

The Statistical Data Viewer

Volker Nannen

April, 2003

Abstract

This is the manual to the *Statistical Data Viewer*. The motivation for writing this application and an example of its actual use are discussed in my master's thesis [Nan03] on machine learning. If you simply want to get started and explore the application by yourself, read Section 2.3 of my thesis. It takes you through all the steps of setting up a simple experiment. The thesis, the source code of the application and the binaries are all available at

<http://volker.nannen.com/work/mdl>

Contents

Contents	1
1 Experimental verification	2
1.1 The Statistical Data Viewer	2
1.2 A simple experiment	5
2 Manual	11
2.1 The editor	11
2.2 The scientific plots	12
2.3 Defining a process	13
2.4 Taking a sample	17
2.5 Working with polynomials	19
2.6 The file format	23
2.7 The Vandermonde object	24
3 Some implementation details	24
3.1 Program requirements	24
3.2 Functional division	25
3.3 The programming environment	26
3.4 Core algorithms	27
References	29
Index	30

1 Experimental verification

As an expert on statistics and machine learning you are asked to supply a method for model selection to some new problems:

A number of deep sea mining robots have been lost due to system failure. Given the immense cost of the robots, the mining company wants you to predict the risk of a loss as accurate as possible. Up to now about a hundred of the machines have been lost, under very different conditions. Decennia of experience with deep sea mining have taught the mining company to use some sophisticated risk evaluation models that you have to apply. Can you recommend MDL?

A deep sea mining robot has got stuck between some rocks. The standard behaviors that were supposed to free the robot have failed. Some of the movements it made have won it partial freedom, others have made the situation only worse. So far, about a hundred movements have been tried and the outcomes have been carefully recorded. To choose the next action sequence, can you recommend MDL?

Before we are going to use MDL on problems that are new to us, we want to be sure that it is indeed a strong theory, valid even without the need of any preprocessing of the data or other optimizations that are common if a method is repeatedly applied to the same domain. In the case of MDL there are countless ways of expanding and compressing data and sooner or later we will find one that matches good models to short descriptions in a specific domain. But this does not convince us that it can be universally applied.

To experimentally verify that MDL would be a good choice to solve the problems above, we want to

1. test it on a broad variety of problems
2. prove that we use the shortest description possible

1.1 The Statistical Data Viewer

There are two factors that limit the type of data that is usually used for statistical experiments: availability and programming constraints.

Data availability. A major problem in machine learning and in statistics in general is the availability of appropriate data. One of the basic principles of statistics says that a sample that has been used to verify one hypothesis cannot be used to verify another one. And one and the same sample can never be used to both select and to prove a hypothesis. If we would do so, we would simply optimize on the sample in question.

This applies to model selection as well. Not only are the individual models hypotheses in the statistical sense, a general method for model selection can

1 EXPERIMENTAL VERIFICATION

itself be viewed as a hypothesis. If we want to experimentally verify methods for model selection we need a large supply of unspoiled problems and data.

Mathematical programming. A major obstacle to an objective diversity of the data is the way the common mathematical packages work. They are based on scripting languages that make heavy use of predefined function calls. This integrates nicely with most mathematical concepts, which are usually defined as functions. But it is not the preferred way to handle complex data structures or to conduct sophisticated experiments. It also severely reduces the quality of the outcome of an experiment. It is quite difficult and tiring to manipulate the graphical representations of data by using function calls.

Another problem of a function oriented approach is that it is the responsibility of the user to keep the definitions and the results of an experiment together. Users are often reluctant to play with the settings of complex experiments as that requires an extensive version control of scripts and respective results. And instead of spending a lot of time on writing a lot of different scripts for a lot of different experiments, researchers tend to do only minor changes to an experiment once it works correctly, and repeat it on large amounts of similar data. It is easier to try a method ten thousand times on the same problem space than to try it a few times on two different problem spaces.

A visual approach. To overcome the limitations of mathematic scripting and to guarantee diversity of the learning problems in an objective way we developed the *Statistical Data Viewer*: an arbitrary precision math application with a modular graphical user interface that can visualize all the abstract difficulties of model selection. It is well documented and can be downloaded for free at

<http://volker.nannen.com/work/mdl>

The *Statistical Data Viewer* makes all the definitions and results of an experiment available through a sophisticated editor. Experiments can be set up in a fast and efficient way. Problems can visually be selected for diversity. The performance of selection methods can be analyzed in a number of interactive plots. All graphical representations are fully integrated into the control structure of the application, allowing the user to change views and to select and manipulate anything they show.

To develop a working application within reasonable limits of time, the first version of the application is limited to the model family of polynomials and to two-dimensional regression problems, which are easier to visualize. Polynomials are used widely throughout science and their mathematics are well understood. They suffer badly from overfitting.

Great care was taken to give uninitiated students and interested lay persons access to the theory as well. No scripting language is needed to actually set up an experiment. The predefined mathematical objects—problems, samples, models and selection methods—are all available as visible objects. They can be specified and combined through the interactive editor which is extremely easy to use. The essential versions of MDL are implemented. They can be analyzed and compared with another independent and successful method for model selection,

cross-validation. In addition, the user can try his or her own penalization term for two-part MDL.

The progress of an experiment is observable and the execution can be disrupted at any moment. If possible, the graphs in the plots show the state of an experiment during its execution. It is possible to reproduce everything with little effort. All the data are saved in a standardized well documented format that allows for verification and further processing by other programs. The graphs are of scientific quality and available for print.

Available learning problems. To provide the user with a broad range of interesting learning problems, a number of random processes are available. They include

- smooth functions which can be approximated well by polynomials, e.g. the sinus function
- functions that are particularly hard to approximate like the step function
- fractals
- polynomials

When taking a sample from a process it can be distorted by noise from different distributions: Gaussian, uniform, exponential and Cauchy, which generates extreme outliers. The distribution over the support of a process can also be manipulated. It can be the uniform distribution or a number of overlapping Gaussian distributions, to simulate local concentrations and gaps.

Samples can also be loaded from a file or drawn by hand on a canvas, to pinpoint particular problems.

Objective generalization analysis. To map the performance in minimizing the generalization error in an objective way it is possible to check every model against a test sample drawn from the same source as the training sample. This has drawn some criticism from experts as the conventional way to measure the generalization error seems to be to check a model against the original source if it is known. I opted against this because:

- When speaking of generalization error the check against a test sample gives a more intuitive picture of the real world performance.
- For data drawn by hand or loaded from a file the original distribution might be unknown. In this case all we can do is to set aside part of the data as a test sample for evaluation. Depending on whether the source is known or unknown we now would have two different standards, the original function for known sources and a test set for unknown sources.
- If the source is known the test sample can be made big enough to faithfully portray the original distribution (by the law of large numbers) and give as fair a picture of the generalization error as a check against the original process.

1 EXPERIMENTAL VERIFICATION

- When using the original source the correlation between model and source has to be weighted against the distribution over the support set of the source. Especially in the case of multiple Gaussian distributions over the support this can be extremely difficult to compute.
- Although we concentrate on i.i.d. samples, this method allows us also to train a model on a sample with Gaussian noise and to measure the generalization error on a test sample that was polluted by Cauchy noise (extreme outliers) and vice versa or to use different distributions over the support set.

An independent method to compare. Comparing the predictions of MDL with the generalization analysis on an i.i.d. test set does not show whether MDL is a better method for model selection than any other method. For this reason the application includes an implementation of cross-validation. Cross-validation is a successful method for model selection that is not based on complexity theory. It can be seen as a randomized algorithm where we select the model that is most likely to have a low generalization error. The method divides the training sample a couple of times into a training set and a test set. For each partition it fits the model on the training set and evaluates the result against the respective test set in the same way as the generalization analysis uses an independent test sample. The results of all partitions are combined to select the best model.

1.2 A simple experiment

To make you familiar with the *Statistical Data Viewer* and to introduce you to some of the basic problems of model selection, this section will take you through all the steps of a simple experiment.

The sinus wave is very common in nature and comparatively easy to model by a polynomial. To find out whether MDL is a useful method for data prediction we conduct our first experiment on a sinus wave of a single frequency.

Experiment 1: validity of MDL

Hypothesis: MDL can minimize the generalization error.

Source: a sinus wave with frequency $f = 0.5$ and amplitude 1.

Noise: normal (Gaussian) with variance $\sigma^2 = 1$.

Support: uniform over the interval $[0, 10]$.

