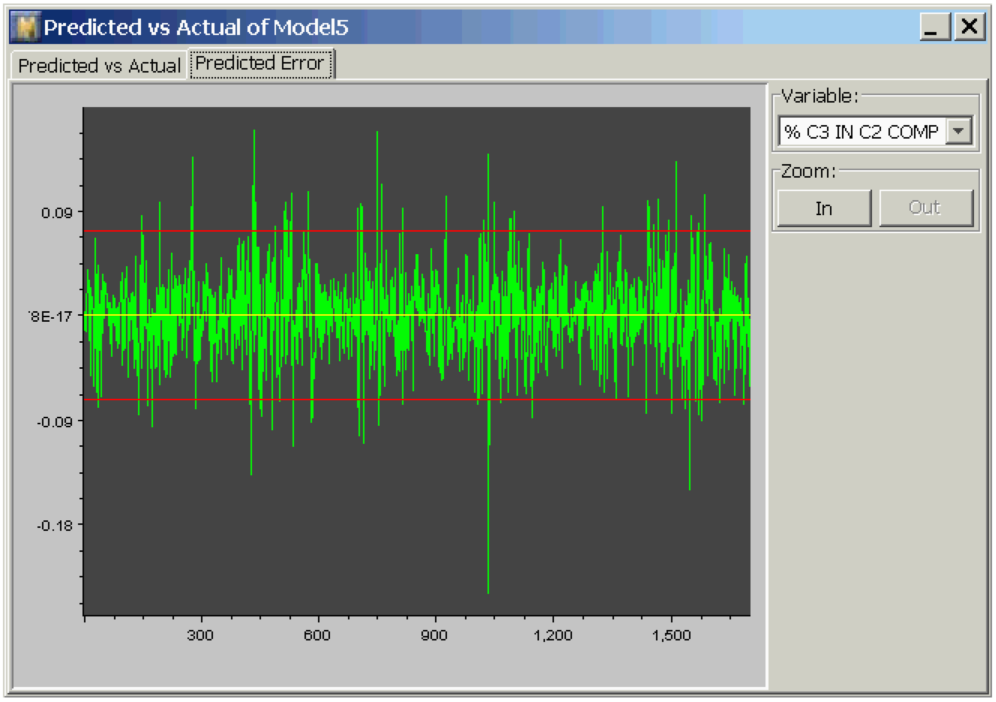
Exporting Predictions

You can save the model predictions to a file, using the Export Comparisons button. The data that is saved reflects the predictions as seen on the screen when you perform this action. For example, if you are displaying the predictions for the test data subset when you export the predictions, only selected rows of the data will be written. The “gaps”—corresponding to samples not chosen for the test set—will be written as not-a-number (NaN). When you click the All Outputs button, you save as many columns of data as there are outputs in the model. The Displayed Outputs button saves only the currently displayed output. Timestamps are included in the output, regardless of the option selected. The actual output values corresponding to the specified predictions are also included in the exported file. In this way, you can make comparisons between exported files.

When writing the file, you can choose either binary (*.bds*) or text (*.ds*) formats. You specify which option by specifying the file extension in the Save File dialog. A example format of the text file is shown below:

```
Time                10A100C3.PV_pred         10A100C3.PV
Time                % C3 IN C2 COMP_pred     % C3 IN C2 COMP
Millisecond         None                     None
        8.71964E+11             1.225174967              1.385540366
        8.71964E+11             1.286989442              1.385540247
        8.71964E+11             1.366936914              1.385540247
        8.71964E+11             1.374557197              1.382564783
        8.71965E+11             1.393119682              1.382488489
        8.71965E+11             1.409462661              1.367229223
        8.71965E+11             1.417492178              1.412197351
        8.71966E+11             1.439030908               1.42521441
        8.71966E+11             1.466799704               1.42521441
```

Viewing the Predicted Error



In the Predicted Error dialog, you can show the prediction error of any output as a line chart, as shown above. You change outputs by selecting from the Variable dropdown list. In the chart, the 95% upper and lower limits for the predicted error are also shown in red.

Zooming

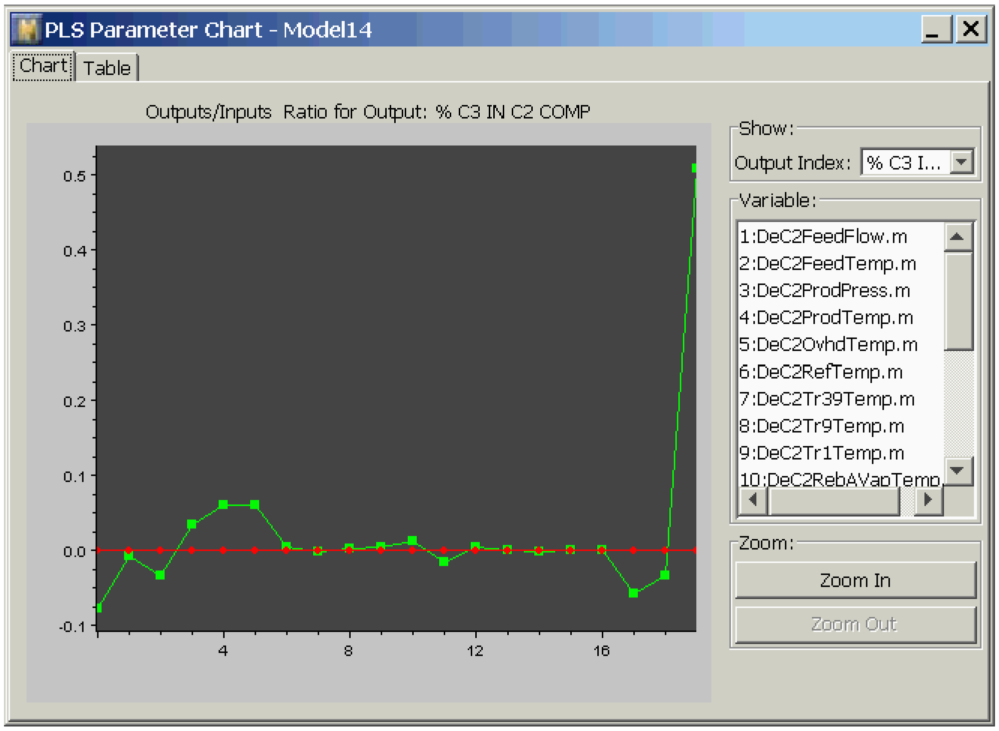
All of the charts in the predicted vs. actual dialog allow you to interactively zoom in/out and scroll through your data. When you zoom in, the location of the point in the center of the plot does not change. Thus, if you want to zoom in on a particular point, scroll the plot horizontally until the point is shown in the center of the screen, then select the zoom in button. When you zoom in, plot symbols representing the individual data points appear. In a line chart, you cannot magnify the y axis by zooming, whereas in a scatter chart you can.

Validating a Model Against Another Data Series

You can validate a PLS model against a new data series. The validation process is the same as with a predictive model. For details, see Validating a Model Against Another Data Series.

Viewing Inputs/Outputs Ratio

To see the inputs/outputs ratio, click the Inputs/Outputs Ratio button on the model properties dialog:

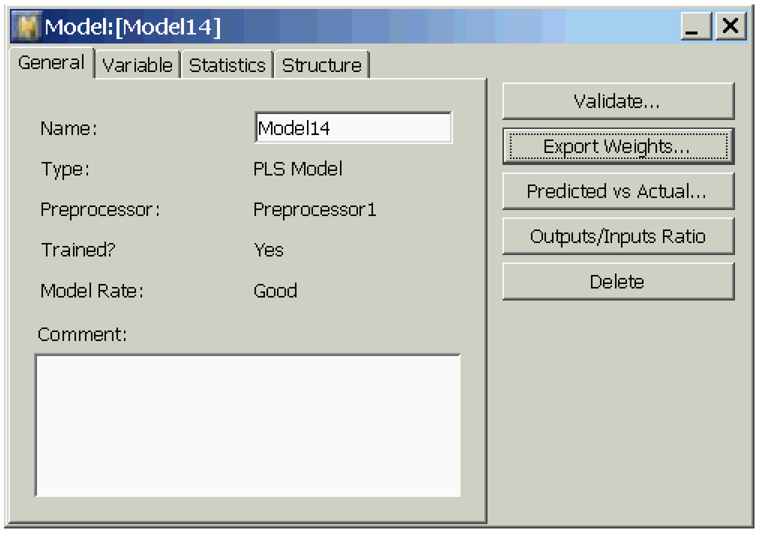


Exporting a Model

To deploy a PLS model in a G2 environment, you should save the weights of the PLS model to a text file. With NOL Studio and G2 connected, you can upload the PLS model parameters with a procedure call in G2. For more information on deploying a PLS model, see Model Deployment.

To export the model weights:

1 Click the Export Weight button in the Model Properties dialog:



Name: Model14
Type: PLS Model
Preprocessor: Preprocessor1
Trained? Yes
Model Rate: Good
Comment:

2 Specify a path and file name in the dialog.

The model weight will be saved in a `.pls` file.

Your model weight is now ready for deployment in a G2 environment.

Performing Simulations with a PLS Model

You can validate the PLS model by using simulations in NOL Studio. Simulations allow you to specify some data as inputs to a model, inspect the output generated from that data, then save and analyze the results. For details, see Performing Simulations with a Trained Model.

Creating a Principal Component Analysis Model

Describes how to create a principal component analysis model.

Introduction

Principal component analysis (PCA) involves a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. Mathematically, the main use of PCA is to reduce the dimensionality of a data set while retaining as much information as is possible. It computes a compact and optimal description of the data set. PCA is a useful statistical technique that has found application in fields such as process performance monitoring, abnormal situation root cause detection, face recognition and image compression, and is a common technique for finding patterns in data of high dimension. For a brief mathematical explanation of PCA, see What is PCA?.

Similar to all types of models, you can start building PCA models after importing data, labeling and filtering data by using a preprocessor, and creating formulas that condition the data in the same preprocessor. This chapter describes how you set up and build a PCA model.

NOL Studio helps you to build a PCA model through a model building wizard. One NOL Studio project can contain any number of PCA models. You can build a PCA model for any one set of variables. You can save out PCA models for online deployment.

Creating Principal Component Analysis Models

To create a PCA model, you follow the steps in the modeling wizard. The wizard guides you through these steps to create a model:

1. Name the model.
2. Select whether to use old model parameters.
3. Specify the preprocessor for the model.
4. Specify the data series for the model.
5. Classify the variables as used or unused.

These steps are detailed in the following sections.

To launch the wizard:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Principal Component Analysis Model.
Toolbar:	Click the New Principal Component Analysis Model button.
Tree View:	Right-click the Principal Component Analysis Model node and choose New from its menu.

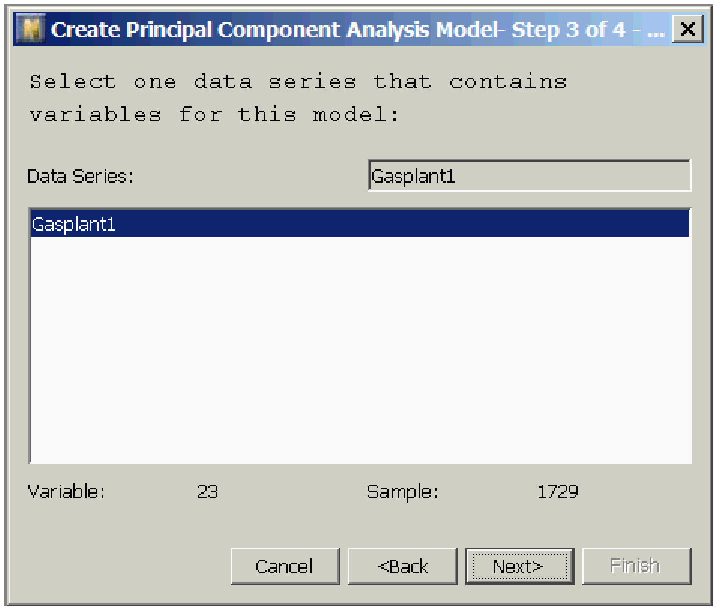
All of these actions display the model wizard.

Note After the wizard is displayed, follow the first three steps in the procedure summarized in Creating a Predictive Model.

Selecting the Data Series

Next, you select the data series containing the variables from which you want to build a PCA model. The variables used in a PCA model have to be within a single data series. There is no consideration of time stamps for the model. All the samples within the data series are treated as independent from previous and later samples.

To select data series, click the desired data series, for example:


Create Principal Component Analysis Model- Step 3 of 4 - ...

Select one data series that contains variables for this model:

Data Series: Gasplant1

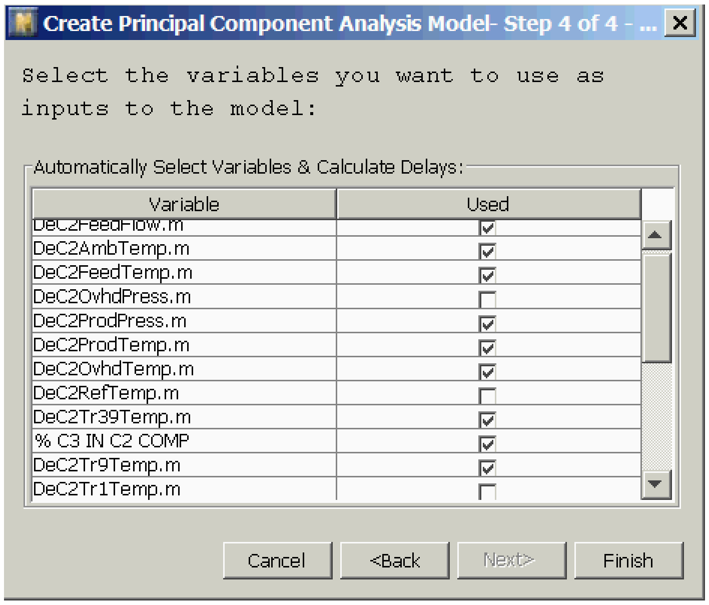
Gasplant1

Variable: 23 Sample: 1729

Classifying Variables

After you select the data series, the last step is to classify which variables you want to use to build the PCA model. The default setting is to use all variables in the selected data series. To exclude any variable, you need to manually disable the variable in the table.

When you get to this step, the wizard shows following dialog:


Create Principal Component Analysis Model- Step 4 of 4 - ...

Select the variables you want to use as inputs to the model:

Automatically Select Variables & Calculate Delays:

Variable	Used
DeC2FeedFlow.m	✓
DeC2AmbTemp.m	✓
DeC2FeedTemp.m	✓
DeC2OvhdPress.m	☐
DeC2ProdPress.m	✓
DeC2ProdTemp.m	✓
DeC2OvhdTemp.m	✓
DeC2RefTemp.m	☐
DeC2Tr39Temp.m	✓
% C3 IN C2 COMP	✓
DeC2Tr9Temp.m	✓
DeC2Tr1Temp.m	☐

Cancel
<Back
Next>
Finish

After classifying the variables, click Finish.

Note The Principal Component model is a linear statistical model. It does not require a multistep training process. There is no training console to show the status of PCA model training.

Preparing the Training Set

When you begin building a PCA model, NOL Studio prepares a data set according to your specified variables. The method to prepare the data set for all models is the same. See Preparing the Training Set.

Viewing the Model Properties

When you click the Finish button in the new model wizard, a principal component analysis model object is created and added to the tree view. You can view the properties dialog for any model by double-clicking the corresponding tree view node, or by choosing Object > Go To > Models > Principal Component Analysis Model.

General Properties

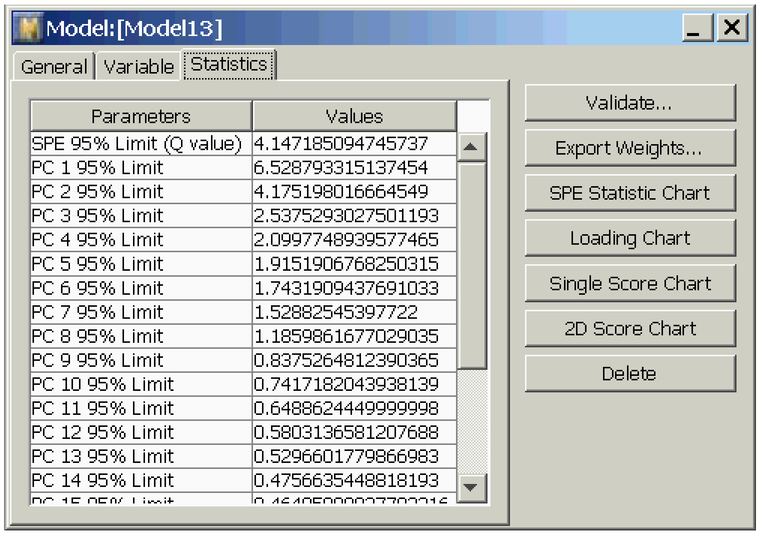
The General Properties tab on the model properties dialog gives general information about the model. This is the same as for the predictive model. See General Properties.

Model Variables

The Variables tab on the model properties dialog shows variables used for this model. See Model Variables.

Statistics

The Statistics tab provides statistics for the PCA model. The specific statistics shown are the general statistics, such as 95% Squared Predicted Error (SPE) (Q) value, 95% T^2, and 95% limits for each principal component. Because the data are center scaled before building PCA model, the principal components are also centered around its axis. For example:



Parameters	Values
SPE 95% Limit (Q value)	4.147185094745737
PC 1 95% Limit	6.528793315137454
PC 2 95% Limit	4.175198016664549
PC 3 95% Limit	2.5375293027501193
PC 4 95% Limit	2.0997748939577465
PC 5 95% Limit	1.9151906768250315
PC 6 95% Limit	1.7431909437691033
PC 7 95% Limit	1.52882545397722
PC 8 95% Limit	1.1859861677029035
PC 9 95% Limit	0.8375264812390365
PC 10 95% Limit	0.7417182043938139
PC 11 95% Limit	0.6488624449999998
PC 12 95% Limit	0.5803136581207688
PC 13 95% Limit	0.5296601779866983
PC 14 95% Limit	0.4756635448818193
PC 15 95% Limit	[illegible]

Performing Operations on the Model

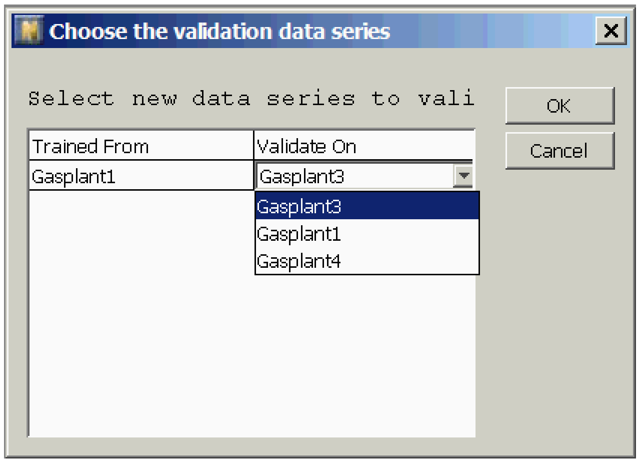
This section describes the operations you can perform on the model, using the buttons along the right side of the model properties dialog.

Validating a Model Against Another Data Series

If you are satisfied with the performance of the model on the present data series, then you may want to test the model on another data series. You can use any data series other than the one used for training the model, as long as it contains variables with the same tags as the original training data series. This is because validation is meant to be performed with a data series from the same underlying data sources, but from a different time range or set of operating conditions. Normally, you are trying to look at whether the principal component values are out of their limits.

To start a validation session from the main model dialog:

1 Click the Validate button to display this dialog:



Select new data series to vali

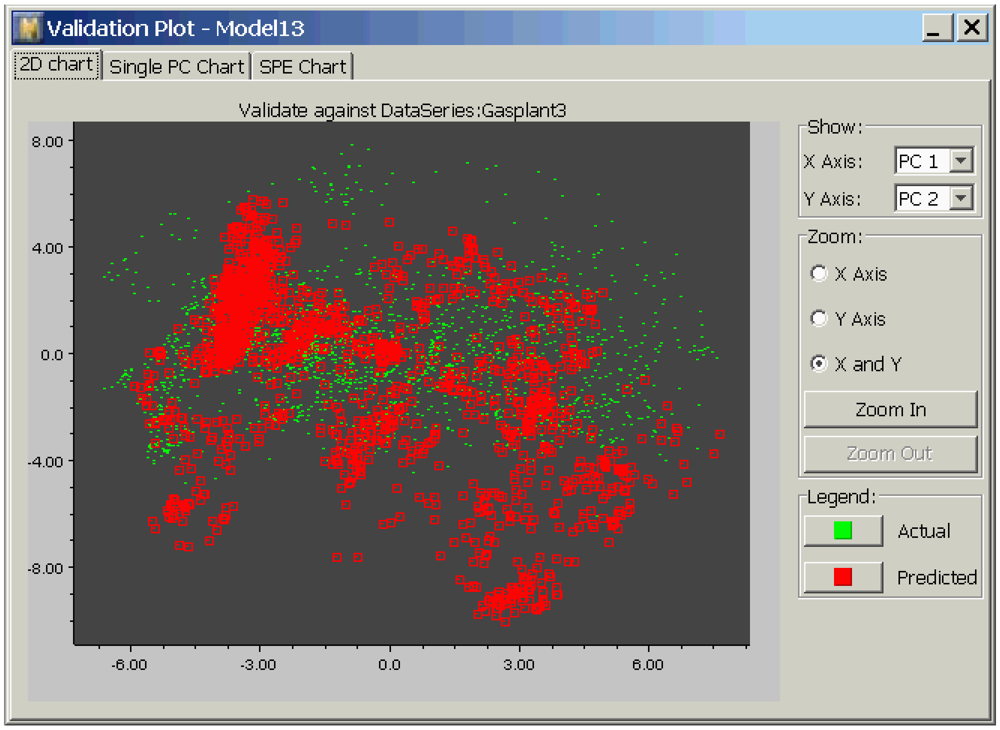
Trained From	Validate On
Gasplant1	Gasplant3

Gasplant3
Gasplant1
Gasplant4

Only valid data series appear in the dropdown list on the dialog.

2 Click OK once you have chosen a data series for the validation.

The next step is to view the performance of the model on the new data series. There are several items of information displayed in the following dialog:



X Axis: PC 1
Y Axis: PC 2
Zoom: ( ) X Axis ( ) Y Axis [✓ X and Y]

Legend:
🟩 Actual
🟥 Predicted

This dialog shows the name of the model and the name of the validation data series. There are three tabs to show the single score chart with control limits, 2D score chart, and the SPE chart with limits. These control limits are calculated from the training data.

2D Score Chart

By default, the first two principal components, PC1 and PC2, are displayed in the plot, since these variables contain most of the information on the variation in your data set. However, you can also view less significant principal components (PC3 - PCn) by using the Show selection boxes in the upper right of the score chart view.

