The *winGamma* ™ User Guide

Abstract: This document is the user guide for the *winGamma* software and is updated to version 1.97. *winGamma* is a state of the art non-linear analysis and modelling tool.

Keywords: Smooth model, data analysis, prediction, Gamma test, Feature selection, Noise estimation.

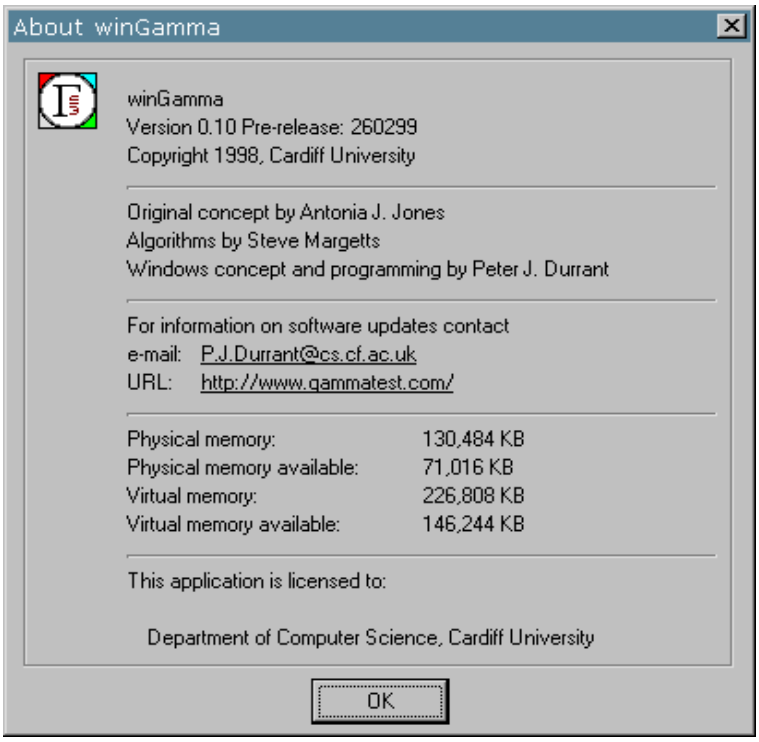
Communication regarding this document should be directed to:

Antonia J. Jones

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF WALES, CARDIFF
PO BOX 916
Cardiff CF24 3XF

Telephone: +44-292-087-4812
Telefax: +44-292-087-4598

Or by email to: Peter J. Durrant at p.j.durrant@cs.cf.ac.uk



NOTICE

This program is experimental and should be used with caution. All such use is at your own risk. To the extent permitted by applicable laws, all warranties, including any express or implied warranties of merchantability or fitness for a particular purpose, are hereby excluded. The authors and distributors of this software disclaim all liability for direct, indirect, consequential, or other damages in any way resulting from this software.

This program is protected by copyright. You may not copy this program or accompanying documentation without the express written permission of the copyright holder. You may not modify this program.

Acknowledgements

Thanks are due to many people who have contributed to the development of *winGamma*. In particular we should like to mention:

Aðalbjörn Stefánsson whose conviction that *MSError* could be determined without trial and error sparked the whole idea in 1995.

Antonia J. Jones and Nenad Končar, who slaved away to turn an idea into a useable prototype.

Alban Tsui who explored the outer reaches of the technique using incredibly complicated surfaces and a long double precision version of the Gamma test.

Steve Margetts who generated the original version of the amazingly tricky tree code in a matter of days and who put an unreasonable amount of effort into the algorithms that are the heart of *winGamma*.

Peter Durrant who put far more into the front end development than any of us had a right to expect and who gave the product the look and feel it now has. The collaboration between Steve and Pete has been critical.

McCann-Erickson (UK) Who generously funded a Research Studentship without which *winGamma* would not now be available.

Nick Fiddean the HoD of Computer Science at UWC whose unfailing support made the job possible.

Tina Thomas who put up with an invasion of computer scientists at the research farm over several years and cared for us all so well.

Howard James who saw the potential and used his business expertise to help turn *winGamma* into a product.

Nick Bourne of the RACD division at UWC who has steered us enthusiastically through the legalities that computer scientists blithely tend to ignore but which are essential if the product is ever to enjoy widespread use.

Dr. Nicola M. Pearsall who kindly gave permission for us to use the example data in *solar.csv*. Data for this example were provided by Newcastle Photovoltaics Applications Centre at the University of Northumbria at Newcastle, UK. These data were collected as part of a project with funding from the European Commission (THERMIE Programme) and the UK Department of Trade and Industry.

To all of you we express our thanks and hopes that *winGamma* will make a contribution worthy of your efforts.

The *winGamma* User Guide

CONTENTS

List of Figures

List of Tables

CHAPTER I Getting Started

1.1 Introduction

Data or *observations* can be considered as a spreadsheet of numbers in which the columns are divided into two types: *input columns* and *output columns*. In any row we might wish to determine the values of the outputs when these are not known but the corresponding inputs are known.

A *data model* data model is an algorithm constructed from a set of observations (for which all inputs and outputs are known) which enables us to predict the outputs from a given set of inputs. This software is concerned with constructing data models of a particular type.

1.1.1 The Purpose of the Software

winGamma is a software package which in the first instance estimates the least Mean Squared Error (*MSError*)[1] that any smooth data model (e.g. a trained feed forward neural network) can achieve on the given data without over-training.

winGamma can be used with multiple column *Input/Output* data files and single or multiple *Time Series*.

winGamma assumes that non-determinism in a smooth model from inputs to outputs is due to the presence of statistical noise on the outputs. Not all phenomena that one might seek to model fall into this category. For example, if the outcome that one is trying to predict from observations is *highly* probabilistic then the model produced by *winGamma* will not be satisfactory as a prediction tool.

- However, the software is able to detect this situation[2].

The models that *winGamma* is designed to produce are of phenomena (more exactly outputs) that are smoothly determined by the input variables. Mostly the limiting factor on the predictive accuracy of the model will be measurement noise or insufficient data.

For a given data set the *winGamma* software executes the *Gamma Test* which estimates the variance of the noise on each output. This will be an estimate of the best *MSError* that a smooth model can achieve for the corresponding output.

- Inputs and outputs should be continuous variables.

---

[1] See Appendix V for definitions.

[2] It will be reflected in a high *Gamma statistic* or a *Vratio* close to 1.

The estimate of that part of the variance of an output that cannot be accounted for by a smooth data model is called the *Gamma statistic*. As the number of data samples increases the *Gamma statistic* invariably[3] approaches an asymptotic value which is the variance of the noise on the particular output.

The goal of *model identification* for a particular output is to choose a selection of inputs that minimises the asymptotic value of the modulus[4] of the Gamma statistic. All things being equal this should result in a model which has minimal *MSError* when used to predict the output using input data not seen in the model construction process.

What happens if the final conclusion is that the noise variance on the output we are trying to predict is unsatisfactory? We can attempt one or all of the following:

- Increase the accuracy of measurements of both the inputs and the outputs. The *effective* noise variance on the output may be the result of measurement error on the inputs.

- Ask if we have included all the principal causative input variables liable to affect the output. If some obviously important factor has been missed then this may well explain why we are currently unsuccessful in predicting the output variable.

- For a time series prediction we could increase the rate of sampling or consider if there are other time series which may have predictive value for the time series we are interested in predicting (such time series are often called *leading indicators*).

One reason the Gamma test is so useful is that it can immediately tell us *directly from the data* whether or not we have sufficient data to form a smooth non-linear model and *how good that model is liable to be*. If the result is that the error of prediction is too high, no matter how much data we are given, then we must address the above issues.

For each choice of inputs investigated, as the number of data points increases we attempt to establish the asymptotic *Gamma statistic* for each output. We then choose the set of inputs for a particular output that has the *minimum* asymptotic Gamma statistic - this is known as *model identification*. Having established the best selection of inputs for each output, using the *winGamma* software, models may be built by:

- Static local linear regression (fixed model).
- Dynamic local linear regression (model updated as new data becomes available).

or by using one of four different types of neural network training algorithms:

---

[3] Convergence in probability.

[4] Because of sampling error if the variance of the noise level on an output is very small the *Gamma statistic* may sometimes be negative, even though a variance can never be negative. If this occurs we use the absolute value or modulus of the Gamma statistic.

- Two layer back propagation
- Meta-backpropagation (Not included in the Beta release)
- Conjugate gradient descent
- BFGS neural network

Predictions on new input data for which the outputs are unknown can also be made using one or more of the models.

1.1.2. The range of applicability.

The software is designed to analyse data with the goal of producing a near optimal smooth function from inputs to outputs using only the data provided. Both the inputs and outputs should be continuous real variables from some bounded range. The software will be much less effective if some of the input or output variables take only categorical values (e.g. 0 or 1). The underlying function is presumed smooth and this means bounded first and second derivatives. If the unknown function has regions of very high curvature it will be much harder to produce an accurate predictive model.

It is also assumed that the noise variance on each of the outputs is bounded and independent of the input values. If the independence condition is false this is not necessarily fatal, the Gamma test will return an average noise variance over the whole input space.

Subject to these conditions *winGamma* can be applied to a wide variety of non-linear modelling problems. It is particularly useful in the research and design of non-linear control systems.

1.2 Loading data files.

winGamma can analyse two basic types of numeric data files: *Input/Output data*, where each column corresponds to an input or an output, and *Time Series data* where each column corresponds to a particular time series and successive rows represents successive values in time for each series.

Note all data files must contain only numerical data arranged in one of the allowed formats. (For more details of data file formats see Appendix II.) To load a data file launch the application from the Start menu. Click on **'File**' and then '**Open Analysis Data Set**'.

1.2.1 Comma separated variable (*.csv) files from spreadsheets

If the file data is in the *.csv format (e.g as exported from *Excel*), on loading the file you will be asked to specify which of the columns are outputs. Because a *.csv file does not indicate which columns are inputs and which are outputs, if the file is an *Input/Output* file it is necessary to give this information to *winGamma*. Each column has to be tagged as an input or output column. This is done as indicated in Figure 1.1.To change an input (default) to an output select it with the mouse or cursor keys and press the `**Enter'** (or **'Return'**) key, or toggle with a double click on the left mouse button.

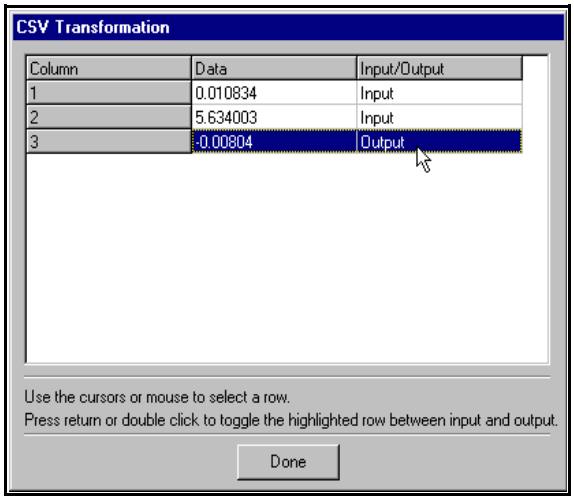


Figure 1-1 Toggle inputs to outputs as required when loading a *.csv file as Input/Output data.

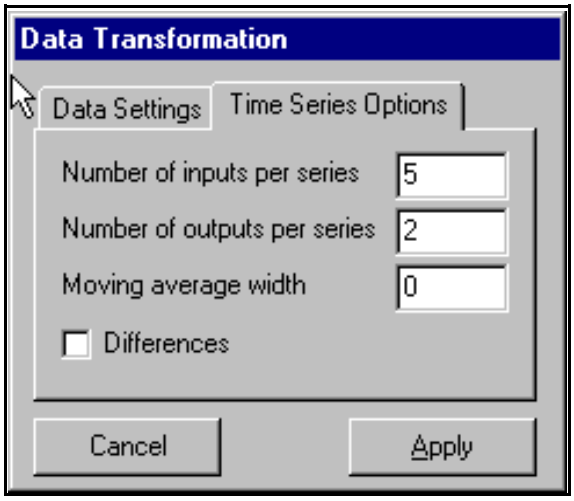


Figure 1-2 Selecting the number of inputs and outputs per time series.

For *Time Series* data *specify all columns as inputs*. As in Figure 1.2 *winGamma* will then ask you to specify the number of inputs and outputs per series. At present these are the same for all series. In Figure 1.2 we are choosing to use 5 previous values of every time series to predict the next 2 values for each of the time series. Choosing more outputs will produce predictions further into the future. The nature of things is such that the further we try to predict into the future the less accurate these predictions will be. This is reflected in a higher Gamma statistic for more distant future predictions.

1.2.2 *Input/Output* data in standard format (*.asc) files.

Standard format for an *Input/Output* file is DOS ASCII in the following form. In each row the inputs are separated by spaces and the list of inputs terminated by a comma. The list of outputs then follows, each separated by spaces. The end of a row is signified by CR/LF. File data in standard *Input/Output* form will be automatically recognised as such. At present the numbers in the file must be in simple decimal format.

The first decision to be made after specifying the file name is whether or not to '**Transform**' (i.e. to scale or normalise) the data. To normalise check the appropriate box as indicated in Figure 1.3

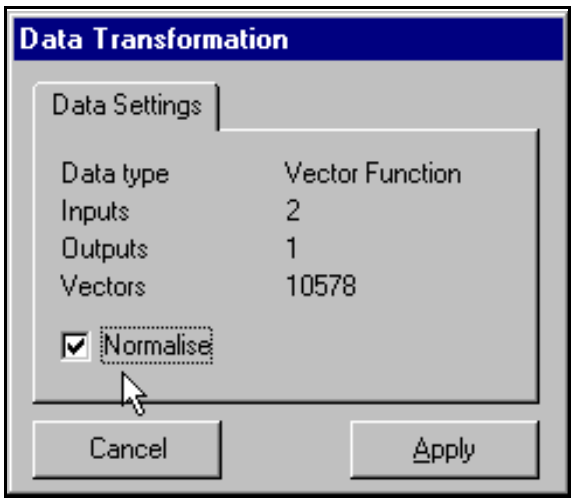


Figure 1-3 The **`Normalise'** check box.

For a full discussion of the effects of scaling and whether or not to scale in any particular case see section 2.14. In an initial investigation it is usually a good idea to scale *Input/Output* or multiple *Time Series* data.

1.2.3 *Time Series* data in standard format (*.asc) files.

Standard format for a *Time Series* file is DOS ASCII in the following form. Each column represents an individual time series. The rows represent values for each of the time series, successive rows being successive values in time. Within a row each numeric value is separated by spaces. The end of a row is signified by CR/LF.

1.2.4 Partitioning the data.

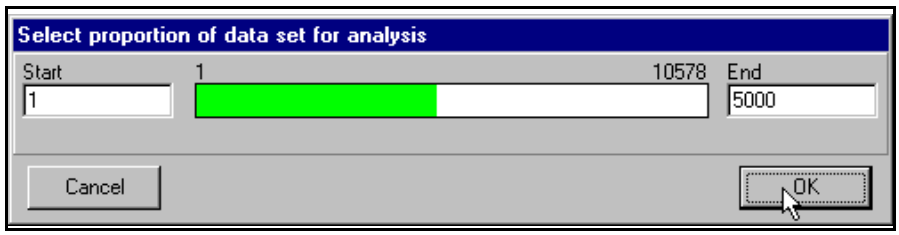


Figure 1-4 Selecting a proportion of the data for initial analysis.

It is sometimes convenient to perform the initial analysis on a subset of the whole data file. This could happen for example where the data set was very large. Therefore *winGamma* will next ask the user to select the proportion of the data which should currently be used for analysis, see Figure 1.4. We can later separate training and test data.

1.3 A first experiment.

Load the 2-input/1-output data file *solar.csv* and select column 3 as output. Initially do not normalise. Select all the data for analysis, there are 10578 data points in the file. After the data has been successfully loaded *winGamma* displays the main screens, as in Figure 1.5.

The **Experiments** window in the **Analysis Manager** shows the different kinds of data analysis that can be performed. We shall discuss the meaning of these experiments and the interpretation of their results fully in Chapter II. For the present we shall simply illustrate the basic Gamma test experiment.
The **Data Set Manager** shows the data that has been loaded as in Figure 1.5, where the windows have been tiled.. Because data files may be very large the data rows are divided into `pages' of 100 rows each. In Figure 1.5 the first page has been selected. Each column represents a column of inputs or outputs and is labelled as such. The first four rows give the *Mean*, *Standard Deviation*, M*inimum* and *Maximum* of each column for the whole of the data selected for analysis. The name of the current data file is also displayed at the top of this window.

Handy Tip. Note that most of the windows and sub-windows including the column separators in the **Data Set Manager** data display can be resized using click and drag.