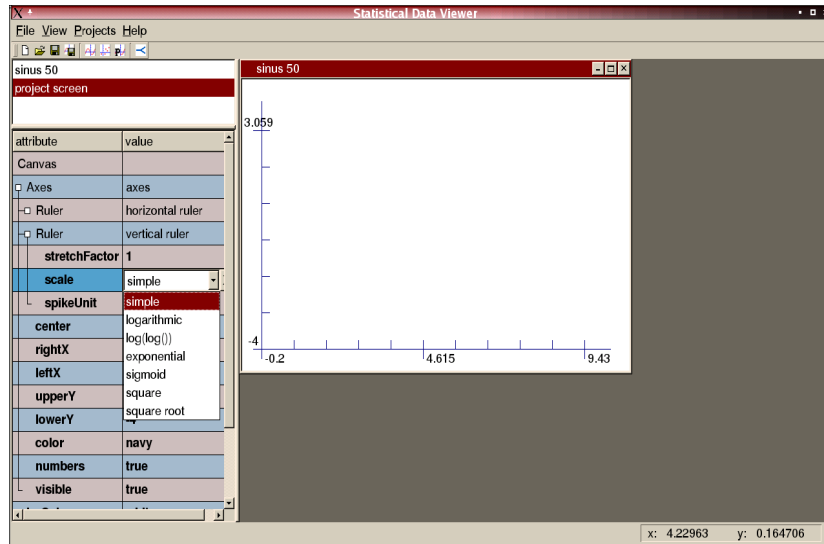
Sample: three samples of 50, 150 and 300 points.

Test set: i.i.d., 3,000 points.

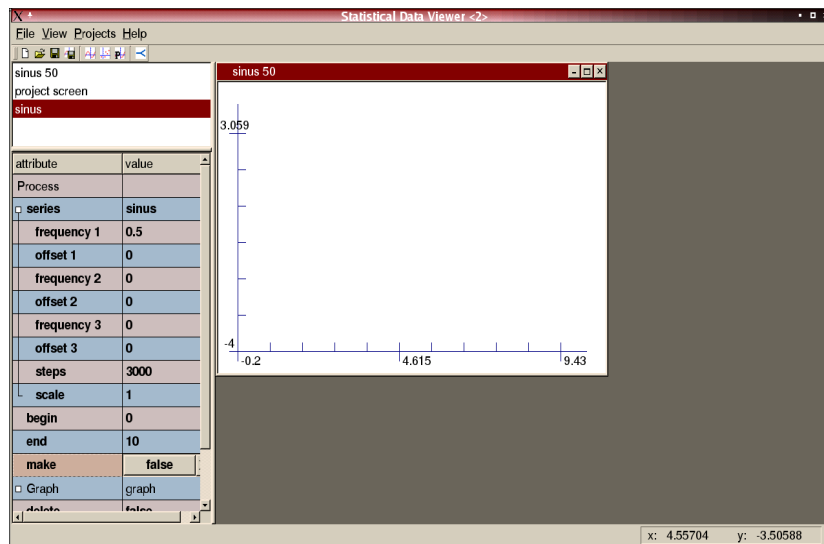
Figure 1 shows how the source signal is created in the application and Figure 2 shows how the samples are drawn from the source and how an experiment for model selection is started.

1.2 A simple experiment

Figure 1: Creating the source signal



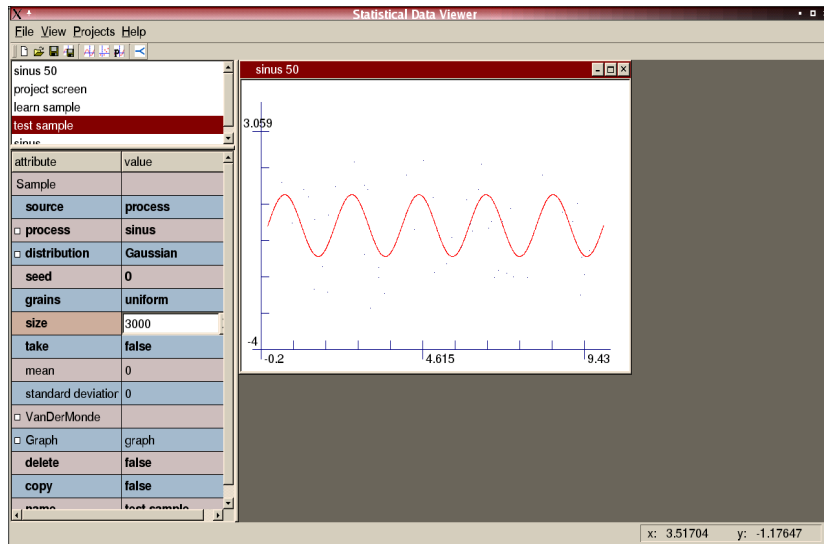
Clicking on the NEW PROJECT icon has created a new project. The project name can be defined and the attributes of the main plot can be set, e.g. range, size, scale, color and the origin of the rulers. The list view makes it possible to organize the attributes into a comprehensive hierarchy and to access them easily.



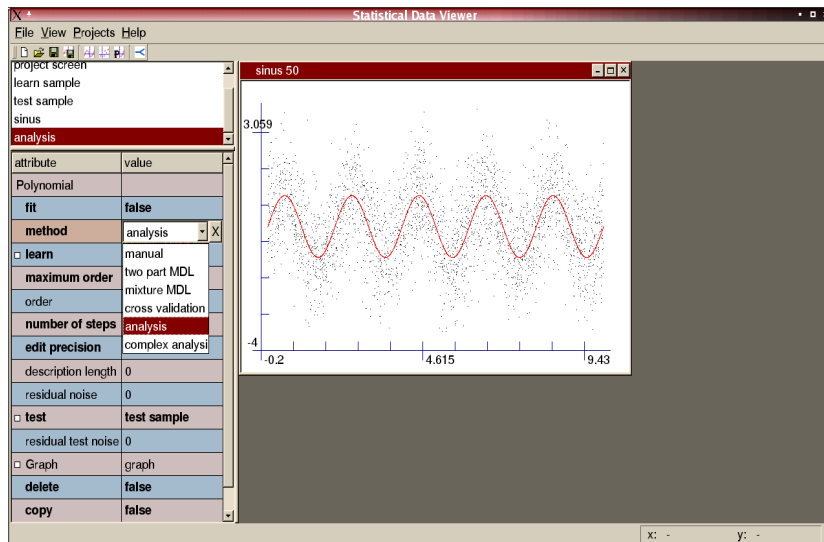
Clicking on the PROCESS icon has loaded a new process into the editor. From the available processes the sinus wave has been selected and the attributes were set: one frequency of $f = 0.5$ and support interval $[0, 10]$. A click on the MAKE button in the editor has created the signal.

1 EXPERIMENTAL VERIFICATION

Figure 2: Creating the samples and starting an experiment



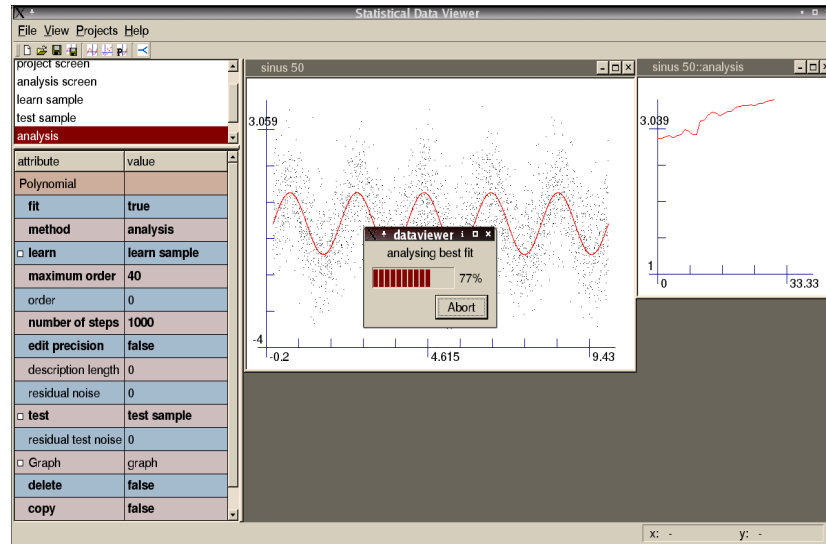
Clicking on the SAMPLE icon loads a new sample into the editor. A sample of 50 points has already been created and is visible on the main plot. Now the test sample is being defined: source and noise are specified and the size is entered. Clicking on the TAKE button in the editor will draw the sample.