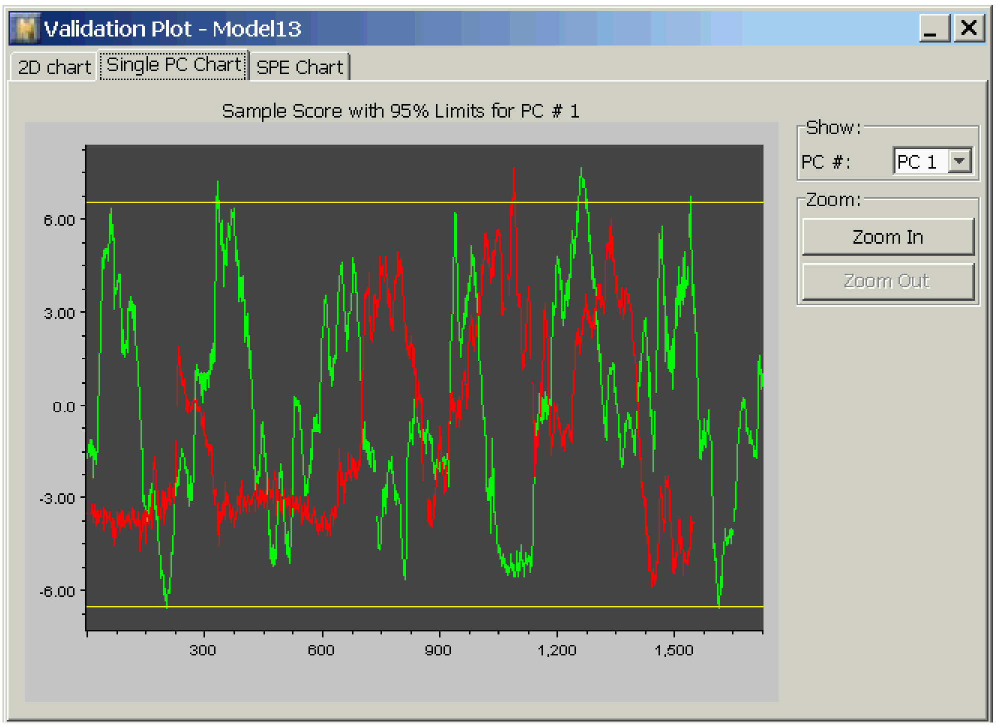
The scores from validation data series will be superimposed over the original plot. The green dots represent the PCs calculated from the training data series, and the red dots represent the PCs from the validation data series.

Single Score Chart

By default, the first principal component, PC1, and its limits are displayed in the plot. You can also view less significant principal components (PC1 - PCn) by using the Show selection boxes in the upper right of the projection chart view. The scores from validation data series will be superimposed over the original plot.

The green line represents the PCs calculated from the training data series, and the red line represents the PCs from the validation data series. The controls for the chart are arranged on the right-hand side. You can choose which score variable is plotted. You can zoom in and out for details.

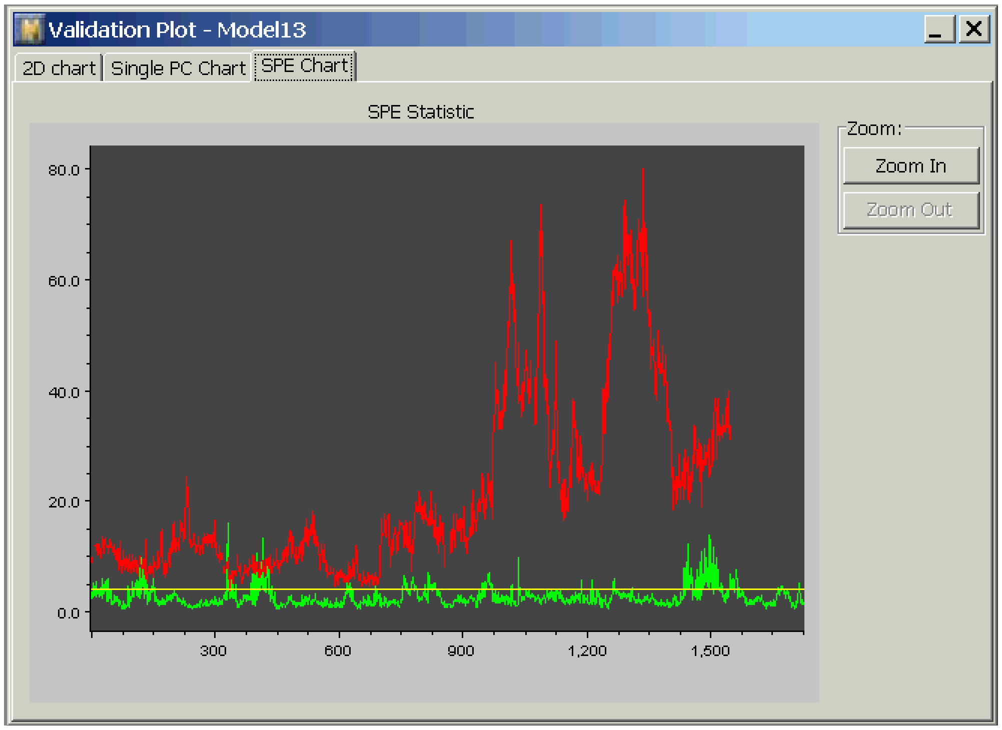
If the scores from validation data series are outside their corresponding control limits, then the process that generate the validation data series may operate outside the normal operating range defined by the training data series.



SPE Chart

SPE refers to the Squared Predicted Error of the variables based on the PCA model. The green line represents the SPE calculated from the training data series, and the red line represents the SPE from the validation data series. The controls for the chart are arranged on the right-hand side. You can choose which score variable is plotted. You can zoom in and out for details.

If the SPE from validation data series are outside Q limit, calculated from training data, then the process that generates the validation data series may operate outside the normal operating range defined by the training data series.

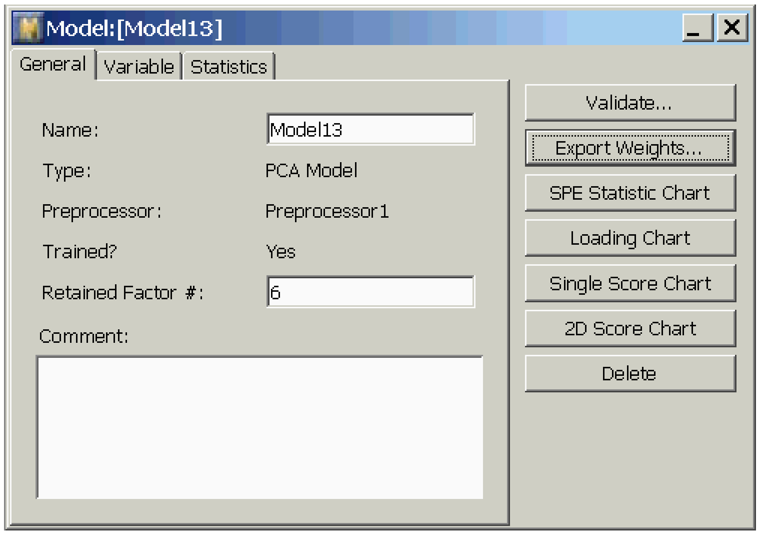


Exporting the Model Parameters

The PCA should be deployed in G2 only. There are two ways to deploy PCA models in G2. You can save the parameters of the PCA model to a text file, then load the parameter file into PCA object in G2. When NOL Studio and G2 are connected, you can upload the PCA model parameters with a procedure call in G2 without saving them into a file. For more information on deploying a PCA model, see Model Deployment.

To export the model weights:

1 Click the Export Weight button in the Model Properties dialog:



Name: Model13

Type: PCA Model

Preprocessor: Preprocessor1

Trained? Yes

Retained Factor #: 6

2 Specify a path and file name in the dialog.

The model parameters will be saved in a *.pca* file.

Your model parameters is now ready for deployment in G2.

SPE Statistic Chart

SPE refers to the Squared Predicted Error of the variables based on the PCA model. It is the error between the original data and reconstructed values from the PCA model. With PCA model, the training data X(n x m) are reduced to an f-dimensional matrix T (n x f), where $f < n$. The scores matrix T is produced by multiplying X by a projection matrix P (m x f), as follows:

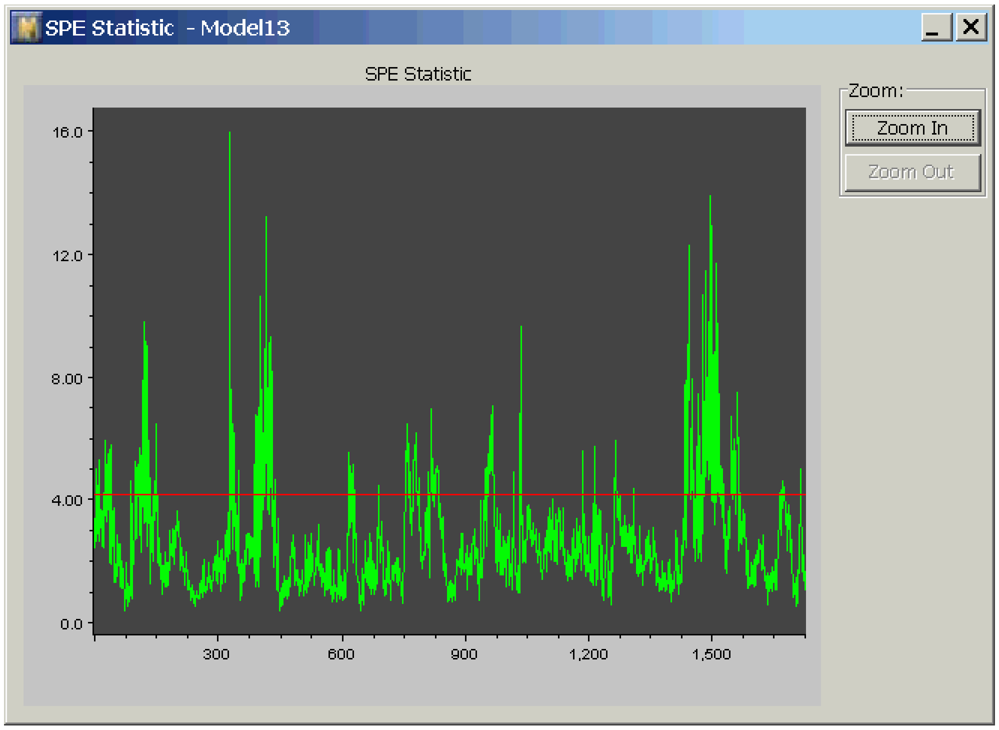
$$T = XP$$

The P matrix is chosen so that T is an optimum projection, in the sense of minimizing the "lost information" that results from reducing the number of columns of X. Mathematically, the lost information can be quantified by a matrix E (n x m), calculated as follows:

$$E = X - TP^t = X(I - PP^t)$$

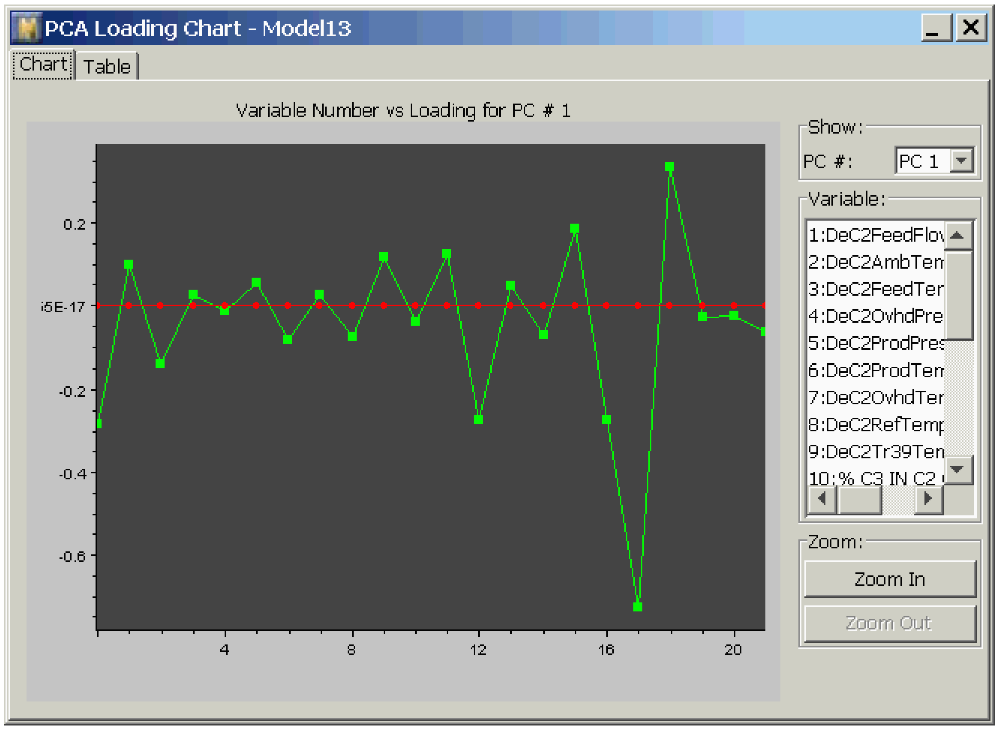
PCA chooses the projection matrix P so the 2-norm (equivalent to the mean of the squares of the elements) of E (SPE) is minimized. In addition, the 95% statistic (Q value) for SPE can be calculated based on the training data set. This Q value can be used to monitor the performance of new data samples.

To show a plot of Squared Predicted Error for the training data, click the SPE Statistic Chart button to show this dialog:



Loading Chart

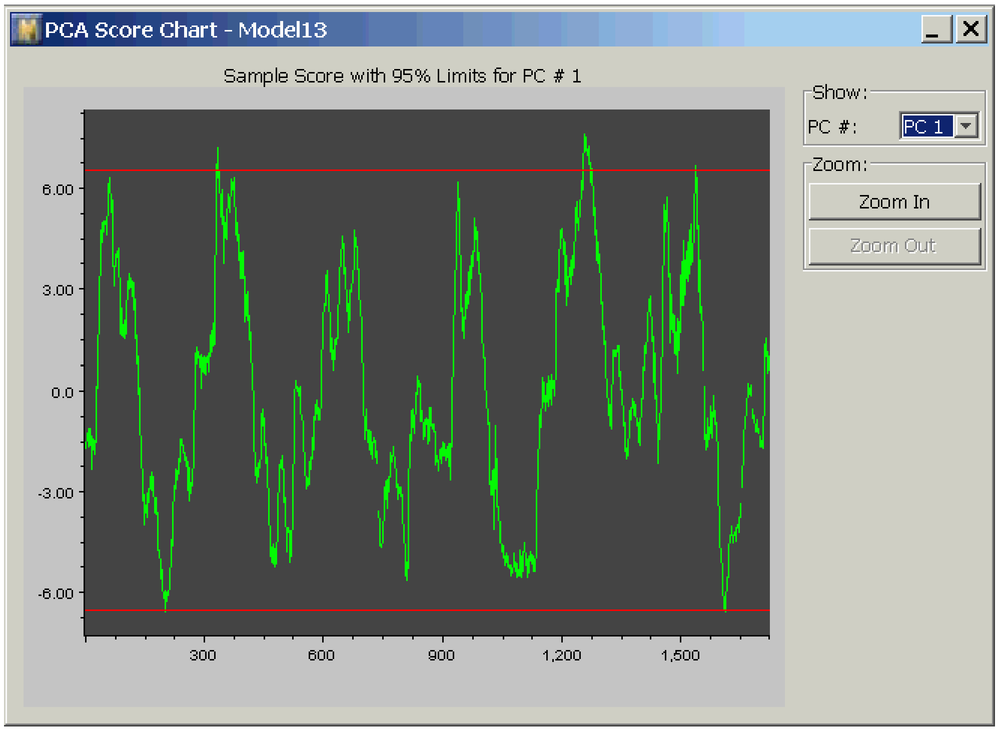
If you want to know how much each variable contributes to the principal component factors, you can look at the correlations between the variables and the factors. These correlations are also called factor loadings. Click the Loading Chart button to show this dialog:



To show loading factors of variables to different principal components, you can use the Show selection boxes in the right of the chart view to select the component index. The variable list box shows the index to variable names. If the number of variables is very large, you can use the Zoom In and Zoom Out buttons to exam the loadings in details.

Single Score Chart

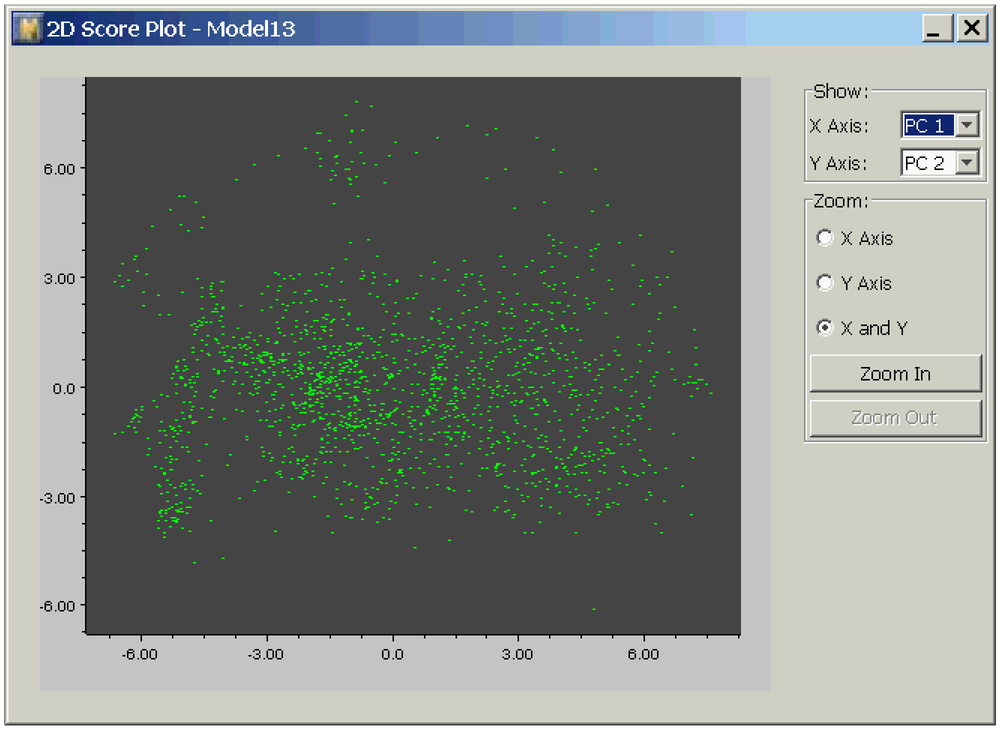
A Single Score Chart allows you to view one principal component calculated from training data. Score line charts are useful for showing the trends of the scores and also the trend against the 95% control limits. For example:



To show different score variables, use the Show selection boxes in the right of the chart view. The line chart allows you to interactively zoom in/out and scroll through your data. When you zoom in, the location of the point in the center of the plot does not change. Thus, if you want to zoom in on a particular point, scroll the plot horizontally until the point is shown in the center of the screen, then click the Zoom In button.

2D Score Chart

A 2D Score Chart is the same type of chart as a Projection chart for a data series. For detailed explanation and benefits of the chart, you can refer to the Viewing Data in Projection Charts. Click the 2D Chart button to show this dialog:



14

Optimization

Describes how to use a model and user-defined criteria to determine correct setpoints for manipulated variables.

Introduction

At this point, you have trained and verified a model that accurately predicts one or more performance characteristics of your process. The model can be directly deployed to make this prediction on-line, in real time, acting as a virtual sensor. Virtual sensor predictions can be used as part of your existing control system, or as an input to a supervisory control system.

You may also want to use the model to drive the predicted value to a desired set point. Or, you might have a more complex economic objective that should be maximized or minimized. In terms of network inputs and outputs, you are seeking input values that achieve the desired output value or optimize the economic objective. However, the predictive model *cannot* be used directly for this purpose, because some of the inputs to your predictive model might be functions of other inputs, and therefore, cannot be varied independently.

For example, in a distillation column, you might use measurements of feed flow rate, reflux rate, reboiler steam flow, and several tray temperatures to predict overhead product composition. However, tray temperatures are functions of feed flow rate, reflux rate, reboiler steam flow, not truly independent. So, if you want to find optimal settings for the reflux rate and reboiler steam flow, you must know how tray temperatures are affected by these variables. Your predictive model does not contain this information.

So the first job in solving an input optimization is to develop a model that better organizes dependent and independent variables, which we call an *optimization model*. Given this model, you specify desired values and cost for input and output variables, initial values for the manipulated variables, and optional constraints and parameters, and NOL Studio provides setpoints for manipulated variables that produce the best solution to your specification.

In this Chapter, we first discuss the classification of variables in optimization models. Next, we discuss the process of creating an optimization model. Then, we describe the objective function used in NOL Studio optimizations. Finally, we show how you create an objective function and run optimizations based on your optimization model.

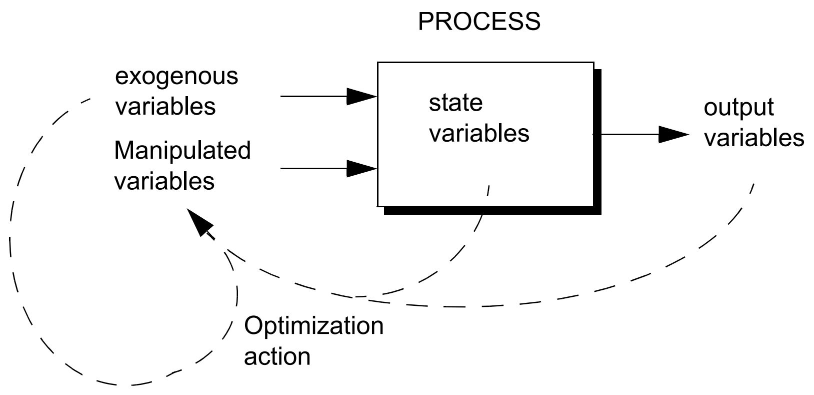
Variable Classification for Optimization

When you set up an optimization problem, you must classify your measured variables into four categories:

Category	Description
Exogenous (external) variables	These variables affect the output and state variables, but cannot be controlled. Examples are outside air temperature or feedstock properties.
Manipulated variables	These are input variables that affect output and state variables, and can be deliberately changed to meet production goals. Examples are valve positions, pump speeds, and flow ratios.

Category	Description
State (internal) variables	These variables are affected by exogenous and manipulated variables, but cannot be directly controlled. Examples are distillation column tray temperatures, specific reaction rates, and fermentation cell densities.
Output variables	These are variables are affected by exogenous, manipulated, and state variables. They are to be predicted by the optimization model. Usually, the output variables have specific target values that are either pushed near constraints, or held at a set point.

The following figure illustrates the relationship between these variables:



To solve the optimization problem, two models must be developed:

- The first model predicts the state variables as a function of the exogenous and manipulated variables.
- The second model predicts the output variables as a function of state, exogenous, and manipulated variables.

During the optimization, when NOL Studio considers hypothetical changes to the manipulated variables, the first model is used to predict the corresponding changes to the state variables, given the current measured values of exogenous variables. The second model is then used to predict the value of the output variables, corresponding to the predicted state variables, the hypothetical values of manipulated variables, and current measured values of exogenous variables. The resulting values of all variables are used to evaluate the objective function.

Developing an Optimization Model

The process for developing an optimization model is quite similar to the process of developing a predictive model. To create an optimization model, you follow the steps in the modeling wizard.

To launch the wizard:

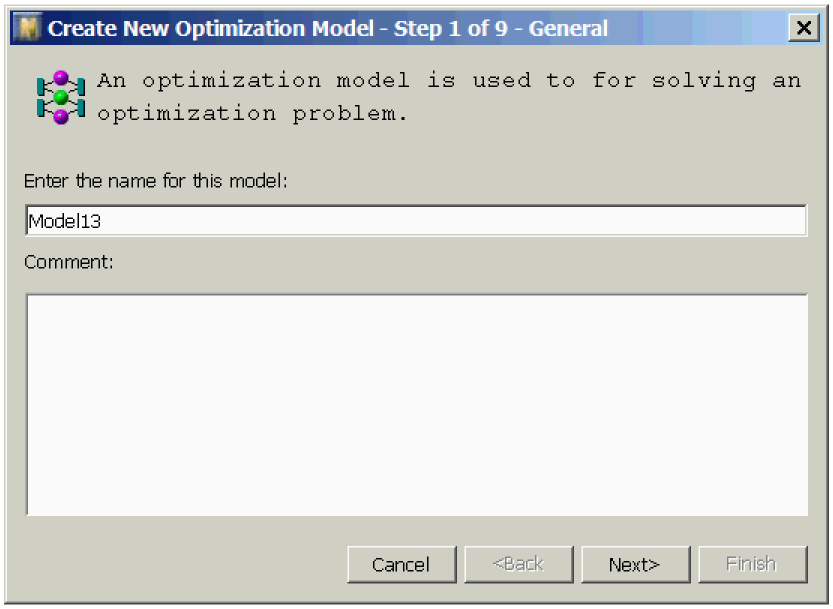
➔ Do one of the following:

Menu Bar:	Choose Object > New > Optimization Model.
Toolbar:	Click the New Optimization button.
Tree View:	Right-click the Optimization Models node and choose New from its menu.