To perform a Gamma test select the **Analysis Manager** and then **Experiments**. Highlight *Gamma test* and select **'New'**. We can now toggle between the Experiment tab and the Mask tab. The only option to be set from the Experiment tab in this experiment is the number of near neighbours. For

the present leave this set to 10. The Mask tab is used to select which inputs to include. Leave this set to '11' (i.e. both inputs are included).

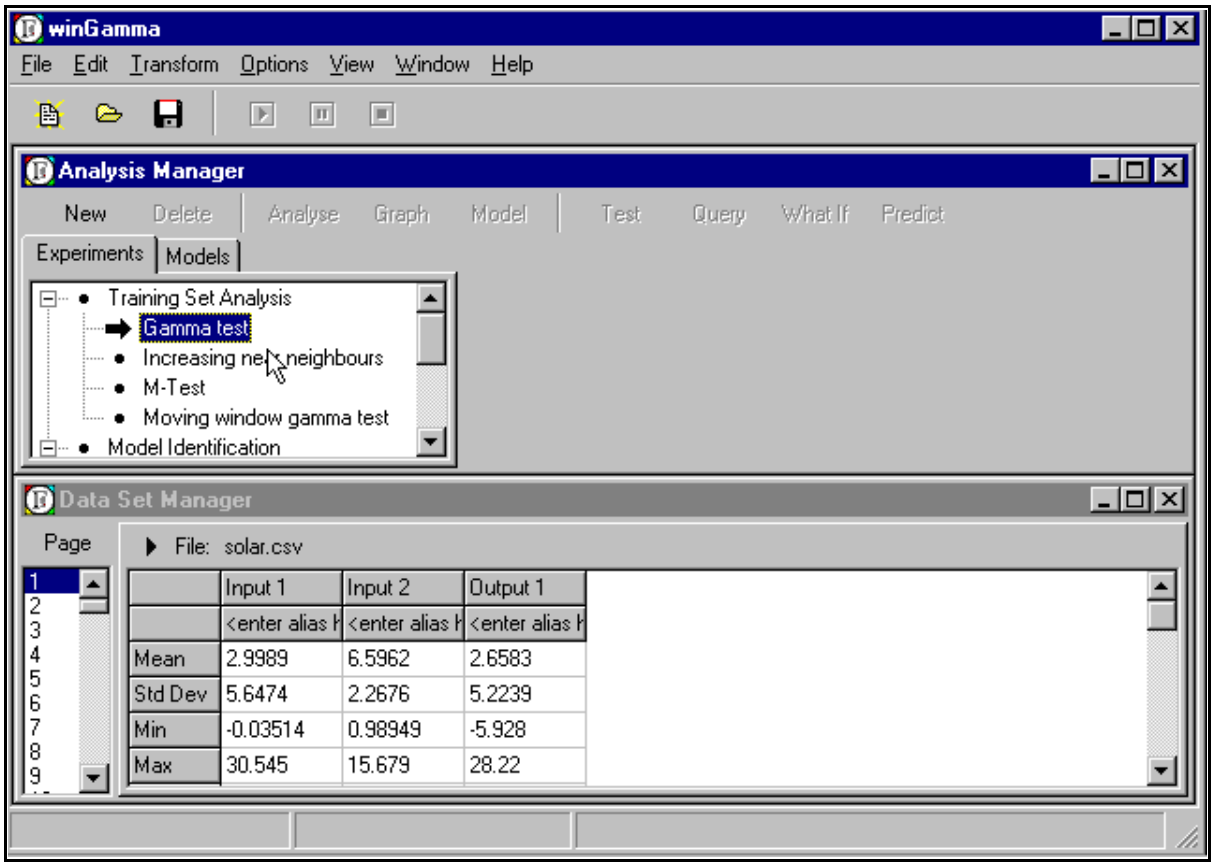


Figure 1-5 The **Analysis** and **Data Set Managers** after loading a data file.

When these steps are complete click on **'Execute'**. Under the **Analysis Manager** the **Settings** window will now show the settings for the current experiment. This is shortly augmented by a **Results** window which shows the results of this experiment. We can switch between the **Settings** and **Results** windows using the appropriate tabs. These results for the single output are presented in a Results/Settings window along a single row (because there is only one output) and are shown here in the first column of Table 1-1. If there is more than one output the software generates a similar set of results for each output.

Finally '**Transform'** the data and repeat the experiment to obtain the scaled results in the last column of Table 1-1.

1.3.1 Interpreting the results.

To interpret these results it helps to have some idea of how the *Gamma statistic* is calculated. We shall describe this more fully in Chapter II, but for now it is enough to know that the *Gamma statistic* is calculated by determining a regression line based on near neighbour statistics derived from the data - see Figure 1.6.

Gamma

The first row of Table 1-1 gives the *Gamma statistic* (pmax = 10) for the output as evaluated over the data selected for analysis (in this case the whole data set). As one can see from Figure 1.6 the *Gamma statistic* is actually the vertical intercept of the regression line in the figure. This is the estimated variance of the errors for any smooth model built on the data. Since the output variable range is approximately [0, 30] this is a relatively small error variance. It means any smooth model built on this data will have a standard deviation of the prediction error of about $\sqrt{0.020761} = 0.144$ on the unscaled data - which is about 0.5% of the range.

Table 1-1 Gamma test results with pmax = 10 for unscaled and scaled *solar.csv* data.

	Unscaled	Scaled
Gamma	0.020761	0.000196
Gradient	0.244242	0.256910
Standard Error	0.002360	2.328010E-5
Vratio	0.000760	0.000785
Near Neighbours	10	10
Start	1	1
Unique Points	10578	10578
Evaluated Output	1	1
Zero Near Neighbours	0	0
Upper 95% Confidence	0	0
Lower 95% Confidence	0	0
Mask (remaining entries)	11	11

In general it is helpful to distinguish two cases:

- First, where the true noise variance is zero. In this case the *asymptotic Gamma statistic* should approach zero and there is no limit to how good a model we can build provided only that we have more and more data of arbitrarily high precision. For example, this can happen with artificially generated data for chaotic time series.

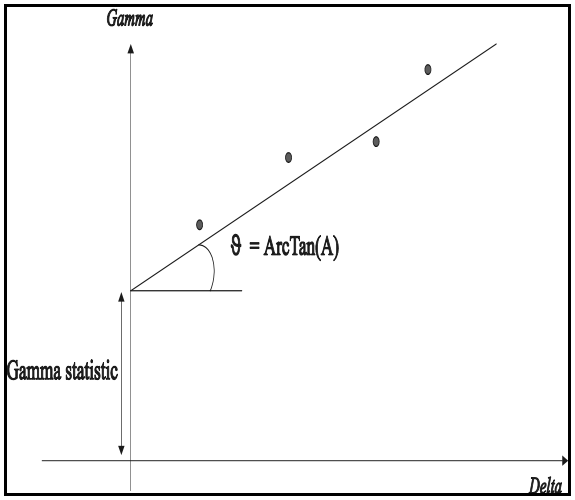


Figure 1-6 The *Gamma statistic* and the *Gradient/Slope*.

- Second, and more realistically, where the true noise variance is positive. In this case the *asymptotic Gamma statistic* should also be positive and there will come a point where using more data to build our model will not actually improve the quality of the predictions when compared with the measured values of the output.

In the case of a *positive asymptotic Gamma statistic* we can determine the minimum amount of training data required to build a smooth model with this *MSError* using the *M*-test described in section 2.5.

Gradient

The *Slope* or *Gradient* is the slope of the regression line in Figure 1.6 used to calculate the *Gamma statistic*. It is actually a rough measure of the complexity of the smooth function we are seeking to construct. In this case the gradient of $A = 0.244$ indicates that the output is a rather simple function of the two inputs. It is generally best to look at the *Gradient* for the scaled data since this refers to a standardized output range.

Like the *Gamma statistic* the *Gradient* will eventually asymptote to a fixed value. However, the number of data samples required to get a stable asymptote for the *Gradient* will usually be much larger than the number required to get a stable asymptote for the *Gamma statistic*.

Standard Error (*SE*)

This is the usual goodness of fit applied to the regression line in Figure 1.6. If this number is close to zero we have more confidence in the value of the *Gamma statistic* as an estimate for the noise variance on the given output. In this case an $SE = 0.00236$ represents a good fit for the regression line.

Vratio

The *Vratio* is defined as *Gamma/Var(output)*. It thus represents a standardised measure of the *Gamma statistic* and enables a judgement to be formed, independently of the output range, as to how well the output can be modelled by a smooth function. In comparing different outputs, or outputs from different data sets, the *Vratio* is a good number to study because it is independent of the output range. A *Vratio* close to zero indicates a high degree of predictability (by a smooth model) of the particular output. If the *Vratio* is close to one the output is equivalent to random noise as far as a smooth model is concerned. In this case $Vratio = 0.00076$ indicates low noise data which we should be able to model quite accurately.

Near Neighbours (number of pmax)

This is the one user settable parameter in the Gamma test. When estimating the *Gamma statistic* pmax should be selected in relation to the size of the data set. For large data sets, in the interests of getting a more accurate *Gamma statistic*, we can afford to take the number of near neighbours somewhat larger (this depends on a number of factors discussed in Chapter II). In general in a Gamma test experiment we should keep the number of near neighbours less than 30. Usually 10-20 is a good choice.

Start

This indicates the row identifier for the first vector selected.

Unique Points

In some data sets the same input vector may occur several or many times. This indicates how many distinct input vectors are present in the data (see the discussion on zeroth near neighbours below).

Evaluated Output

Indicates to which output the results relate. In a file with multiple outputs all these results are calculated for each output.

Zero(th) Near Neighbours

In some data sets the same input vector may occur several or many times. If an input vector appears multiple times then, if it has the same output value(s), it might be construed as a *repetition* or it may be a *separate independent observation*. In the first case there is no extra information and the data vector should be deleted. In the second case there is useful information in the two vectors because they are telling us that for these inputs the outputs are identical, and so presumably subject to low or zero noise variance. If one or more outputs are *different* for the *same input* vector then again there is useful information, because enough vectors of this type could give us an immediate grip on the noise variance.

Therefore because it is important for an analyst to know if the same input vector occurs multiple times *winGamma* provides this information by stating the maximum number of non-unique input vectors. If this number is small in relation to the size of the data set it can safely be ignored on a first pass. If it is large then the data should be subjected to some analysis outside of *winGamma*.

Upper 95% Confidence/Lower 95% Confidence

In the case where zeroth near neighbours are present these results are the lower and upper bounds at the 95% confidence levels for the *Gamma statistic* estimated directly from the zeroth near neighbours. Unless the data file has many repeated input rows these values can be ignored. If the file has many repeated inputs then these values can be compared with the normal *Gamma statistic* (which is computed in an entirely different way).

1.4 The basic controls of *winGamma*.

The use of these options will discussed fully in Chapter II.

The Analysis Manager.

> **Experiments** These are options used to determine the Gamma statistic and to investigate how reliable this statistic is, i.e. to determine. the quality of a model which might be built using the data and a given selection of inputs. To invoke any of these options after loading a data set simply select the **Analysis Manager** and highlight the option required. Then click on **`New'**. For any particular option there are probably other parameters which require to be set before invoking **`Execute'**.
>
> *Gamma test*: Finds the *Gamma statistic* and other relevant measures.

Increasing near neighbours: Finds how the *Gamma statistic* varies with the number of near neighbours used to compute it.

M-test: Shows how the *Gamma statistic* estimate varies as more data is used to compute it. This test can also tell us how much data we are likely to need to obtain a model of a given quality.

Moving Window Gamma test: Shows how the estimate for the *Gamma statistic* using a fixed number of data points varies as we move a fixed length window along the data file. This is used to check the stability of the Gamma statistic as we move along a large file.

Model Identification. These options are used to select those inputs which can best be used to predict a selected output (some inputs may be noisy or irrelevant). The use of model identification techniques is discussed in Chapter II.

Full Embedding
Genetic Algorithm
Hill Climbing
Sequential Embedding
Increasing Embedding

Other features. Are captioned in Figure 1.7.

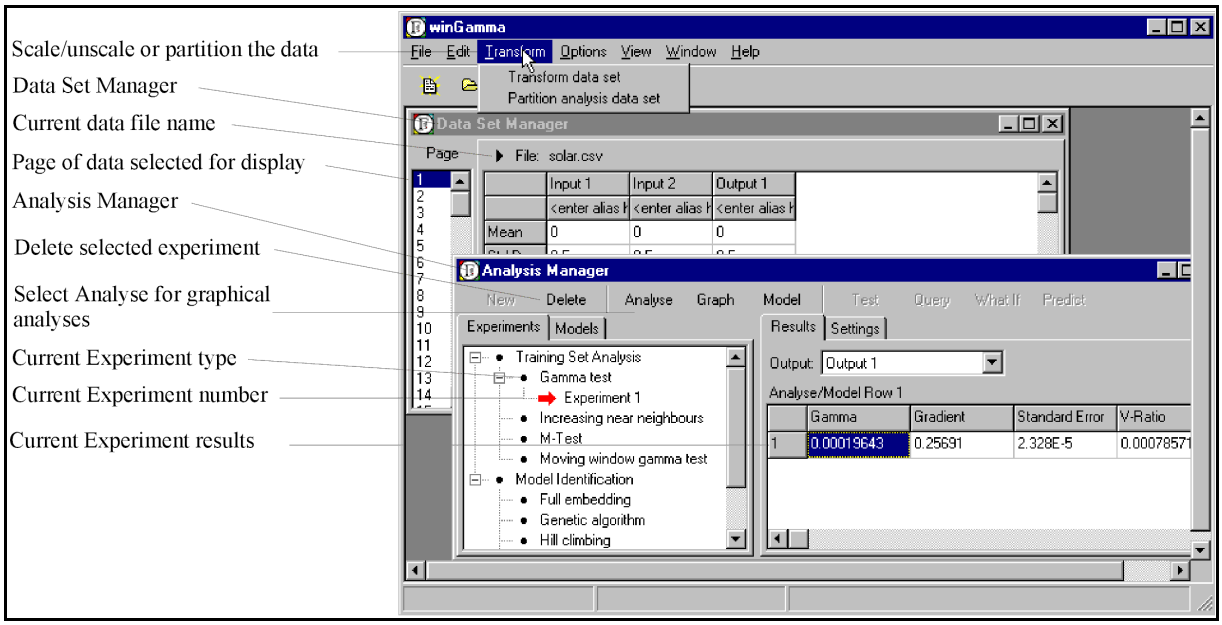


Figure 1-7 The **Analysis** and **Data Set Manager** windows after performing the initial experiment.

1.5 Two simple examples

In this section we further illustrate the use of *winGamma* using two test files provided with the software.

1.5.1 An *Input/Output* file.

The data for the file *Sin500.asc* was created (via the *Mathematica file DataGen.nb*) using the function y = Sin(x) and then adding uniformly distributed noise with a theoretical variance of 0.075 to the y values. A point plot of the data is shown in Figure 1.8.

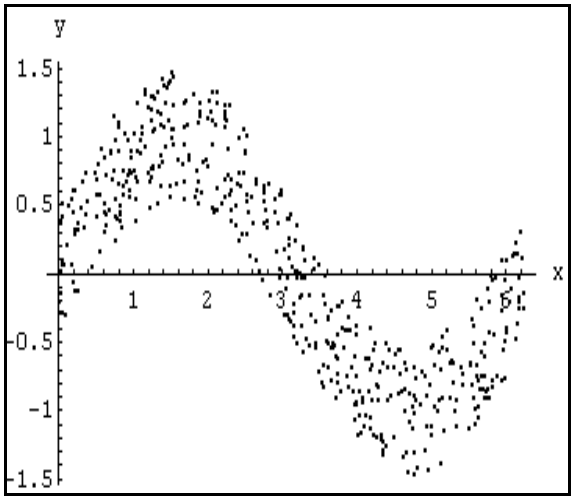


Figure 1-8 The noisy sine data.

1.5.1.1 The basic steps

Load the data file and run a simple Gamma test with the number of near neighbours set at 10 (default), as described in section 1.3. Do not scale the data. Note that we do not need to specify the number of inputs and outputs because this file is in standard format.

The *Gamma statistic* in the **Results** window is 0.07355 which is quite close to the theoretical noise variance. The *Vratio* of 0.12762 suggests that we will not be able to predict the value of an output very accurately, which in view of the data plot in Figure 1.8 is not too surprising. The *SE* is 0.0037651 which indicates a fair degree of reliability in this assessment.

Now click on **Analyse**. This enables us to see three analytical graphical displays which are described more fully in Chapter II. The first of these displays is the *Gamma scatter plot* and regression line of Figure 1.6. The other two tabs give a 3*D Histogram* and an *Angle histogram*. These are different ways of viewing the data in the scatter plot.