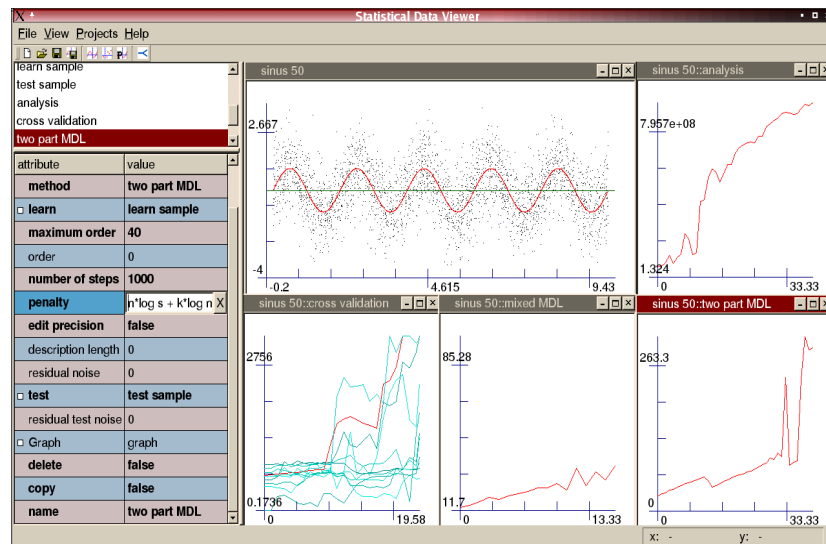
Clicking on the POLYNOMIAL icon has loaded a new polynomial into the editor. The training and the test sample are already specified and now the method for model selection is chosen from the pull-down menu. After the maximum order for the selection process is entered, a click on the FIT button in the editor will start the experiment.

1.2 A simple experiment

Figure 3: Sinus wave, witnessed by 50 points



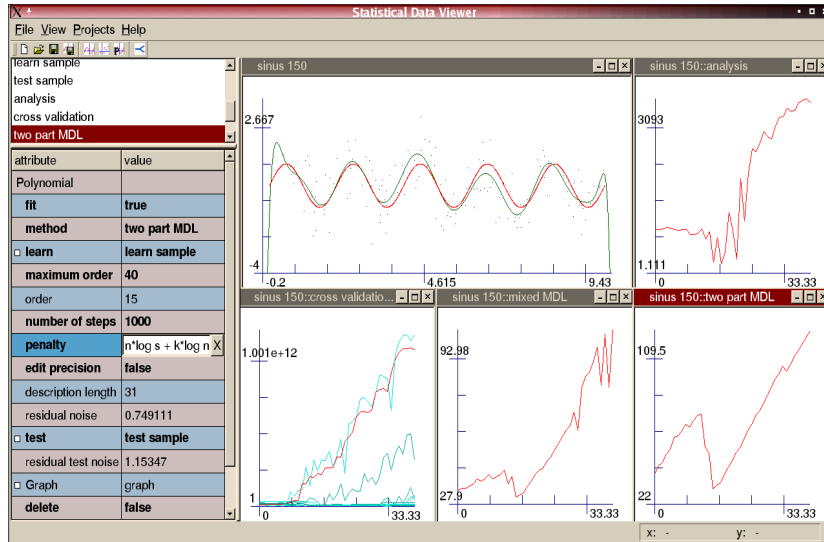
The generalization analysis is started: all polynomials in the range 0–40 degrees are fitted to the 50 training points and then checked against the 3,000 test points. The results are immediately plotted. The progress dialog allows to cancel the execution if it takes too long or if the early results call for a different design of the experiment.



4 experiments have been conducted on the 50 point sample and the results were plotted into separate plots. The generalization analysis in the upper right shows that the 0-degree polynomial is the best choice. In the lower part from left to right are the three results from cross-validation, mixture MDL and Rissanen's original two-part MDL. All agree with the analysis that zero degrees is the best choice.

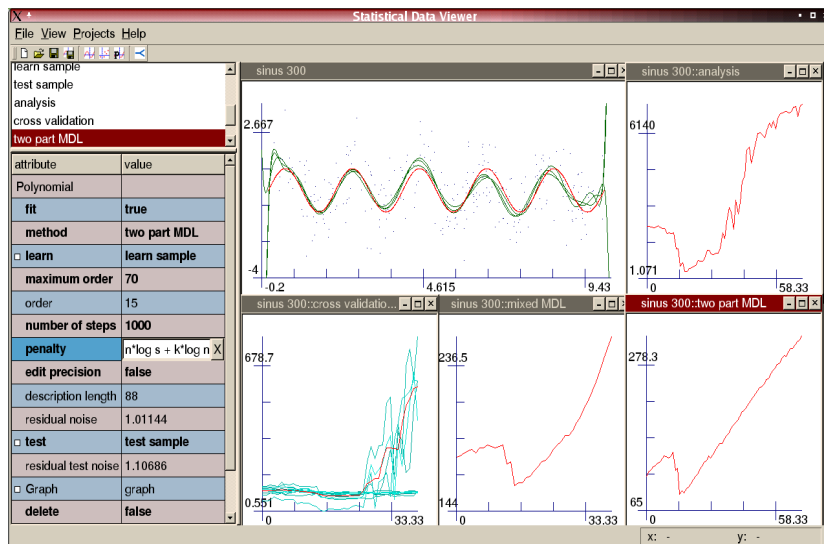
1 EXPERIMENTAL VERIFICATION

Figure 4: Sinus wave, witnessed by 150 and 300 points



150 point training sample. The generalization analysis shows 17 degrees as the optimum. Mixture MDL and Rissanen's original version slightly underfit and choose 15 as the optimum.

Cross-validation shows essentially the same picture as with 50 points. Anything from 0 to 10 degrees is good but 0 degrees is best.



300 point training sample. The generalization analysis now shows 18 degrees as optimal. 15-25 degrees are less than 5 percent worse than the optimum error. From 40 degrees on the generalization error increases rapidly.

All methods now generalize well. Rissanen's original version and mixture MDL choose 15 degrees, cross-validation 20 degrees.

50 points. A sinus wave with $f = 0.5$ shows 10 alternating peaks and valleys over the interval $[0, 10]$. An n -degree polynomial can have at the most $n - 1$ alternating peaks and valleys. We therefore expect polynomials with a low generalization error to be of at least 11 degrees.

Figure 3 shows that a training set of 50 points is too small to capture this threshold. 11 degrees do not show a local decrease in generalization error. On the contrary, from this point on the generalization error starts to get really bad. The best generalization error is achieved for the 0-degree polynomial, which is nothing but the mean of the training sample. For degree 0–10 the generalization error is almost as good and never worse than two times the optimum. But for 11 degrees the expected error rises to ten times the optimum and then increases so fast that the LOG-LOG scale is needed to make the result readable.

All three methods agree that the 0-degree polynomial is optimal. Cross-validation even predicts correctly that exactly from 11 degrees onwards the generalization error will get very bad.

150 points. With 150 points the generalization analysis shows that the optimum lies at 17 degrees. The error between 14 and 18 degrees is less than half way between the optimum and the error of the 0-degree polynomial. From 21 degrees onwards the generalization error is never again lower than that of a 0-degree polynomial. After that it increases so rapidly that the LOG-LOG scale has to be used again.

Mixture MDL and Rissanen’s original version give a good picture of the situation. Both choose 15 degrees as the optimum and predict a low generalization error for models between 14 and 18 degrees. Cross-validation is a disappointment. It shows essentially the same picture as with 50 points: almost constant generalization error for 0–10 degrees, optimum for 0 degrees and a very high generalization error beyond 10 degrees.

300 points. With 300 points we see all methods generalizing well. The generalization analysis now shows 18 degrees as optimal. 13–33 degrees are less than half way between a 0-degree polynomial and the optimum. From 40 degrees on the generalization error increases rapidly. Rissanen’s version chooses 17 degrees, mixture MDL 15 degrees and cross-validation 20 degrees, all of which are less than 10 percent worse than the optimum.

The three methods not only make a good choice, they also capture the general outline of the generalization analysis. Polynomials of degree 15–25 are shown as good candidates by all the three methods. They also show that anything from 35 onward is worse than taking the 0-degree polynomial.