The Create New Optimization Model wizard appears to guide you through the steps necessary to create the optimization model.

Naming the Model

The first panel in the wizard prompts you to specify a name for your model, and enter a comment. Use the comment to help you remember what data was used for training, how the data was preprocessed, and other special characteristics of the model.


Create New Optimization Model - Step 1 of 9 - General

An optimization model is used to for solving an optimization problem.

Enter the name for this model:

Model13

Comment:

Selecting the Preprocessor

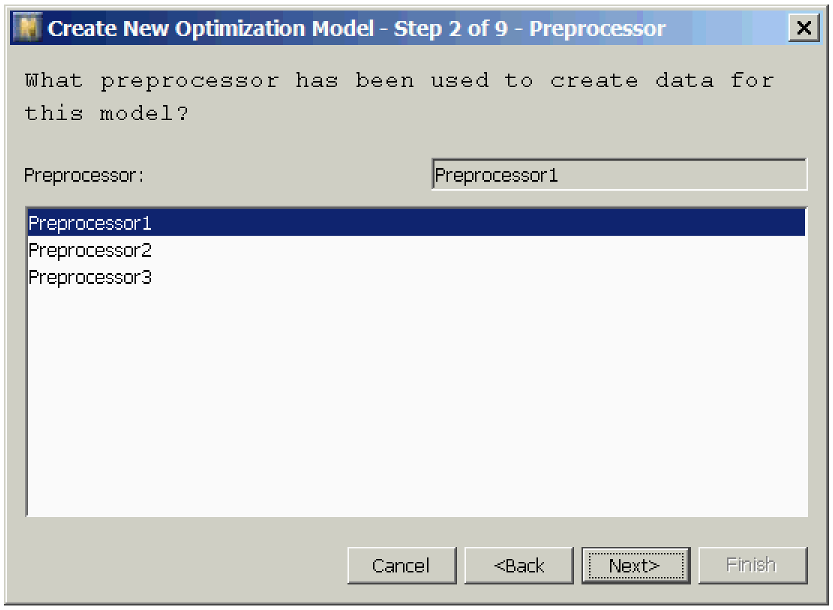
In the second step of the wizard, specify the preprocessor that provides the training data.

Each model requires a specific preprocessor as the source of training data. The preprocessor must contain all variables used in your model. You cannot take data from more than one preprocessor, and you cannot train a model on raw data. If you do not want to preprocess the raw data, simply create a preprocessor with no filter, and no formulas, and use this preprocessor as the data source for your model.

To select the preprocessor:

➔ Click on the desired preprocessor.

For example:



What preprocessor has been used to create data for this model?

Preprocessor: Preprocessor1

Preprocessor1
Preprocessor2
Preprocessor3

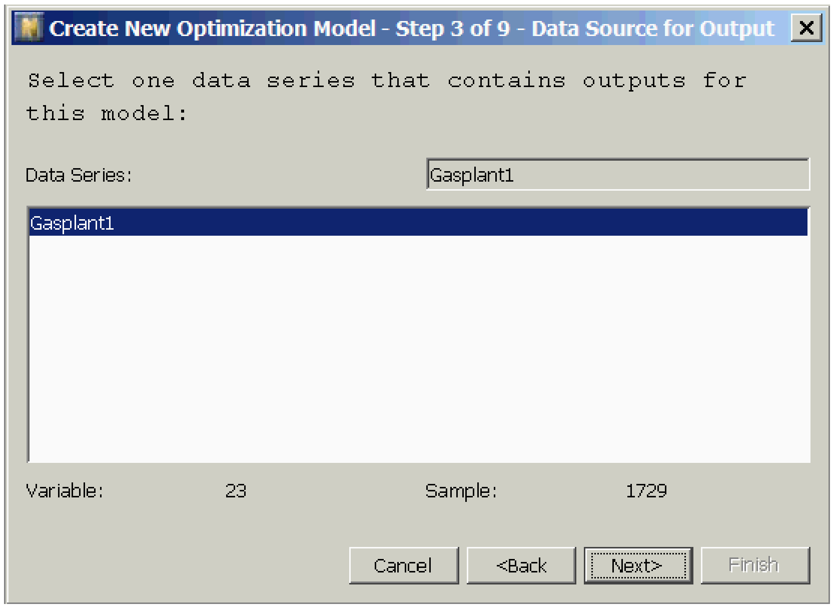
Selecting the Output Data Series

Next, you select the data series containing the output variable or variables. Only variables associated with the selected data series can be outputs of your model. However, both inputs and outputs can be contained in the selected data series. If you want to model variables from different data series, you must create different models.

To select data series:

➔ Click on the desired data series.

For example:



Create New Optimization Model - Step 3 of 9 - Data Source for Output

Select one data series that contains outputs for this model:

Data Series: Gasplant1

Gasplant1

Variable: 23 Sample: 1729

Cancel
<Back
Next>
Finish

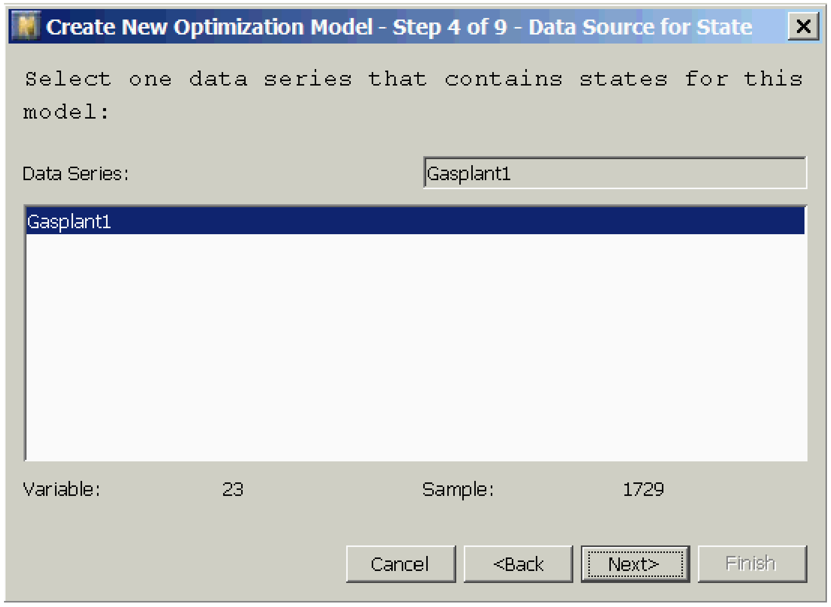
Selecting the State Variable Data Series

Next, you select the data series containing the state variables. Only variables associated with the selected data series can be state variables of your model.

To select data series:

➔ Click on the desired data series.

For example:


Create New Optimization Model - Step 4 of 9 - Data Source for State
Cancel
<Back
Next>
Finish

Select one data series that contains states for this model:

Data Series: Gasplant1

Gasplant1

Variable: 23 Sample: 1729

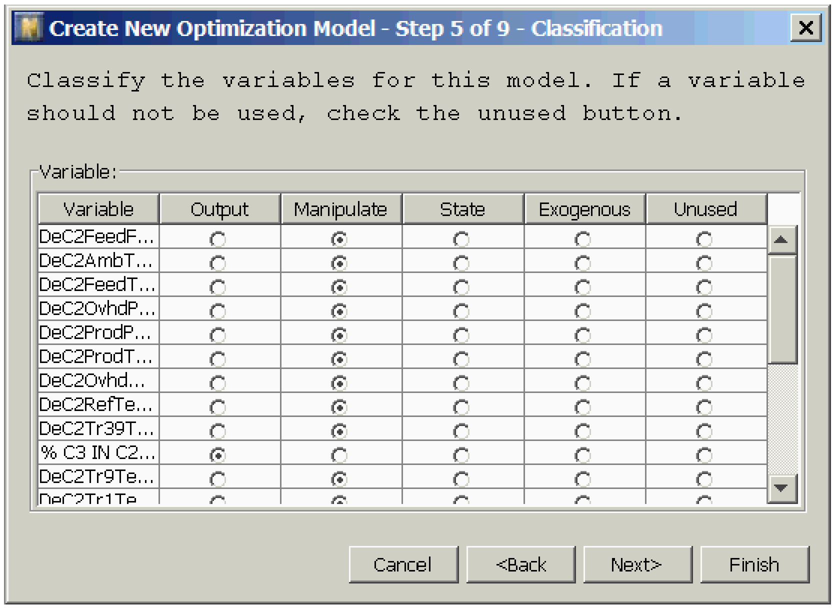
Classifying Variables

In the fifth step of the wizard, you specify which variables will be exogenous, manipulated, state, and output variables, and which variables are not used in the model. The list of variables include all variables in the selected data series, including derived variables defined by the preprocessor formulas.

To classify the variables:

➔ Click the appropriate radio buttons.

For example:



Create New Optimization Model - Step 5 of 9 - Classification

```
Classify the variables for this model. If a variable
should not be used, check the unused button.
```

Variable:

Variable	Output	Manipulate	State	Exogenous	Unused
DeC2FeedF...	○	●	○	○	○
DeC2AmbT...	○	●	○	○	○
DeC2FeedT...	○	●	○	○	○
DeC2OvhdP...	○	●	○	○	○
DeC2ProdP...	○	●	○	○	○
DeC2ProdT...	○	●	○	○	○
DeC2Ovhd...	○	●	○	○	○
DeC2RefTe...	○	●	○	○	○
DeC2Tr39T...	○	●	○	○	○
% C3 IN C2...	●	○	○	○	○
DeC2Tr9Te...	○	●	○	○	○
DeC2Tr1Te	○	●	○	○	○

Specifying Time Delays

Specification of time delays is needed for both state and output models. When specifying delays in the state model, the delays are the time lag between the inputs (manipulated and exogenous variables) and the state variables. When specifying the delays for the output variables, the delays are the time lags relative to the output variables.

You cannot specify delays for row-based data.

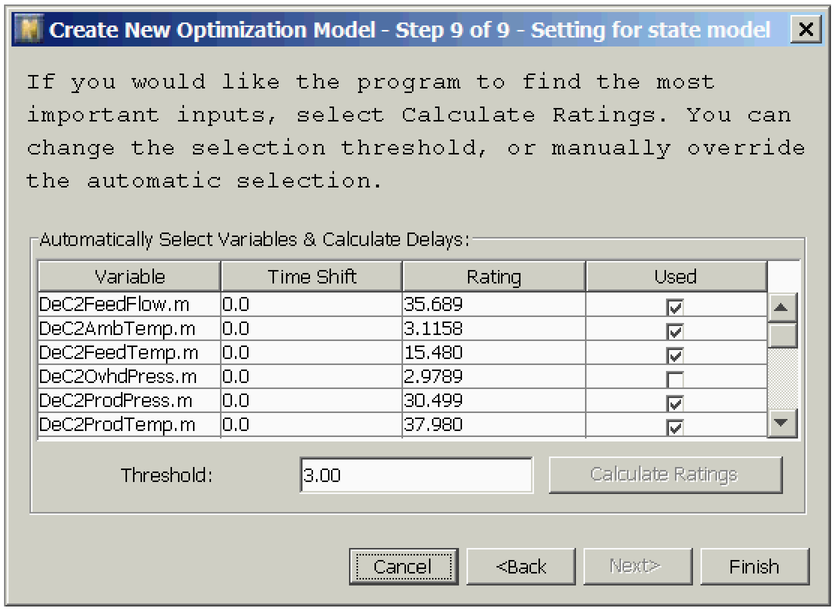
For more information on specifying time delays, see Specifying Time Delays.

Automatic Selection of Inputs and Delays

At this stage, you have selected a set of tentative input variables, possibly including some delayed variables, for both state and output variables. NOL

Studio will automatically select the model inputs and optimal delays from your tentative input set, in the same manner as for predictive models, as described in Automatic Selection of Inputs and Delays.

Here is the dialog for rating and final specification of inputs to the state model:



Create New Optimization Model - Step 9 of 9 - Setting for state model

If you would like the program to find the most important inputs, select Calculate Ratings. You can change the selection threshold, or manually override the automatic selection.

Automatically Select Variables & Calculate Delays:

Variable	Time Shift	Rating	Used
DeC2FeedFlow.m	0.0	35.689	☑
DeC2AmbTemp.m	0.0	3.1158	☑
DeC2FeedTemp.m	0.0	15.480	☑
DeC2OvhdPress.m	0.0	2.9789	☐
DeC2ProdPress.m	0.0	30.499	☑
DeC2ProdTemp.m	0.0	37.980	☑

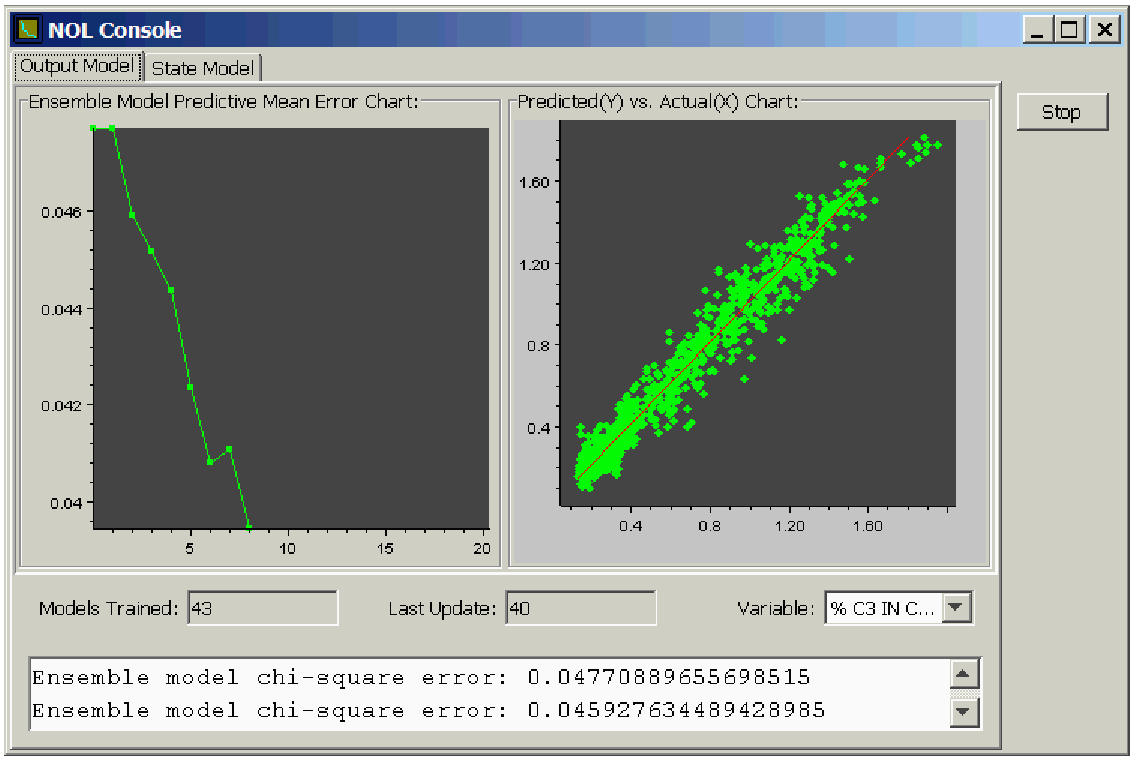
Threshold: 3.00

After this step, When you exit the wizard, you will be prompted to select a training time. When you train the model, be sure you specify adequate time, since internally, two models are being trained.

The Training Console

When you train an optimization model, the training console is slightly different than the console for a predictive model. In this console, there are two tabs, one for the state model and one for the output model. You can track the progress of either model by switching between tabs.

This is an example of the console:



Models Trained: 43
Last Update: 40
Variable: % C3 IN C...

```
Ensemble model chi-square error: 0.04770889655698515
Ensemble model chi-square error: 0.045927634489428985
```

The Optimization Objective Function

The following objective function, which NOL Studio minimizes, is defined with respect to the outputs, state variables and manipulative variables, *z*:

$$F(\overrightarrow{z_i}) = \sum_i^N f_i(z_i)$$

subject to:

$$LB_i < z_i < UB_i$$

where LB_i and UB_i are the lower and upper bounds of z_i, respectively, where *i* ranges over all of the variables, excluding the exogenous variables.

The function $f_i(z_i)$ is defined as:

$$f_i(z_i) = w_{i0} z_i + w_{i1}(z_i - z_{isp})^2 + w_{i2}\max(0, z_i - SUB_i)$$
$$+ w_{i3}\max(0, SLB_i - z_i)$$

in which the w_i are user-defined weights, z_{isp} is the user-defined setpoint of variable *i*, and SLB_i and SUB_i are the soft lower bounds and soft upper bounds of z_i, respectively.

There are four user-defined weights for each variable:

- A linear weight, w_0, which multiplies the value of the variable. If w_0 is positive, the variable will tend to be driven to its lower bound. If it is negative, the variable will tend to be driven to its upper bound.
- A quadratic weight, w_1, which multiplies the deviation from a variable's setpoint. Usually, this weight is non-zero only for output variables, whose values you want to control.
- Two linear weights, w_2 and w_3, which penalize violation of soft upper and lower bounds, respectively. The contribution of these weights is zero if the variable is not in the region between the soft bound and the hard bound.

Any weight can be positive or negative, but typically, all weights are positive except w_0, which can be either positive or negative.

Creating an Optimization Problem

During creation of an optimization problem, you will be prompted to fill out values for the weights. Generally, you determine the targets for each output variable, and constraints on inputs. It is the task of the optimization routine to find the best setpoints for the input variables. You only need to give NOL Studio initial values for the variables to start the calculation.

The setpoints that NOL Studio determines are optimal will always be between the hard lower and upper bounds that you set for each variable, provided there is a feasible solution. These limits may correspond to the physical limitations imposed by the equipment, such as a valve or thermostat. NOL Studio also uses both upper and lower soft bounds. Violation of the soft bounds imposes a cost.

If you know something about the solution to the optimization problem, you can speed the computation by providing a starting point close to a solution. Usually the current operating point is a good starting point.

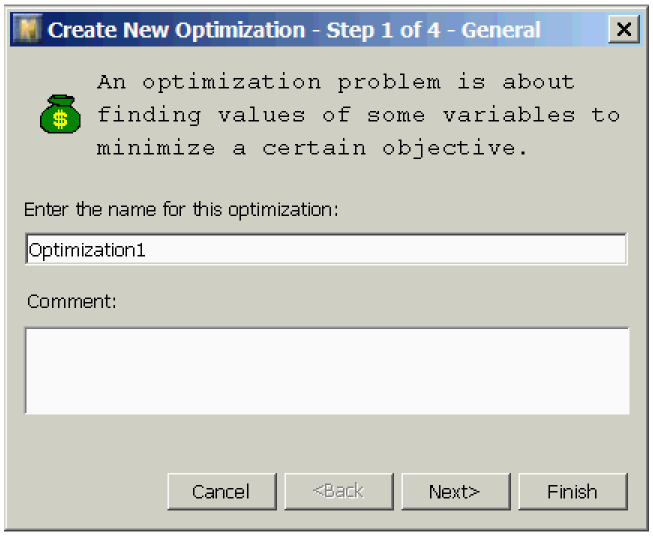
Using the Optimization Wizard

To create an optimization:

➔ Do one of the following:

Menu Bar:	Choose Object > New > Optimization.
Toolbar:	Click the New Optimization button.
Tree View:	Right-click the Optimization node and choose New from its menu.

The Create New Optimization wizard appears, for example:


Create New Optimization - Step 1 of 4 - General

An optimization problem is about finding values of some variables to minimize a certain objective.

Enter the name for this optimization: Optimization1

Comment:

To configure an optimization:

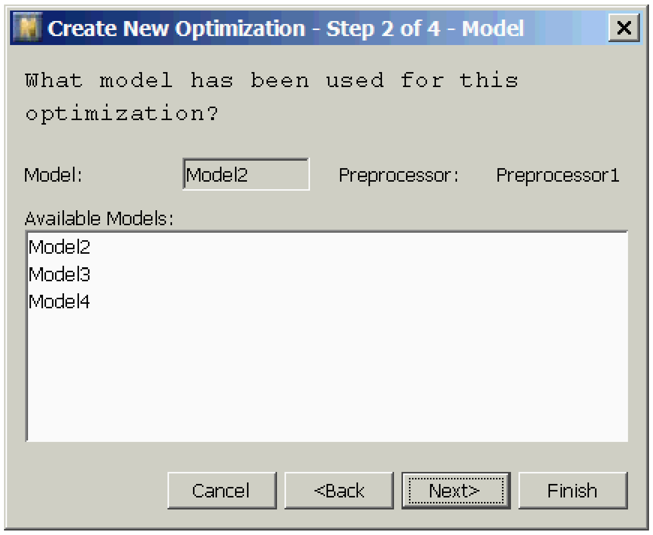
1. Enter a name for this optimization.

   Every new optimization command creates an optimization object. The default name is Optimization*N*, where *N* is a count of the number of optimizations you have thus far created.

2. Enter a comment associated with this optimization to remind yourself of any information you may wish to remember in the future.

3 Click Next and specify the model to be used in creating your optimization.

For example:



Create New Optimization - Step 2 of 4 - Model

What model has been used for this optimization?

Model: Model2 Preprocessor: Preprocessor1

Available Models:

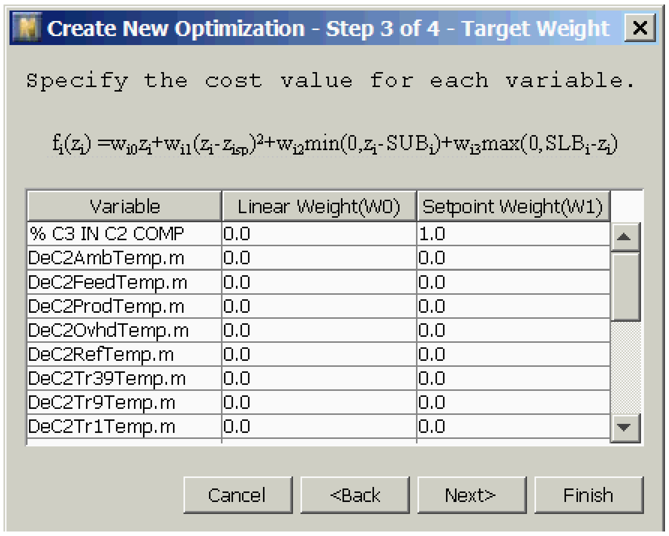
Model2
Model3
Model4

4 Click Next and define the first two adjustable weights for the optimization.

The initial values for all the linear weights are 0.0. The initial setpoint weights for outputs are 1.0 and 0.0 for inputs.

You can enter new weights in this panel and the weights can be changed later from the optimization property workspace. The values for linear weights can be either positive or negative. The weights for setpoints should almost always be positive.

For example:



Create New Optimization - Step 3 of 4 - Target Weight

Specify the cost value for each variable.