How stable is the Gamma statistic (with 10 near neighbours) as the number of data points varies? We can answer this question by clicking on the *Experiments* tab and then highlighting ***M*-test**. This will run the Gamma test for an increasing number (M) of data points. Now click on **`New'** to begin setting up the *M*-test. leave the number of near neighbours set to 10 and click on the *M-test* tab. Set the initial sample size to 10, the final sample size to 500, and the steps size to 10. Now click on **Execute** to begin the Experiment. After the Results window comes up click on **`Graph'** to obtain a graph of the *Gamma statistic* values against the

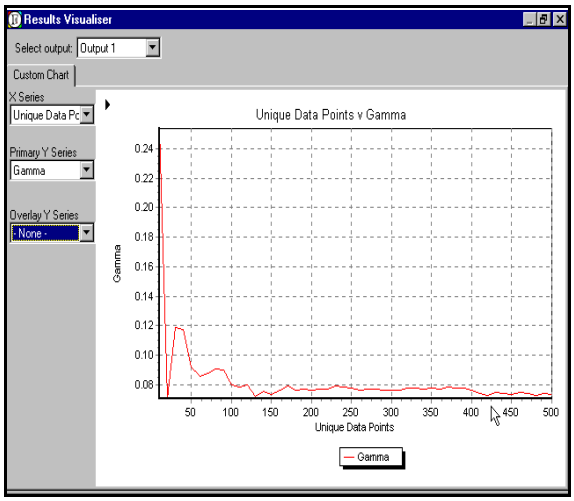


Figure 1-9 An *M*-test on the noisy sine data.

number of data points. This is shown in Figure 1.9 and we can see that after around 425 points the graph is fairly stable. The fact that the data is rather noisy means we should try to optimise the number of near neighbours for the Gamma test if we wish to obtain a more accurate Gamma statistic and we shall see how to do this in Chapter II.

1.5.2 A chaotic *Time Series*.

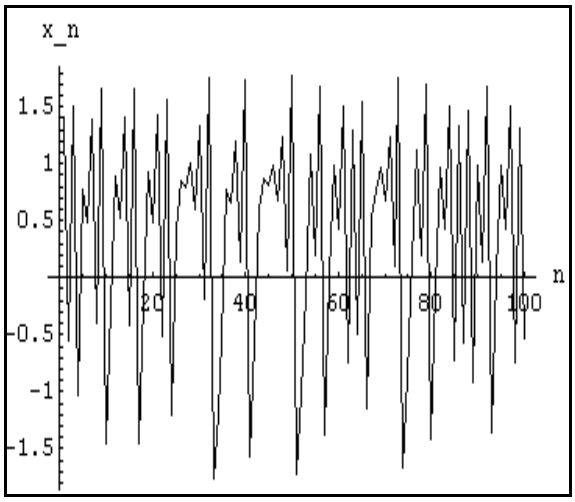


Figure 1-10 The first 100 points of the *Hen500.asc* time series.

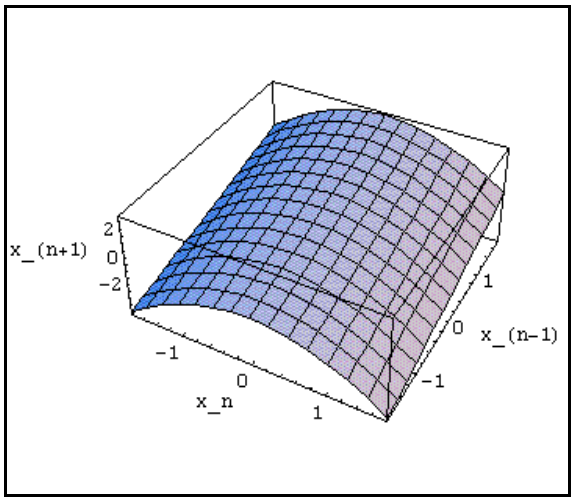


Figure 1-11 The surface which defines x_{n+1} in the Henon map as a function of the two previous values.

Here we use the file *Hen500.asc*. This file contains time series data generated by iterating the Henon map. It is described in more detail in **The Gamma test and how to use it: a practitioners guide**.

To get some idea of what the time series data looks like we graph the first 100 points of the time series using any convenient software as in Figure 1.10. Although this time series looks quite unpredictable, nevertheless the underlying model which takes us from two successive values to the next is a smooth function of the two successive inputs and therefore does not violate the requirement of the Gamma Test, see Figure 1.11.

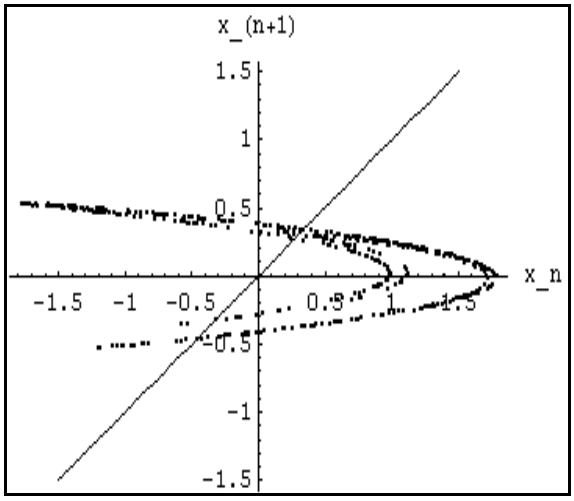


Figure 1-12 The distribution of points in the input space for the Henon map.

A very important factor to consider when building a non-linear model is the distribution of sample points in the input space. In some cases these points will be uniformly distributed but in many cases this will not be the situation. If we plot the distribution of the points (x_{n-1}, x_n) for the Henon map data from the file we obtain Figure 1.12 Peculiar distributions of data like this can be very helpful in high

dimensional input spaces, as it often means that we need less data to build a good model than would be the case if the data were uniformly distributed over the whole space.

It is precisely the surface of Figure 1.11 which is the model that we can seek to construct using *winGamma*. We could take the time series and create a *2-Input/1-Output* data structure (x_{n-1}, x_n) -> x_{n+1}. In fact any time series that evolves according to some smooth iterative or dynamic process can be treated this way, provided only that we can determine the number of previous values of the time series required to predict the next value (this is called the *embedding dimension*)[5]. In this example we shall pretend that we do not know the embedding dimension and show how *winGamma* can be used to get some idea of which previous inputs are likely to produce a good model.

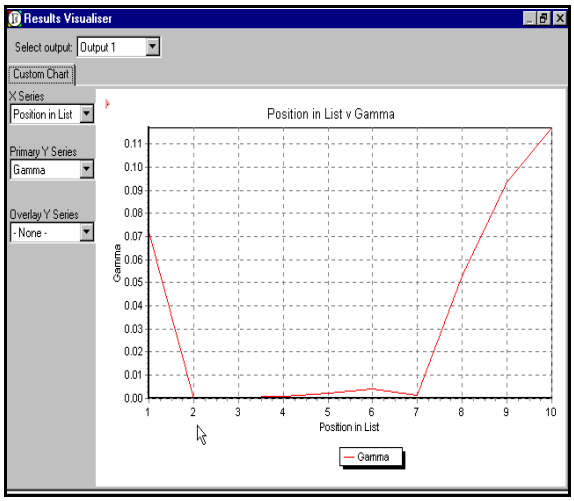


Figure 1-13 The result of an **Increasing Embedding** on *Hen500.asc* with a maximum of 10 inputs (using 10 nearest neighbours).

Note that the data in the file *Hen500.asc* is high precision and not subject to noise.

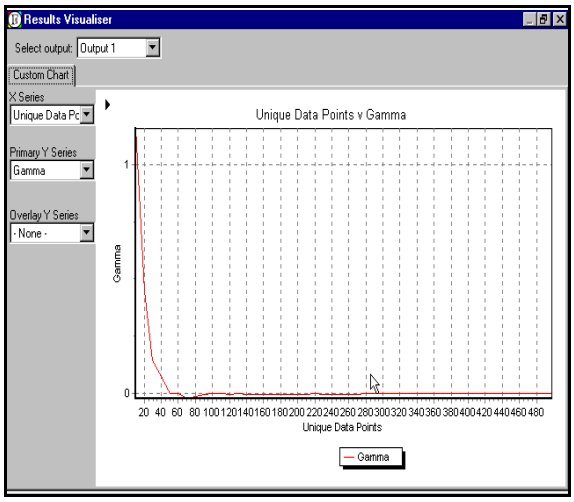


Figure 1-14 The result of an *M*-test on *Hen500.asc* with 2 inputs (using 10 nearest neighbours).

1.5.2.1 The basic steps

1. Load *Hen500.asc* with **'Open Analysis Data Set'**.

2. Set the number of inputs to 10 in the Time Series tab.

3. Do not enable **'Normalisation'** in the check box. Since the data is a single time series and each sample is comparable we should not expect much gain from scaling.

4. When prompted to select a proportion of the data set for analysis use all the data (1-490) for the initial analysis and click 'OK'

5. Select **`Gamma test'** from the **Experiments Manager** and then click on **`New'**

[5] The fact that this is so is by no means obvious. It is a consequence of a fairly deep theorem due originally to Takens (1981).

Initial results: The initial result gives a *Gamma statistic* of 0.117143 and a *Vratio* of 0.185337 which is not very encouraging. However, the real reason for this is that most of the inputs we have selected for the model are irrelevant or not very helpful.

6. Next in the **Experiments Manager** under *Model Identification* highlight **Increasing Embedding**. and then click on **`New'**. Leave the number of near neighbours set at 10 and click on **`Execute'**. What this experiment does is to compute the *Gamma statistic* for a succession of models based on 1-Input (the previous value), 2-Inputs (the previous two values), etc. up to the maximum number of inputs we have set (which is 10), where in each case the output is the current value.

Results. This gives us a succession of Gamma values which we can graph by clicking on **`Graph'**. The result is shown in Figure 1.13. Here it becomes clear that the best of these models (i.e the one having a *Gamma statistic* closest to zero) is the one which uses just the two previous values. The *Gamma statistic* for this model is approximately -0.000161 which is very close to zero. The *Vratio* is -0.0001648 which again is close to zero.

7. Now that we have identified the relevant inputs pull down the **`Transform'** menu and click on **`Transform the data set'**. Under the *Time Series Options* select 2 inputs and 1 output and then leave the proportion of data set for analysis set to 1-498.

8. Next in the **Analysis Manager** under *Training Set Analysis* select *M*-test and then click on **`New'**. In the **Experiment Editor** click on the *M*-test tab and set the Initial sample size to 10, leave the final sample size set to 498, and set the step size to 10. Now click on **`Execute'**. We should like to see how stable the Gamma statistic is and how much data we are likely to need to get a good quality model. Finally when the results window comes up click on **`Graph'** to see the result of the experiment. This is shown in Figure 1.14

Results. We see from the graph that we could probably build a pretty good model using only around 100 points. However, if we want to be sure then we should choose around 280 points because from this point onwards the variations in the *M*-test graph are very small. 280 points gives a *Gamma statistic* of -0.001054 and a *Vratio* of -0.001017.

8. Highlight the result in question in row 28 and click **'Analyse'**. The scatter plot and regression line is shown in Figure 1.15.

Handy Tip. By left clicking and dragging the mouse down and to the right we can zoom in on any selected part of these graphs as shown in Figure 1.15. We can also move the contents up/down and left/right by right clicking and dragging. To restore the original view simply left click and drag the mouse up and to the right.

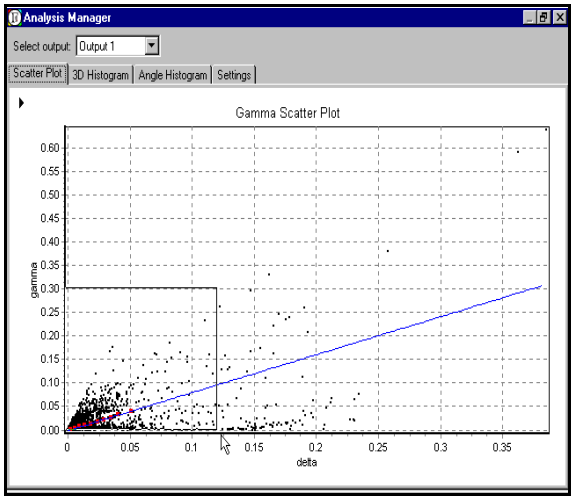


Figure 1-15 The scatter plot for 280 test points on *Hen500.asc* with 2 inputs (using 10 nearest neighbours).

It is interesting to see the remarkable difference between this scatter plot and corresponding scatter plot for the noisy sin data we used in section 1.5. Here in Figure 1.15 we cannot fail to observe the almost empty wedge in the top left hand corner of the plot. We shall see in Chapter II that such a feature in the scatter plot is strongly indicative of a noise free smoothly determined process. This observation is reinforced by the very small *Vratio*.

9. Finally examine and compare the other graphs produced by the **Analyse** tabs with those produced for the noisy Sin data in section 1.5.1.1. We shall examine these tools more fully in Chapter II.

1.6 Linear models

winGamma is a *non-linear* modelling tool and makes very few assumptions about the nature of the model. Because of this fact it generally needs far more data than parametric analysis where the model is presumed to have a particular form. If it is safe to assume that the model is linear then a simple linear regression model should be built and tested using some other standard software (e.g. *Mathematica* has very good linear regression facilities).

If you know nothing at all about the data being analysed it is always a good idea to check the linear regression model before using *winGamma*.

If the data is fundamentally linear then *winGamma* will perform quite well using local linear regression. However, *winGamma* will make less efficient use of the data available than global linear regression.

1.7 Exporting results for use by other software

Data produced by *winGamma* is either Graphics or data such as predictions.

Data Files can be exported in:

1. *Mathematica* compatible format e.g. {}s are embedded to format lists and arrays.
2. *Excel* and spreadsheet compatible comma separated variable (*.csv) format.

These Export functions are available as an option under 'Edit' in the main *winGamma* parent window, a right click on the mouse button in the appropriate context, or by clicking the '➤' tab in the top left corner of many of the graphics windows.

1.8 Customising the file and project directories

To customise the locations of data files and project files (discussed in Chapter II) pull down the **`Options'** menu and click on **`Customize'** You can modify the number of data files and project files kept in the history (in is usually best to set these to their maximum of 9). Now under data files click on **`Modify'** and select the directory that should first appear when the process of loading a data file is initiated. When the desired location has been selected click on `OK'. Go though a similar procedure to locate the project directory. If you wish the windows settings to be saved each time *winGamma* is closed down then check the appropriate box. Finally click on **`Apply'** and exit the program.

CHAPTER II Performing an analysis

2.1 Introduction

An **Experiment** is a particular type of calculation performed on the analysis data. A new experiment is started by highlighting the type of experiment required and then selecting **`New'** in the A**nalysis Manager** window. If we want to perform the same calculation but with different parameters (e.g. the number of nearest neighbours) or a new method (e.g. *M*-test) then a new experiment is started.

In this chapter we discuss each type of Experiment, how to set the parameters and how to interpret the results. Each Experiment is discussed using an example and illustrated with screen shots.

2.1.1 The user cycle

The user cycle for a full analysis is not completely fixed and can be varied according to circumstances. However the general approach can be summarised by the following steps.

Input/Output files.

1. Load the data and on the full data set (if not exceedingly large) do a simple *Gamma test* scaled and unscaled with the number of nearest neighbours set to the default of 10. If the data set is very large use a subset of the data for initial experiments.

2. Run an *Increasing Near Neighbours tes*t and use the minimum *SE* between (say) pmax = 5 and pmax = 50 to determine the most accurate Gamma statistic.

3. Using the value for pmax determined in Step 2 run an *M-test* to determine how stable the Gamma statistic is with increasing data set size.

4. If the *M*-test produces a stable asymptote decide if the noise variance is likely to be:

- Zero (arbitrarily good models possible with enough high precision data).

or

- Non-zero (not much point is using more data than necessary to give a model which predicts at the Gamma statistic level).

On this basis decide how much data is likely to be needed to build a model.

5. Can we get a better *Gamma statistic* by discarding some of the input? To answer this question run a *Full Embedding* if the number of inputs is small enough to allow this (say ≤ 10 -15). Otherwise try the heuristic search techniques, such as Hill-climbing or Sequential embedding (see 2.7.2 - 2.7.4), ending up with a long *GA* run.

6. If a better embedding is found then repeat steps 2, 3 and 4 to refine those conclusions.

Time series files.

Time series analysis is complicated by the fact tat we probably do not know how far back in time we should look to build our prediction model. This initial decision is not irrevokable and should be guided by some degree of commonsense analysis on what is likely to be the case for the given data set and how much data is available.