Conclusion. Our simple experiment has shown that both our versions of MDL are good methods for data prediction and error minimization. When compared to cross-validation they can be called at least of equal strength.

2 Manual

The main window. The main window of the application is divided into a main menu, a workbench with multiple plots and a sophisticated editor. The user can work simultaneously on a number of different experiments, alias projects. They can be created, opened, selected and deleted via the main menu, by using the pull-down menus or the standard GUI icons. The way plots are shown on the workbench can also be changed via the main menu. Icons allow the user to add processes, samples or models to an experiment.

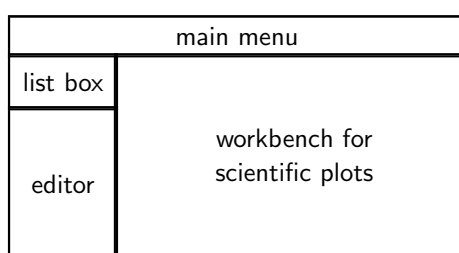


Figure 5: The main window of the application

When a new project is created, a new main plot is put on the workbench and the project is loaded into the editor. The main plot will show all the processes, samples and models of an experiment. For each selection method used, a new plot is added which shows the predictions of that method. A plot initially has a white background and an x and an y -axis with normal scale. These properties can be changed.

Above the editor is a list box that contains all the objects of an experiment. Creating an object via the main menu, clicking on its graph in one of the plots or selecting it from the list box will load the object into the editor. Once in the editor, the attributes of an object can be evaluated and changed. The plots and the project itself are also in the list box and can be loaded into the editor as well.

2.1 The editor

The editor shows the attributes of an object in a hierarchical, comprehensive way. Attributes include names, the color of a graph, size and scale of the plot, the number of points of a sample and, very important, the parameters of functions, distributions and selection methods. The seed for the random generator is also an attribute of objects that include random processes.

The editor shows the attributes in two columns. The left column shows the name of an attribute and the right column shows its respective value. If the value can be manipulated it is printed in bold. Clicking on an attribute opens a little editor in its place. For boolean attributes this is a button that can toggle its state. Other attributes can be selected from a list or entered into a text

field, depending on the type. If the attribute has a default value this value is available by clicking on a small button to the right of the editor.

Some attributes represent actions. Actions include copying or deleting the object, taking a sample from a process or starting a selection method. Clicking on an action attribute will start the action. If the action involves a random process, the SEED attribute is important. If the seed is zero, which is the default, the random generator will be fed with the computer time. Otherwise it will be fed with the actual seed. An action that includes a random process will always give the same results if fed with the same seed.

Objects that are visible on one of the plots have a GRAPH attribute that defines its color, style and visibility. Any of the many color names that are defined in the file *rgb.txt* on all Linux/Unix systems is a valid input. Numerical input of the format *#rrggbb* is also accepted.

Objects can even have other objects as attributes. When a sample is taken from a process, the process and the distribution around the process are its attributes. The object attributes of a model are the sample that it is trained on and the method that defines the number of parameters. Such object attributes can usually be selected from a pull-down menu. Once selected, clicking on the object attribute will expand it and its own attributes become visible. An object and its object attributes can be browsed much like a file tree on the common windowing systems.

2.2 The scientific plots

The functionality of the scientific plots is of central importance to the application. Graphs have always played an important role in mathematics. Sometimes the accessibility of a theory depends heavily on the availability of appropriate graphs. A scientific application should not only produce graphs as an optional add-on but give them a central position in the control structure.

In the *Statistical Data Viewer* a plot can be loaded into the editor by selecting it from the list box or by clicking on one of its axes. It has the obvious attributes height, width, background color and margin. No graphs are shown on the margin. It also has two AXIS objects as attributes. The attributes of an axis are color and visibility, location on the plot and the range. If a plot has to show everything in the range 0–10 along the x -axis, this is the place to define it. For every axis a number of scales can be chosen: SIMPLE (which is normal), LOGARITHMIC, LOG-LOG, EXPONENTIAL etc.

When moving the mouse over a plot, the location of the mouse along the x and y -axis is shown in a small box at the bottom of the main window. This is very helpful when analysing a graph. Hovering over a graph in the plot will show its name. Clicking on it with the left mouse button will load it into the editor. Clicking on a plot with the middle mouse button will zoom into the section around the mouse. Moving the mouse while pressing the middle button will move the area that is zoomed into. Releasing the middle mouse button will let the zoom disappear.

The behaviour of the right mouse button depends on which object is currently loaded into the editor. If it is the plot itself, clicking on the right button and then moving the mouse over a section of the plot will create a rectangle between the mouse and the point where the button was pressed first. If the mouse is not on the plot anymore when the button is released, nothing happens. But if it is, the section that is within the rectangle is defined as the new range of the axes. In this way portions of the plot that are of particular interest can be magnified again and again. To go back to the old range of the axes the editor has to be used. Clicking on one of the axes will load the plot into the editor.

A plot is saved by selecting the `SAVE PLOT` option from the main menu. The application saves all plots in the `PNG` format. Use the `convert` program on Linux/Unix or your favourite image processor to convert it to another format.

2.3 Defining a process

Besides the standard attributes, a process has a `SERIES` object that defines the type of the process, two attributes that define the beginning and the end of the range over the x -axis of the process and a `MAKE` action which will create the process once its properties are defined.

Seven different processes are available as predefined two-dimensional learning problems. Most of them are common in nature. They include sinus waves, time series, fractals and the step function. Some are smooth, others have sharp corners. Some have extreme values, others deviate only slightly from the main path. Using the default parameters of a process and the random generator will generate a broad range of interesting problems. Manipulating the process specific parameters will generate an even broader variety of different problems.

Besides the process specific parameters, all processes have a `SYSTEM NOISE`, a `SEED`, a `STEPS` and a `SCALE` attribute. Because the evolution of a time series often involves some system noise, most processes depend on the random generator. The system noise must not be confused with the deviation of the points of a sample along the y -axis. It is a major factor in defining the shape of a time series. The `SYSTEM NOISE` attribute defines the variance of the Gaussian distribution over this noise. Setting the `SEED` attribute to a value other than zero will always produce the same shape for such a process.

The `STEPS` attribute defines how fast the process will evolve over the defined range of the x -axis. The default values are set in a way that most series can be approximated well by a 50 degrees polynomial. More steps will let the series evolve faster and a higher degree polynomial will be needed to learn it. The `SCALE` attribute defines the amplitude of a process along the y -axis.

In all graphs the horizontal x -axis shows the time t and the vertical y -axis shows the value y as a function of t . When more than one variable changes with time, only the y value is visualized in the graph and considered for model selection.

Autoregression. Autoregressive time series are particularly common in nature. The value y_t at time t of such a series is the weighted sum

$$y_t = a_0 + \sum_{i=1}^n a_i y_{t-i} + \epsilon \quad (1)$$

over n previous values of y . ϵ is the system noise. The *Statistical Data Viewer* allows to specify six different parameters a_0 – a_5 but three parameters is usually quite enough. Figure 6 shows an example with $a_1 = 0.5$, $a_2 = 0.5$. The other parameters are equal to zero:

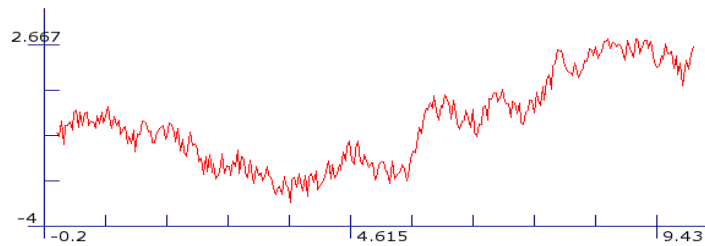


Figure 6: Autoregression

Sinus wave. The sinus wave is also very common. Five frequencies can be specified, each with an individual offset. The resulting function is

$$f(x) = \sum_{i=1}^5 \sin(f_i x + o_i) \quad (2)$$

The example in Figure 7 uses zero offsets and the four frequencies $f_1 = 1.05$, $f_2 = 0.8$, $f_3 = 0.55$ and $f_4 = 0.15$:

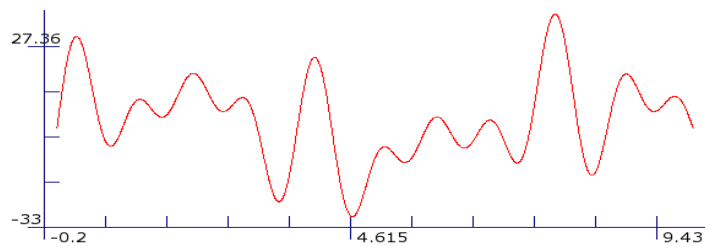


Figure 7: Sinus wave

Logistic map. A logistic map is a time series of the form

$$y_t = a y_{t-1} (1 - y_{t-1}) + \epsilon \quad (3)$$

2 MANUAL

with ϵ the system noise. It was first published by the Belgian mathematician Pierre Verhulst sometime between 1838 and 1850.