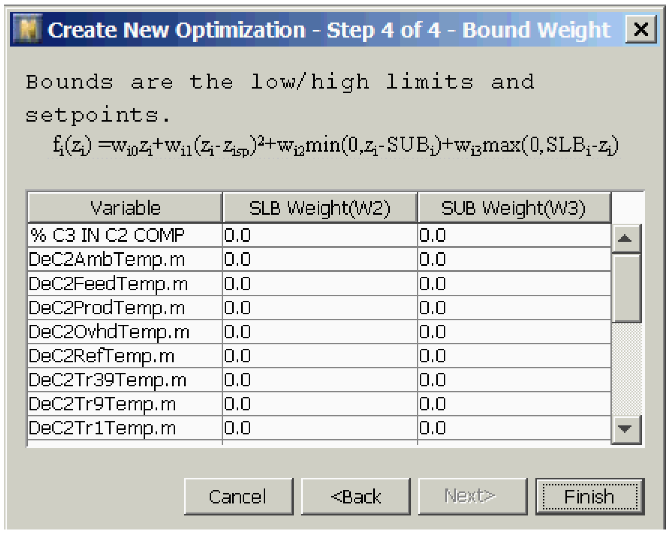
$$f_i(z_i) = w_{i0}z_i + w_{i1}(z_i - z_{isp})^2 + w_{i2}\min(0, z_i - SUB_i) + w_{i3}\max(0, SLB_i - z_i)$$

Variable	Linear Weight(W0)	Setpoint Weight(W1)
% C3 IN C2 COMP	0.0	1.0
DeC2AmbTemp.m	0.0	0.0
DeC2FeedTemp.m	0.0	0.0
DeC2ProdTemp.m	0.0	0.0
DeC2OvhdTemp.m	0.0	0.0
DeC2RefTemp.m	0.0	0.0
DeC2Tr39Temp.m	0.0	0.0
DeC2Tr9Temp.m	0.0	0.0
DeC2Tr1Temp.m	0.0	0.0

5 Click Next to define the final two adjustable weights for the optimization.

Again the weights entered through this panel can be changed later from the optimization property workspace.

For example:



Create New Optimization - Step 4 of 4 - Bound Weight

Bounds are the low/high limits and setpoints.

$f_i(z_i) = w_{i0} z_i + w_{i1}(z_i - z_{isp})^2 + w_{i2}\min(0, z_i - SUB_i) + w_{i3}\max(0, SLB_i - z_i)$

Variable	SLB Weight(W2)	SUB Weight(W3)
% C3 IN C2 COMP	0.0	0.0
DeC2AmbTemp.m	0.0	0.0
DeC2FeedTemp.m	0.0	0.0
DeC2ProdTemp.m	0.0	0.0
DeC2OvhdTemp.m	0.0	0.0
DeC2RefTemp.m	0.0	0.0
DeC2Tr39Temp.m	0.0	0.0
DeC2Tr9Temp.m	0.0	0.0
DeC2Tr1Temp.m	0.0	0.0

6 Click Finish to complete the creation of the optimization object.

Once you have created an optimization, you can access it from the tree view.

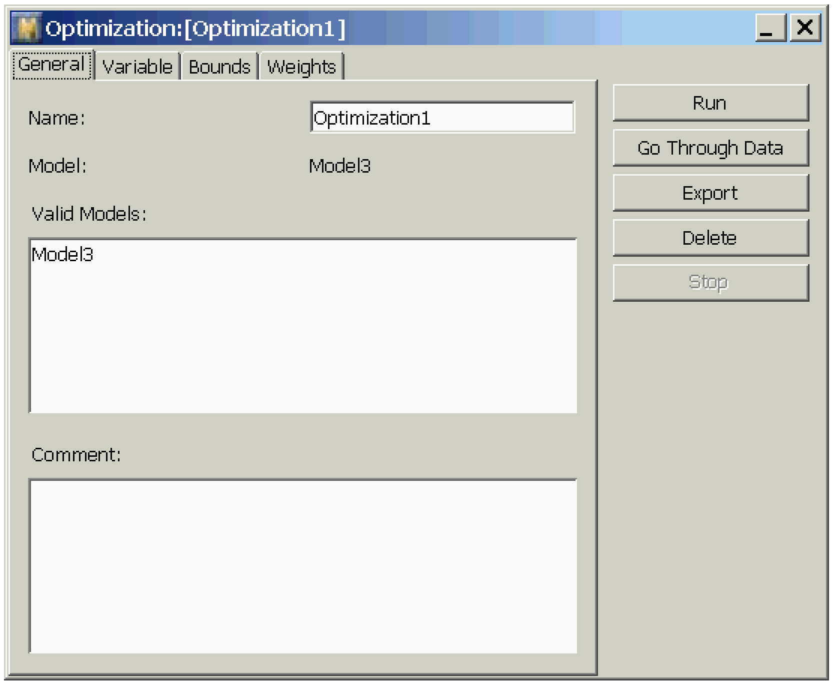
To display the optimization property workspace from the tree view:

➔ Do one of the following:

- Double-click the name of the optimization object.
- Right-click the name of the optimization object and choosing Go To from its menu.

The optimization property workspace provides general information and access to the values of weights and bounds, which include upper and lower limits, soft upper and lower limits and setpoints.

For example:



Name: Optimization1

Model: Model3

Valid Models:

Model3

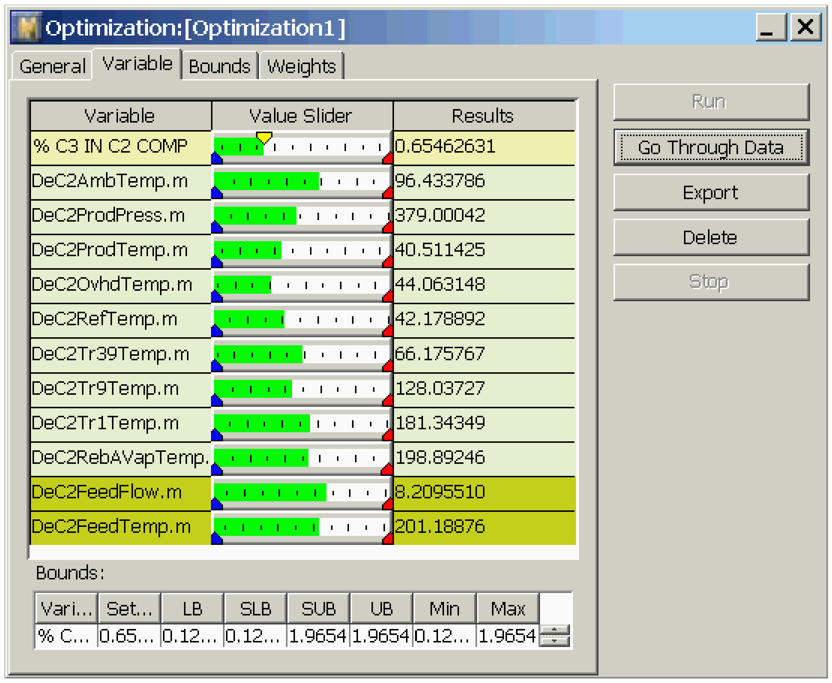
Comment:

Upon any changes to the optimization settings, the Run button becomes active and can be re-run to compute a new set of setpoints. If no changes are made the Run button remains disabled and the previously computed setpoint is still valid. You can change the name and comment of this optimization by typing new strings in corresponding text fields.

Running an Optimization

To calculate the optimization, click Run in the optimization workspace.

The following example shows the variable panel of the optimization property workspace. This panel allows you to select a variable and modify that variable’s Lower Bound (LB), Upper Bound (UB), Soft Lower Bound (SLB), and Soft Upper Bound (SUB).



Variable	Value Slider	Results
% C3 IN C2 COMP		0.65462631
DeC2AmbTemp.m		96.433786
DeC2ProdPress.m		379.00042
DeC2ProdTemp.m		40.511425
DeC2OvhdTemp.m		44.063148
DeC2RefTemp.m		42.178892
DeC2Tr39Temp.m		66.175767
DeC2Tr9Temp.m		128.03727
DeC2Tr1Temp.m		181.34349
DeC2RebAVapTemp.		198.89246
DeC2FeedFlow.m		8.2095510
DeC2FeedTemp.m		201.18876

Bounds:

Vari...	Set...	LB	SLB	SUB	UB	Min	Max
% C...	0.65...	0.12...	0.12...	1.9654	1.9654	0.12...	1.9654

The light-yellow background indicates the state variables. The dark-yellow background indicates the output variables, with all other variables being manipulated variables. The exogenous variables will be indicated by white background color. The initial value for each variable is its mean. The range of the slider is defined by the Min and Max and is originally derived from the data set.

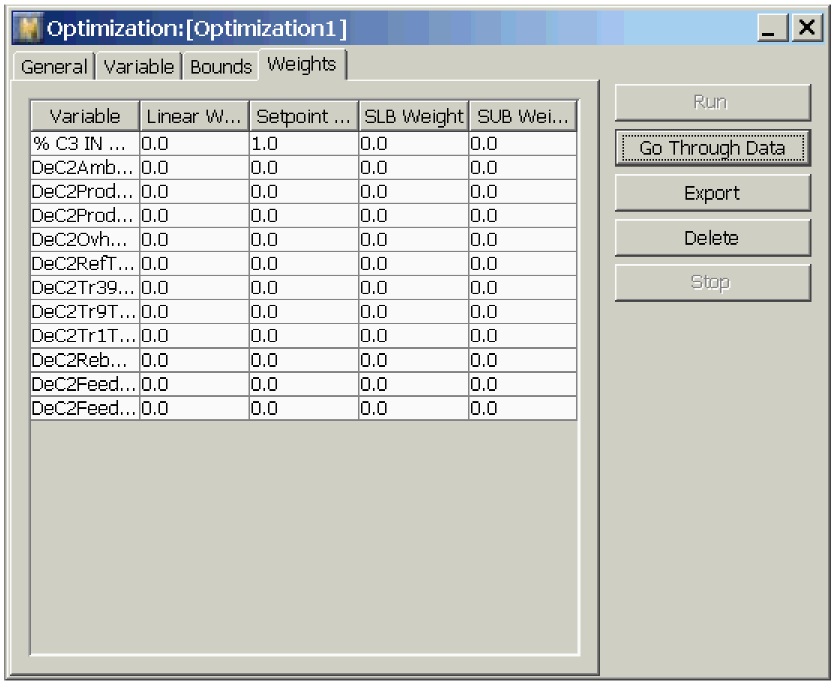
You can change the setpoints and limits in two ways:

- Use the mouse to move the thumb for each limit or setpoint in the variable slider. While moving the slider you will note that setpoint changes.
- Enter the value for each limit or setpoint in the variable bounds table. You select to show the values of the bounds for one variable by click on the variable row in the variable list.

After calculation, the results will appear in the variable list. The results for manipulated variables are the best setpoints for these variables. The results for state variables are the values based on the setpoints of manipulated variables and the results for outputs are the achieved best values from the manipulated and state variables

The results normally show by green color bars. If the any result for a variable reaches the upper bound, the color bar will turn dark gray. If the result reaches lower bound, the color bar will turn into red color.

The Bounds tab panel in the optimization property workspace allows you to edit/modify weights that you entered previously for the limits and setpoints. For example:



Variable	Linear W...	Setpoint ...	SLB Weight	SUB Wei...
% C3 IN ...	0.0	1.0	0.0	0.0
DeC2Amb...	0.0	0.0	0.0	0.0
DeC2Prod...	0.0	0.0	0.0	0.0
DeC2Prod...	0.0	0.0	0.0	0.0
DeC2Ovh...	0.0	0.0	0.0	0.0
DeC2RefT...	0.0	0.0	0.0	0.0
DeC2Tr39...	0.0	0.0	0.0	0.0
DeC2Tr9T...	0.0	0.0	0.0	0.0
DeC2Tr1T...	0.0	0.0	0.0	0.0
DeC2Reb...	0.0	0.0	0.0	0.0
DeC2Feed...	0.0	0.0	0.0	0.0
DeC2Feed...	0.0	0.0	0.0	0.0

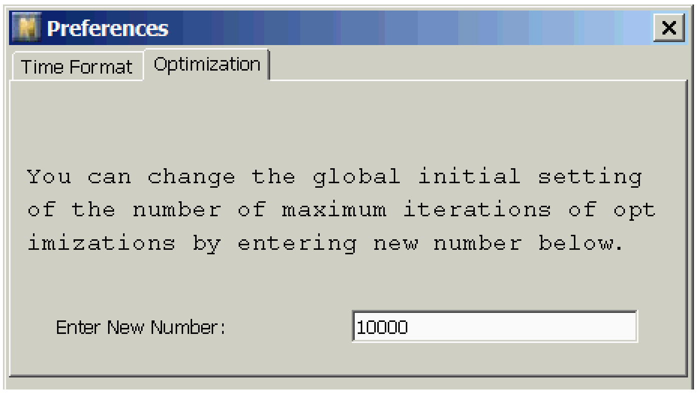
Initial Condition and Error Handling in Optimization Calculation

A automatic approach for setting the initial condition ensures that the initial condition is within the hard lower and upper bounds. The default values for the initial condition are the mean values of every variable from the data series. If any hard bound is changed and any value of the initial condition is outside its hard bounds, this value is discarded in favor of the mean of the upper and lower hard bounds.

If an error occurs during optimization, a status code is returned after the calculation is called.

Maximum Iterations

When an optimization calculation reaches the maximum number of iterations, a warning dialog appears. When this occurs, you can increase the maximum iteration and allow the optimization to continue. In this instance, the new number is only used for the current optimization; it is not used as a global preference.



You can change the global initial setting of the number of maximum iterations of opt imizations by entering new number below.

Enter New Number: 10000

Running through an Existing Data Set

You can run the optimization with one particular setting through an existing data series. The values of output variables in the data series serve as proposed setpoints for outputs. The values of exogenous variables are those in the data series and the values of manipulated and state variables are not used. The results are saved in a data series, which you can access from the tree view and the Go To option of the View menu.

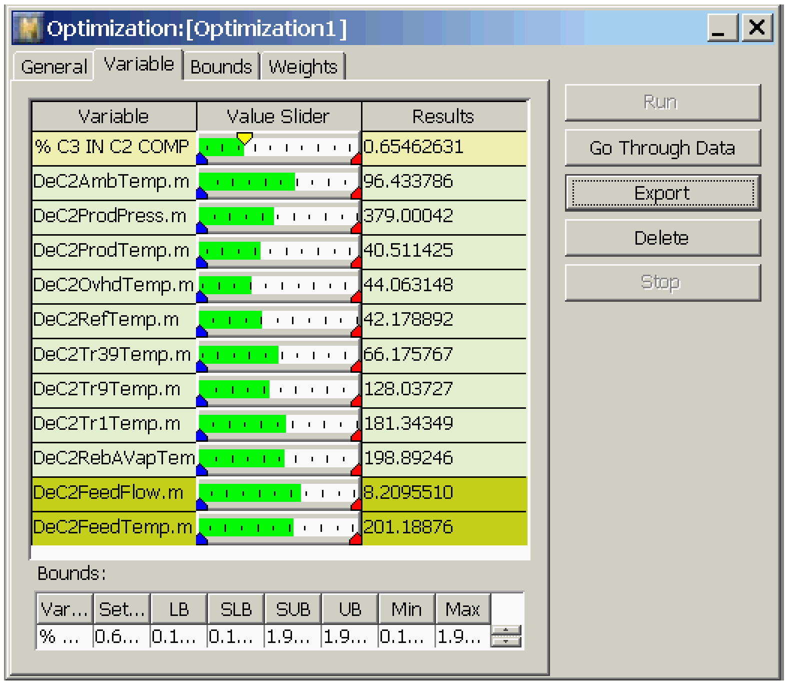
Note This will be a time-consuming process, please be patient.

Saving an Optimization

After you test the settings of the optimization problem to your satisfaction, you can export the objective function and settings as an optimization object for on-line use as an Active X control or with the G2 API, click Export on the right side of the Properties dialog. This will export the objective function and the settings of bounds and weights into a file. For more information on deploying an optimization object, see Model Deployment.

To export the optimization:

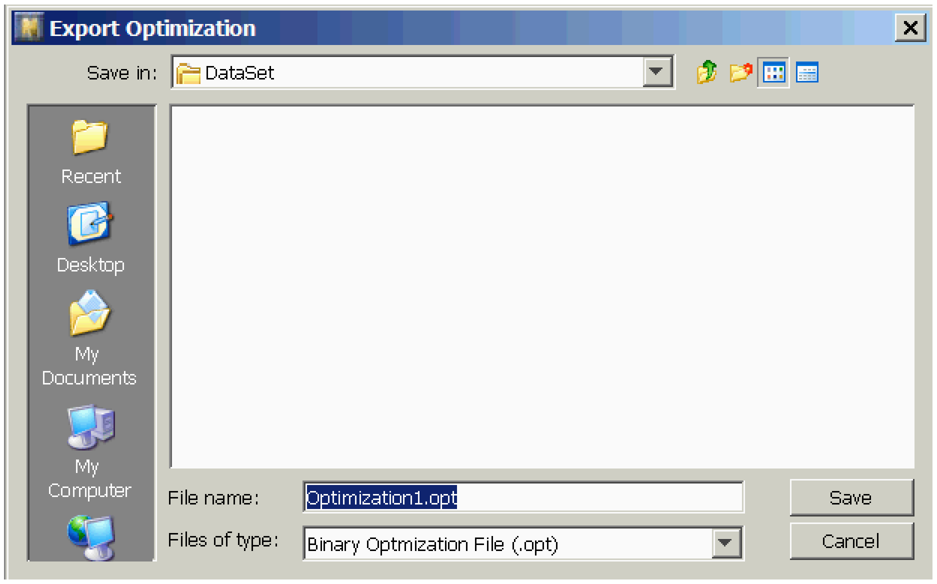
1 From the Optimization Properties workspace, click Export.



Variable	Value Slider	Results
% C3 IN C2 COMP		0.65462631
DeC2AmbTemp.m		96.433786
DeC2ProdPress.m		379.00042
DeC2ProdTemp.m		40.511425
DeC2OvhdTemp.m		44.063148
DeC2RefTemp.m		42.178892
DeC2Tr39Temp.m		66.175767
DeC2Tr9Temp.m		128.03727
DeC2Tr1Temp.m		181.34349
DeC2RebAVapTem		198.89246
DeC2FeedFlow.m		8.2095510
DeC2FeedTemp.m		201.18876

Var...	Set...	LB	SLB	SUB	UB	Min	Max
% ...	0.6...	0.1...	0.1...	1.9...	1.9...	0.1...	1.9...

2 Specify a path and file name in the dialog. The model will be saved in a *.opt* file.



Save in: DataSet

File name: Optimization1.opt

Files of type: Binary Optmization File (.opt)

Your optimization object is now ready for deployment.

Model Deployment

Describes how to export and deploy a predictive model in ActiveX and G2.

Introduction

When you have trained and validated your model to satisfaction, you can deploy it for online use. There are two environments where deployment can occur:

- COM (Component Object Model)
- G2

COM is Microsoft's standard component interface specification. Many databases, historians, DCSs, and applications such as Microsoft Office applications, Visual Basic, Delphi, OSI's Process Book, Aspen Technology's InfoPlus.21, and Honeywell's PHD are COM compliant. Any application that is COM compliant can be executed on the native Windows NT, Window 2000, or Window XP platform. You may choose to use this environment for the deployment of the NOL model if you have a ready source of data compliant with COM.

G2 is Gensym's flagship real-time expert system. Combined with the Gensym Neural Network Engine and integrated with tools such as NOL Studio, G2 makes it easy to configure online deployment, acquire data, and control the context of applying the neural network. G2 provides the capability of reading data from virtually any source through its family of bridge products. It also provides a whole range of products for data processing, from basic data filters to applications such as Optegrity.

G2 is a development environment, as well as a real-time expert system. You may choose to write an application in G2 and embed the deployed model there. In doing so, you will combine G2's powerful inference engine, rule-based analysis, and GNNE's neural network prediction and optimization capabilities into one whole entity - your application.

Your Choice of deployment environment will be based on many factors. In addition, you should consider the following: end-user interface, location and transfer of data, pre- and post-processing of data, and the larger application which may contain the deployed model as an embedded control. However, no matter what your choice is, you will find the deployment of your model easy and smooth.

Exporting Your Model

Once you have a trained and validated your model, one of the options is to export it, then to load it into your deployment environment. You can export the model in several ways:

- For model deployment with a preprocessor, you can export the model to a *.mod* file. You can load this file directly into a COM environment or into a G2 environment, using JavaLink.
- You can export the model weights into an ASCII file, and load the weights into neural network block in GNNE or NOL Classic environment. For information on how to do this, see Saving a Model.
- You can connect the NOL Studio and G2, and export models to GNNE environment through procedure calls.

Note You can deploy a predictive model in both COM and G2 environments. You can deploy a backpropagation net, autoassociative net, radial basis function net, and rho net in Gensym Neural Network Engine or NeurOn-Line Classic.

Note Once the model is exported, you cannot make any changes to the architecture of the model or to the associated preprocessor. You can update the weight parameters by retraining the model with new data series.

Deploying Your Model in ActiveX

NOLOnline, the deployment component of NOL Studio, is shipped with two ActiveX controls, NOLPredictor and NOLOptimizer. The NOLPredictor control is the one you will use to deploy predictive models in your ActiveX applications. The NOLOptimizer control is for deploying optimizations, described later.

When you install NOL Studio on your machine, the ActiveX controls are not automatically registered on your machine. The next section describes how to register and use the controls. All of the code examples are performed in Visual Basic; however, this control can be used in any COM-compliant environment.

Registering the ActiveX Control

When you are ready to deploy a predictive model in the ActiveX environment, you first need to register the NOLOnline ActiveX controls on the machine where the deployment will take place.

To register the ActiveX Controls on Windows NT platforms:

1 Make sure that your machine has been rebooted after NOLStudio has been installed.

2 Choose Start > Programs > Gensym G2 2011 > G2 NeurOn-Line > Register G2 NeurOn-Line Control.

A dialog appears to inform you that the controls have been registered successfully.

Now, you are ready to use the controls in an ActiveX container application.

Note The Install Controls command registers both NOLPredictor and NOLOptimizer ActiveX controls.

Using NOLPredictor in Visual Basic

Before you can deploy your NOL model, you need to add the NOLOnline ActiveX controls to the application toolbox.

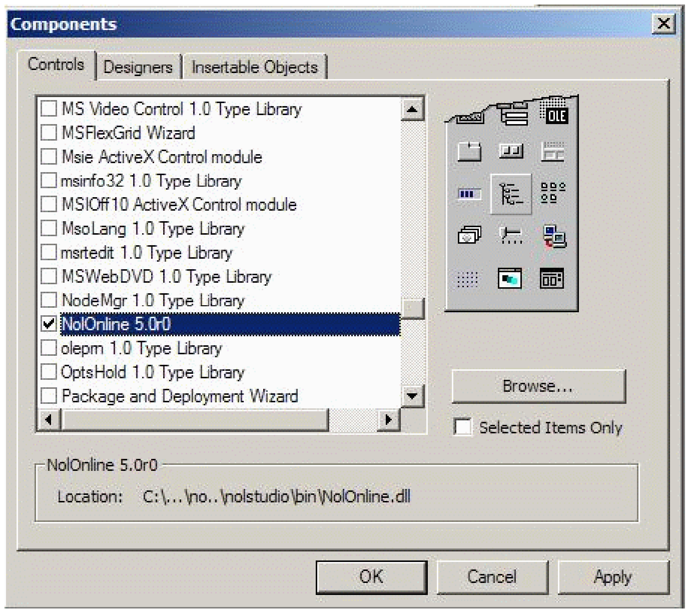
To add the controls to component toolbox:

1 Start a Visual Basic application, or start Visual Basic with a new project.

2 Right-click on the Toolbox and choose Components from its menu.

3 Select NolOnline 5.1r0 from the list of controls and click OK.

For example:



MS Video Control 1.0 Type Library
MSFlexGrid Wizard
Msie ActiveX Control module
msinfo32 1.0 Type Library
MSIOff10 ActiveX Control module
MsoLang 1.0 Type Library
msrtedit 1.0 Type Library
MSWebDVD 1.0 Type Library
NodeMgr 1.0 Type Library
[✓] NolOnline 5.0r0
olepm 1.0 Type Library
OptsHold 1.0 Type Library
Package and Deployment Wizard

Selected Items Only

NolOnline 5.0r0
Location: C:\...\no..\nolstudio\bin\NolOnline.dll

Both the *NOLPredictor* and the *NOLOptimizer* components are added to the Toolbox.