E.g. For a single time series with annual periodicity, where the samples are weekly, we might set the number of inputs to 104 - the equivalent of two years. However, 104-dimensional space has 'a lot of room up there' and we should need a data set going back many years to make this worthwhile. If only a few years data is available then perhaps we should first consider a model over the last several months or weeks.

1. Load the data and do not initially normalise if it is a single time series. Set the number of inputs to a reasonable maximum in the light of the data and the number of outputs initially to one. Now perform a simple *Gamma test* on the full data set (if not exceedingly large) with the default number of near neighbours set to 10, to get an initial idea. If the data set is very large use a subset of the data for initial experiments.

2. Run an *Increasing Embedding test* to determine a likely embedding dimension.

3. **Transform** the data set to reset the maximum number of inputs to the largest number from the Increasing Embedding Experiment which still gives a comparatively small *Gamma statistic*.

4. Run a *M-test* to check the stability of the Gamma statistic If the *M*-test produces a stable asymptote decide if the noise variance is likely to be:

- Zero (arbitrarily good models possible with enough high precision data).

or

- Non-zero (not much point is using more data than necessary to give a model which predicts at the Gamma statistic level).

On this basis decide how much data is likely to be needed to build a model.

5. Can we get a better Gamma statistic by discarding some of the input? To answer this question run a *Full Embedding* if the number of inputs is small enough to allow this (say <= 10 -15). Otherwise try the heuristic search techniques ending up with a long *GA* run.

6. If a better embedding is found then repeat steps 4, 5 and 6 to refine those conclusions.

7. Refine the number of near neighbours for the final estimate of the Gamma statistic using an *Increasing Near neighbours test*.

2.2 The Gamma test

This finds the *Gamma statistic* and other relevant measures. These are principally the *Gradient*, the *Vratio* and the *Standard Erro*r as described in Chapter I.

Once the inputs have been determined, either with preliminary Gamma tests or because these are set by the structure of the data, as in multiple *Input/Output* data sets, the only parameter to optimise is the number of near neighbours (often denoted by pmax). It is a remarkable fact that for many data sets the default of pmax = 10 nearest neighbours is often nearly optimal.

A suitable size for pmax in the Gamma test principally depends on two factors. The *number of data samples M*: if *M* is large the local number of data points close to a given point can be expected to be high The *local curvature of the surface* described by the unknown function *f*: other things being equal, for a surface with high curvature we cannot afford to take neighbours too far away, so that pmax will require to be smaller.

Systematic ways to determine the best choice for the number of near neighbours are described later in section 2.4.

Note that the size of pmax in *modelling* the unknown function *f* using local linear regression is determined by other factors described in section 3.2. Whilst for the Gamma Test it is usually the case that we want to take pmax *small*, for local linear regression at *high* noise levels we will need to take pmax *much larger*.

2.3 The Gamma Test analysis graphs.

After performing an experiment highlight the row containing the Gamma result to be scrutinised and click **'Analyse'**.

Clicking on the tabs will provide the other plots that are discussed below. In an experiment where there are multiple Gamma results the graphs and plots will relate to the highlighted Gamma result.

- Therefore it is important to highlight the required result in the **Results** window.

2.3.1 The scatter plot and regression line.

The critical graph to look at first is the scatter plots and (δ(p), γ(p)) regression line, see Figure 2.1. The scatter plot shows point pairs (δ, γ), where δ is the squared distance of an input (**x**) from one of its near neighbours and γ is one half of the squared distance between the two corresponding scalar output (y) values. The points to which the regression line is fitted are calculated by finding the mean $\delta(p)$ of δ and $\gamma(p)$ of γ, where p refers to the first nearest-neighbour (p = 1), the second nearest neighbour (p =2) and so on up to the maximum number of near neighbours (pmax) which has been set by the user.

A good regression line with points ($\delta(p)$, $\gamma(p)$) approaching (δ, γ) = (0, 0) indicates that the scalar output values of input-near-neighbours are close. If the regression line has a steep slope this indicates that the modelling function *f* that we seek to approximate is liable to be quite difficult to construct and

a large number of data points *M* will be required. If the line is almost horizontal the function is quite simple.

A particular feature to look for here is an empty 'wedge' in the top left corner of the scatter plot. If there are points in the top left corner it means that there are input points in the original data set which have |**x**(i) - **x**(j)| small (i.e. **x**(i) and **x**(j) are close together) but their corresponding output values y have |y(i) -- y(j)| large (i.e. y(i) and y(j) are far apart). This is very bad from the viewpoint of constructing a smooth model. It may be a reflection of a high intrinsic noise level on y (a high gamma) or it may just be that there is no smooth underlying model.

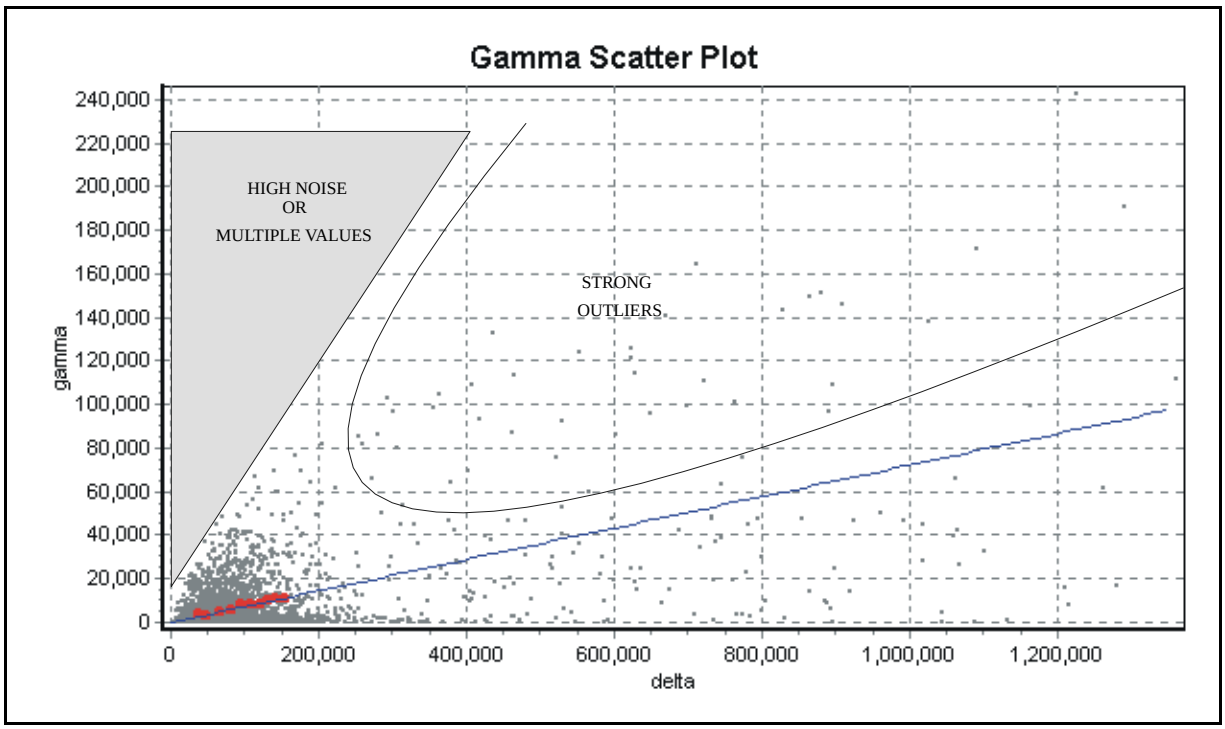


Figure 2-1 Main Features of the scatter plot and regression line

An example where the underlying model is not intrinsically smooth might be a logic function of the input variables, e.g XOR or *m*-bit parity. In *m*-bit parity the inputs are the vertices of the *m*-dimensional unit hypercube and the outputs are 1 or 0. In fact one can put a smooth surface through these points but this is a rather meaningless exercise. Problems with a large number of discrete input or output variables are best tackled via a *decision tree* approach rather than trying to use smooth modelling techniques.

The scatter plot can also give important clues on the nature of the data. For example it can happen in some control applications that the system being modelled goes through two or more different dynamical regimes. In one instance the scatter plot revealed that there were really two different regression lines each corresponding to a different dynamical regime. Moreover, each regime corresponded to a distinct part of the input space. By spitting up the input space and building a different model for each part a vast improvement on modelling capability was obtained.

It is interesting to note that by taking the number of near neighbours pmax much larger than is necessary (or desirable) for the Gamma test, the scatter plot can also reveal *periodicities* on different scales present in the data (although for large pmax the resulting *Gamma statistic* estimate will be essentially meaningless). Consider for example the data provided in *ModSin5000.asc*. This is a 1-Input/1-Output file derived from sampling the graph in Figure 2.2.

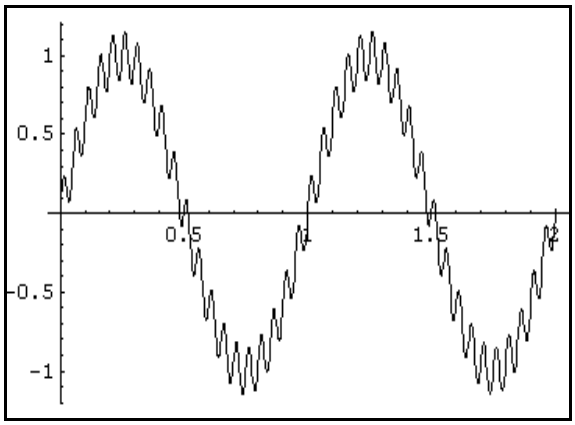


Figure 2-2 Modulated sine curve used to generate the Input/Output file *ModSin5000.asc*

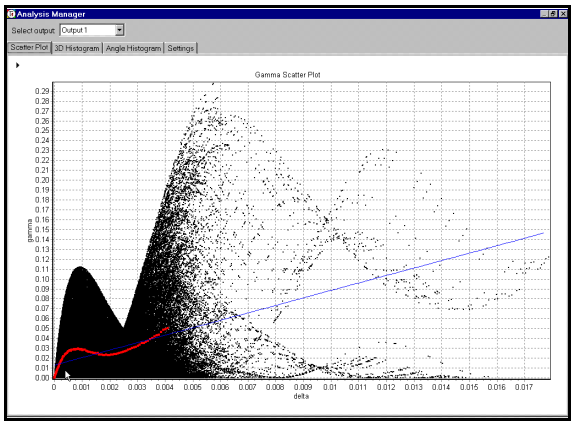


Figure 2-3 Scatter plot with pmax = 100 for *ModSin5000.asc*.

The scatterplot with pmax = 100 is shown in Figure 2.3. This illustrates both levels of periodicity and also shows why to get an accurate Gamma statistic we should take pmax fairly small.

2.3.2 The 3D histogram.

This is just another way of viewing the scatter plot. The software can also display the scatter plot as a 3D histogram, as for example in Figure 2.10, which can be rotated and examined from different viewpoints. Click the left and right pointing red arrows to rotate the viewpoint. Default is to display frequency values linearly on the vertical axis but there is also an option for a logarithmic vertical scale.

This can illustrate more clearly the 'wedge shaped' area. It can also be used to quickly ascertain the distribution of outliers. We shall call point pairs with large δ (each is a long way from its nearest neighbour) and large γ (the y values of close inputs are far apart) *strong outliers* and techniques for identifying and eliminating such points will be discussed in a later version of this document.

2.3.3 The angle histogram.

To help to further analyse the situation the software also produces an '*angle histogram*'', as for example in Figure 2.11, for each point in the scatter plot we imagine joining the gamma intercept on the vertical axis of the regression line plot to the scatter point. The angle the resulting line makes with the positive horizontal axis is then computed. This angle lies between $[-\pi/2, \pi/2]$. A histogram of the resulting angles is then displayed. The feature to look for in this histogram is the frequency of angles close to the right-hand end, i.e. close to $\pi/2$. If there are no points close to $\pi/2$ (= 90 degrees) this is

a good indicator for smooth modelling. If there are many points close to $\pi/2$ this is a very bad sign. The importance of the distribution close to $\pi/2$ in the angle histogram is another way to visualise the upper left hand wedge of the scatter plot.

The remaining types of Experiment which can be performed are described in the following sections.

2.4 Increasing near neighbours

This experiment shows how the *Gamma statistic* (and the other results returned by the Gamma test) varies with the number of near neighbours used to compute it. It is used to get some idea of how accurate the *Gamma statistic* is liable to be.

If we perform this experiment and use the graphing facility to plot the *Gamma statistic* and the *SE* against the number of near neighbours, by examining the graphs together we can usually see which choice for the number of near neighbours is likely to produce the most accurate estimate.

For example in Figure 2.4 (produced from *Sin500.asc* we see that the *SE* first increases and then for a while plateaus before (eventually) beginning to steadily increase. The range of the plateau is roughly between 7-27 near neighbours and it minimises at around pmax = 17 with a *Gamma statistic* slightly larger than 0.074, which we know (from the way the data was constructed) is close to the correct value.

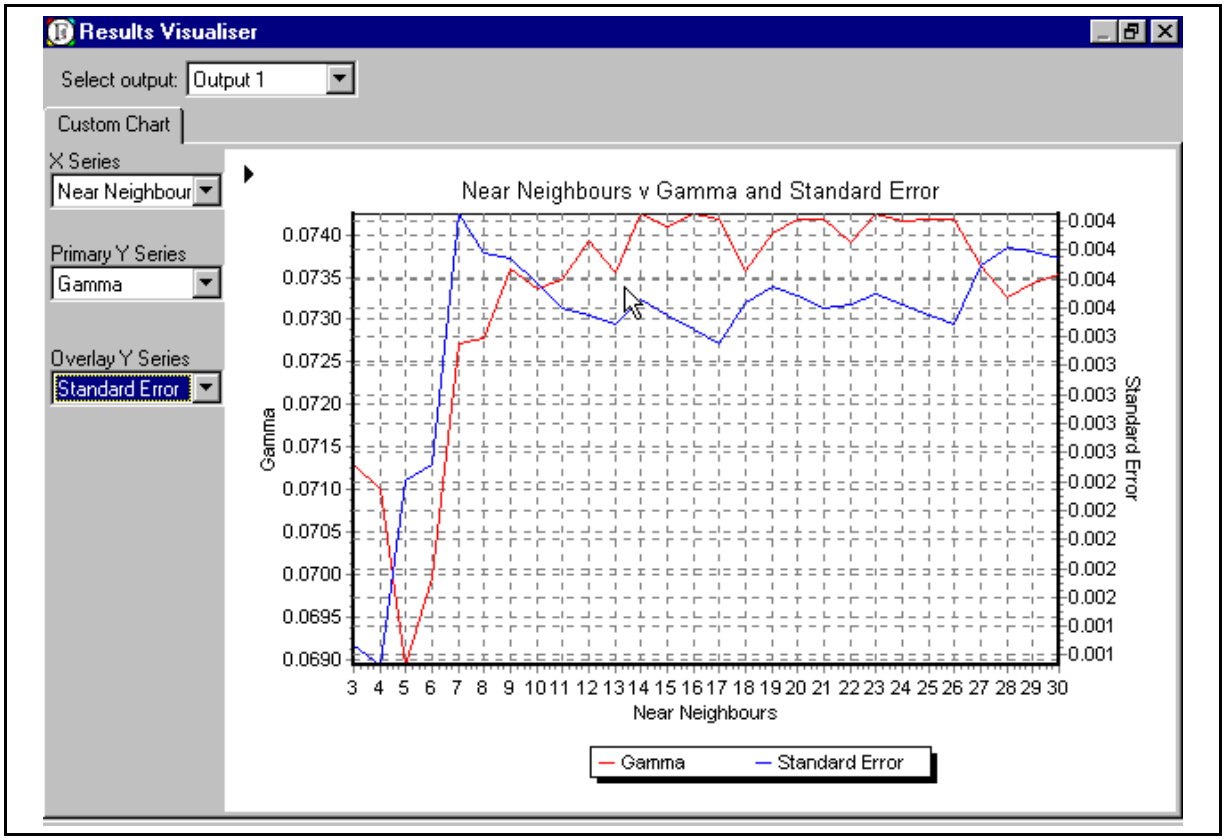


Figure 2-4 The variation of Gamma and *SE* as the number of near neighbours increases.

We also note that the *Gamma statistic* is reasonably stable in the same range.

It is also sometimes interesting to observe when the *Gamma statistic* is at a local minimum and the *Gradient* is at a local maximum as the number of near neighbours varies[6]. This criterion seems to be sensitive to noise on different scales of distance in input space.

2.5 *M*-test

This test is used to show how the *Gamma statistic* (and the other results returned by the Gamma test) estimate varies as more data is used to compute it. Eventually, if enough data is used the *Gamma statistic* should asymptote to the true noise variance on the output for which it has been computed.