The example in Figure 8 has $a = 0.5$:

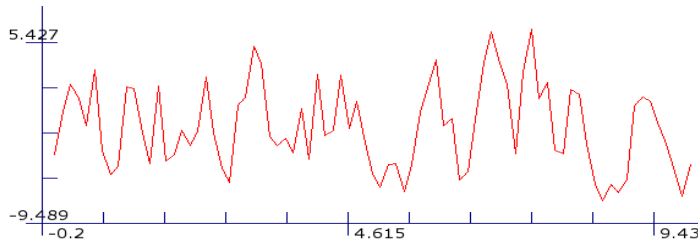


Figure 8: Logistic map

Lorenz attractor. The famous Lorenz attractor is a self similar object. It is also a time series. E.N. Lorenz discovered it when he was working on models of the weather [Lor63]. Its evolution is governed by the equations

$$\begin{aligned} y_t &= a(z_{t-1} - y_{t-1}) + \epsilon \\ z_t &= by_{t-1} - z_{t-1}w_{t-1} + \epsilon \\ w_t &= y_{t-1}z_{t-1} - cy_{t-1} + \epsilon \end{aligned} \quad (4)$$

z and w are not shown in the graph and are not considered in the experiments. The default values are $a = 10$, $b = 28$, $c = 2.667$:

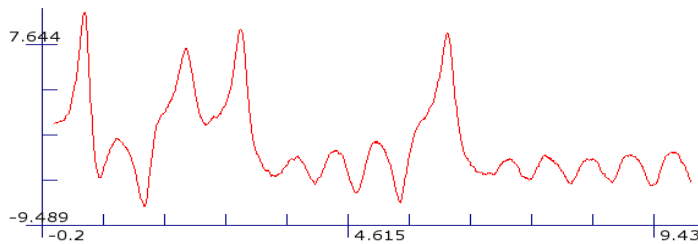


Figure 9: Lorenz attractor

Pendulum. The movement of a noisy pendulum with orbit o and frequency f is defined as

$$y_t = \sin(y_{t-2}) - cy_{t-1} + f \cos(o(t-2)) + \epsilon \quad (5)$$

The example in Figure 10 has $c = 0.2$, $f = 0.5$ and $o = 0.67$.

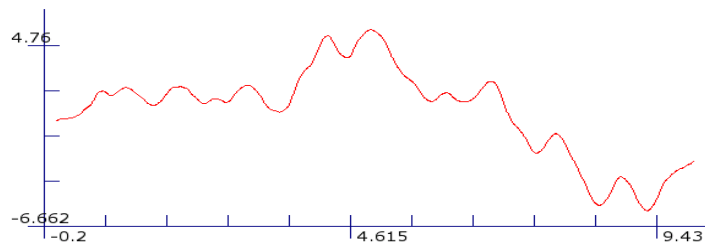


Figure 10: Pendulum

Step function. The step function oscillates n times between two values. The example in Figure figure-step-function has $n = 10$ and oscillates between minus ten and ten. The points where the function switches between values are chosen at random. The same non-zero seed will produce the same step function.

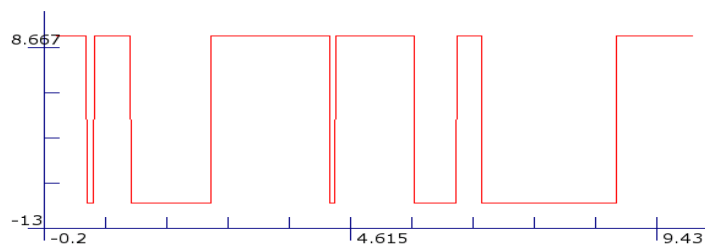


Figure 11: Step function

Thom map. The Thom map is also a time series. R. Thom [Tho93] discovered it when he searched for a simple discrete equivalent to the Lorenz equations which were defined for continuous time.

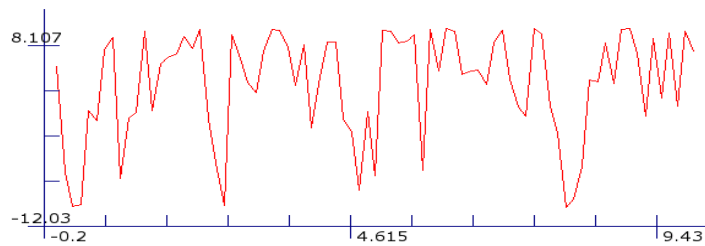


Figure 12: Thom map

$$\begin{aligned} y_t &= a y_{t-1} + b z_{t-1} + \epsilon \\ z_t &= c y_{t-1} + d z_{t-1} + \epsilon \end{aligned} \tag{6}$$

The z -axis is not shown in the graph and is not considered in the experiments. The example in Figure 12 has $a = 0.5$, $b = 0.3$, $c = 0.3$ and $d = 0.4$.

2.4 Taking a sample

There are five different ways to take a sample:

1. loading a sample from a file,
2. taking it from one of the predefined processes,
3. taking it from a polynomial,
4. drawing it by hand and
5. merging two or more samples into a new sample.

The METHOD attribute defines which one will be used.

Loading a sample from a file. If the FILE method is selected, the user has to select a file through the standard file browser. The file has to be in ASCII format. Once the file is selected, a new window pops up and shows the contents of the file. Now there are two ways to select the data: by regular expression or by column. The options can be selected by radio buttons. You should have some experience with regular expressions when using this option. An introduction and a list of special characters are available at <http://doc.trolltech.com/3.1/qregexp.html#1>.

If the regex option is chosen, a regular expression has to be entered for both the x and the y values. All lines of the file are parsed. For each line that is matched by both expressions a point is added to the sample. In order to work correctly, the expression usually has to match more than only the intended value. The number of the opening parenthesis that encloses the portion of the expression that actually contains the value must be entered in a separate input field. You can check whether the expressions capture the correct values by pressing the APPLY button. It will show the captured portions of the text in different colors. If you are satisfied, press the OK button.

Example: we want to study the number of sold products as a function of the consumer price. The lines of our input file have the following format:

consumer price: 543.21, production cost: 342.23, items sold: 4300

A regular expression that will match the x -value is

$$\text{\textasciitilde}\backslash D+(\backslash d+\backslash.?\backslash d*)$$

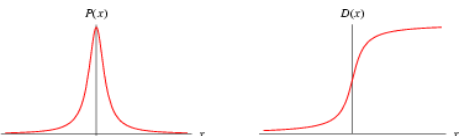
and the first opening parenthesis captures the value. The y -value will be matched by the expression

$$\text{\textasciitilde}\backslash D+\backslash d+\backslash.?\backslash d*)\{2,2\}\backslash D+(\backslash d+\backslash.?\backslash d*)$$

where opening parenthesis number three captures the value.

The other option is to use columns. Every line of the text is divided into a number of columns by a regular expression. This expression is usually very simple, e.g. white space or a colon. The column numbers that specify the x and the y -value have to be entered. For each line of the file that has real values in both columns a point is added to the sample.

Taking a sample from a process. In order to use this option, a process must already have been specified. It can then be selected as an attribute of the sample. A distribution over the measurement noise should also be selected. There are four distributions available: Gaussian, uniform, exponential and Cauchy. The Cauchy distribution is important to simulate far outliers. Like the other distributions the Cauchy distribution is symmetric around the mean and non-increasing. But it has no calculable variance. Its density function is

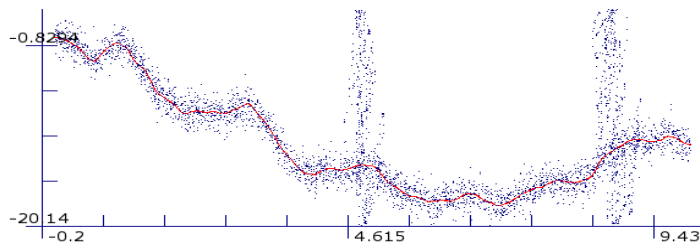
$$P(x) = \frac{s}{\pi((x - \mu)^2 + s^2)},$$

(7)

where μ is the mean and s the distance between the mean and the point that has half the probability of the mean.