Loading the NOL Model

The NOLPredictor control that you have added to your Toolbox is not the actual model that you exported from NOL Studio. It is a generic component capable of loading and running any NOL model. To use the control with your model, first you need to place the control on a form, then you need to load the model. You can programmatically call the LoadModel method on the NOLPredictor instance to load the model with specified path and file name.

To create a NOLPredictor instance:

1 Clone the NOLPredictor control and place it on the form. The control is invisible at run-time, and appears as a string on your form.

2 Examine the Property Window of the control. You need to name your NOLPredictor control (e.g. “gasplant”), although VB will provide a default name, such as NOLPredictor1.

To load the exported NOL predictive model:

1. Double-click on the form to show the code-window for the form.
2. In the Form_Load subroutine, write the following code:

   `gasplant.loadModel` *file, path*

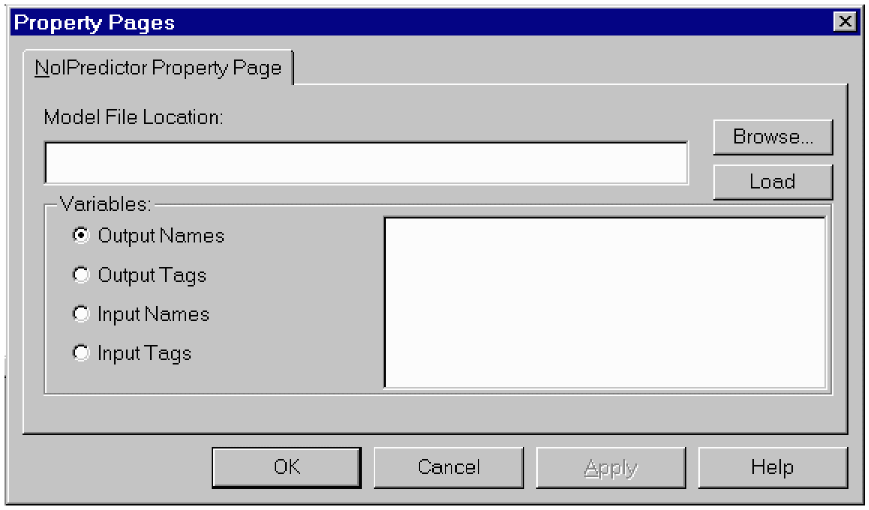
   where `gasplant` is the name of your control, and *path* and *file* are the two arguments to the method, pointing to the location where the `.mod` file is stored.

You can also load the model at design time, to verify that you have the correct one, for example.

To verify the model:

1. Select the NOLPredictor object in your form.
2. Choose View->Property Pages to display its property page, or click on the Custom design property.

   The following dialog appears.

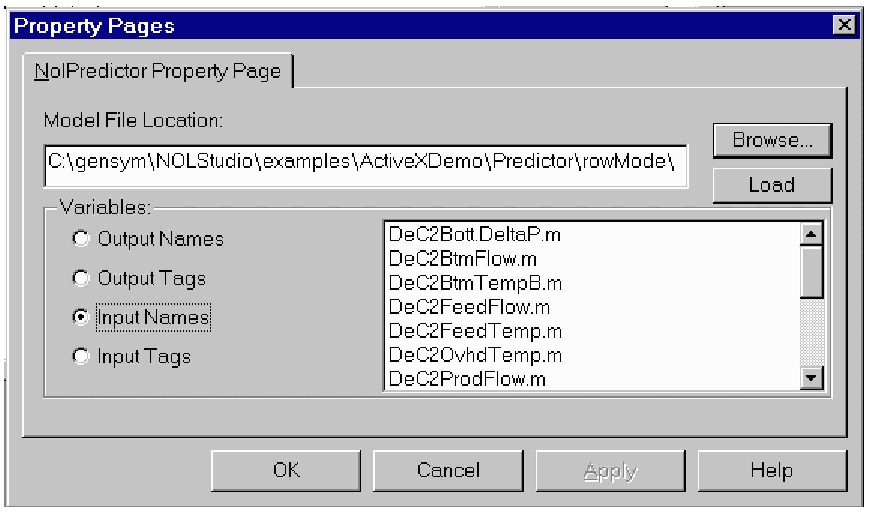


3. Click the Browse button to display the File Load dialog.
4. Locate and select your model, and then click the Open button.

   The model file name and path appear in the Model File Location input box.
5. Click the Load button to load the model from this location.

You may now look at the variable names of the model itself in the property pages of the control.

By clicking any of the Variables radio buttons, you can display the needed information, stored in your model. Keep in mind that the names and tags should appear in the same order as in the trained model.



Model File Location: C:\gensym\NOLStudio\examples\ActiveXDemo\Predictor\rowMode\

Variables:
Output Names
Output Tags
[✓ Input Names]
Input Tags

DeC2Bott.DeltaP.m
DeC2BtmFlow.m
DeC2BtmTempB.m
DeC2FeedFlow.m
DeC2FeedTemp.m
DeC2OvhdTemp.m
DeC2ProdFlow.m

Note We recommend using the *loadModel* method to load the model parameters during run time, rather than using the properties dialog to load model during design time.

In the ActiveX demo provided in NOL bundle CD-ROM, a Visual Basic application shows all of the previous actions. The loading of the model and displaying some of its properties can also be done at run-time with some simple code. Here is an example of code that performs the these actions on an instance of an NOLPredictor named gasplant.

```
Private Sub Form_Load()
    gasplant.loadModel "gasplant.mod""gasplant.mod", _
    "c:\gensym\g2-2011\nolstudio\examples\ActiveDemo"
    outputs = gasplant.getNumberOfOutputs()
    inputs = gasplant.getNumberOfInputs()
    MsgBox "The number of inputs =" & inputs & ", the number of "
        outputs = " & outputs"
    MsgBox "The first input name = " & gasplant.inputNames(0)
    MsgBox "The first output name = " & gasplant.outputNames(0)

End Sub
```

Running the Model in ActiveX

The basic mechanism by which you run the model consists of three steps:

1 Provide the model with input values.

2 Calculate the outputs.

3 Request output values.

Data Input

Look in the Visual Basic Object Browser to examine the methods associated with the NOLPredictor control. You will see several methods for data input:

```
setTimeFormat(TimeFormat As String)

setInputAtTimeByIndex(index As Long, InputValue As Double, Time As
String)

setInputAtTimeByName(name As String, InputValue As Double, Time As
String)

setInputAtTimeByTag(Tag As String, InputValue As Double, Time As
String)

setInputsAtTime(Inputs, Time As String)

setInputForRowByIndex(index As Long, InputValue As Double)

setInputForRowByName(InputName As String, InputValue As Double)

setInputForRowByTag(Tag As String, InputValue As Double)

setInputsForRow(Inputs)
```

The first method sets the time format for all other methods that require a time stamp as one of the inputs. The next four methods set the input values to the model for a specific time stamp. The last four methods allow you to send input values to a row-based model for a specific row. You can find detailed descriptions of these methods at the end of this section.

There are also methods for setting data input for recursive models. Use following methods to set data into the output variable buffer for the predictor with a recursive model.

```
setVariableValueAtTimeByName(name As String, InputValue As Double, Time
As String)

setVariableValueAtTimeByTag(tag As String, InputValue As Double, Time
As String)
```

Before you send any time values to the NOLPredictor, you must set a time format, which specifies the form of the time stamp. For example:

```
gasplant.setTimeFormat "M/d/y H:m:s"
```

If you wish to specify a different time format, you can call the setTimeFormat method at any time.

Caution Your time format must include both date and time information, otherwise incorrect values or exceptions may occur around midnight. If a12-hour clock is used, the format code "a" (for am/pm) must be specified, or incorrect values may occur around noon or midnight.

Think of setting an input as sending an array of input values and a time stamp to the model. For example, if your model has 4 inputs: Oven1Temperature, Oven2Temperature, Oven3Temperature, CookieTemperature, here are the lines of code you would need to set the inputs for a specific time stamp:

```
Dim inputs As Variant
Dim inarray(0 to 3) as Double

CookieModel.setTimeFormat "mm/dd/yy hh:mm:ss"
inarray(0) = 350.23 'Oven1Temperature value
inarray(1) = 370.42 'Oven2Temperature value
inarray(2) = 310.99 'Oven3Temperature value
inarray(3) = 367.54 'CookieTemperature value

'write the inputs array to the variant
inputs = inarray

CookieModel.setInputsAtTime inputs, "3/12/97 10:31:45"
```

Note that the array you pass to the method inputs must be of type Variant, but that the values must be written to an array of Doubles. This is why you need to include the line:

```
inputs = inarray
```

For row based data, no time stamps are needed. However, be aware that you can only set one row of data at a time. In other words, you set all the inputs for a specific row, then calculate the output for this row. Next, set all the inputs for another row and calculate its output. If you do not calculate the output between row sets, then the next time you set the input data, it will override the data you had set previously.

Here is an example for the same Cookie Model, if it was row based:

```
Dim inputs As Variant
Dim inarray(0 to 3) as Double

inarray(0) = 350.23 'Oven1Temperature value
inarray(1) = 370.42 'Oven2Temperature value
inarray(2) = 310.99 'Oven3Temperature value
inarray(3) = 367.54 'CookieTemperature value
```

```
'write the inputs array to the variant
inputs = inarray

CookieModel.setInputsForRow inputs
```

Should you wish to set just one input value, then you have to use the methods setInputAtTime, or setInputForRow. These methods are a little trickier to use, since you need to understand how the Index parameter works. The Index is the order-number of the input variable as it was trained in the model, starting at zero. So, for example, if your model has the same 4 inputs: Oven1Temperature, Oven2Temperature, Oven3Temperature, and CookieTemperature; then the Index of Oven1Temperature is 0, and the Index of Oven3Temperature is 2. Here is a code example that would perform the same functionality as shown above, but setting one input at a time:

```
CookieModel.setTimeFormat "mm/dd/yy hh:mm:ss"
' 350.23 is Oven1Temperature value
' 370.42 is Oven2Temperature value
' 310.99 is Oven3Temperature value
' 367.54 is CookieTemperature value

CookieModel.setInputAtTimeByIndex 0, 350.23, “3/12/97 10:31:45“
CookieModel.setInputAtTimeByIndex 1, 370.42, “3/12/97 10:31:45“
CookieModel.setInputAtTimeByIndex 2, 310.99, “3/12/97 10:31:45“
CookieModel.setInputAtTimeByIndex 3, 367.54, “3/12/97 10:31:45“
```

Similarly, if the CookieModel predictive model, accepted row-based data, you could set the inputs one at a time as follows:

```
' 350.23 is Oven1Temperature value
' 370.42 is Oven2Tempeature value
' 310.99 is Oven3Temperature value
' 367.54 is CookieTemperature value

CookieModel.setInputForRowByIndex 0, 350.23
CookieModel.setInputForRowByIndex 1, 370.42
CookieModel.setInputForRowByIndex 2, 310.99
CookieModel.setInputForRowByIndex 3, 367.54
```

The question is, when would you set data in a complete array, vs. a single value? The methods that allow you to send the full array of input values to the model are extremely useful if you already have your data in a spreadsheet. For example, in Microsoft Excel you can specify a “range” of values, which could be a row of inputs. You could write that range to a Variant data type, and send it directly to the model. On the other hand, if your data is coming in asynchronously from some process, and you would like to send the values to the model as they come in, then it makes sense to use the methods that allow you to set one input at a time.

Testing Whether the Model is Time-Based Model

Models built from time-based data require different method calls for setting inputs. You can use a method call to find out whether the model you load is time-based model.

```
isTimeBasedModel() As BOOLEAN
```

Calculating Output

As mentioned in the previous section, for row-based data, you need to calculate output prior to setting inputs for the next row. For time based data, you can calculate output on request. Let’s examine the API.

For row-based data:

```
calculateOutputsForRow() As Variant
```

For time-based data:

```
calculateOutputsAtTime(Time As String) As Variant
```

For auto-recursive models, which use delayed output variables as model inputs, you can set the time to a time in the future, as long as you have enough historical data to perform a prediction for the first unavailable outputs. The predictor automatically finds the last available data samples, calculates intermediate output values, and feeds them back into the model as inputs until the calculated timestamp reaches the time specified by the method’s input parameter.

Obtain Results

Both of these methods return a Variant data type, which holds an array of values. These values are of type Double, and they are in the order of outputs specified in the trained model. So, for example, if your model returns two outputs, CookieMoisture and ColorIndex, your code may look like this:

```
Dim outputs as Variant
Dim counter as Integer

outputs = CookieModel.calculateOutputsAtTime ( “3/12/97 10:31:45“)
For counter = 0 to CookieModel.NumberOfOutputs - 1
   MsgBox "the output " & counter & " is: " & outputs(counter)
Next counter
```

Once you have the output value, you can write it back to your spreadsheet, or plot it, or process it according to your application.

Clear Data Buffer

Models built from time-based data rely on a data buffer to store data from specific periods of time in order to construct the input data. The data buffer uses a queue structure. The data are stored in the buffer in the order of their timestamps. You can only push new data sample with its timestamp later than the earliest data sample into the buffer. If you need to push earlier data into the buffer, you need to clear the data buffer first.

```
clearDataBuffer()
```

Training Predictive Models in Real Time

In a real-time application, you may want to continue performance improvement of an existing model as new data is collected. The new data may come from real-time changes in target process such as slow deactivation of a catalyst, mechanical wear of components, and change in raw materials properties.

Learning should happen in two situations:

- Process enters new X regime, so new areas of the same function Y = f (X) can be learned. For this case, you should continue the training from where it is stopped and present the data series containing both new and old data to the model for training.
- The function itself changes, so Y=FNew(X) does not equal FOld(X) in some portion of the X space. In such a case, you should start the training after initializing the model and use new data only.

You can trigger the training of a predictive model in the ActiveX environment if you detect changes in your process.

Note You cannot change the model input and output structure, or the contents in the preprocessor. The online training just adjusts the model parameters.

Note For additional API methods of the NOLPredictor class and an explanation of all methods, see Appendix A, NOLPredictor Class.

Deploying Your Model in G2

You can deploy your model in G2 in one of three ways:

- Using Gensym Neural Network Engine (GNNE)
- Using NeurOn-line Classic
- Using G2 JavaLink

You can deploy all types of model except optimization models in GNNE or NOL Classic. Deployment through G2 JavaLink is only for predictive and optimization models and involves using G2, G2 JavaLink, and `nolstudio.kb`. The following section describes how to deploy models in G2 JavaLink and NOL Classic. For information on deploying models in GNNE, see the *Gensym Neural Network Engine*

Deploying Your Model in NOL Classic

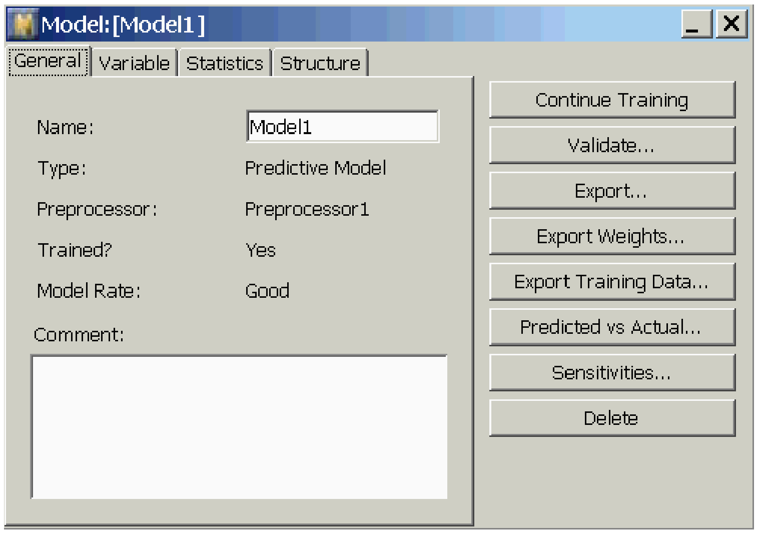
NOL Studio allows you to build all four types of neural network models in NOL Classic. The Backpropagation net, Autoassociative net, Radial Basis Function net, and Rho net have direct mappings between these two packages. You can export the predictive model weights and load them into an Ensemble net block, which you access through the Neural Networks palette.

To use the deployment option with NOL Classic, you must be licensed to use NeurOn-Line Classic. This section assumes knowledge of G2 and the NeurOn-Line block language.

Caution For all model types, data preprocessing steps (formulas and delays) are not implemented by the network blocks in NOL Classic. Only the model itself is imported into G2. If you specified time delays in your model, you must implement them by using time delay blocks in your G2 application. Likewise, in NOL Classic, preprocessor formulas must be implemented in the NOL block language. You can also use blocks from G2 Diagnostic Assistant (GDA).

Exporting Your Model as a Weight File

To use the G2 deployment option, first save your model as a weight file. To save a model as a weight file, go to the model's property table, and select the Export Weights button, as shown below:



Name:	Model1
Type:	Predictive Model
Preprocessor:	Preprocessor1
Trained?	Yes
Model Rate:	Good
Comment:	

This will save your model as a text file, which contains a description of your model that you can load into G2 via the network blocks in GNNE or NOL Classic.

Using the NOL Model

When NOL is loaded, you have access to the Neural Net blocks that represent NOL Studio models. You implement your model by cloning this block from the palette and configuring it.

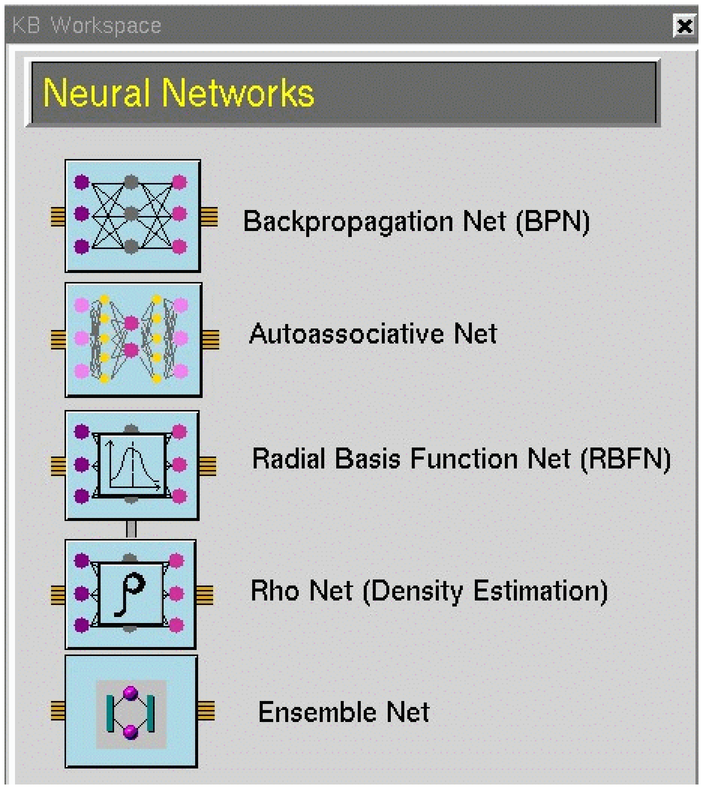
To load and use the Backpropagation net, Autoassociative net, Radial Basis Function net, and Rho net blocks, see the description of each of these blocks in the *NeurOn-Line Reference Manual*.

The following section describes how to use the Ensemble Model block in NOL Classic.

Note The Ensemble Model block is included on the Neural Net Blocks palette. There is no separate palette for NOL Studio blocks.

To create and configure the NOL Studio Ensemble Model block:

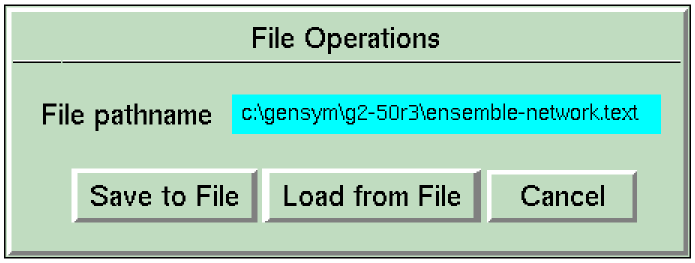
1 From the NOL top level menu bar, choose Palettes > Neural Networks > Neural Net Blocks to display this palette:


KB Workspace
Neural Networks
Backpropagation Net (BPN)
Autoassociative Net
Radial Basis Function Net (RBFN)
Rho Net (Density Estimation)
Ensemble Net

2 Click the Ensemble Net block. A new instance of the block will be attached to the mouse.

3 Move the block to the desired location on the workspace which you will be creating your diagram. Click to place the block on the workspace

4 Choose file operations from the block's menu to display this dialog:



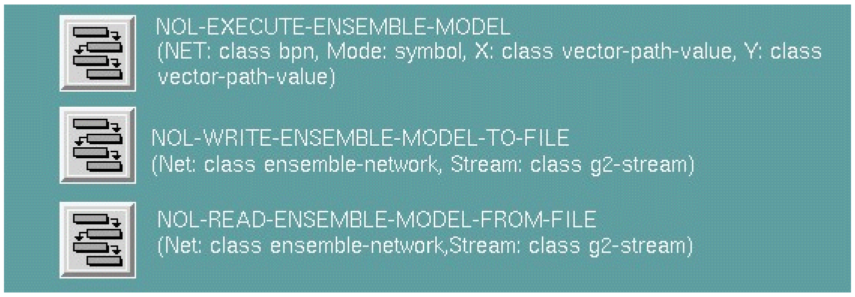
c:\gensym\g2-50r3\ensemble-network.text

5 Enter the filename containing your model, and select Load from File.

When you complete the last step, the block loads the information from the designated model file. You are now ready to provide your model with data, and to calculate the results.

Loading Models Programmatically

You can programmatically load information in the model from a file or save it to a file, using API procedures. To display these procedures, choose Main Menu > Get Workspace > Nol API, and click Neural Networks API.



NOL-EXECUTE-ENSEMBLE-MODEL
(NET: class bpn, Mode: symbol, X: class vector-path-value, Y: class vector-path-value)

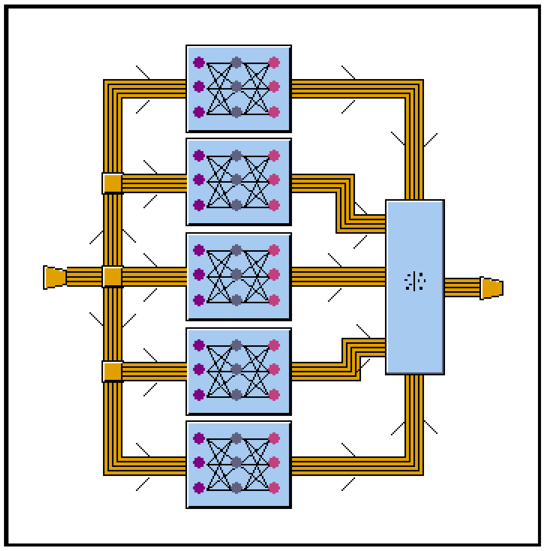
NOL-WRITE-ENSEMBLE-MODEL-TO-FILE
(Net: class ensemble-network, Stream: class g2-stream)

NOL-READ-ENSEMBLE-MODEL-FROM-FILE
(Net: class ensemble-network,Stream: class g2-stream)

Examining Your Model

Default NOL Studio predictive models are ensemble models, composed of several submodels, whose outputs are combined to make the final prediction. The combination operator is the median of the prediction of the submodels. In the NOL Classic block language, this is implemented as an encapsulation block.