The *M*-test can also tell us how much data we are likely to need to obtain a model of a given quality, in the sense of predicting with a *MSError* around the noise level. In Figure 2.5 we see that in this sense a perfectly adequate model can be built using anywhere from 150-200 data points, since the variance of the *Gamma statistic* after this stage is relatively small compared with its actual value.

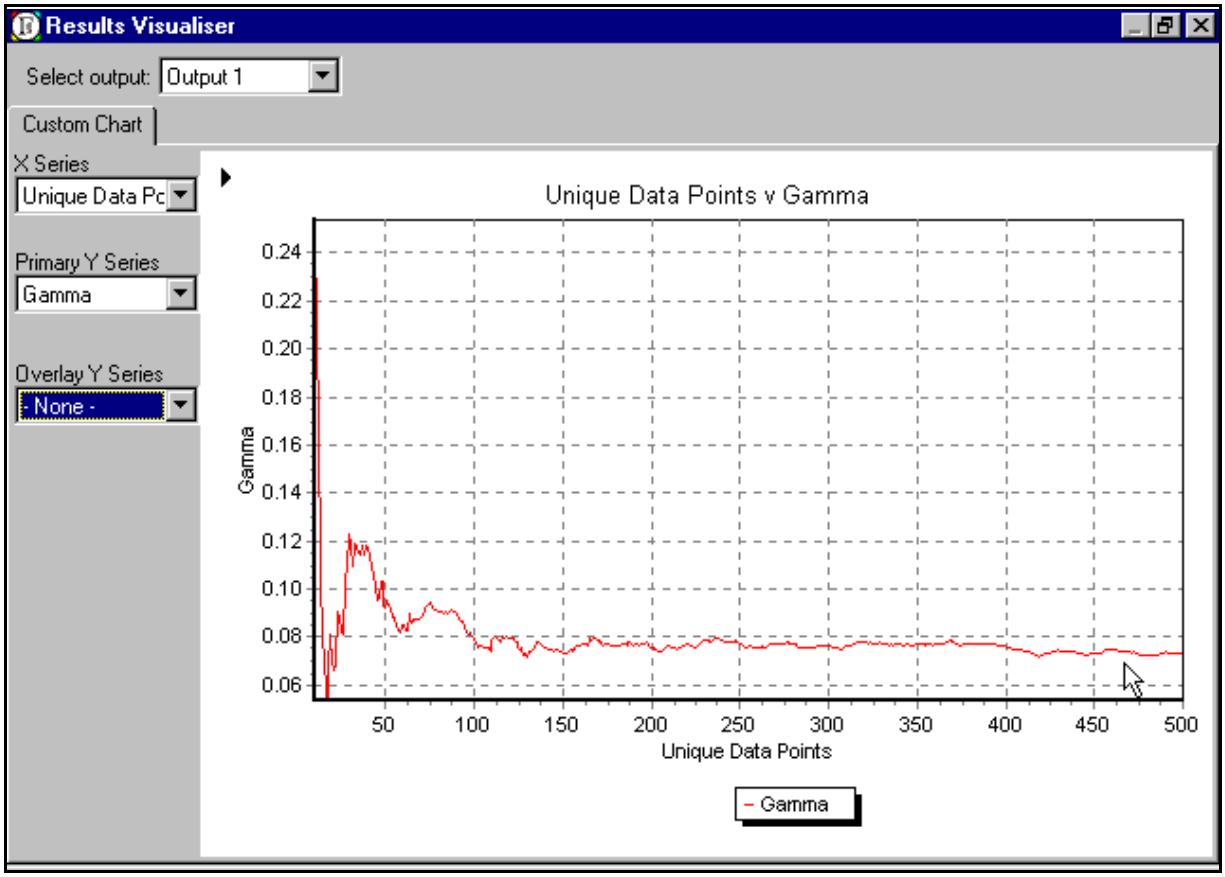


Figure 2-5 *M*-test graph for *Sin500.asc* Note the relatively stable asymptote.

[6] We call these the `Terry points' after John Terry who first observed the phenomenon.

Of course, using more data we can actually often progressively improve the *model* (this can easily be checked by building a local linear regression model and using the **WhatIf** option to recover a quite good approximation of the original sine curve), but it is not necessarily helpful to have an extremely accurate model if the output data we are comparing it with is subject to large amounts of noise.

2.6 Moving Window Gamma test

The **Moving Window Gamma test** shows how the estimate for the *Gamma statistic* (and other relevant results returned by the Gamma test) using a fixed number of data points varies as we move along the data file. It gives some indication of how stable the *Gamma statistic* is when estimated for different subsets of the data all having the same size.

The remaining sections deal with *model identification*, i.e.(in this context) the best choice of inputs for predicting a given output.

2.7 Model identification

2.7.1 Full embedding.

An *embedding* is a selection of inputs chosen from all the possible inputs. In *winGamma* an embedding is designated by a string of '1's and '0's called a *mask*. Thus if there five inputs the mask 10111 indicates that all inputs are to be used are to be used in the embedding except the second.

A *full embedding* tries every combination of inputs to determine which combination yields the smallest absolute Gamma value. It returns the number of results requested. If there are m scalar inputs then there are 2^m - 1 possible embeddings (the embedding where no inputs are chosen can obviously be omitted). If m = 20 this is around one million. To do a full embedding we therefore have to perform one million or so Gamma tests, which is fairly time consuming, although it can be done in about a week on a fast PC.

Even if m is sufficiently small to make this practical (say $m \leq 20$), before we perform a full embedding (assuming say $m > 10$) we should ask if we have *sufficient data* to justify it - because looking at around one million Gamma values the differences between many of them will probably be quite small and so we should ask if our estimates of the Gamma values are *accurate enough* to be able to make these distinctions. Whether or not the estimates are sufficiently accurate to choose the absolutely best embedding will mainly depend on how much data is available. It practice the best few embeddings will usually have little to choose between them.

Because a full embedding on a large number of inputs is often pointless or impractical *winGamma* offers a number of excellent heuristic methods to find a good embedding and these are described in the following sections.

A useful feature associated with a full embedding or GA search is the *Embedding Histogram*, which shows the frequency of embeddings with a specific *Gamma statistic*. If the choice of embedding is largely determined by statistical variations in the data this histogram tends to have a Gaussian or

Normal distribution (see Figure 2.33). If on the other hand there are clear underlying dynamics in the data then the histogram often shows a bimodal or multimodal distribution (see Figure 2.36).

2.7.2 Genetic Algorithm

This option searches the space of all masks using a Genetic Algorithm (GA) to find good embeddings. The parameters which can be used to control this search are (default values of parameters are given in brackets):

Population Size (100) The size of the population of masks being used throughout the search.

Mutation Rate (0.01). The probability that an individual bit will be mutated during the reproduction process.

Crossover Rate (0.5) The chance of inserting a random length run of bits from a parent mask to a child mask (i.e. the probability that a crossover event occurs during the reproduction process).

Gradient Fitness (0.1) The weighting in the GA fitness function for masks giving a low gradient in the Gamma Test. Increasing this weighting will place more emphasis on the relative simplicity of the modelling function

Intercept Fitness (0.8) The weighting in the GA fitness function for masks with a low absolute value of the *Gamma statistic*. Increasing this weighting will place more emphasis on the model accuracy.

Length Fitness (0.1) The weighting in the GA fitness function for masks with a given number of '1's. Increasing this weighting will encourage the selection of masks with fewer '1's and thereby place more emphasis on simpler models.

Note the three weightings selected for GA fitness should sum to 1.

Run Time (5 minutes) The (approximate) maximum time selected to perform the GA.

Setting the population larger may improve the final fitness of the best mask found but only if a large run time is permitted. For long masks (i.e. a large number of inputs) and large data sets the GA will require runs of several hours.

2.7.3 Hill Climbing

In hill climbing a mask is taken (default is all ones for the current number of inputs) and each bit is flipped in turn calculating the Gamma until the end of mask is reached. This is repeated until no single it flip gives an improvement on the Gamma. This is a relatively fast heuristic but takes longer than a sequential embedding.

2.7.4 Sequential Embedding

Here a single pass through the current mask is made, flipping each bit only if there is an improvement over the *Gamma statistic* obtained with the original mask. Again default is a mask of all ones equal to the length of the current input vector- though as in the previous method an initial mask of any kind can be used provided its length does not exceed the current number of inputs. This is very fast.

2.7.5 Increasing Embedding

The Increasing Embedding algorithm starts with the mask obtained by taking only the rightmost input (in the case of a time series this is the most recent) and obtains a *Gamma statistic* for this mask. It progressively increases the number of bits set in the mask working from right to left performing a Gamma test for each new mask. It runs to the maximum number of inputs and stops. We can then examine the *Gamma statistic* for each mask. The best embedding found will be the one whose *Gamma statistic* is closest to zero. This is useful in a time series to discover the underlying embedding dimension as we saw in section 1.5.2.

In the next sections we shall give example analyses using these various options.

2.12 Analysing Input/Output data

2.12.1 The *Ran500.asc* data.

We begin with a data set which is a type of 'worst case' in the sense that there is no smooth data model for this example. The file *Ran500.asc* is is a 4-Input/1-Output file containing 500 I/O pairs of completely random data generated using a uniform distribution in [-1, 1] via the *Mathematica*™ test file *DataGen.nb*. The output is actually pure noise having a true variance of 0.333333. A point plot of the output is given in Figure 2.6.

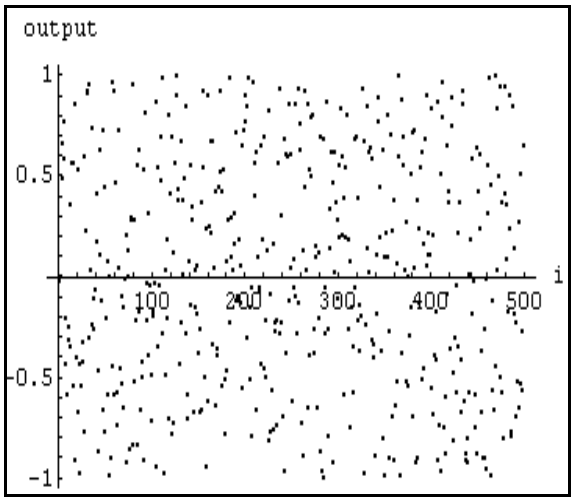


Figure 2-6 The *Ran500.asc* output plotted against the position in the file.

If we run a simple Gamma test with pmax = 10 near neighbours we obtain the results in Table 2-1.

The estimated *Gamma statistic* ≈ 0.31793 indicates a high noise level as does *Vratio* ≈ 0.97821 which is very close to one. The regression line with slope $A \approx 0.0575$ on scaled data is close to horizontal.

With pure random non-smooth data the slope of the regression line will gradually increase as the number of data points M is increased - this is because the continuity condition is not satisfied.

Taken together, particularly with *Vratio* so close to one, these are clear indicators that it is pointless to try to model the data with a smooth function. Next we examine the standard analysis tests in turn.

Table 2-1 The results of a simple Gamma test on the file *Ran500.asc* for unscaled and scaled data.

	Unscaled	Scaled
Gamma	0.31793	0.24838
Gradient	0.08426	0.057506
Standard Error	0.01429	0.012318
Vratio	0.97821	0.99353
Near Neighbours	10	10
Start	1	1
Unique Points	500	500
Evaluated Output	1	1
Zero Near Neighbours	0	0
Upper 95% Confidence	-	-
Lower 95% Confidence	-	-
Mask (remaining entries)	1111	1111

The **Increasing Near Neighbours** plot for pmax = 3 to 30 is given in Figure 2.7. This suggests the best estimate for the *Gamma statistic* is obtained at around pmax = 10. The *M*-test result of Figure 2.8 was obtained starting at $M = 50$ and increasing M to 500 in steps of 10. This consistently gives a Gamma statistic of around 0.3, but ideally as the graph has not yet settled to an asymptote we should need more points to obtain an accurate estimate for this 4-dimensional data.

The scatter plot in Figure 2.9 contains points with small δ but large γ which also supports the conclusion. At the same time the regression line fit is rather poor. The 3D histogram in Figure 2.10 shows no real indicators of an `empty wedge' and supports the general conclusions that the data is extremely noisy. The same is true of the angle histogram in Figure 2.11.

Finally the **Moving Window Gamma test** using a window size of 300 in steps of 10 in Figure 2.12 consistently shows a *Gamma statistic* between 0.29 and 0.38.

- These results together indicate that there is no point in going on and trying to produce a smooth model for this data.

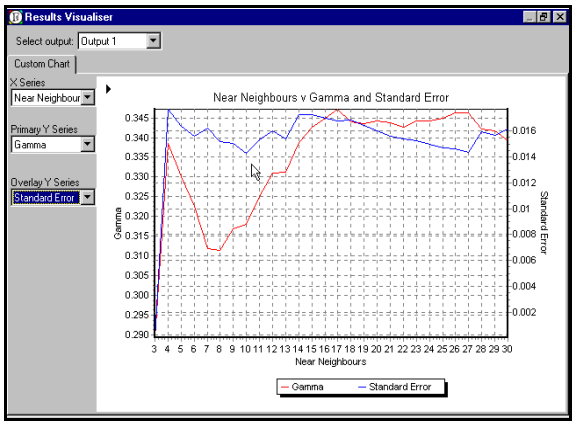


Figure 2-7 Increasing near neighbours (3-30) on *Ran500.asc* Gamma/*SE*

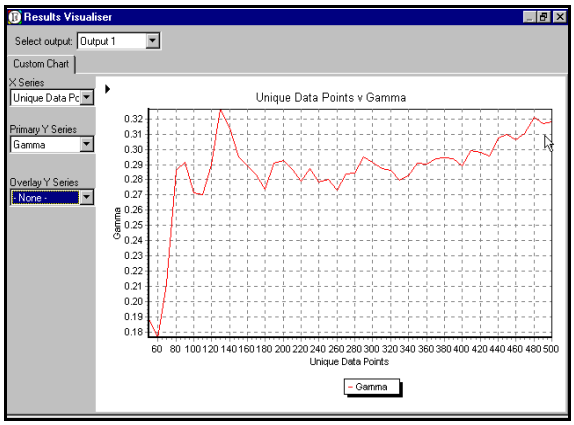


Figure 2-8 *M*-test (pmax = 10) on *Ran500.asc*.

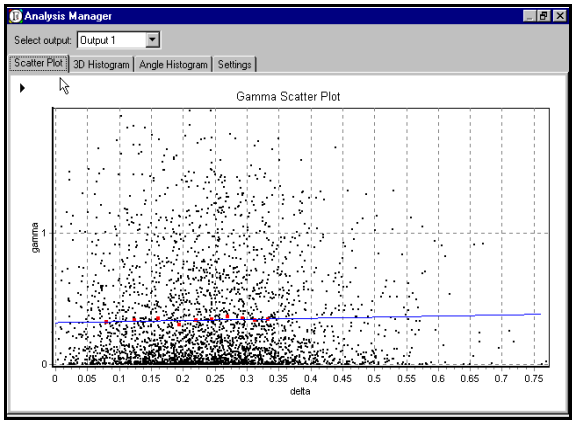


Figure 2-9 Scatterplot and regression line (pmax = 10) for *Ran500.asc*.

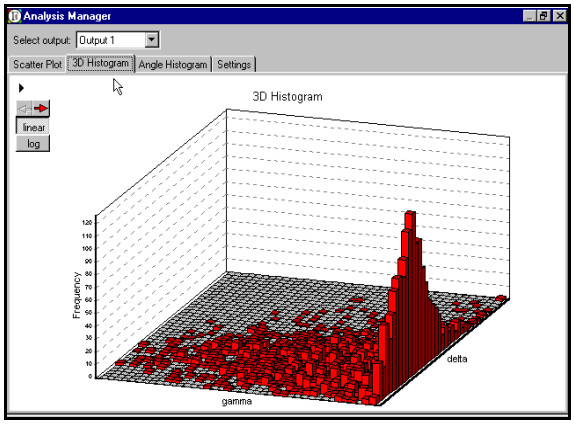


Figure 2-10 3D Histogram (pmax = 10) for *Ran500.asc*.

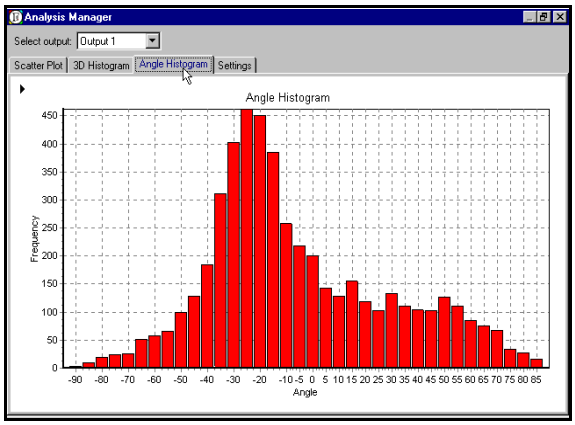


Figure 2-11 Angle histogram for *Ran500.asc* (pmax = 10).