The points will be taken at random over the support, which is the range along the x -axis for which the process is defined. The GRAINS attribute specifies the distribution over the support. It can either be uniform or a number between one and nine. If there is one grain, the distribution will be Gaussian around the center of the support. If there are two grains, there will be a center at one third and another at two thirds of the support range. The distribution over the support will be the joint distribution of two Gaussian distributions. Three grains will have a center at 1/4, 2/4 and 3/4 of the range etc. With nine grains the joint distribution of nine Gaussian distributions is almost uniform.

After the desired size of the sample has been entered at the SIZE attribute, a click on the MAKE action will take the sample from the process. To guarantee that all experiments can be repeated, values of the SEED attribute other than zero will always produce the same sample under the same conditions.

Drawing a sample. Drawing a sample is straightforward. When this option is chosen and the sample is in the editor, moving the mouse over the main plot and pressing the right mouse button will produce points on the plot. Points can also be drawn after a sample has already been loaded from a file or been taken from one of the predefined processes. In this way particular problems can be pinpointed, e.g. far outliers. You also can simulate noise along the entire range of the y -axis for a limited range of x . In the example 3,000 points were taken from a pendulum in the usual way. After that, 1,000 points were added by hand to simulate corruption of the measurements at $x = [4.6, 5.2]$ and $x = [8.4, 9.3]$



Pendulum with corrupted measurement

Taking a sample from a polynomial. Taking a sample from a polynomial is very similar to taking a sample from a process. The only difference is that a polynomial must be specified in a different way. Entering the parameters of a polynomial by hand usually does not lead to the desired results. Instead, the following procedure should be followed: first, draw a line by hand on the plot that has the outline of the desired polynomial. Then, fit a polynomial of the desired degree on this hand drawn line. How to do this will be explained in the section on polynomials. This polynomial should then be used to take other samples from it. The original hand drawn line can be deleted.

If you still insist on entering the parameters by hand, this can be done as well. Save a project that contains a polynomial of the desired degree. All projects are saved in XML format and can easily be edited by hand. Replace the parameters of the saved polynomial by your own values and reload the experiment.

Merging samples. When the MERGE method is selected, up to four different samples can be specified that will be merged into a new sample. As usual, the TAKE action will create the new sample from the specified source samples. The source samples are not changed or deleted.

Splitting samples. A sample can also be split in two. This is useful when a sample has been taken from a file or drawn by hand and we need to split it into a training and a test sample. Splitting is not a method of a sample. You need to select the SPLIT operator from the main menu.

Once created, the SPLIT operator is loaded into the editor. The sample that will be split has to be selected and the number or percentage of points that will go into subsample *A* must be entered. Clicking on the SPLIT action will split the selected sample at random. The specified number of points will go to subsample *A* and the rest will go to subsample *B*. The new samples are visible on the main plot and are available from the list box. The selected sample is not changed or deleted.

2.5 Working with polynomials

Create a POLYNOMIAL object by clicking on the respective icon of the main menu. Before using it, the METHOD and the LEARN attribute must be specified. For the LEARN attribute one of the samples of the project has to be selected as the training sample. The optional TEST attribute allows to select another

sample as the test sample. Once the polynomial is fitted to the training sample the mean squared error on the training sample is shown as the RESIDUAL NOISE attribute. If a test sample is selected, the mean squared error on the test sample will be shown as the RESIDUAL TEST NOISE attribute.

The METHOD attribute allows the user to select among six options:

- manual:** specify the degree and precision of a polynomial by hand.
- two-part MDL:** find the optimum degree by penalization.
- mixture MDL:** by using the minimax algorithm for mixture MDL.
- cross-validation:** to compare with the performance of the MDL methods.
- analysis:** calculate the generalization error on a test sample.
- complex analysis:** repeat the generalization analysis over training samples of different size.

Using a method other than MANUAL will open an additional plot that shows the results of the method. Clicking on the graph of the additional plot will load the polynomial into the editor, just like clicking on the polynomial in the main plot.

The manual method. The MANUAL method was specifically added to allow the user to play with the parameters of a polynomial. You can specify the degree of a polynomial and the precision in digits per parameter. Clicking on the FIT action will calculate the least square fit of the polynomial on the sample. Playing with the precision will show you the effects of rounding. Usually, for less than $k/3$ digits per parameter the polynomial cannot approximate the sample. The mean squared error on the training sample becomes very high. $k/2$ gives an almost optimum result. Higher precisions will let the mean squared error decrease only marginally. Section 3.3 on page 28 tells you more about rounding problems.

Clicking on the PRECISION EDITOR attribute will allow you to specify the precision for individual parameters. A window pops up with the value and the precision of each individual parameter. The precisions can be changed and the polynomial can be fitted again, resulting in the parameters that give the best result for the specified precision.

The two-part MDL method. If you use this method or any of the other methods that calculate an optimum degree, you cannot manually change the degree or the precision of the polynomial. Rather, you specify the maximum degree that will be looked at and the method will search for the best degree. 50 to 100 degrees as a maximum are usually quite enough to work with. The rounding procedures give precise results for at least a 150 degrees¹ but computations become awfully slow. If processor speed continues to grow at the current

¹ more on this in Section 3.4

2 MANUAL

rate we will soon watch a 1,000 degree polynomial being fitted at a speed fast enough for direct interaction. But this will not significantly alter our results.

The `PENALTY` attribute specifies the way the total code length is calculated. The default is Rissanen's original term, as calculated in Section 2.2 of the thesis [Nan03].

$$n \log \sigma^2 + k \log n \tag{8}$$

You can modify this term or replace it by the AIC, which is $n \log \sigma^2 + 2k$. The `PENALTY` attribute will accept every function name that is specified in the C/C++ math libraries: `log()`, `ln()`, `sin()`, `cos()`, `sqrt()`, and so on. The standard operators `+`, `-`, `/`, `*`, `^` are defined. In addition, the following single letter symbols are available:

- n** the size of the sample
- k** the degree of the polynomial
- s** the mean squared error σ^2 of the polynomial on the sample
- p** the real number π
- e** the real number e

Clicking on the `FIT` button will calculate the least square polynomial for every degree smaller than the maximum degree. The complexity is calculated according to the penalty term and immediately shown on the additional plot. Once the complexity for all degrees has been calculated, the model that had the shortest two-part complexity is selected as the good model. It is calculated and plotted into the main graph.

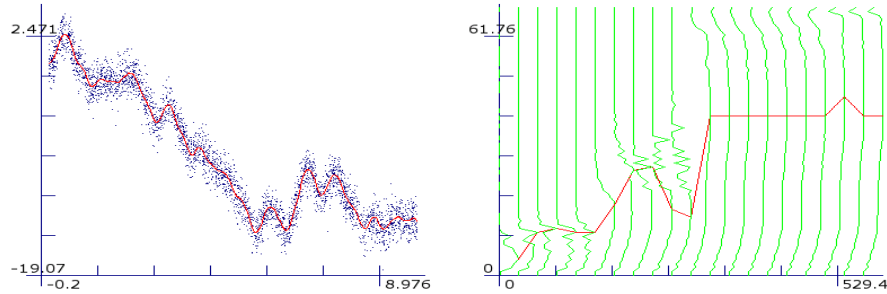
If you want to save time and don't want to look at every degree, specify the `NUMBER OF STEPS` parameter. If it is bigger than the maximum, all degrees will be looked at. If it is smaller, only this number of degrees will be looked at. Starting at the 0-degree polynomial and continuing to the maximum degree, superfluous degrees will be left out at equal intervals.

The mixture MDL method. The `MIXTURE MDL` method works exactly like the `TWO-PART MDL` method, except for the fact that no penalty term can be specified.

The cross-validation method. This method also works like the `TWO-PART MDL` method without the penalty term. Future versions may allow to modify the smoothing algorithm and the number of partitions used in the algorithm.

The analysis method. This method also works like the `TWO-PART MDL` method without the penalty term. But the test sample is now mandatory. Keep in mind that it should contain at least 1,000 points. Don't make it too big as this will slow down calculations. 3,000 points have been proven to be a good size for

Figure 13: Complex generalization analysis



Left: Pendulum with 600 point training set and 3,000 point test set.