For example, this is the diagram for the encapsulation block:



You can open the subworkspace by selecting view diagram from the block's menu choices. Then you can examine each submodel by selecting configure from the subblock's menu choices. Consider this information to be read-only; do not make any changes to the block's configurations.

Saving a Model In Your KB

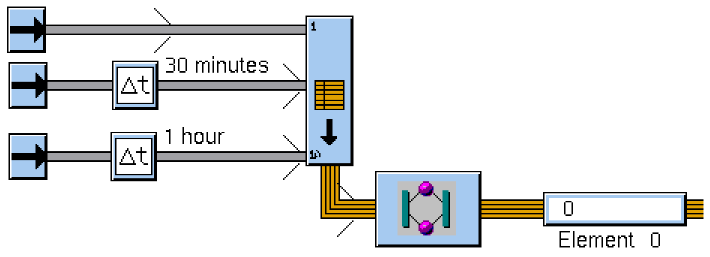
If a NOL block you create is used as part of a diagram, that block will be saved with the rest of your diagram when you save the KB file. However, the weights will not be stored as a permanent part of the KB unless you select the block's make permanent menu choice. If you neglect to make the weights permanent, and you attempt to run the model, an error will inform you that you must configure the block (see the Handling Errors section in the *NeurOn-Line User's Guide*).

Running the Model in G2

To run the model in NOL, you must connect the configured model to other blocks in NOL. When a vector of data is received by the block, it will pass the current output value of the model to the output vector path of the block. In addition, a model can be asked to produce a current output value based on the current inputs available, by manually evaluating the block. The *NeurOn-Line User's Guide* contains detailed information on creating, connecting and running diagrams in the NOL Classic environment.

To determine the size and order of elements for the input vector, consistent with the number and order of inputs of the model, refer to the model's property table, in NOL Studio.

The following is a small example wherein three inputs are passed into the NOL Studio model. The first input has no delay, the second has 30 minutes delay, and the last input has 1 hour delay:



In this example, a Processor Block has been created and configured as outlined in the previous sections. To its input vector path, we have connected a Vectorizer. This is a standard NOL block which allows you collect scalar inputs into a vector. At the input of the vectorizer, we have connected time delay blocks to two of the inputs. Each input is receiving values from a scalar entry point. To the output of the Processor Block we have connected a Vector Path Display. This block is configured to display the first element (element 0) of the output vector of the Ensemble Model block.

Note No scaling blocks are required when you use an NOL Studio ensemble model block. Required scale factors are incorporated into the block itself.

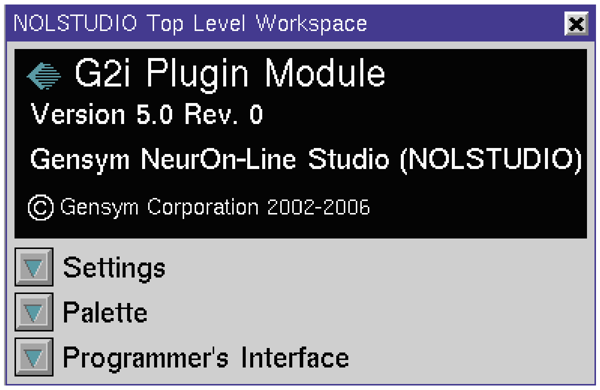
Deploying in G2 using G2 JavaLink

To deploy the NOL Studio model in G2 through G2 JavaLink, you must have a license for G2 JavaLink, which is part of the G2 Bundle. You also need to have JavaLink installed on your machine. Go to the Readme file of NOL bundle, and follow the directions for installing JavaLink. You will need to restart your machine before proceeding. This section assumes knowledge of G2 and JavaLink.

Loading the Necessary KBs

All the kbs are located in the `nol` directory. The top-level module is called `nolstudio.kb`. However, do not make `nolstudio.kb` the top level module in your application. Create a module, or a hierarchy of modules, as your application, then merge `nolstudio.kb` into your application, and make it a required module.

This is the top-level workspace of the *nolstudio.kb*:



NOLSTUDIO Top Level Workspace

G2i Plugin Module

Version 5.0 Rev. 0

Gensym NeurOn-Line Studio (NOLSTUDIO)

© Gensym Corporation 2002-2006

It provides a palette of objects such as the Predictive Model, the Optimization, the Module Settings, the PCA model, the PLS model, and the API for all types of models:

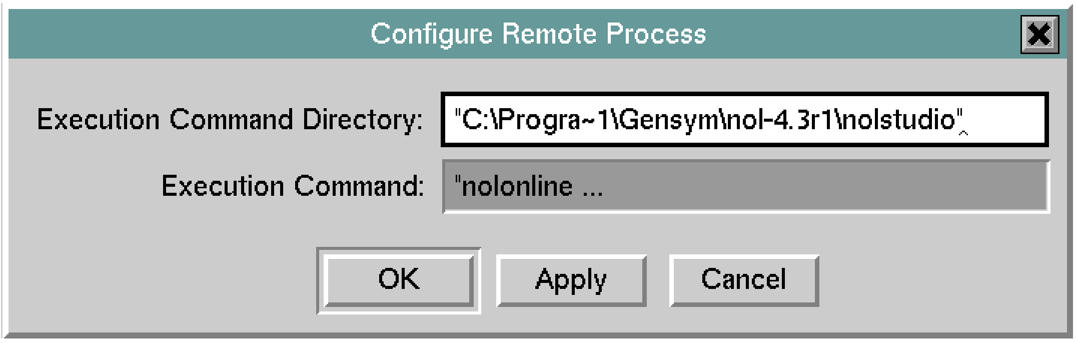
The basic process of using an NOL Studio model in G2 is to:

- Connect to an interface.
- Initialize the model.
- Send data to the model.
- Receive/request outputs.
- Retrain or build a model, if necessary.

Each of these steps is detailed in the following sections.

Launching a Remote Process at Startup

Before you can run your model in G2, you need to connect to a remote process, which performs the model calculations. To launch the remote process from the local machine at startup automatically, set the designated boolean parameter in the NOL Studio top-level workspace to true. The following dialog appears for configuring the path to your remote process.



Configure Remote Process

Execution Command Directory: "C:\Progra~1\Gensym\nol-4.3r1\nolstudio"

Execution Command: "nolonline ...

When you click OK, the remote process launches and is ready to use.

Launching a Remote Process Using Procedure

You can also launch the remote process, using a procedure. You provide the home directory, the listener port, and the name of the interface. To facilitate this process, the NolG2Gateway class is available in the *`nolstudio\com\gensym\nols\deploy`* directory. You can also use the module settings object to launch the remote process, using a procedure.

To set up module settings:

1. Start G2.
2. Clone a module settings object from the NOLStudio palette, and place it on a workspace in your top-level module, *not* *`nolstudio.kb`*.
3. Edit the table attributes of the module settings object as follows:

Attribute	Description
nols-studio-home-directory	The directory in which NOL Studio is installed, such as "c:\Progra~1\Gensym\g2-2011\nolstudio".
nols-remote-process-listener-port	The port that the NolG2Gateway class uses, which is 22044, by default.
nols-connection-timeout	A connection timeout, which is 10 seconds, by default.
nols-interface-object-name	The name of the nols-interface object that provides communication, which is a subclass of gsi-interface. This object is created transiently by the launch procedures and referenced by the initialization procedures.
nols-execution-command	The name of execution file to launch the interface.
others	Not used in this version.

4. Restart G2 to initialize the settings.

If you click the NOLS Programmers Interface button on the NOLStudio palette, then click the Remote Process Management Procedures button, you will see four procedures on a workspace:

- nols-launch-remote-process() = (float)
- nols-launch-remote-process-by-setting(settings:class nols-settings) = (float)
- nols-launch-remote-process-with-message(settings:class nols-settings, Client: class ui-client-item) = (float)
- nols-kill-remote-process ()

The first procedure uses the default nols-settings, and the next two use the setting you create for your application. You can create action buttons on a workspace in your module to start these procedures. When the remote process is started successfully, you should see the following message in the background window of G2:

```
CREATED NolG2Gateway, count = 1; connected to G2 OK.
```

If you terminate the remote process successfully, the following message should appear in the G2 background window:

```
G2 Connection has been closed.
```

Now, you are ready to set up a NOL Studio model in G2 and use it.

The Predictive Model and its API

To set up the predictive model you need to clone its icon off the NolStudio palette, and fill in its attributes. Then, use the API methods to send the input data to the model and receive the output values. The API is split up into three parts: the general methods - such as getting the name and comment of the model, the sets - such as setting the inputs by row or by time, by tag or by name of the variable, and the gets - such as calculating the outputs and receiving them into G2.

To set up a predictive model:

1. Make sure G2 is running, then clone the predictive model onto a workspace in your application. Make sure the workspace does not belong to the nolstudio module.
2. Edit the attribute table of the model to fill in the following attributes:
   a. Names: the name of the model; you will need to refer to it later
   b. Nols directory name: the directory of the model in quotes
   c. Nols file name: the file name of the model, such as “model1.mod”

Now, you are ready to write procedures using the predictive model API to send and receive data. Click on the Nols Programmers Interface button on the palette, then on the Methods for Predictive Models buttons. Examine the methods on this workspace.

Method to Initialize the Predictive Model

The first method you must call before you can do anything further is nols-initialize.

nols-initialize
(*model*: class nols-predictive-model, *interface*: class nols-gateway)

Initializes the predictive model. The *model* is an attribute of the predictive model. The *interface* is the nols-interface-object-name referenced in your module settings.

General Methods

The general methods do not send or receive actual model data.

nols-get-name
(*model*: class nols-predictive-model)
-> *model:* text

Returns the name of the model created in NOL Studio.

nols-get-comment
(*model*: class nols-predictive-model)
-> *comment:* text

Returns the comment associated with the model created in NOL Studio.

nols-get-number-of-inputs
(*model*: class nols-predictive-model)
-> *inputs:* integer

Returns the number of inputs in the model. You may need to call this method to resize your arrays.

nols-get-number-of-outputs
(*model*: class nols-predictive-model)
-> *outputs:* integer

Returns the number of outputs in the model. You may need to call this method to resize your arrays.

nols-get-input-names
(*model*: class nols-predictive-model)
-> *inputs*: class text-array

Returns an array of input names for the model.

nols-get-input-tags
(*model*: class nols-predictive-model)
-> *input-tags*: class text-array

Returns an array of input tags for the model.

nols-get-output-names
(*model*: class nols-predictive-model)
-> *output-names*: class text-array

Returns an array of output names for the model.

nols-get-output-tags
(*model*: class nols-predictive-model)
-> *output-tags*: class text-array

Returns an array of output tags for the model.

nols-get-input-units
(*model*: class nols-predictive-model)
-> *input-units*: class text-array

Returns an array of input units for the model.

nols-get-output-units
(*model*: class nols-predictive-model)
-> *output-units*: class text-array

Returns an array of output units for the model.

Methods to Send Input Data to a Model

These methods send input data to a model. There are two issues involved with these methods.

First, all of these methods will signal an error of class nols-error, if something goes wrong. To capture this error, you should enclose these methods in begin...end on error statements. The error has three attributes: class-of-error, description-of-error, and backtrace-of-error. All of these attributes are texts, which you can save to a log file, print to a message board or workspace, or set up gfr-error-handlers for.

Second, you need to know if your model is time-based or row-based. For time-based data, you need to set the time format of the time text-string that you will provide in the inputs for time based models.

nols-set-time-format
(*model*: class nols-predictive-model, *time-format*: text)

Sets the time format for a time-based model. For example:

```
call nols-set-time-format(model1, "M/d/y H:m:s).
```

You must set the format before calling the other methods that use time.

Methods to Set Input Values for Time-Based Models

For the following methods, you can choose to set the input values by index (the number of the variable, starting at zero), by name, by tag, or set all of the input values in one array. These are again, for time-based models only:

nols-set-input-at-time
(*model*: class nols-predictive-model, *index*: integer, *val*: float, *time*: text)

Sets a value of an input by its index. The index is the number of the variable in the input array, starting at zero.

nols-set-input-at-time-by-name
(*model*: class nols-predictive-model, *var-name*: text, *val*: float, *time*: text)

Sets a value of an input by its input name. If the name is incorrect, an error is signaled.

nols-set-input-at-time-by-tag
(*model*: class nols-predictive-model, *tag*: text, *val*: float, *time*: text)

Sets a value of an input variable by the variable's tag. If the tag is incorrect, an error is signaled.

nols-set-inputs-at-time
(*model*: class nols-predictive-model; *vals*: class float-array, *time*: text)

Sets all the input values at once. Make sure that the values are in the same order as the variable names or tags. This method is particularly useful if you are reading values from a file, where the variable values come in simultaneously.

nols-set-variable-value-at-time-by-name
(*model*: class nols-predictive-model, *name*: text, *val*: float, *time*: text)

Sets a value of a variable by the variable's name. If the name is incorrect, an error is signalled. You use this method to set the delayed output value for a recursive model. You can also use it to set the input value of a recursive model.

nols-set-variable-value-at-time-by-tag
(*model*: class nols-predictive-model, *tag*: text, *val*: float, *time*: text)

Sets a value of a variable by the variable's tag. If the tag is incorrect, an error is signalled. You can use this method to set the delayed output value for a recursive model. You can also use it to set values for input variables of a recursive model.

Method to Test Whether it is a Time-Based Model

nols-has-time-stamps
(*model*: class nols-predictive-model)
-> *timestamps:* truth-value

Determines whether a particular model is a time-based or a row-based model. This method returns true if the model is time-based, false if it is row-based.

Methods to Set Input Values for Row-Based Models

For row-based models, these methods set the inputs by index, by name, by tag, or as an array.

nols-set-input-for-row
(*model*: class nols-predictive-model; *index*: integer, *val*: float)

Sets a value of an input by its index. The index is the number of the input variable in the inputs array, starting at zero.

nols-set-input-for-row-by-name
(*model*: class nols-predictive-model; *var-name*: text, *val*: float)

Sets the input variable value by the variable name. If the name is incorrect, an error is signaled.

nols-set-input-for-row-by-tag
(*model*: class nols-predictive-model; *tag*: text, *val*: float)

Sets the input variable value by the variable tag. If the name is incorrect, an error is signaled.

nols-set-inputs-for-row
(*model*: class nols-predictive-model; *vals*: class float-array)

Sets all the inputs at once. Make sure that the input values order corresponds to the order of input names or tags in the model.

Methods to Calculate Outputs

These methods calculate the outputs for both the time-based and the row-based models. The output of these methods is a float-array with the output values in order of the outputs specified in the model. These methods will also signal an nols-error if such occurs.

nols-calculate-outputs-at-time
(*model*: class nols-predictive-model; *time*: text)
-> *outputs:* class float-array

Calculates the outputs for a time-based model. For a time based model, you can request a time that does not necessarily correspond to a specific input. In that case, the output values will interpolated between the closest time steps.

nols-calculate-outputs-for-row

(*model*: class nols-predictive-model)
outputs: class float-array

Calculates the outputs for a row based model. You must call this method immediately after you set all the inputs for a row which you want to calculate. If you set a particular input more than once, without calculating the output between the sets, the calculate method will take the latest input values for the row.

Training Predictive Model at Run Time

These methods train predictive models at run time. The common parameters in these APIs are *time*, *auto-stop*, *initial-training*, and *training-display*. The *time* tells the training algorithm how long you want to spend to train the model. The *auto-stop* indicates whether you want the training algorithm to stop training automatically if no further improvement is detected. The *initial-training* indicates whether you want to clear the model and start new training, or to continue training from last stop state. The *training-display* indicates whether to show the training console during the training.

nols-train-model

(*model*: class nols-predictive-model, *xmatrix* : sequence , *ymatrix*: sequence, *time*: float, *auto-stop*: truth-value, *initial-training*: truth-value, *training-display*: true-value)

Trains the model online with pre-formatted input and output matrices. The sequence type for *xmatrix* and *ymatrix* is a sequence of float-arrays. The column number of *xmatrix* must be the same as the input number with each delay, if applicable, served as a input. The column number of *ymatrix* must be the same as output number.

This method is used to train a predictive model when the data are collected and formatted inside G2 and pass it through a nols-gateway interface.

nols-train-model-from-file

(*model*: class nols-predictive-model , *input-file*: text, *output-file*: text, *time-in-minutes*: float, *auto-stop*: truth-value, *initial-training*: truth-value, *training-display*: true-value)

Trains a predictive model from a data file. Use this method when you have saved the data to a file that contains numerical values only. The format of the file must be a comma separated ASCII file with the same numerical values at each line. The column number of the input data file must be the same as the input number with each delay, if applicable, served as an input. The column number of output data file must be the same as the output number.

nols-train-model-from-file

(*model*: class nols-predictive-model , *data-series-files*: sequence, *time-in-minutes*: float, *auto-stop*: truth-value, *initial-training*: truth-value, *training-display*: true-value)

Trains a predictive model from a set of data series files. Use this method when you have saved the data series to files with a fixed `.ds` format. The *data-series-files* contains a set of text strings, which represent to path of the data file. The data files need to contains all input and output variables. For time-based model, the output variables has to in one data file. For row-based model, the sequence should just contains one data file path.

Note Every training method initially sets the nols-complete-training attribute of the model to false, then spawns the training process through the nols-gateway. When the training finishes, nols-complete-training is set to true. If an error occurs during the training, the nols-has-error attribute of the model is set to true and an error message will be set to nols-error-message. The common arguments are *time-in-minutes*, *auto-stop*, *initial-training*, and *training-display*.

Parameter	Description
time-in-minutes	The number of minutes you want to spend to train this model.
auto-stop	Whether the training process automatically stops based on a converge criterion. When false, the training process continues to the end set by the *time* argument.
initial-training	When true, the model weights are initialized to a set of random numbers. When false, the training process starts with the existing model weights.
training-display	When true, a training console with error information is displayed during training.

Additional Methods for Predictive Model

nols-clear-data-buffer

(*model*: class nols-predictive-model)

Clears the data buffer of the online predictor. This method is used only for models developed from time based data.

nols-get-max-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the maximum statistic of the variable by its name. The returned value is the maximum value of that variable in the data series used to build the model.

nols-get-max-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the maximum statistic of the variable by its tag. The returned value is the maximum value of that variable in the data series used to build the model.

nols-get-min-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the minimum statistic of the named variable. The returned value is the minimum value of that variable in the data series used to build the model.

nols-get-min-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the minimum statistic of the named variable. The returned value is the minimum value of that variable in the data series used to build the model.

nols-get-mean-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the mean statistic of the named variable. The returned value is the mean value of that variable in the data series used to build the model.

nols-get-mean-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the mean statistic of the named variable. The returned value is the mean value of that variable in the data series used to build the model.

nols-get-median-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the median statistic of the named variable. The returned value is the minimum value of that variable in the data series used to build the model.

nols-get-median-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the median statistic of the named variable. The returned value is the median value of that variable in the data series used to build the model.

nols-get-sum-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the sum statistic of the named variable. The returned value is the sum of that variable in the data series used to build the model.

nols-get-sum-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the sum statistic of the named variable. The returned value is the sum of that variable in the data series used to build the model.

nols-get-std-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the standard deviation statistic of the named variable. The returned value is the standard deviation value of that variable in the data series used to build the model.

nols-get-std-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the standard deviation statistic of the named variable. The returned value is the standard deviation value of that variable in the data series used to build the model.

nols-get-variance-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the variance statistic of the named variable. The returned value is the variance value of that variable in the data series used to build the model.

nols-get-variance-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the variance statistic of the named variable. The returned value is the variance value of that variable in the data series used to build the model.

nols-get-kurt-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the kurt statistic of the named variable. The returned value is the kurt value of that variable in the data series used to build the model.

nols-get-kurt-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the kurt statistic of the named variable. The returned value is the kurt value of that variable in the data series used to build the model.

nols-get-skew-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the skew statistic of the named variable. The returned value is the skew value of that variable in the data series used to build the model.

nols-get-skew-value-by-tag
(*model*: class nols-predictive-model, *tag*: text)
-> *value:* float

Returns the skew statistic of the named variable. The returned value is the skew value of that variable in the data series used to build the model.

nols-get-range-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the range statistic of the named variable. The returned value is the range value of that variable in the data series used to build the model.

nols-get-range-value-by-name
(*model*: class nols-predictive-model, *name*: text)
-> *value:* float

Returns the range statistic of the named variable. The returned value is the range value of that variable in the data series used to build the model.

nols-get-formulas
(*model*: class nols-predictive-model)

Returns the string of formulas inside the preproccessor used for building the model.

Handle Error Exceptions

The following is a list of all methods that signal nols-error when an error occurs. Ensure that you encapsulate all of these method calls into begin-end on error statements to capture the errors.

nols-calculate-outputs-at-time
nols-calculate-outputs-for-row
nols-set-time-format
nols-set-inputs-at-time
nols-set-inputs-for-row
nols-set-input-at-time
nols-set-input-at-time-by-name
nols-set-input-at-time-by-tag
nols-set-variable-value-at-time-by-name
nols-set-variable-value-at-time-by-tag
nols-set-input-for-row-by-name
nols-set-input-for-row-by-tag
nols-has-time-stamps

The Statistical Models and their API

Two statistical model types exist in the NolStudio deployment environment. They are the Principal Component Analysis (PCA) model and the Partial Least Squares (PLS) model.

Building Online Statistical Models

You can build PCA or PLS models in NolStudio, export them into files, and load into G2 for execution, or you can build PCA and PLS in G2 using online data. To do this, you need to launch the NolStudio remote process. After the remote process is established, you should initialize the statistical calculator for model training process. If you load the model parameter from the text file exported from NolStudio, you don’t need to initialize the calculator.

If you have a NOL Studio connected to G2, you can directly export the PCA and PLS parameters into a model object in G2. For detailed description, see “Partial Least Squares Model” and “Principal Component Analysis Model” in Chapter 3 of the *Gensym Neural Network Engine*.

Method for Initializing the Statistical Calculator

nols-initialize-statistical-calculator()

This method initializes an internal statistical calculator used for building PCA and PLS models.

Principal Component Analysis (PCA) Model

Principal components analysis (PCA) is a statistical technique applied to a set of variables to discover which sets of variables form coherent subsets that are relatively independent of one another. These subsets, principal components, are thought to be representative of the underlying processes that have created the correlations among variables. For this reason, PCA models are useful for analyzing data offline, as well as for online process monitoring.

To set up a PCA model:

1. Make sure G2 is running, then clone the PCA model onto a workspace in your application.
2. Make sure the workspace does not belong to the nolstudio module.

Now, you are ready to write procedures that use the PCA model API to do the PCA calculation.

Methods for PCA Model

These are the methods you use to build a PCA model and run data through the model.

nols-load
(*model*: class nols-pca-model, *stream*:class g2-stream)

Load the PCA parameter exported from the NolStudio projection chart.

nols-learn
(*model*: class nols-pca-model, *x*: class item-array)

Builds the PCA model from data matrix *x*, which is an item array of float arrays.

nols-rescaler-input-vector
(*model*: class nols-pca-model, *input-vector*: class float-array, *output-vector*: class float-array)

Scales the input data before feeding it into the PCA model. The PCA model stores the scale weights internally.

nols-execute
(*model*: class nols-pca-model, *x*: class float-array, *pcs*: class float-array)

Runs the scaled input data through the PCA model. *Pcs* provides the results of the calculation.

nols-execute
(*model*: class nols-pca-model, *x*: class float-array, *pcs*: class float-array, *nfactor*: integer)

Calculates first *nfactor* principal components for the scaled input. *Pcs* provides the results of the calculation.