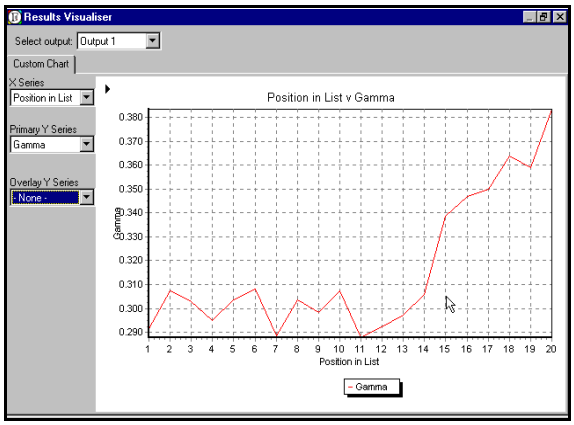


Figure 2-12 Moving window Gamma test (pmax = 10) on 300 points in steps of 10 from *Ran500.asc*.

2.12.2 The *Sin500.asc* data.

If we run a simple Gamma test with pmax = 10 near neighbours we obtain the results in Table 2-2.

Table 2-2 The Gamma test result (pmax = 10) for unscaled and scaled data on the file *Sin500.asc*.

	Unscaled	Scaled
Gamma	0.07335	0.03190
Gradient	0.71122	4.0386
Standard Error	0.00376	0.00163
Vratio	0.12762	0.12762
Near Neighbours	10	10
Start	1	1
Unique Points	500	500
Evaluated Output	1	1
Zero Near Neighbours	0	0
Upper 95% Confidence	-	-
Lower 95% Confidence	-	-
Mask (remaining entries)	1	1

The estimated *Gamma statistic* ≈ 0.07335 indicates a moderate noise level as does Vratio ≈ 0.12762.

The regression line on the scaled data with slope $A \approx 4.0386$ indicates definitely non-linear data. However if we pull up the *Gradient* plot we see that it is still highly variable - so one should not have too much confidence in this observation.

Taken together, these results indicate noisy but manageable non-linear data.

The **Increasing Near Neighbours** plot for pmax = 3 to 30 is given in Figure 2.13. This suggests the best estimate for the *Gamma statistic* is obtained at around pmax = 17. The *M*-test result of Figure 2.14 was obtained starting at $M = 50$ and increasing M to 500 in steps of 10. This consistently gives a *Gamma statistic* of around 0.07, but ideally as the graph has not yet settled to an asymptote we should need more points to obtain an accurate estimate for this noisy 1-dimensional data.

The scatter plot in Figure 2.15 contains points with small δ but large γ which also supports the conclusion. At the same time the regression line fit is rather poor. The 3D histogram in Figure 2.16 shows partial indicators of an `empty wedge' and supports the general conclusions that the data is noisy. The same is true of the angle histogram in Figure 2.17. Finally the **Moving Window Gamma test**, using a window size of 300 in steps of 10, in Figure 2.18 consistently shows a Gamma statistic between 0.072 and 0.076.

These results together indicate that we have noisy non-linear data but that model construction is quite feasible.

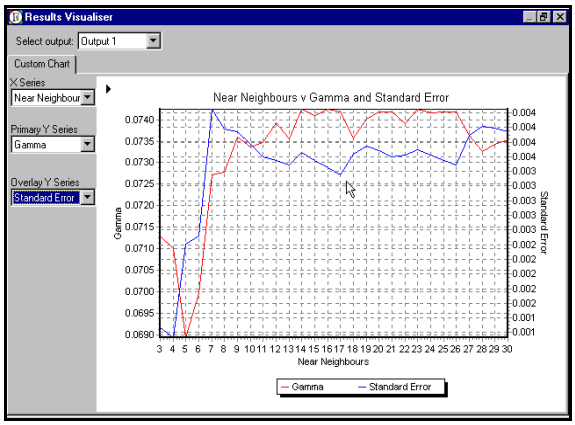


Figure 2-13 Increasing near neighbours (3-30) on *Sin500.asc* Gamma/*SE*

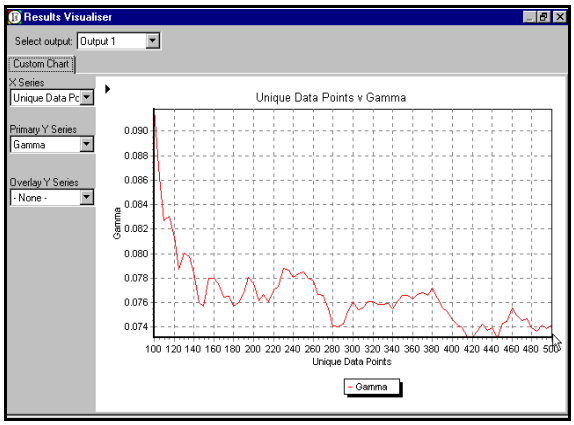


Figure 2-14 *M*-test (pmax = 17) on *Sin500.asc*.

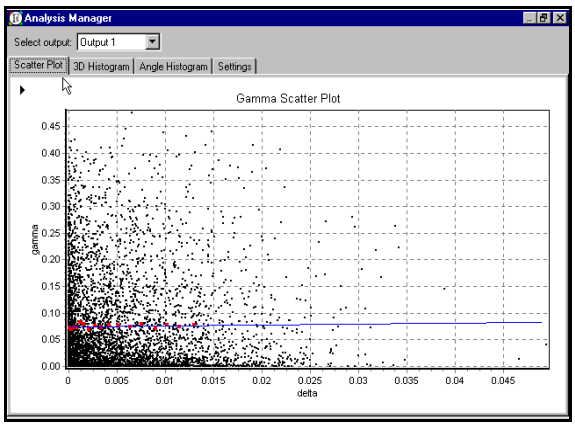


Figure 2-15 Scatterplot and regression line (pmax = 17) for *Sin500.asc*.

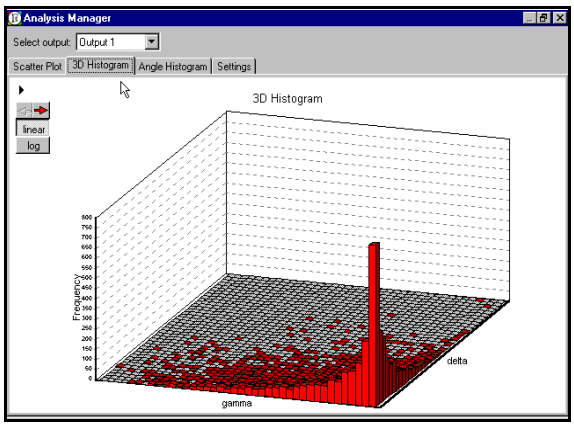


Figure 2-16 3D Histogram (pmax = 17) for *Sin500.asc*.

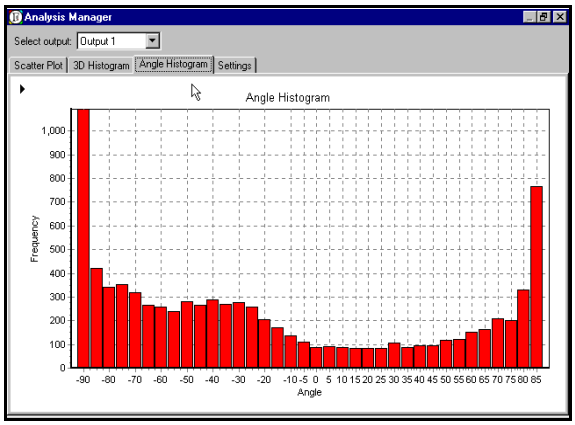


Figure 2-17 Angle histogram for *Sin500.asc* (pmax = 17).

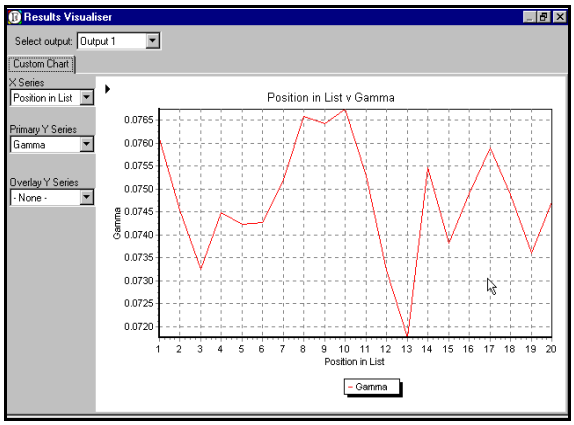


Figure 2-18 Moving window Gamma test (pmax = 17) on 300 points in steps of 10 from *Sin500.asc*.

2.12.3 The *solar.csv* data[7].

The data considered in this 2-input/1-output file relates to the generation of electrical power by an array of solar cells. The inputs are a measure of light intensity (South Plane Irradiance) to a precision of ± 0.01 kW/m^2, and the current temperature in degrees C to a precision of ± 0.5 ˚C.

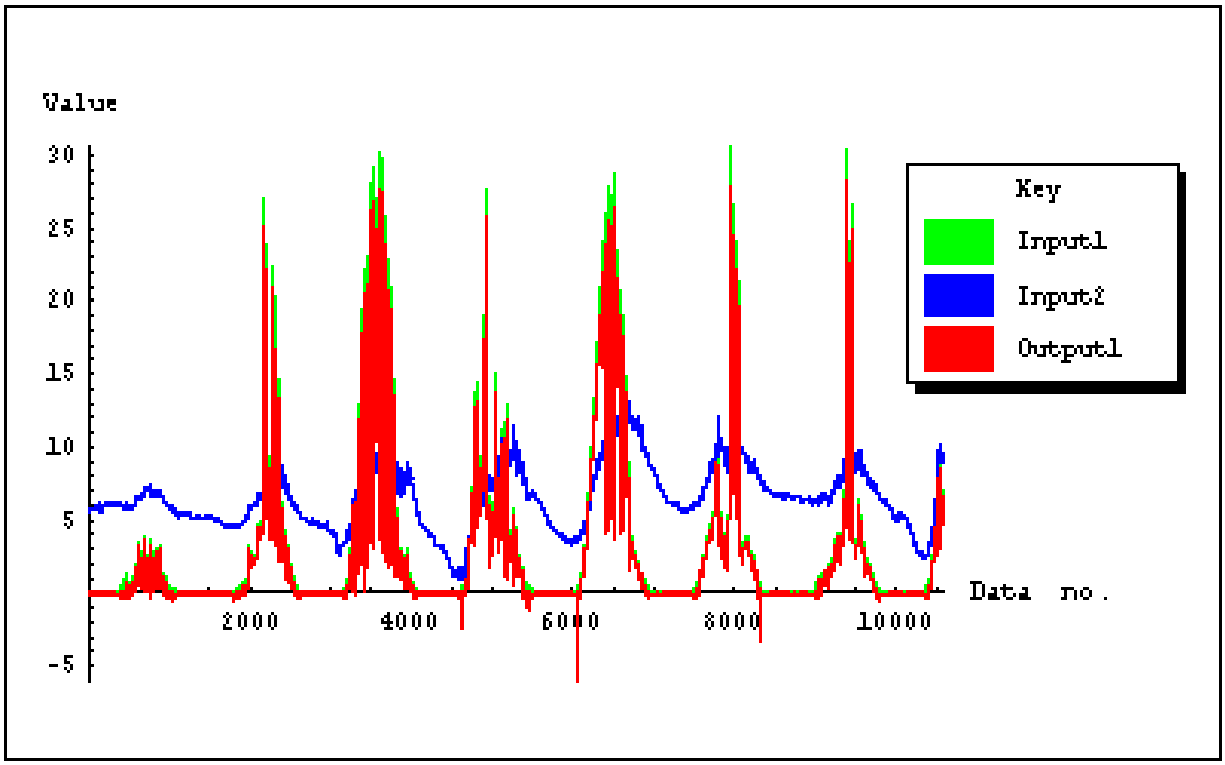


Figure 2-19 Plot against time of the irradiance and temperature from the training data file *solar.csv*.

The output is the voltage inverter AC power output measured to a precision of ±0.01 kW. The file consists of these values sampled every minute.

Figure 2.19 illustrates the graphs of the two inputs and the output against time (position in the file). We note that at low Irradiance the recorded power output values are irregular and sometimes negative. This is a result of the fact that intelligent circuits are attempting to determine whether or not to initialise the system as the sun rises or sets. The effect is to produce noise on the output power at low Irradiance levels. We are just using the data as an example, but if one wanted to use the data to build a really accurate model obviously one should filter out the data having low or zero Irradiance.

[7] Data for this example were provided by Newcastle Photovoltaics Applications Centre at the University of Northumbria at Newcastle UK. These data were collected as part of a project with funding from the European Commission (THERMIE Programme) and the UK Department of Trade and Industry.

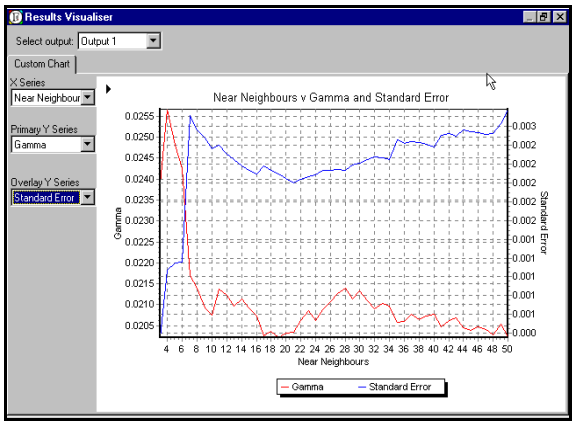


Figure 2-20 Increasing near neighbours (3-50) on *solar.csv* Gamma/*SE*

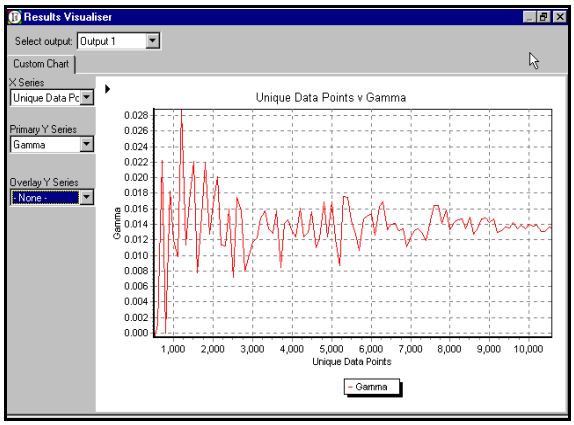


Figure 2-21 *M*-test (pmax = 20, Randomised, 2 repetitions) on *solar.csv*.

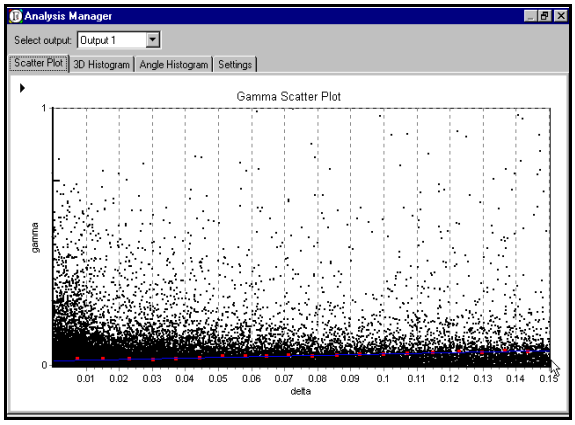


Figure 2-22 Scatterplot and regression line zoomed (pmax = 20) for *solar.csv*.

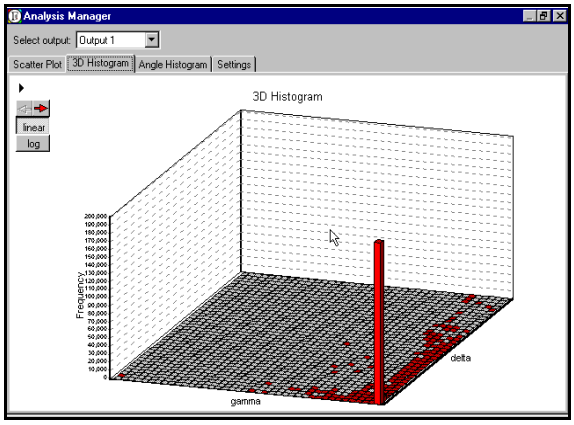


Figure 2-23 3D Histogram (pmax = 20) for *solar.csv*.