Right: Complex analysis with 20 different samples of size 30–600 points. Each vertical line represents the generalization error for one sample, starting on the left with the smallest sample. The lower the error the more the line deviates to the right. The single horizontal line goes through all the optima. From a sample size of 330 points onwards the optimum stays almost constantly at 41 degrees.

most learning problems. The optimum model according to the generalization analysis is plotted into the main plot.

The complex analysis method. This last method was not introduced during the experiments. Like the ANALYSIS method it calculates the generalization error for all degrees. But it does so for training samples of varying size. The NUMBER OF SAMPLES attribute specifies how many samples are used. These samples are subsets of the selected training sample. If the training sample has 600 points and the number of samples is 20, the first set contains 30 points, the second 60, the third 90 and so on. The test sample is not changed.

The plot of this method is three-dimensional. The x -axis shows the sample size, the y -axis shows the degree and the z -axis shows the generalization error. The view is tilted so that a high value along the z -axis results in a shift of a point to the left. For each sample the generalization error is a vertical line along the y -axis. The lower the generalization error, the more this line goes to the right and the higher the error the more the line goes to the left. To mark the optimum number of degrees per sample size, a single horizontal line connects all the optima.

To get more reliable results for the generalization error per sample size, for small samples the average is taken over multiple samples of the same size. For sample sizes smaller than half the size of the original training sample the average is taken over two independent samples of the same size. For samples smaller than a third the average is taken over three and for samples smaller than a fourth the average is taken over four independent samples of the same size.

The COMPLEX ANALYSIS method makes it possible to study the behavior of the generalization error as a function of the size of the training set. If polynomials

converge only slowly with the original problem, the optimum will rise continuously. But smooth functions like the sinus wave, the pendulum and the Lorenz attractor usually show an initial increase followed by an almost flat region.

2.6 The file format

All projects are saved in XML². The standard file extension used by the *Statistical Data Viewer* is `.sdv`. XML is a well defined standard, there is no shortage of fast and convenient XML parsers and XML can be read by humans. If a file is corrupted or you miss a certain functionality in the editor you can easily modify an `.sdv` file in a text browser.

XML is much more than just a human readable file format. It allows an application to organize the contents of a project in a robust, standardized and comprehensive way. Data can easily be exchanged, shared and combined with other applications on different platforms. With the proper tools an XML file can also easily be transformed into a webpage or a \LaTeX file.

An XML file looks much like an HTML file with a lot of tags that are enclosed in brackets. Unlike HTML, all XML tags must either have a closing slash or be matched by a closing tag of the same name preceded by a slash. In an `.sdv` file each mathematical object has its own tag name. Its attributes are listed as name-value pairs of the format `name="value"`. Objects and attributes have the same name as in the application. As a rule, object names start in upper case and attributes are in lower case. The example is a `PROCESS` object whose attributes have not yet been specified:

```
<Process seed="0" begin="0" copy="false" end="10"
  series="---" ID="3" name="process &lt;1>" make="false" />
```

Some of the attributes are not visible in the editor, for example the `ID` attribute. Don't touch them, especially the `ID` attribute. If you keep the file consistent, the application will read and accept your changes. Even if it doesn't understand them, the worst thing that will happen is that the application resorts to the default values.

If an object has other objects as attributes, the initial tag doesn't close with a slash, only with a closing bracket. Next come the nested objects. The object is ended by repeating the tag name in brackets, preceded by another slash. In the example below the process was defined as a pendulum without system noise. Changing the system noise in the file is equivalent to changing it through the editor.

```
<Process seed="0" begin="0" copy="false" end="10"
  series="pendulum" ID="2" name="pendulum" make="false" >
  <TimeSeries frequency="0.5" steps="1000" offset="0"
    seed="0" orbit="0.67" scale="10" system_noise="0"
    constant="0.2" ID="2" name="Pendulum system" />
</Process>
```

² Extensible Markup Language. Look at <http://www.w3.org/XML> for literature and specifications.

2.7 The Vandermonde object

Every sample has a `VANDERMONDE` object. The importance of the Vandermonde matrix is explained in Section 3.3 on page 27. It is essential to the algorithms that work on the sample: least square fitting, calculating the Fisher information and taking the mean squared error. Building the Vandermonde matrix and especially putting it in triangular form are time expensive operations³. A Vandermonde matrix of size $n \times n$ is useful for polynomials of degree $n - 1$ or lower. But, at least with the algorithms known to the author of this application, for a higher degree polynomial a bigger Vandermonde matrix has to be built from the sample and again put into triangular form. The attributes of the `VANDERMONDE` object specify the dimension of the matrix and whether it has already been built and put into triangular form.

Normally you shouldn't use this object. Selecting a sample for a polynomial and fitting an n -degree polynomial on it will build the $n + 1$ degree Vandermonde matrix for you. The matrix is preserved and will be reused whenever needed. It is saved and loaded together with the project. But remember that first fitting a low degree polynomial on a sample and then fitting a high degree polynomial will result in the expensive recalculation of the Vandermonde matrix.

3 Some implementation details

If you plan to add your own functionality to the *Statistical Data Viewer* you should of course refer to the documentation that comes with the actual distribution. The following pages give only a general outline of design of the application.

3.1 Program requirements

These are the program requirements that stood at the basis of the *Statistical Data Viewer*:

open source: high priority is given to programming techniques that encourage others to improve the program and to adapt it to their own needs. In particular, this implies

- a single well known programming language
- use of well documented open source libraries
- a modular, object oriented design
- clean programming interfaces
- self documenting code

machine independence: the application has to work on the majority of computer systems popular in academic research.

functional division: the interface must show a clear functional division.

³ The time complexity of putting it into triangular form is $O(n^3)$.

3 SOME IMPLEMENTATION DETAILS

direct manipulation: all essential mathematical attributes of an object have to be available for immediate manipulation and interpretation.

observability: all visible objects must reflect their current state. Progress bars must give estimates of the duration of time consuming executions and it must be possible to stop them at random.

usability: the application has to be accessible to students that are unfamiliar with the theory.

reproducibility: it must be possible to reproduce an experiment in all its aspects. It must be possible to randomly load and save all states of an experiment in a robust and fault tolerant way.

standard file format: all experiments are saved and loaded in XML. XML is a well defined standard, there is no shortage of fast and convenient XML parsers and XML can be read by humans.

broad input: it must be possible to enter samples in the following ways:

- read samples from a file
- take samples from a broad and interesting set of predefined processes
- draw and manipulate samples by hand

scientific output: the application must provide high quality plots which can be used for publication.

scientific documentation: the used algorithms have to be fully documented as they might be influencing the results.

3.2 Functional division

As shown in Figure 14, the application is divided into four independent modules:

a main menu that satisfies all the standard expectations. It allows the user to start a new project and to save, load or close an existing one. It should eventually have a help section and should allow for customization of the application.

a project which consists of a number of objects like processes, distributions, samples, models and methods for model selection. They can be created, edited, copied, saved and deleted at random.

All mathematical computations are done within the project module.

an editor that allows for easy analysis and direct manipulation of all objects. Attributes are arranged in a comprehensive hierarchy. Actions show immediate response. Standard input methods like choice lists, buttons and text fields are used when appropriate. It is also possible to restore default values after a change.

3.3 The programming environment

a plot where processes, samples, models, methods and experiments can be observed and randomly selected for analysis and manipulation. The execution of an experiment must be observable on the plot. The graphical output must be of scientific quality and available for print.

Communication between the menu, a plot, a project and the editor is done in XML. This allows for independent implementation of the different modules. The editor could be an HTML page. A plot could be a Java applet. It is also possible to let the mathematical objects communicate via XML. This helps to distribute calculations over a network.

This first version of the *Statistical Data Viewer* is a proof of concept and hopefully not the final application. There are exceptions to the XML-communication that should be eliminated. The programming interfaces can be simplified. It should be possible to distribute calculations over a network. The final idea is to have an open source system for statistical analysis that can be customized and extended much like the L^AT_EX system, with packages and front ends that can change with the purpose of the system.

3.3 The programming environment

With ongoing standardization in the Linux/Unix world we now witness two strong GUI libraries that run on almost all Linux and Unix machines: the Gnome libraries and the KDE libraries, the latter build around a core of Qt libraries. The *Statistical Data Viewer* is build on the core Qt libraries. They are open source⁴, fast, strictly object oriented, well documented and to my judgment quite beautiful. They are written in C++. A powerful programming environment for KDE and Qt is available, KDevelop, which improves and speeds up the development of large applications. Finally, Qt is platform independent. Trolltech, the Norwegian company which develops Qt, takes great care to guarantee correct execution of code written for its Qt libraries on the major Linux and Unix distributions as well as on MacIntosh and Microsoft Windows machines. Though the code has to be compiled again on each system, the fast and reliable performance justifies this extra step.