Partial Least Squares (PLS) Model

Partial Least Squares (PLS) is a statistical technique for building a linear regression model. The linear PLS model is simple and robust for correlating input variables.

To set up a PLS model:

1 Make sure G2 is running, then clone the PLS model onto a workspace in your application.

2 Make sure the workspace does not belong to the nolstudio module.

Now, you are ready to write procedures that use the PLS model API to make the model prediction.

Methods for PLS Model

These are the methods you use to build a PLS model and run data through the model to get model predictions.

nols-load
(*model*: class nols-pls-model, *stream*: class g2-stream)

Loads the PLS parameter exported from NolStudio predictive model, which is trained as a linear model only.

nols-learn
(*model*: class nols-pls-model, *x*: class item-array, *y*: class item-array, *nfactor*: integer)

Builds the PLS model from data matrix *x* and *y* with a specified number of internal factors. The data matrix should be an item array of float arrays.

nols-rescaler-input-vector
(*model*: class nols-pls-model, *input-vector*: class float-array, *output-vector*: class float-array)

Scales the input data before feeding it into the PLS model. The PLS model stores the scale weights internally.

nols-rescaler-output-vector
(*model*: class nols-pls-model, *input-vector*: class float-array,
output-vector: class float-array)

Scales the output data back to their normal range after PLS model execution. The PLS model stores the scale weights internally.

nols-execute
(*model*: class nols-pls-model, *x*: class float-array)
-> <u>*value:*</u> class float-array

Executes a PLS model with the given input data *x* and returns the output.

nols-execute
(*model*: class nols-pls-model, *x*: class item-array, *y*: class item-array)

Executes a PLS model with the given input data matrix *x*, where *y* provides the resulting matrix. The data matrix should be an item array of float arrays.

Optimization Deployment

Describes how to export and deploy an optimization in ActiveX and G2

Introduction

After you test your settings of an optimization to satisfaction, you can deploy it online to solve real time optimization problem. Same as deploying predictive models, there are two environments where deployment can occur: COM and G2.

Exporting Optimization

Once you have a tested optimization, you need to export it. Exportation saves the optimization object in a `.opt` file. You will use this file later to load your optimization in the deployment environment. Exporting an optimization from the optimization workspace has been explained in the optimization chapter.

Note After an optimization is exported, you can still change the bounds and weights of the objective function. You can also update the output and state model weights by retraining them with new data.

Deploying in ActiveX

The ActiveX control class of optimization is called NOLOptimizer. You need to register this control before you can deploy NOLOptimizer in COM environment. You can follow Registering the ActiveX Control section in model deployment chapter to register NOLOptimizer control. The registering procedure registers both NOLPredictor and NOLOptimizer.The following section describes how to use the NOLOptimizer control in ActiveX environment. All of the code examples are performed in Visual Basic. However, this control can be used in any COM compliant environment.

Using NOLOptimizer in Visual Basic

Before you can deploy your NOL optimization, you need to add the NOLOnline ActiveX controls to the application Toolbox.

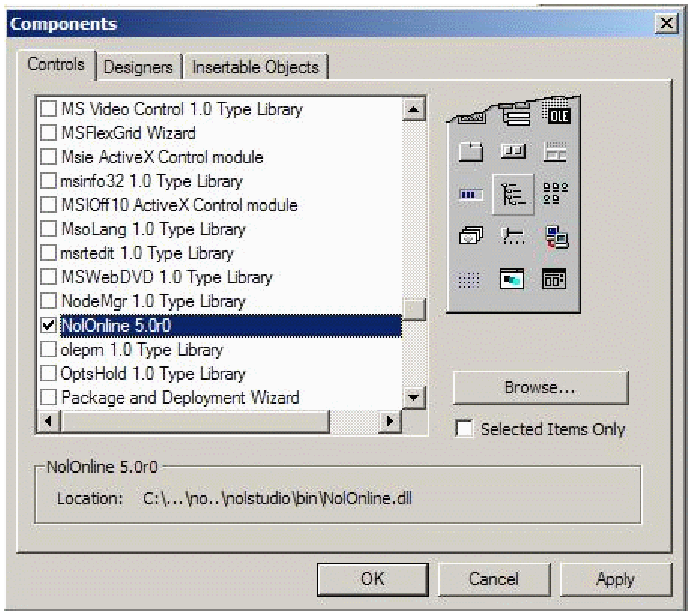
To add the controls to the component toolbox:

1 Start a Visual Basic application, or start Visual Basic with a new project.

2 Right-click on the Toolbox and choose Components from its menu.

3 Select NOLOnline 5.1r0 from the list of controls and click OK.

For example:



- [ ] MS Video Control 1.0 Type Library
- [ ] MSFlexGrid Wizard
- [ ] Msie ActiveX Control module
- [ ] msinfo32 1.0 Type Library
- [ ] MSIOff10 ActiveX Control module
- [ ] MsoLang 1.0 Type Library
- [ ] msrtedit 1.0 Type Library
- [ ] MSWebDVD 1.0 Type Library
- [ ] NodeMgr 1.0 Type Library
- [x] NolOnline 5.0r0
- [ ] olepm 1.0 Type Library
- [ ] OptsHold 1.0 Type Library
- [ ] Package and Deployment Wizard

NolOnline 5.0r0

Location: C:\...\no..\nolstudio\bin\NolOnline.dll

Both the NOLPredictor and the NOLOptimizer components are added to the Toolbox.

Loading the NOL Optimization Object

The NOLOptimizer control that you have added to your Toolbox is a generic component. To run the optimization, first you need to place the control on a form, then you need to load the objective function with specified bounds and weights. The objective function is stored in an optimization object, which stores the objective function and associated parameters. You can programmatically call the LoadOptimization method on the NOLOptimizer instance to load the optimization object with specified path and file name.

To create a NOLOptimizer instance:

1 Clone the NOLOptimizer control and place it on the form. The control is invisible at run-time, and appears as a string on your form.

2 Examine the Property Window of the control. You need to name your NOLOptimizer control (e.g. "gasplant"), although VB will provide a default name, such as NOLOptimizer1.

To load the exported optimization object:

1 Double-click on the form to show the code-window for the form.

2 In the Form_Load subroutine, write the following code:

```
gasplant.loadOptimization file, path
```

where `gasplant` is the name of your control, and *path* and *file* are the two arguments to the method, pointing to the location where the `.opt` file is stored.

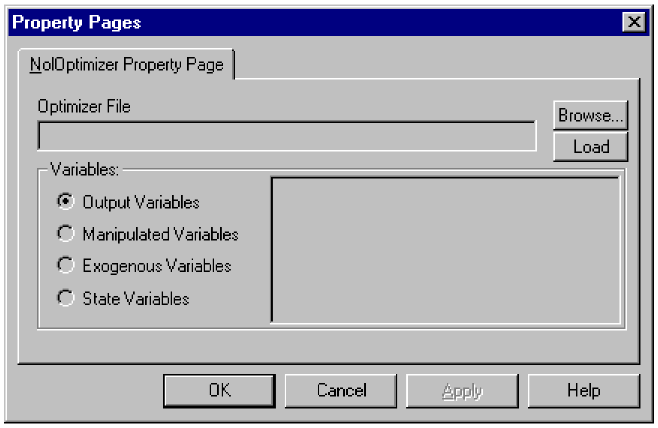
You can also load the model at design time, to verify that you have the correct one, for example.

To verify the Optimization Model

1 Select the NOLOptimizer object in your form.

2 Choose View > Property Pages to display its property page, or click on the Custom design property.

The following dialog appears:



3 Click the Browse button to display the File Load dialog.

4 Locate and select your optimization model, and then click the Open button.

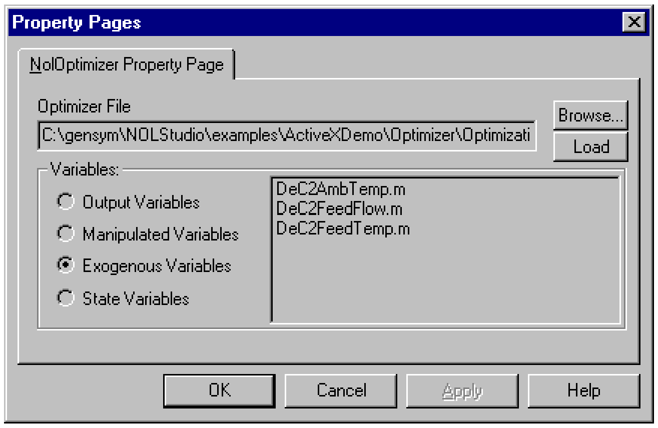
The file name and path appears on the dialog.

5 Click the Load, OK, or Apply button to load the model from this location.

You may now look at the variable names of the optimization model itself in the property pages of the control.

By clicking any of the Variables radio buttons, you can display the needed information in your optimization model. Keep in mind that the names and tags should appear in the same order as in the trained model.

For example:



Optimizer File: C:\gensym\NOLStudio\examples\ActiveXDemo\Optimizer\Optimizati

Variables: Output Variables [✓ Exogenous Variables] Manipulated Variables State Variables

DeC2AmbTemp.m
DeC2FeedFlow.m
DeC2FeedTemp.m

In ActiveX demos provided as part of the NOL Studio package, a Visual Basic application shows all of the previous actions. The loading of the optimization object and displaying some of its properties can also be done at run-time with some simple code. Here is an example of code that performs the above actions on an instance of an NOLOptimizer that has been named gasplant.

```
Private Sub Form_Load()
   gasplant.loadOptimization "gasplant.opt", _
      "c:\gensym\g2-2011\nolstudio\examples\ActiveDemo"
   outputs = gasplant.getNumberOfVariables(0)
   For counter = 0 to outputs
      ouotputName(counter) = gasplant.outputNames(counter)
      MsgBox "The output name [" & counter & "]" & outputName(counter)
   Next counter
End Sub
```

Running the Optimization

You need to perform following three steps to run the optimization.

1 Provide the setpoint values for outputs.

2 Calculate the optimization.

3 Request the calculated input values.

Data Input

Look in the Visual Basic Object Browser to examine the methods associated with the NOLOptimizer control. You will see several methods for data input:

```
setBoundsByName(BSTR name, VARIANT value);

setBoundsByTag(BSTR tag, VARIANT value);

setValueByName(BSTR name, double value);

setValueByTag(BSTR tag, double value);

setValues(short index, VARIANT values);

setWeightsByName(BSTR name, VARIANT values);

setWeightsByTag(BSTR tag, VARIANT values);
```

You can use these method to set the desired values to the optimization. The detailed description of these methods can be found in appendix.

Caution The bounds and weights parameters in above methods are double arrays of all the bound and weight values for given variable. To set particular values within the array without changing others, you can get the values first and change the corresponding ones with the array you just get, and set the values back with this array.

Here are the some demo of code you would use to set the specification for an optimization object:

```
value = 1.16 'setpoint for % C3 IN C2 COMP

name = gasplant.outputNames(0) 'variable name for % C3 IN C2 COMP

bounds = gasplant.getBoundsByName(name)

bounds(2) = value 'Setpoint is the third element of bounds.

gasplant.setBoundsByName name, bounds
```

Calculating Optimization

You use one method to calculate the optimization.

```
calculate();
```

Obtaining Results

The results of an optimization problem contain the desired values for manipulated variables, the achieved values for state and output variables. All of these values can be obtained from following methods:

```
double getValueByName(BSTR name);

double getValueByTag(BSTR tag);
```

Once you have the desired value, you can write it back to your spreadsheet, or plot it, or process it according to your application.

Training Optimization Models in Real Time

In a real-time application, you may want to update your optimization model as new data is collected. For detailed information, see Training Predictive Models in Real Time. For the API for training optimization models, see Appendix B, NOLOptimizer Class.

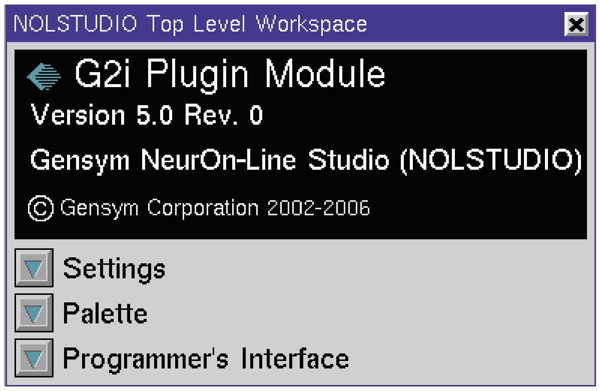
Deployment in G2 using G2 JavaLink

In order to deploy the NOL Studio optimization in G2 through G2 JavaLink, you must have JavaLink license. You also need to have JavaLink installed on your machine. Go to the Readme file of NOL bundle, and follow the directions for installing JavaLink. You will need to restart your machine before proceeding. This section assumes knowledge of G2 and JavaLink.

Loading the Necessary KBs

All the KBs are located in the `nol` directory. The top-level module is called `nolstudio.kb`. However, do not make `nolstudio.kb` the top level module in your application. Create a module, or a hierarchy of modules, as your application, then merge `nolstudio.kb` into your application, and make it a required module.

This is the top-level workspace of the `nolstudio.kb`:


NOLSTUDIO Top Level Workspace

G2i Plugin Module

Version 5.0 Rev. 0

Gensym NeurOn-Line Studio (NOLSTUDIO)

© Gensym Corporation 2002-2006

It provides a palette of objects such as the Predictive Model, the Optimization, the Module Settings, and the API for both the predictive and optimization models.

The basic process of using an nols model in G2 is to:

- Connect to an interface.
- Initialize the model.
- Send data to the model.
- Receive/request outputs.

Each of these steps is detailed in the following sections.

Launching a Remote Process

Launching a remote process for optimization is the same as for any model. For details, see Launching a Remote Process at Startup and Launching a Remote Process Using Procedure.

The Optimization Model and its API

To set up an optimization model you need to clone its icon from the NolStudio palette, and fill in its attributes. Then, use the API methods to send data to the optimization and receive values. The API is split up into three parts: the general methods - such as getting the name and comment of the model, the sets and gets of the values, and the sets and gets of the weights and bounds.

To set up an optimization model:

1 Make sure G2 is running, then clone the Optimization onto a workspace in your application.

Note Make sure the workspace does not belong to the nolstudio module.

2 Edit the attribute table of the model to fill in the following attributes:

Attribute	Description
names	The name of the optimization, which you reference in the API procedures.
nols-directory-name	The directory of the optimization, in quotes.
nols-file-name	The file name of the optimization, for example, "model1.opt".

Now, you are ready to write procedures using the optimization API to send and receive data. Click the Nols Programmers Interface button on the palette, then click the Methods for Optimizations button. The following sections describe each API procedure.

Method to Initialize the Optimization

The first method you must call before you can do anything further is nols-initialize.

nols-initialize
(*opt*: class nols-optimization, *interface*: class nols-gateway)

Initializes the optimization. The *model* is an attribute of the optimization. The *interface* is the nols-interface-object-name referenced in your module settings.

General Methods

The general methods do not send or receive actual model data.

nols-get-name
(*opt*: class nols-optimization)
-> *model:* text

This method returns the name of the model created in NOL Studio.

nols-get-comment
(*opt*: class nols-optimization)
-> *comment:* text

This method returns the comment associated with the model created in NOL Studio.

nols-get-number-of-variables
(*opt*: class nols-optimization, *type*: integer)
-> *number:* integer

This method returns the number of specified variables in the model. The type argument determines the variables you want:

0 - outputs
1 - manipulated
2 - exogenous
3 - state.

You may need to call this method to resize your arrays.

nols-get-variable-names
(*opt*: class nols-optimization, *type*: integer)
-> *names:* class text-array

This method returns an array of variable names specified by the type argument:

0 - outputs
1 - manipulated
2 - exogenous
3 - state

nols-get-variable-tags
(*opt*: class nols-optimization, *type*: integer)
-> *tags:* class text-array

This method returns an array of variable tags specified by the type argument:

0 - outputs
1 - manipulated
2 - exogenous
3 - state.

nols-get-variable-units

(*opt*: class nols-optimization, *type*: integer)
-> *units*: class text-array

This method returns an array of variable units specified by the type argument:

0 - outputs
1 - manipulated
2 - exogenous
3 - state

Methods to Set Weights and Bounds on Variables

These methods set the weights and bounds on the variables in the optimization. You can set or get either the weights or the bounds by the variable name or tag. The weights are sent to the optimization as a float array in the following order:

linear weight, setpoint weight, soft upper bound weight, soft lower bound weight

The bounds are also sent to the optimization as an array of floats in the following order:

hard lower bound, soft lower bound, setpoint, soft upper bound, hard upper bound

All of the methods in this group will signal an nols-error if something goes wrong with the method. You must enclose the calls to these methods into a begin-end-on-error clause to capture the error. The nols-error has three text attributes that you can view once the error is captured. These attributes are: the class-of-error, the description-of-error, and the backtrace-of-error.

nols-set-weights-by-name

(*opt*: class nols-optimization, *variable-name*: text,
variable-weights: class float-array)

Sets the weights for a particular variable. Remember the order of the weights: linear weight, setpoint weight, soft upper bound weight, soft lower bound weight.

nols-set-weights-by-tag

(*opt*: class nols-optimization, *variable-tag*: text,
variable-weights: class float-array)

Sets the weights for a particular variable, using the variable's tag for identification. Remember the order of the weights: linear weight, setpoint weight, soft upper bound weight, soft lower bound weight.

nols-get-weights-by-name
(*opt*: class nols-optimization, *variable-name*: text)
-> *weights:* class float-array

Returns the weights set for a particular variable, using that variable's name. The float values returned will be in the same order: linear weight, setpoint weight, soft upper bound weight, soft lower bound weight.

nols-get-weights-by-tag
(*opt*: class nols-optimization, *variable-tag*: text)
-> *weights:* class float-array

Returns the weights set for a particular variable, using that variable's tag for identification. The float values returned will be in the same order: linear weight, setpoint weight, soft upper bound weight, soft lower bound weight.

nols-set-bounds-by-name
(*opt*: class nols-optimization, *variable-name*: text,
variable-bounds: class float-array)

Sets the bounds for a variable using the variable's name for identification. The bounds should be all floats, in the following order: hard lower bound, soft lower bound, setpoint, soft upper bound, hard upper bound.

nols-set-bounds-by-tag
(*opt*: class nols-optimization, *variable-tag*: text,
variable-bounds: class float-array)

Sets the bounds for a variable using the variable's tag for identification. The bounds should be all floats, in the following order: hard lower bound, soft lower bound, setpoint, soft upper bound, hard upper bound.

nols-get-bounds-by-name
(*opt*: class nols-optimization, *variable-name*: text)
-> *bounds:* class float-array

Returns the bounds set for a particular variable, by that variable's name. The array of floats returned, contains the bounds in the following order: hard lower bound, soft lower bound, setpoint, soft upper bound, hard upper bound.

nols-get-bounds-by-tag
(*opt*: class nols-optimization, *variable-tag*: text)
bounds: class float-array

Returns the bounds set for a particular variable by that variable's tag. The array of floats returned, contains the bounds in the following order: hard lower bound, soft lower bound, setpoint, soft upper bound, hard upper bound.

Methods to Get/Set Variable Values

This group of methods allows you to get and set variable values. You can get/set a value for a particular variable by its name or tag, or you can get/set values for a group of variables, such as all manipulated variables or all outputs. The values set by this group of method are used as the initial starting point for the optimization calculation. The values return from get methods show the current status of the optimization object. For example, if called after one calculation, the values from get methods for manipulated variables will be the current results returned from that calculation.

Note Because in the optimization calculation the exogenous variables only provide a set of fixed values for the calculation, the set value methods are used to set these fixed values, which are also saved as their setpoints at the same time.

All of the methods in this group will signal an nols-error if something goes wrong with the method. You must enclose the calls to these methods into a begin-end-on-error clause to capture the error. The nols-error has three text attributes that you can view once the error is captured. These attributes are class-of-error, description-of-error, and the backtrace-of-error.

nols-set-value-by-name
(*opt*: class nols-optimization, *variable-name*: text, *variable-value*: float)

Sets a value for a variable using the variable's name for identification.

nols-set-value-by-tag
(*opt*: class nols-optimization, *variable-tag*: text, *variable-value*: float)

Sets a value for a variable using the variable's tag for identification.

nols-get-value-by-name
(*opt*: class nols-optimization, *variable-name*: text)
-> *value:* float

Returns a value set for a particular variable, using the variable's name for identification.

nols-get-value-by-tag
(*opt*: class nols-optimization, *variable-tag*: text)
-> *value:* float

Returns a value set for a particular variable, using the variable's tag for identification.

nols-set-values
(*opt*: class nols-optimization, *variable-type*: integer,
variable-values: class float-array)

Sets the values for all variables of a particular type. The variable-type argument is decoded as follows:

0 - outputs
1 - manipulated variables
2 - exogenous variables
3 - state variables.

Make sure that the order of the values in the array corresponds to the order of those variables in the optimization model.

nols-get-values
(*opt*: class nols-optimization, *variable-type*: integer)
-> *values:* class float-array

Returns an array of values set for variables of a particular type. The variable-type argument is decoded as follows:

0 - outputs
1 - manipulated variables
2 - exogenous variables
3 - state variables.

The order of the values returned is the same as the order of variables in the optimization model.

Methods to Calculate the Optimization

These methods calculate the optimization. The calculation only runs if something has changed since the last calculation, such as a weight, a bound, or a value has been changed by the user. These methods also signal an nols-error if something goes wrong.

nols-calculate-optimization
(*opt*: class nols-optimization)

Calculates the optimization once you have set the weights, bounds, and/or values as desired.

nols-calculate-optimization-with-flag
(*opt*: class nols-optimization)
-> *status:* integer

Use this method to calculate the optimization. A integer value is returned to indicate the status flag of the optimization calculation.

0 - converge normal
1 - Feasible, but not converge (maximum iteration)
2 - could not find feasible solution (maximum iteration)
3 - other error condition.

Training Optimization Models in Real Time

The common parameters in these API are *time*, *auto-stop*, *initial-training*, and *training-display*. The *time* tells the training algorithm how long you want to spend to train the model. The *auto-stop* indicates whether you want the training algorithm to stop training automatically if no further improvement is detected. The *initial-training* indicates whether you want to clear the model and start new training or continue training from last stop state. The *training-display* indicates whether to show the training console during the training.

nols-train-output-model
(*model*: class nols-predictive-model, *xmatrix* : sequence , *ymatrix*: sequence, *time*: float, *auto-stop*: truth-value, *initial-training*: truth-value, *training-display*: true-value)

Trains the output submodel of the optimization. Use this method when you collect data inside G2 and pass the data through a G2 interface. The sequence type for *xmatrix* and *ymatrix* is a sequence of float-arrays. The column number of *xmatrix* and *ymatrix* must be the same as the input number and output number.

nols-train-output-model-from-file
(*model*: class nols-predictive-model , *input-file*: text, *output-file*: text, *time-in-min*: float, *auto-stop*: truth-value, *initial-training*: truth-value, *training-display*: true-value)

Trains the output submodel of the optimization model. Use this method when you save the data into data files that contain numerical value only. The format of the file must be a comma separated ASCII file with the same numerical values at each line.

nols-train-state-model
(*model*: class nols-predictive-model, *xmatrix*: sequence , *ymatrix*: sequence, *time*: float, *auto-stop*: truth-value, *initial-training*: truth-value, *training-display*: true-value)

Trains the state submodel of the optimization. This method does nothing if there is no state submodel. Use this method when you collect data inside G2 and pass the data through G2 interface. The sequence type for *xmatrix* and *ymatrix* is a sequence of float-arrays. The column number of *xmatrix* and *ymatrix* must be the same as the input number and output number.

nols-train-state-model-from-file
(*model*: class nols-predictive-model , *input-file*: text, *output-file*: text, *time-in-min*: float, *auto-stop*: truth-value, *initial-training*: truth-value, *training-display*: true-value)

Trains the state submodel of the optimization. This method does nothing if there is no state submodel. Use this method if you save the data into data files. The data file should contain numerical value only. The format of the file must be a comma separated ASCII file with same numerical values at each line.

nols-train-model-from-file
(*model*: class nols-predictive-model , *data-series-files*: sequence, *time-in-min*: float, *auto-stop*: truth-value, *initial-training*: truth-value, *training-display*: true-value)

Trains the whole optimization model. Use this method if you save the data into data files. The data files contain the data series with fixed `.ds` format. The method will generate training data for both output and state submodels from these data series.