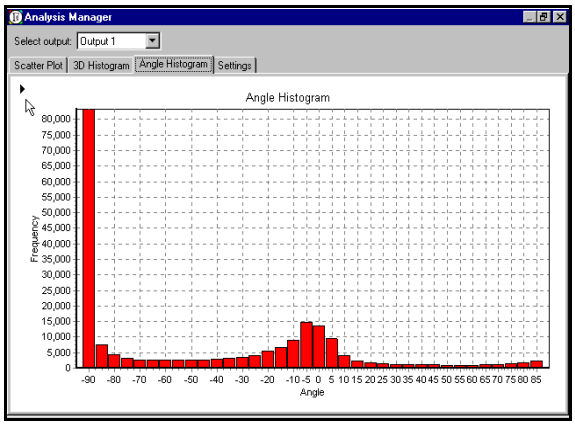


Figure 2-24 Angle histogram for *solar.csv* (pmax = 20).

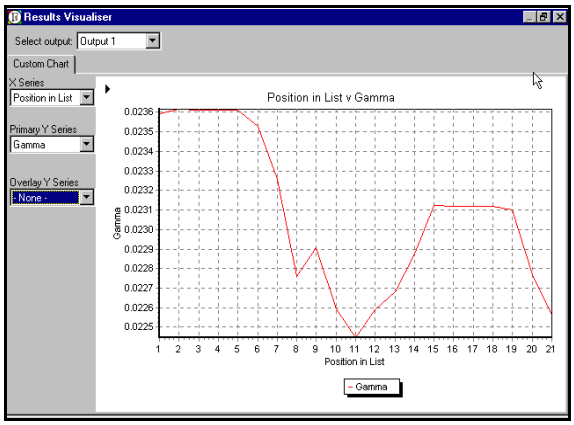


Figure 2-25 Moving window Gamma test (pmax = 20) on 8400 points in steps of 100 from *solar.csv*.

There are some other factors of interest in this particular situation. As it happens the sensor which measures Irradiance is on the roof of the building, whereas the solar array is in a nearby location. The solar array is shaded at certain times of day by a chimney and a nearby building and this shading is not measured by the Irradiance sensor. Since the shading is obviously a function of the time of day and the time of year this has the effect of introducing smooth non-linearities into the situation, which would be extremely hard to model analytically. One could imagine including the time of day and year into the data set and then building a different and much more accurate model. By examining the difference between the two models we could actually quantify the effect of shading without having an analytic model. This would be a good example of one type of application of *winGamma* to scientific data. However, since the data in this file only runs over about one week we do not consider this extra complication here

If we run a quick Gamma test on the full data set with pmax = 10 near neighbours we get the results of Table 1-1 in Chapter I.

The unscaled *Gamma statistic* of 0.020761 seems high but in view of the output range (approx [0, 30]) is actually quite good. A better measure is the V_{ratio} = 0.000760 (defined as the ratio Gamma/Var[output]), which is low and shows that the output is highly predictable from the inputs. Because the data clearly falls into two distinct classes (day and night) we should be aware that representative training and test data should include both types. The point to grasp here is that although the time series data varies from moment to moment (as clouds obscure the sun) the relationship between sunlight input at a given temperature and power output is a smooth (almost linear) model.

The next step in a more careful analysis is to run an **Increasing near Neighbours** test. This will give us some idea of the best pmax to choose to give the most accurate Gamma test results. Figure 2.20 shows the result of the increasing near neighbours test run for pmax = 3 to 50. We note the *SE* first increases and then plateaus. Along the plateau a minimum *SE* occurs at around pmax = 20, which from now on we take as the best pmax for further analysis of this data.

Table 2-3. The results of the Gamma test (pmax = 20) for unscaled and scaled data from *solar.csv*.

	Unscaled	Scaled
Gamma	0.020328	0.000221
Gradient	0.250261	0.230184
Standard Error	0.002051	3.095267E-5
Vratio	0.000744	0.000884
Near Neighbours	20	20
Start	1	1
Unique Points	10578	10578
Evaluated Output	1	1
Zero Near Neighbours	0	0
Upper 95% Confidence	0	0
Lower 95% Confidence	0	0
Mask (remaining entries)	11	11

Figure 2.21 Shows the *M*-test and we can see that for $M \approx 9000$ we are beginning to get a stable asymptote. From this we infer that around 9000 data points will be required to build a model which will predict with an accuracy about equal to the noise level. The result of a Gamma test on pmax = 20 near neighbours using the full data set is shown in Table 2-3.

Figure 2.22 shows the scatter plot and regression line (zoomed). This is typical of slightly noisy data but the regression line fit is good. The desirable empty wedge is the top left corner is not obviously present, another indication that the data is somewhat noisy.

The 3D Histogram in Figure 2.23 shows similar features except that we can see the actual frequency distribution of points in the scatter plot and this shows that strong outliers or `empty wedge' points although present are relatively infrequent. The Angle histogram of Figure 2.24 shows a roll-off in frequency as we approach angles close to $\pi/2$

The final **Moving Window Gamma Test** of Figure 2.25 is intended to show the relative stability of the Gamma test result. In this case it really fails to do so because the order of the data should really be randomised (since it is very time periodic). Even so when we examine the vertical scale of Figure 2.25 we see that the relative variation is not very large.

We see shall later in Chapter III how to take the results of this analysis and build and test models using the *solar.csv* file.

2.13 Analysing *Time Series* data

2.13.1 The *DH(34)5000.asc* data (Delayed Henon Map).

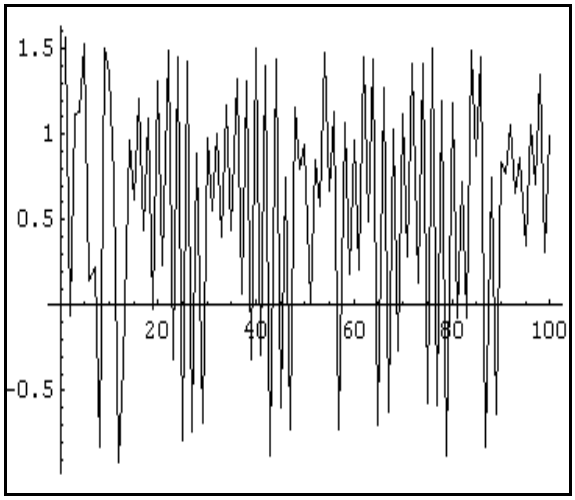


Figure 2-26 The first 100 points of the delayed Henon map time series.

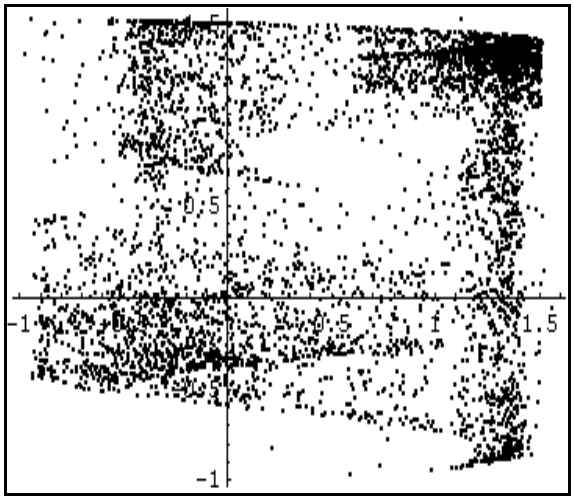


Figure 2-27 The return map (x_{n-1}, x_n) for the delayed Henon map.

This Time Series data was generated by a process very similar to the Henon map, except that where the current value of the Henon map time series depends the last two values of the series, for the Delayed Henon map the current value is determined by the values three and four steps in the past. This changes things in a number of respects.

The plot of the time series is given in Figure 2.26 and Figure 2.27 shows the return map for (x_{n-1}, x_n) which is analogous to Figure 1.12. We observe that this distribution looks quite different.

We now proceed through the steps outlined in section 2.1.1 for Time Series.

1. Load the data and do not initially normalise (it is a single time series). Set the number of inputs to 20 the number of outputs initially to one. Now perform a simple *Gamma test* on the full data set (which gives 4985 I/O pairs with 20 inputs) to get an initial idea. If the data set is very large use a subset of the data for initial experiments.

Results. This gives an initial *Gamma statistic* of 0.0042614 and a Vratio of 0.007481. The *SE* for this result is 0.00125. These initial results are encouraging.

2. Next run an *Increasing Embedding test* to determine a likely embedding dimension.

Results. If we zoom in on the resulting graph we see Figure 2.28 and infer that a good model is likely to be obtained with 4 or 5 previous values.

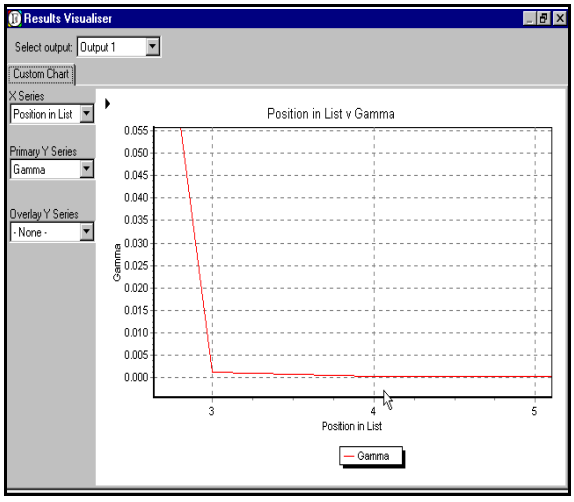


Figure 2-28 The result of an Increasing Embedding for the delayed Henon map.

The *Gamma statistic* for 4 is 0.00019635, for 5 it is 0.0002997 but the lowest value of all is for 7 past values which gives -1.5E-7. These very low values suggest that the time series is consists of very low noise, or noise free, data. Examination of the scatter plot and associated graphics supports this view.

3.We next **Transform** the data set to reset the maximum number of inputs to 8.

4. We next run a *M-test* to check the stability of the *Gamma statistic* If the *M*-test produces a stable asymptote we can decide if we really have enough data to support these conclusions. A reasonable choice is to start at 100 in steps of 100 until the end of the data.

Results. The *M*-test graphs of the *Gamma statistic* together with the *Gradient* are shown in Figure 2.29. From this we see a good asymptote and conclude that with 8 inputs a good model can be obtained using around 3000 points. It also looks likely that we have an essentially zero noise time series.

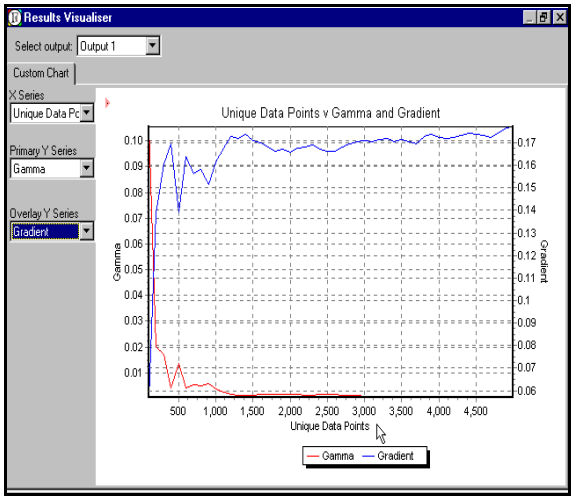


Figure 2-29 The M-test graph (pmax =10 number of inputs = 8) for the delayed Henon map.

5. Can we get a better *Gamma statistic* by discarding still more of the inputs? To answer this question we run a *Full Embedding* on 7 inputs. (To do this we have to transform the data again).

Results. If we make a small table of the best 5 masks found we obtain:

Gamma	2.3228E-6	1.3276E-5	-1.888E-5	-2.176E-5	-2.438E-5
Mask	0001101	1101100	1111111	0001100	1011101

From this we infer that lags 3 and 4 (remember we have to count from the right) are very important but that the marginally best model should be obtained using lags 1, 3 and 4.

It is worth examining the *Embedding Histogram* associated with the embedding search. In this case we see a somewhat irregular multimodal histogram (there are only 15 possible embeddings).

6. If we now fix on the embedding 1101 having a *Gamma statistic* of around 2.3228E-6 then we might next do an **Increasing Near Neighbours Experiment** to optimise the choice of near neighbours in estimating the *Gamma statistic*.

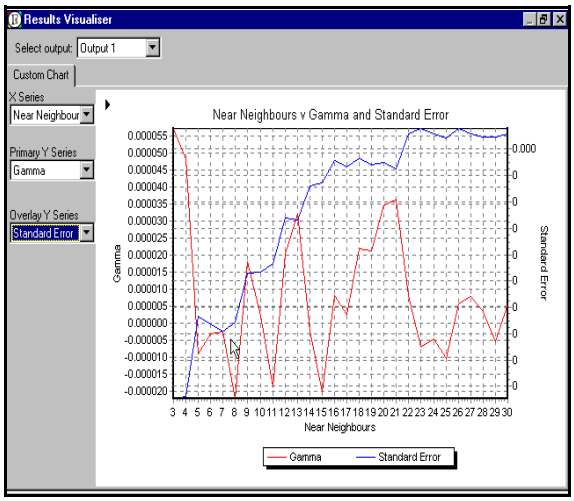


Figure 2-30 An Increasing Near Neighbours test on the embedding 1101 for the delayed Henon map.

Results. The result is shown in Figure 2.30. The minimum *SE* is obtained using pmax = 7 nearest neighbours and corresponds to a final *Gamma statistic* of 2.3228E-6 so that optimising the number of near neighbours hardly changed the *Gamma statistic* at all in this case.

Thus the final analysis of *DH(34)5000.asc* is that it is a noise free time series. Using a few thousand points we should be able to construct a model capable of one step prediction with an estimated *MSError* of around 2.23E-6.

2.13.2 The FTSE weekly closing price data.

The file *FTSEcls.asc* contains the FTSE weekly closing price from 9 May 1988 - 26 January 1998 which gives 508 samples. Figure 2.31 shows the time series over the full run of the data.

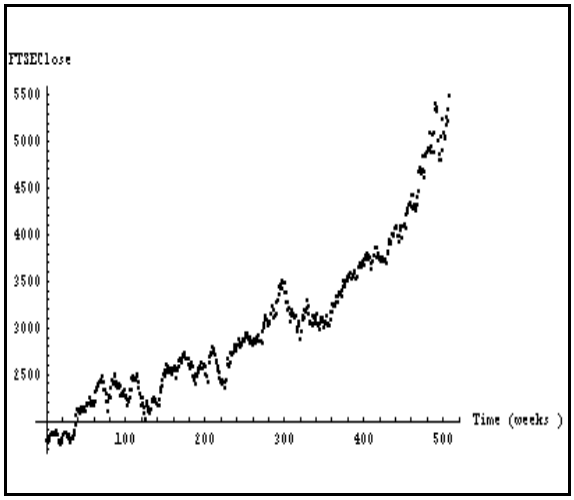


Figure 2-31 A plot of the FTSE close data.

If we perform a full embedding on the last 20 weeks (on a convenient Unix station!) using pmax = 10 we obtain the histogram in Figure 2.33. This has the characteristic Gaussian shape of mainly statistically determined data.

From the table of the best embeddings we select the embedding which gives the smallest positive gamma. This is 11011100110000111101.

Handy Tip. Typing in long masks can be error prone and tedious. There is no need to do this. A mask can be copied to the clipboard and pasted in whenever required using a right click on the mouse.

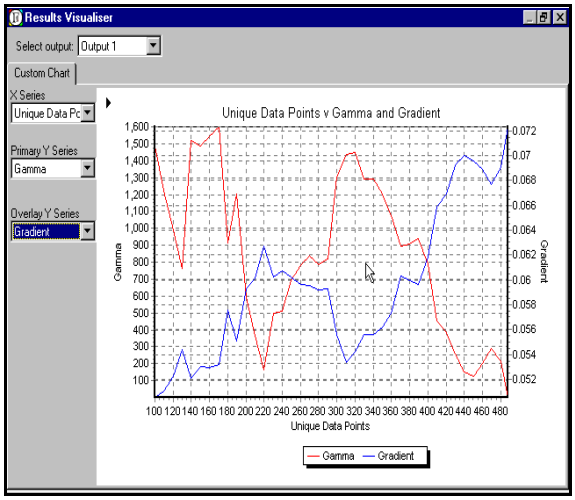


Figure 2-32 The M-test graph for the FTSE data using the best embedding of length 20.