⁴ Under the '*Q Public License*'. If you want to use Qt for commercial software, you have to pay, of course.

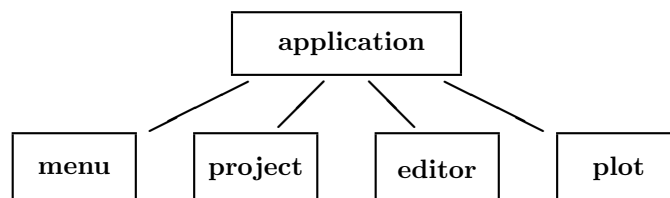


Figure 14: Modules of the application

3 SOME IMPLEMENTATION DETAILS

Arbitrary precision library. To have full control over the arbitrary precision arithmetic and to optimize the essential time consuming algorithms, the application uses an independent arbitrary precision library: MAPM⁵. MAPM is a portable arbitrary precision math library which is open source. While most other arbitrary precision libraries emphasize integer arithmetic, this library was specifically designed for floating point arithmetic. A notable oddity of this library is that it handles precision in digits, not in bits [Rin01].

3.4 Core algorithms

Least square fitting. The method for linear regression used in the *Statistical Data Viewer* is least square fitting based on Gaussian elimination. A k -degree polynomial has $k + 1$ parameters:

$$f_k(x) = a_0 + a_1x + \cdots + a_kx^k \quad (9)$$

The squared error between such a polynomial and an n points sample $\{(x_1, y_1) \dots (x_n, y_n)\}$ is

$$\sigma^2 = \sum_{i=1}^n \left(y_i - (a_0 + a_1x_i + \cdots + a_kx_i^k) \right)^2, \quad (10)$$

the partial derivatives of which are

$$\begin{aligned} \frac{\partial \sigma^2}{\partial a_0} &= -2 \sum_{i=1}^n [y_i - (a_0 + a_1x_i + \cdots + a_kx_i^k)] = 0 \\ \frac{\partial \sigma^2}{\partial a_1} &= -2 \sum_{i=1}^n [y_i - (a_0 + a_1x_i + \cdots + a_kx_i^k)] x_i = 0 \\ \frac{\partial \sigma^2}{\partial a_k} &= -2 \sum_{i=1}^n [y_i - (a_0 + a_1x_i + \cdots + a_kx_i^k)] x_i^k = 0 \end{aligned} \quad (11)$$

which can be rewritten in matrix form as

$$\begin{bmatrix} n & \sum x & \cdots & \sum x^k \\ \sum x & \sum x^2 & \cdots & \sum x^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum x^k & \sum x^{k+1} & \cdots & \sum x^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum y \\ \sum xy \\ \vdots \\ \sum x^k y \end{bmatrix} \quad (12)$$

The left part of (12) is a Vandermonde matrix and it is accessible under this name in the application. A Vandermonde matrix can be calculated in $O(kn)$

⁵ available from <http://www.tc.umn.edu/~ringx004/mapm-main.html>, or type 'MAPM' into your favorite search engine. Special thanks to Michael C. Ring, who designed the library.

time while the number of multiplications per term x^y does not exceed $\log y$, minimizing the rounding error [PFTV92].

Actually, since all algorithms are slowed down by higher precision, the time complexities have to be calculated as a function not of k but of $k \log d$ with d the precision in bits or digits. This is neglected for the sake of readability.

The next step in the algorithm is to put the $k \times k$ Vandermonde matrix into echelon or lower triangular form where all values below the diagonal are zero. During the triangularization all operations on rows of the matrix are applied to the same rows of the solution vector to keep the them consistent. The result:

$$\begin{bmatrix} 1 & v_{0,1} & \dots & v_{0,k} \\ 0 & 1 & \dots & v_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_k \end{bmatrix} \quad (13)$$

with $v_{n,m}$ the values at row n , column m after triangularization and s_n the value of the solution vector after triangularization. Triangularization by Gaussian elimination can be done in $O(k^3)$ time. Once the matrix is in triangular form we store it to calculate any polynomial of $\leq k$ degree at any precision in $O(k^2)$. During the process of Gaussian elimination a vector is set aside by which the determinant of any Vandermonde matrix $\leq k$ can be obtained in $O(k)$, a time saving byproduct which is important for mixture MDL.

The parameters of the r -degree polynomial can be read recursively from the triangular matrix, starting with the highest parameter which is equal to s_r of the right hand solution vector. The recursive formula for parameter p_{r-s} with $s \leq r$ is

$$p_{r-s} = s_{r-s} - \sum_{i=1}^s [v_{r-s,r-i} \times p_{r-i}] \quad (14)$$

Rounding problems. Computation and triangularization of the Vandermonde matrix has to be done at a precision high enough to safeguard against obvious rounding errors but low enough to guarantee a fast performance. As a general rule, a precision of $2k$ digits is used for k -degree matrixes, which works well for every matrix of $k \leq 150$. This is way above the 70–100 degrees recommended for most operations because of slow performance beyond that point.

When we calculate a parameter vector of precision $d < 2k$ it is of great importance where in the algorithm we round off. The choices are:

Late rounding. The vector is calculated at the $2k$ digit precision of the matrix. Only the final result is rounded to d digits.

Early rounding. Each parameter is rounded instantly to d digits and then used to calculate the other parameters.

Before reading on the interested reader is suggested to answer for him or herself which method gives the better result.

I asked four experts in statistics and algorithms and all of them gave the wrong answer. Late rounding does *not* give the better result. I implemented both versions and found that early rounding reduces the number of digits needed to achieve the same performance by a factor of 2. The error of a parameter vector of d digits precision obtained with late rounding is roughly equal to that of a vector of $d/2$ digits obtained with early rounding.

More than anything else, this example should make it clear that the size of an actual implementation cannot be used to estimate the complexity of a model.

References

- [Lor63] Edward N. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–148, 1963.
- [Nan03] Volker Nannen. The Paradox of Overfitting. Master’s thesis, Rijksuniversiteit Groningen, the Netherlands, April 2003.
- [PFTV92] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. Vandermonde Matrices and Toeplitz Matrices. In *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, pages 82–89. Cambridge University Press, Cambridge, England, second edition, 1992.
- [Rin01] Michael C. Ring. MAPM, A Portable Arbitrary Precision Math Library in C. *C/C++ Users Journal*, November 2001.
- [Tho93] René Thom. *Structural Stability and Morphogenesis: An Outline of a General Theory of Models*. Addison-Wesley, Reading, MA, 1993.

Index

- analysis, *see* generalization analysis
- application, *see* Statistical Data Viewer
- arbitrary precision arithmetic, 27
- autoregression, 14
- axis, 12

- color, 12
- cross-validation, 5
 - how to use, 21

- echelon matrix, 28
- editor, 11, 25
- experiment
 - simple, 5

- file format, 25

- Gaussian elimination, 27
- generalization
 - analysis, 4
 - how to use, 21
 - complex analysis, 22

- implementation, 24

- learning problem, *see* process
- least squares, 27
- linear regression, 27
- logistic map, 14
- Lorenz attractor, 15

- magnification, 13
- manual, 11
- MDL
 - mixture MDL
 - how to use, 21
 - two-part MDL
 - how to use, 20
- menu, 25

- noise
 - system, 13

- open source, 24

- penalty, 21
- pendulum, 15
- plot, 6, 12, 25, 26
- polynomials, 27
 - object, 7
 - working with, 19
- precision, 28
- precision editor, 20
- process, 4
 - object, 5, 13
- project, 6, 25

- Qt, 26

- random generator, 13
- regular expression, 17
- rounding, 28

- sample
 - drawing, 18
 - from file, 17
 - from polynomial, 19
 - from process, 18
 - merge, 19
 - object, 7, 17
 - split, 19
- scale, 12
- seed, 13
- sinus wave, 5, 14
- Statistical Data Viewer, 3
 - implementation, 24
 - manual, 11
- step function, 16
- support, 18

- Thom map, 16
- triangular matrix, 28
- Trolltech, 26

- Vandermonde matrix, 27
 - object, 24

- XML, 25

- zoom, 12