Additional Methods for the Optimization

nols-get-max-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the maximum statistic of the named variable. The returned value is the maximum value of that variable in the data series used to build the model.

nols-get-max-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the maximum statistic of the named variable. The returned value is the maximum value of that variable in the data series used to build the model.

nols-get-min-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the minimum statistic of the named variable. The returned value is the minimum value of that variable in the data series used to build the model.

nols-get-min-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the minimum statistic of the named variable. The returned value is the minimum value of that variable in the data series used to build the model.

nols-get-mean-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the mean statistic of the named variable. The returned value is the mean value of that variable in the data series used to build the model.

nols-get-mean-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the mean statistic of the named variable. The returned value is the mean value of that variable in the data series used to build the model.

nols-get-median-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the median statistic of the named variable. The returned value is the minimum value of that variable in the data series used to build the model.

nols-get-median-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the median statistic of the named variable. The returned value is the median value of that variable in the data series used to build the model.

nols-get-sum-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the sum statistic of the named variable. The returned value is the sum of that variable in the data series used to build the model.

nols-get-sum-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the sum statistic of the named variable. The returned value is the sum of that variable in the data series used to build the model.

nols-get-std-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the standard deviation statistic of the named variable. The returned value is the standard deviation value of that variable in the data series used to build the model.

nols-get-std-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the standard deviation statistic of the named variable. The returned value is the standard deviation value of that variable in the data series used to build the model.

nols-get-variance-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the variance statistic of the named variable. The returned value is the variance value of that variable in the data series used to build the model.

nols-get-variance-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the variance statistic of the named variable. The returned value is the variance value of that variable in the data series used to build the model.

nols-get-kurt-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the kurt statistic of the named variable. The returned value is the kurt value of that variable in the data series used to build the model.

nols-get-kurt-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the kurt statistic of the named variable. The returned value is the kurt value of that variable in the data series used to build the model.

nols-get-skew-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the skew statistic of the named variable. The returned value is the skew value of that variable in the data series used to build the model.

nols-get-skew-value-by-tag
(*opt*: class nols-optimization, *tag*: text)
-> *value:* float

Returns the skew statistic of the named variable. The returned value is the skew value of that variable in the data series used to build the model.

nols-get-range-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the range statistic of the named variable. The returned value is the range value of that variable in the data series used to build the model.

nols-get-range-value-by-name
(*opt*: class nols-optimization, *name*: text)
-> *value:* float

Returns the range statistic of the named variable. The returned value is the range value of that variable in the data series used to build the model.

Handle Error Exceptions

The following is a list of all methods that signal the nols-error if something goes wrong. Please ensure that you encapsulate all of these method calls into begin - end on error statements to capture the errors.

nols-calculate-optimization
nols-set-value-by-name
nols-set-value-by-tag
nols-get-value-by-name
nols-get-value-by-tag
nols-set-values
nols-get-values
nols-set-bounds-by-name
nols-set-bounds-by-tag
nols-get-bounds-by-name
nols-get-bounds-by-name
nols-set-weights-by-name

NOLPredictor Class

Describes data members and operations of the NOLPredictor class.

Introduction

This appendix provides a quick reference for the `NOLPredictor` class. The information is listed alphabetically within these two categories:

- Data Members
- Methods

Notation

All *`VARIANT`* variables in class members are used as a reference of double arrays. The *`VARTYPE`* for all *`VARIANT`* values is *`VT_ARRAY`*|*`VT_R8`*.

Data Members

`bstr` *modelName*`;`

Stores the name of this `NOLPredictor` instance.

`bstr` *modelComment*

Stores the comment of this `NOLPredictor` instance.

`bstr` *modelFilePath*`;`

Stores the directory string for the predictive model.

`bstr` *modelFileName*`;`

Stores the file name string for the predictive model.

`bstr` *TimeFormat*`;`

Stores the format string of date/time for time stamps.

`bstr` *inputNames*`(short` *index*`);`

Stores the input names of the predictive model.

Parameter	Description
index	Gives the order of inputs specified in the predictive model.

`bstr` *inputTags*`(short` *index*`);`

Stores the input tags of the predictive model.

Parameter	Description
index	Gives the order of inputs specified in the predictive model.

`bstr` *inputUnits* `(short` *index*`);`

Stores the input units of the predictive model.

Parameter	Description
index	Gives the order of inputs specified in the predictive model.

`bstr` *outputNames* `(short` *index*`);`

Stores the output names of the predictive model.

Parameter	Description
index	Gives the order of outputs specified in the predictive model.

`bstr` *outputTags* `(short` *index*`);`

Stores the output tags of the predictive model.

Parameter	Description
index	Gives the order of outputs specified in the predictive model.

`bstr` *outputUnit* `(short` *index*`);`

Stores the output units of the predictive model.

Parameter	Description
index	Gives the order of outputs specified in the predictive model.

`boolean` *loadOnRun*

Stores the information on whether a model should be loaded automatically at runtime:

- If `true`, the model should be loaded automatically.
- If `false`, the model can only be loaded through the `loadModel` method.

`long` *numberOfInputs*

Stores the number of inputs in the model.

`long` *numberOfOutputs*

Stores the number of outputs in the model.

Methods

NolPredictor::calculateOutputsAtTime
VARIANT calculateOutputsAtTime(bstr Time);

Calculates and returns output values at a specified time. This function can only be called after a predictive model has been loaded and input data have been prepared.

Parameter	Description
Time	String used to specify the time stamp, at which you want to calculate the outputs.

Return Value	Description
VARIANT	Provides the output values for all model outputs. The *VARIANT* variable is a reference to a double array, which has the same order as outputs specified in the predictive model.

NolPredictor::calculateOutputsForRow
VARIANT calculateOutputsForRow();

Calculates and returns output values for models trained from row-based data.

Return Value	Description
VARIANT	Provides the output values for all model outputs. The *VARIANT* variable is a reference to a double array, which has the same order as outputs specified in the predictive model.

This function can only be called after a predictive model has been loaded and input data have been prepared.

NolPredictor::clearDataBuffer
void clearDataBuffer();

This function is used to clear the data buffer for models developed from time-based models.

NolPredictor::getFormulas
bstr getFormulas();

Returns a string representing the formulas in the preproccessor, from which the model is built.

NolPredictor::getInputDelayByName
VARIANT getInputDelayByName(bstr inputName);

Returns the delays of one variable from variable name.

Return Value	Description
VARIANT	Provides all delay settings for the given variable. The *VARIANT* variable is a reference to a long array, which contains the long value of each delay with an unit of millisecond.

NolPredictor::getInputDelayByTag
VARIANT getInputDelayByTag(bstr inputTag);

Returns the delays of one variable from variable tag.

Return Value	Description
VARIANT	Provides all delay settings for the given variable. The *VARIANT* variable is a reference to a long array, which contains the long value of each delay with an unit of millisecond.

NolPredictor::getKurtValueByName
double getKurtValueByName(bstr inputName);

Returns the kurt value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getKurtValueByTag
double getKurtValueByTag(bstr inputTag);

Returns the kurt value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::getMaxValueByName
double getMaxValueByName(bstr inputName);

Returns the maximum value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getMaxValueByTag
double getMaxValueByTag(bstr inputTag);

Returns the maximum value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::getMeanValueByName
double getMeanValueByName(bstr inputName);

Returns the mean value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getMeanValueByTag
double getMeanValueByTag(bstr inputTag);

Returns the maximum value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::getMedianValueByName
double getMedianValueByName(bstr inputName);

Returns the median value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getMedianValueByTag
double getMedianValueByTag(bstr inputTag);

Returns the median value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::getMinValueByName
double getMinValueByName(bstr inputName);

Returns the minimum value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getMinValueByTag
double getMinValueByTag(bstr inputTag);

Returns the minimum value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::getNumberOfInputs
long getNumberOfInputs();

Returns the number of inputs. This function can only be called after a predictive model is loaded.

NolPredictor::getNumberOfOutputs
long getNumberOfOutputs();

Returns the number of outputs. This function can only be called after a predictive model is loaded.

NolPredictor::getRangeValueByName
double getRangeValueByName(bstr inputName);

Returns the range value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getRangeValueByTag
double getRangeValueByTag(bstr inputTag);

Returns the range value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::getSkewValueByName
double getSkewValueByName(bstr inputName);

Returns the skew value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getSkewValueByTag
double getSkewValueByTag(bstr inputTag);

Returns the skew value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::getSTDValueByName
double getSTDValueByName(bstr inputName);

Returns the standard deviation value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getSTDValueByTag
double getSTDValueByTag(bstr inputTag);

Returns the standard deviation value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::getSumValueByName
double getSumValueByName(bstr inputName);

Returns the sum value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getSumValueByTag
double getSumValueByTag(bstr inputTag);

Returns the sum value of one variable from variable tag. The value is defined from the data series used to build the model.

NolPredictor::getVarianceValueByName
double getVarianceValueByName(bstr inputName);

Returns the variance value of one variable from a variable name. The value is defined from the data series used to build the model.

NolPredictor::getVarianceValueByTag
double getVarianceValueByTag(bstr inputTag);

Returns the variance value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolPredictor::isTimeBasedModel
BOOLEAN isTimeBasedModel();

Determines whether model was trained from time-based or row-based data. This function can only be called after a predictive model is loaded.

Return Value	Description
BOOLEAN	Returns *TRUE* if model was trained from time-based data and *FALSE* if model was trained from row-based data.

NolPredictor::loadModel
void loadModel(bstr modelFileName, bstr modelFilePath);

Loads the predictive model from the specified file/directory location.

Parameter	Description
modelFileName	The file name string.
modelFilePath	The directory name string.

NolPredictor::setInputsAtTime
void setInputsAtTime(VARIANT inputs, bstr time);

Prepares input data for models trained from time-based data. This function can only be called after a predictive model is loaded.

Parameter	Description
inputs	A *VARIANT* variable used to provide a double array for all model inputs at the given time.
time	String used to specify the current time stamp.

`NolPredictor::setInputAtTimeByindex`
`void setInputAtTimeByindex(long` *index*`, double` *inputValue*`, bstr` *time*`);`

Prepares input data for models trained from time-based data. This function can only be called after a predictive model is loaded.

Parameter	**Description**
index	Gives the order of inputs specified in the predictive model.
inputValue	The value for the specified input at the given time.
time	String used to specify the current time stamp.

`NolPredictor::setInputAtTimeByName`
`void setInputAtTimeByName(bstr` *name*`, double` *inputValue*`, bstr` *time*`);`

Prepares input data for models trained from time-based data. This function can only be called after a predictive model is loaded.

Parameter	**Description**
name	Gives the name of input variable.
inputValue	The value for the specified input at the given time.
time	String used to specify the current time stamp.

`NolPredictor::setInputAtTimeByTag`
`void setInputAtTimeByTag(bstr` *tag*`, double` *inputValue*`, bstr` *time*`);`

Prepares input data for models trained from time-based data. This function can only be called after a predictive model is loaded.

Parameter	**Description**
tag	Gives the tag of input variable.
inputValue	The value for the specified input at the given time.
time	String used to specify the current time stamp.

NolPredictor::setInputsForRow
void setInputsForRow(VARIANT inputs);

Prepares input data for models trained from row-based data. This function can only be called after a predictive model is loaded.

Parameter	Description
inputs	A *VARIANT* variable used to provide a double array for all model inputs.

NolPredictor::setInputForRowByIndex
void setInputForRowByIndex(long index, double inputValue);

Prepares input data for models trained from row-based data. This function can only be called after a predictive model is loaded.

Parameter	Description
index	Gives the order of inputs specified in the predictive model.
inputValue	The value for the specified input.

NolPredictor::setInputForRowByName
void setInputForRowByName(bstr inputName, double inputValue);

Prepares input data for models trained from row-based data. This function can only be called after a predictive model is loaded.

Parameter	Description
inputName	Gives the name of input variable,
inputValue	The value for the specified input.

```
NolPredictor::setInputForRowByTag
void setInputForRowByTag(bstr tag, double inputValue);
```

Prepares input data for models trained from row-based data. This function can only be called after a predictive model is loaded.

Parameter	Description
Tag	Gives the tag of input variable.
inputValue	The value for the specified input.

```
NolPredictor::setVariableValueAtTimeByName
void setVariableValueAtTimeByName(bstr name, double value,
bstr time);
```

Prepares variable data for models trained from time-based data. This function can only be called after a predictive model is loaded. You use it to set the data to the delayed output of a recursive model.

Parameter	Description
name	Gives the name of variable.
value	The value for the specified variable at the given time.
time	String used to specify the current timestamp.

```
NolPredictor::setVariableValueAtTimeByTag
void setVariableValueAtTimeByTag(bstr name, double value, bstr time);
```

Prepares variable data for models trained from time-based data. This function can only be called after a predictive model is loaded. You use it to set the data to the delayed output of a recursive model.

Parameter	Description
tag	Gives the tag of variable.
value	The value for the specified variable at the given time.
time	String used to specify the current time stamp.

```
NolPredictor::setTimeFormat
void setTimeFormat(bstr timeFormat);
```

Sets the time format within the model for time-based models. This function must be called before setting inputs for time-based models.

Parameter	Description
timeFormat	The format string of date/time for time stamps.

NOLOptimizer Class

Describes data members and operations of the NOLOptimization class.

Introduction

This appendix provides a quick reference for the `NOLOptimizer` class. The information is listed alphabetically within these two categories:

- Data Members
- Methods

Notation

All *`VARIANT`* variables in class members are used as a reference of double arrays. The *`VARTYPE`* for all *`VARIANT`* values is *`VT_ARRAY`*|*`VT_R8`*.

Data Members

`bstr` *optimizationName*`;`

Stores the name of this NOLOptimizer instance.

`bstr` *optimizationComment*

Stores the comment of this NOLOptimizer instance.

`bstr` *optimizationFilePath*`;`

Stores the directory string for the optimization object location.

`bstr` *optimizationFileName*`;`

Stores the file name string for the optimization object location.

`bstr` *exogenousNames*`(short` *index*`);`

Stores the exogenous names of the optimization model.

Parameter	Description
index	Gives the order of exogenous variables specified in the optimization model.

`bstr` *exogenousTags*`(short` *index*`);`

Stores the exogenous tags of the optimization model.

Parameter	Description
index	Gives the order of exogenous variables specified in the optimization model.

`bstr` *exogenousUnits* `(short` *index*`);`

Stores the exogenous units of the optimization model.

Parameter	Description
index	Gives the order of exogenous variables specified in the optimization model.

`bstr` *manipulatedNames* `(short` *index*`);`

Stores the manipulated names of the optimization model.

Parameter	Description
index	Gives the order of manipulated variables specified in the optimization model.

`bstr` *manipulatedTags* `(short` *index*`);`

Stores the manipulated tags of the optimization model.

Parameter	Description
index	Gives the order of manipulated variables specified in the optimization model.

`bstr` *manipulatedUnits* `(short` *index*`);`

Stores the manipulated units of the optimization model.

Parameter	Description
index	Gives the order of manipulated variables specified in the optimization model.

`bstr` *outputNames*`(short` *index*`);`

Stores the output names of the optimization model.

Parameter	Description
index	Gives the order of outputs specified in the optimization model.

`bstr` *outputTags*`(short` *index*`);`

Stores the output tags of the optimization model.

Parameter	Description
index	Gives the order of outputs specified in the optimization model.

`bstr` *outputUnits*`(short` *index*`);`

Stores the output units of the optimization model.

Parameter	Description
index	Gives the order of outputs specified in the optimization model.

`bstr` *stateNames*`(short` *index*`);`

Stores the state names of the optimization model.

Parameter	Description
index	Gives the order of state variables specified in the optimization model.

`bstr` *stateTags* `(short` *index* `);`

Stores the state tags of the optimization model.

Parameter	Description
index	Gives the order of state variables specified in the optimization model.

`bstr` *stateUnits* `(short` *index* `);`

Stores the state units of the optimization model.

Parameter	Description
index	Gives the order of state variables specified in the optimization model.

Methods

NolOptimizer::calculate
void calculate();

Performs the calculation of the optimization problem.

NolOptimizer::calculateOptimization
short calculateOptimization();

Used to calculate an optimization problem. The returned value is the status flag of that calculation.

Return Value	**Description**
short	0=converge normal 1=feasible, but not converge (maximum iteration); 2=could not find feasible solution (maximum iteration); 3=other error condition.

NolOptimizer::getBoundsByName
VARIANT getBoundsByName(bstr name);

Returns the double array for variable bounds.

Parameter	**Description**
name	Stores the variable name.

Remarks: The array is described in the Notation section. This function can only be called after an optimization object is loaded.

NolOptimizer::getBoundsByTag
VARIANT getBoundsByTag(bstr tag);

Returns the double array for variable bounds. The array is described in the Notation section. This function can only be called after an optimization object is loaded.

Parameter	**Description**
tag	Stores the variable tag.

NolOptimizer::getKurtValueByName
double getKurtValueByName(bstr inputName);

Returns the kurt value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getKurtValueByTag
double getKurtValueByTag(bstr inputTag);

Returns the kurt value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getMaxValueByName
double getMaxValueByName(bstr inputName);

Returns the maximum value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getMaxValueByTag
double getMaxValueByTag(bstr inputTag);

Returns the maximum value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getMeanValueByName
double getMeanValueByName(bstr inputName);

Returns the mean value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getMeanValueByTag
double getMeanValueByTag(bstr inputTag);

Returns the maximum value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getMedianValueByName
double getMedianValueByName(bstr inputName);

Returns the median value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getMedianValueByTag
double getMedianValueByTag(bstr inputTag);

Returns the median value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getMinValueByName
double getMinValueByName(bstr inputName);

Returns the minimum value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getMinValueByTag
double getMinValueByTag(bstr inputTag);

Returns the minimum value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getNumberOfVariables
short getNumberOfVariables(short type);

Returns the number of specified variables. This function can only be called after an optimization object is loaded.

Parameter	Description
type	Specify the type of variables. 0: OUTPUT 1: MANIPULATED VARIABLE 2: EXOGENOUS VARIABLE 3: STATE VARAIBLE

NolOptimizer::getRangeValueByName
double getRangeValueByName(bstr inputName);

Returns the range value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getRangeValueByTag
double getRangeValueByTag(bstr inputTag);

Returns the range value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getSkewValueByName
double getSkewValueByName(bstr inputName);

Returns the skew value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getSkewValueByTag
double getSkewValueByTag(bstr inputTag);

Returns the skew value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getSTDValueByName
double getSTDValueByName(bstr inputName);

Returns the standard deviation value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getSTDValueByTag
double getSTDValueByTag(bstr inputTag);

Returns the standard deviation value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getSumValueByName
double getSumValueByName(bstr inputName);

Returns the sum value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getSumValueByTag
double getSumValueByTag(bstr inputTag);

Returns the sum value of one variable from variable tag. The value is defined from the data series used to build the model.

```
NolOptimizer::getValueByName
double getValueByName(bstr name);
```

Returns the double value for given variable. This function is for all variable types. You use this function to get back results after calculating the optimization problem. This function should be called after the calculation.

Parameter	Description
name	Stores the variable name.

```
NolOptimizer::getValueByTag
double getValueByTag(bstr tag);
```

Returns the double value for given variable. This function is for all variable types. You use this function to get back results after calculating the optimization problem. This function should be called after the calculation.

Parameter	Description
tag	Stores the variable tag.

```
NolOptimizer::getValues
VARIANT getValues(short type);
```

Returns the double array for all variables of given type. You use this function to get back results after calculating the optimization problem. This function should be called after the calculation.

Parameter	Description
type	Specify the type of variables. 0: OUTPUT 1: MANIPULATED VARIABLE 2: EXOGENOUS VARIABLE 3: STATE VARAIBLE

```
NolOptimizer::getVarianceValueByName
double getVarianceValueByName(bstr inputName);
```

Returns the variance value of one variable from a variable name. The value is defined from the data series used to build the model.

NolOptimizer::getVarianceValueByTag
double getVarianceValueByTag(bstr inputTag);

Returns the variance value of one variable from a variable tag. The value is defined from the data series used to build the model.

NolOptimizer::getWeightsByName
VARIANT getWeightsByName(bstr name);

Returns a double array of variable weights. The weights and the structure of the array is described in the Notation section. This function can only be called after an optimization object is loaded.

Parameter	Description
name	Stores the variable name.

NolOptimizer::getWeightsByTag
VARIANT getWeightsByTag(bstr tag);

Returns a double array of variable weights. The weights and the structure of the array is described in the Notation section. This function can only be called after an optimization object is loaded.

Parameter	Description
tag	Stores the variable tag.

NolOptimizer::loadOptimization
void loadOptimization(bstr directory, bstr file);

Loads the optimization object from the specified directory/file location.

Parameter	Description
directory	The file name string.
file	The directory name string.

NolOptimizer::setBoundsByName
void setBoundsByName(bstr name, VARIANT values);

Sets the bounds for a variable, using the variable’s name for identification. This function can only be called after an optimization object is loaded. We recommend to first call *getBoundsByTag* to get the bound array.

Parameter	Description
name	Stores the variable name.
values	Stores the double array for variable bounds.

NolOptimizer::setBoundsByTag
void setBoundsByTag(bstr tag, VARIANT values);

Sets the bounds for a variable, using the variable’s tag for identification. This function can only be called after an optimization object is loaded. We recommend to first call *getBoundsByTag* to get the bound array.

Parameter	Description
tag	Stores the variable tag.
values	Stores the double array for variable bounds.

NolOptimizer::setValueByName
void setValueByName(bstr name, double value);

Sets the initial values for manipulated variables and constants for exogenous variables. This function can only be called after an optimization object is loaded.

Parameter	Description
name	Stores the variable name.
value	Stores the initial double value for given variable.

NolOptimizer::setValueByTag
void setValueByTag(bstr tag, double value);

Sets the initial values for manipulated variables and constants for exogenous variables. This function can only be called after an optimization object is loaded.

Parameter	Description
tag	Stores the variable tag.
value	Stores the initial double value for given variable.

NolOptimizer::setValues
void setValues(short type, VARIANT values);

Sets the initial values for manipulated variables and constants for exogenous variables. This function can only be called after an optimization object is loaded.

Parameter	Description
type	Specifies the type of variables. 0: OUTPUT 1: MANIPULATED VARIABLE 2: EXOGENOUS VARIABLE 3: STATE VARAIBLE
values	Stores the initial double array for given variable type.

NolOptimizer::setWeightsByName
void setWeightsByName(bstr name, VARIANT values);

Sets the weights for a variable, using the variable's name for identification. This function can only be called after an optimization object is loaded. We recommend to first call *getWeightsByName* to get the bound array.

Parameter	Description
name	Stores the variable name.
values	Stores the double array for variable weights.

```
NolOptimizer::setWeightsByTag
void setWeightsByTag(bstr tag, VARIANT values);
```

Sets the weights for a variable, using the variable's tag for identification. This function can only be called after an optimization object is loaded. We recommend to first call *getWeightsByTag* to get the bound array.

Parameter	Description
tag	Stores the variable tag.
value	Stores the double array for variable weights.

Index