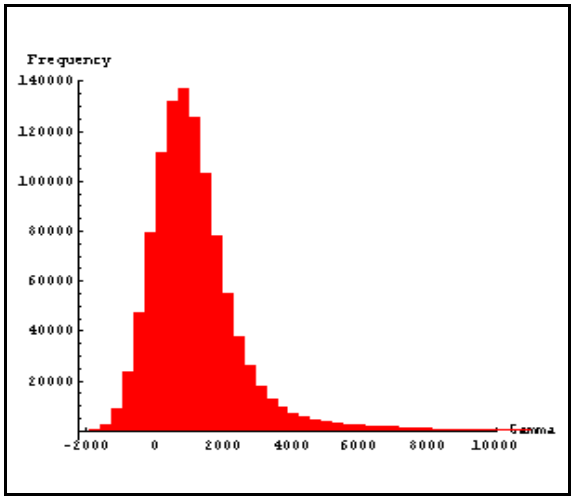


Figure 2-33 The frequency histogram for embeddings of length 20 using the FTSE data.

This choice of embedding should be treated with some caution. The *M*-test graph of Figure 2.32 (pmax = 10, Randomised, 3 repetitions) shows that the estimated gamma values have not yet stabilised (*M* is not sufficiently large) so the error in estimating the *Gamma statistic* for any particular embedding is sufficiently high to make the outcome of a full embedding search itself extremely unreliable. The resulting very low *Gamma statistic* of around 0.007 is an artifact of the statistics of the situation (with over a million embeddings to search we are quite likely to find one with a very small Gamma). The associated *SE* is 517! This clearly illustrates that a low Gamma statistic on a single data set is not enough to ensure a good model - we need to be sure that the *SE* is acceptable and that an *M*-test illustrates the estimate has stabilised.

In reality using the time series alone we are lucky to predict the weekly closing FTSE price to within a standard deviation of 80 (i.e. the true *Gamma statistic* is around 6400).

There is a further complication in that we have no real reason to suppose either that the underlying system is describable by a smooth dynamic model, or that if so the dynamical system is constant. Indeed towards the end of the 10 year period it is noticeable that the local variance of both the system behaviour and (as we shall see in Chapter III) of the errors of predictions increase. From this we

conclude that either the dynamical system is itself varying or at the very least the constant noise variance assumption is suspect.

2.13.3 The sunspot data.

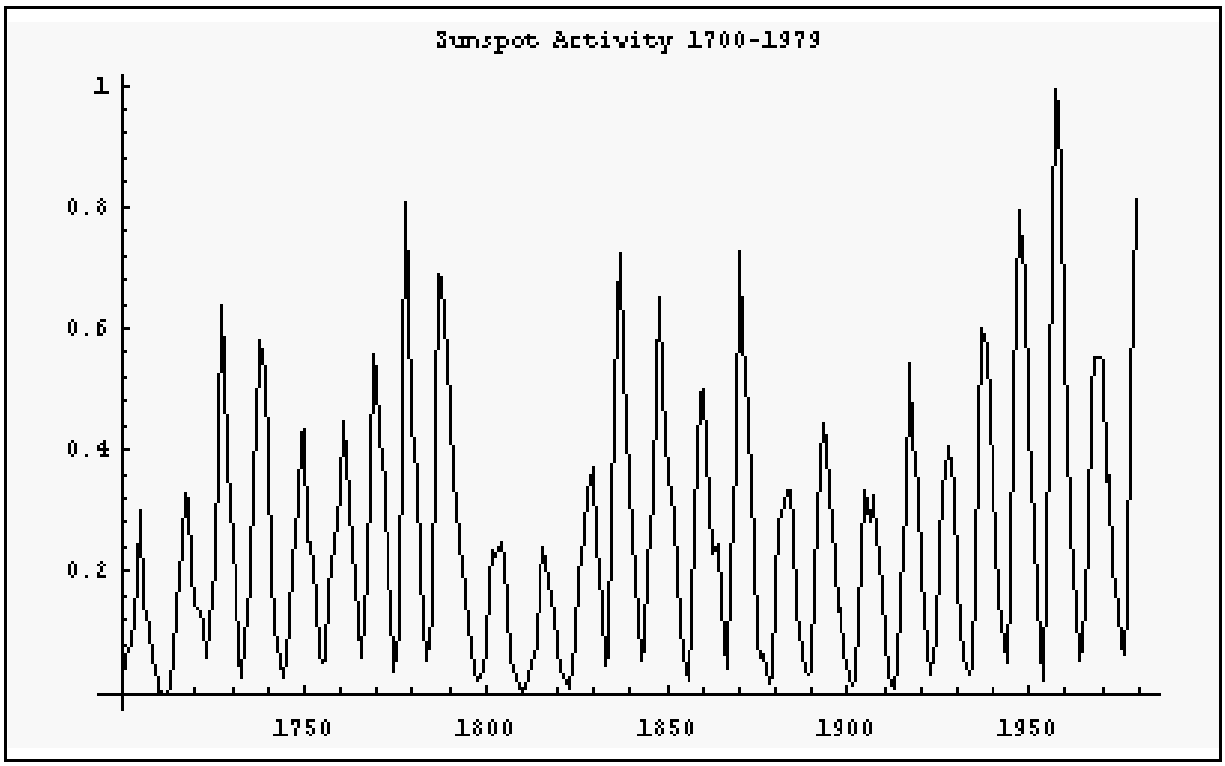


Figure 2-34 Plot of the sunspots data file *Sun280.asc*.

The data used in this experiment was FTP-ed from ftp address: *ftp.santafe.edu*, directory: *pub/Time-Series/data*. Its origin, normalization and training/test regions are described in [Weigend 1990]. The data consists of 280 points representing sunspot activity over the period 1700 - 1979 and was used in [Weigend 1991]. The range of the data has been scaled to [0, 1] and we found the variance to be 0.0410558. Figure 2.34 shows the variation of sunspot activity over the full range of the data.

It is known that the primary sunspot cycle is approximately periodic over 11 years. Other shorter and longer cycles are also known. For radio propagation the short period cycle of 28 days is particularly significant. The data used here is collected from telescopic observations projected onto a white paper card. The sunspots are counted and classified by size and a correction factor applied depending on the magnification of the telescope. The virtue of this data is that it has been regularly collected since 1700. Of course, if one were really interested in predicting sunspot activity much more accurate data is available. The data provided is often used as a test of prediction techniques and can give a reasonable model of gross sunspot activity.

Selecting a best embedding. If we are prepared for a several day run we can use the **Full-Embedding** option of the software to search for a good embedding. In this example we searched over the previous 15 years.

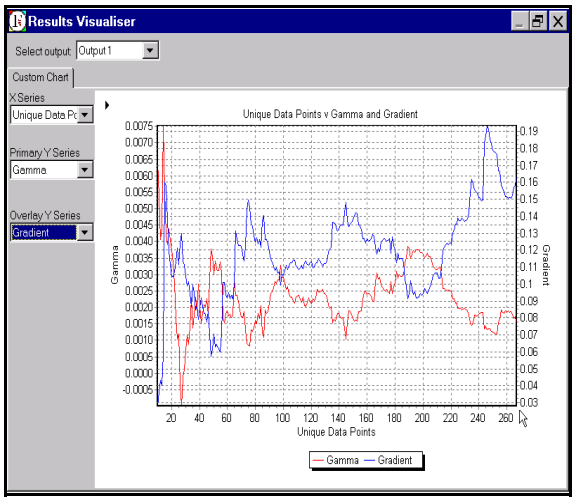


Figure 2-35 The M-test graph for the sunspot data data using the best embedding of length 15.

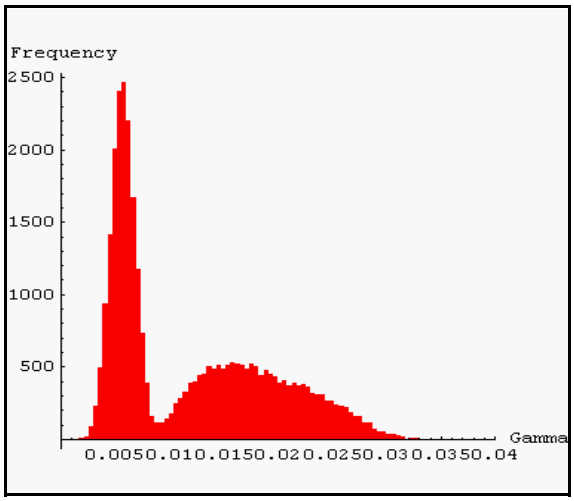


Figure 2-36 The frequency histogram of all embeddings of length 15 using the sunspot data.

The best embedding found was 001001000010111. Here the most recent data comes last. So this embedding says that to predict this year's sunspot activity $x(t)$ we should use the data $x(t\text{-}1)$, $x(t\text{-}2)$, $x(t\text{-}3)$, $x(t\text{-}5)$, $x(t\text{-}10)$ and $x(t\text{-}13)$, an embedding of dimension six. It is interesting to note the bimodal distribution of the **Full Embedding** Histogram of Figure 2.36. The bimodal distribution is partly explained by the observation that only 2.38% of the embeddings with a *Gamma statistic* > 0.008 include $x(t\text{-}1)$ as compared with 99.8% of those having a *Gamma statistic* < 0.008. Put plainly this says that the most important predictive factor for the sunspot activity this year is the value for last year. It is also interesting to see which variables appear in the best few embeddings. These indicate that the last few years, plus the value approximately one 11 year cycle back, plus a value about half way through the previous cycle, give the best results. This is rather impressive since the software has no way of knowing about sunspot cycles.

If we run the Gamma test on the six inputs/one output I/O data file constructed using this mask we get *Gamma* ≈ 0.0015 and V_{ratio} = 0.036 (SE ≈ 0.00093) Note the *M*-test of Figure 2.35 indicates that there is not really enough data (the graph has not stabilized). Therefore if we construct a model and test on unseen data we might expect to get a higher *MSError* than the estimated

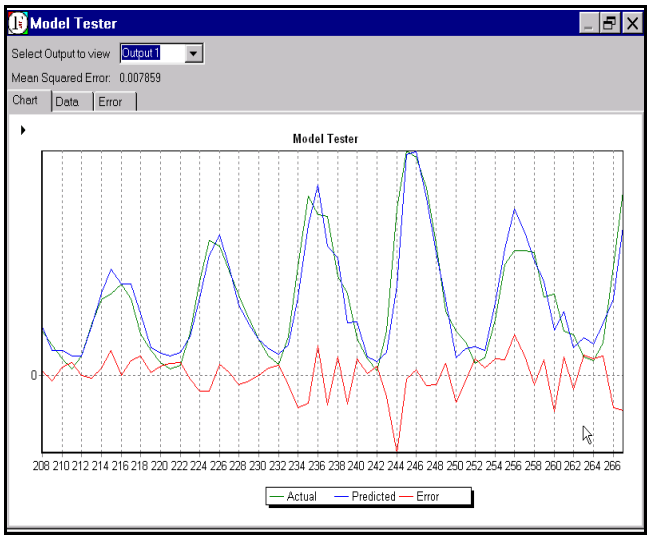


Figure 2-37 A test of the LLR model on the data set *SunPairs.asc* (blue predicted, green actual, red error).

gamma value. If we now predict the last 59 years data, using local linear regression with $k = 60$ near neighbours and a threshold of 0.0001 (we shall see how to do this in the next chapter), on the basis of all the previous years we obtain Figure 2.37 which gives a *MSError* around 0.007. In cases such as this, where there is insufficient data, it is not uncommon to see a *MSError* on unseen data around an order of magnitude greater than the *Gamma statistic*.

2.14 Handling multiple time series.

In later releases we plan to include a TimeSeries Editor to facilitate the direct manipulation of multiple time series. However, using a combination of *winGamma* and *Excel* at present it is possible to accomplish multiple time series manipulation relatively easily.

Suppose we have several time series TS1, ..., TSm, which we wish to use to predict a target time series Target.

Step 1.

Suppose the time series are in an *Excel file* which is structured as follows:

Table 2-4 *Excel* file for multiple time series.

Date	TS1	TS2	...	TSm	Target
10/07/1981	16	10.39		132.06	606.8
10/14/1981	15.5	10.34		132.92	606.8
10/21/1981	15.5	10.38		130.43	626
...	...	...	...	...	...

In *Excel* delete the date column because this is not numerical that can be used as input for *winGamma* notice now that the only factor which preserves the time relationship is the order of the rows.

> **Hint**: It is important when dealing with multiple time series that all the data in a row is sampled at the same time. If one measurement is sampled weekly and another monthly then we can use linear interpolation to construct weekly data samples for the monthly sampled data.

Step 2. Save the file from *Excel* in CSV format - you should include the first row of text descriptors of the time series as *winGamma* can handle these and they will be useful later.

Step 3. Load the file just saved into a text editor which can SaveAs ASC DOS text files and search and replace the commas separating the numeric data only . Do not the replace the commas separating the commas separating the text headers. Save the new file as a ASC DOS text file with the filename suffix *.csv (Edit the file name after saving if the text editor insists on putting an extra *.txt on the suffix.)

Step 4. Decide the maximum lag you are likely to need in analysis and modelling. For example, if you think the maximum lag likely to be needed in two months and the data is sampled weekly then the Number of Inputs to be set in *winGamma* will be 8. Load the *.csv file into *winGamma* (which because the numeric entries are separated by spaces will treat the file as a multiple time series). Selecting the number of inputs as 8 and, assuming that you wish to produce a one week ahead forecast select the Number of Outputs as 1. Do not at this stage normalise/scale the data, this can easily be done later but cannot be undone if it is done at this stage. You also have to decide at this stage whether you want to include a running window average for each time series as an input and whether you wish to include successive differences as possible inputs.

Without the running window average or successive differences, the data loaded into *winGamma* will now be ordered into the following columns:

TS1 : t-8,..., TS1 :t-1, TS2 : t-8, ..., TS2t-1, ..., TSm : t-8, ..., Tsm : t-1, Target :t-8, ..., Target : t-1, TS1 : t, ..., TSm : t, Target : t

where the outputs (at time t) have been underlined. (This is a rather tricky manipulation to do from scratch in *Excel*.)

Notice that although we set out with the intention of trying to model the time series *Target*, we have created a file in which every time series has an out put that we can model. You can delete these extra outputs in **Step 6** if you wish.

You can begin immediately with experiments on this file but, since not all these inputs may be needed for the model, you can also proceed as follows.

Step 5. Use some other software such as *Mathematica* to perform data analysis on the last file using tools not yet provided by *winGamma*. For example, one useful analysis tool is to take the average lagged correlations of successive differences of the target time series with the successive differences of all the time series (e.g. for lags from 1 to 8 in the above example). (This tool is available as *DeltaCorrelation* in the *Mathematica* suite provided with *winGamma*.). We may choose to take only those lags which have the largest absolute lagged delta correlation. This may suggest that some columns could be deleted from the *.csv file we have produced.

Step 6. Load the *.csv file into *Excel* and delete the columns which have been selected as unlikely to be useful. Re-save the result as a *.csv file and proceed (as if from the end of Step 4) with *winGamma* analysis on the resulting file. Further inputs may be set to zero in the mask as a result of *winGamma* analysis.

Notice that when you reload this file into *winGamma* it will be treated as an Input/Output file for which you have to specify the inputs and outputs. Becuase the data is not now recognised as multiple time series data the **Iterate** (Model) option will not be available (at present this can only be used with a single time series).

2.15 To scale or not to scale?

If two input variables are incompatible (e.g. temperature in degrees K and altitude in metres) both in semantics and in range then the effects of a change in one of them can completely outweigh the effects of a change in the other. To ensure that all variables at least start with an equal chance to contribute to an output prediction it is often helpful to apply a standard normalization.

In this software the *standard normalization* is that the mean of each input variable is mapped to zero and the standard deviation to 0.5. In later versions we may include an option for the user to select the standard deviation (this can have some advantages in model building).

The effect of normalizing the data is two fold. First, since the output is also rescaled this will affect the *Gamma statistic* in a trivial way (it will divide it by the square of the new output range). The *Vratio* however will not change due to this effect. Second, rescaling the inputs can change the near-neighbourhood relationships and hence possibly change the associated *Gamma* value. We can detect if this happens as it will also cause *Vratio* to change.

Whether normalization is a good or bad idea depends largely on the circumstances. If input variables are incompatible then it is probably a good idea to normalize.

Normalization of just the input values will not change the *asymptotic Gamma statistic* or *Vratio*, provided we imagine that as the number of data points becomes large we also increase pmax by a suitable constant factor[8], but a good scaling will cause the *M*-test *to converge more rapidly* to the asymptote, so improving the accuracy of the noise estimate for a given amount of data. A good scaling can also improve the accuracy of a model constructed using a fixed amount of data.

The effect of masking is 'all or none' and it may be better to apply a suitable weight to each input variable. For example, it is a general observation regarding near-neighbour classifiers that they perform well given the right weighting of inputs but that at present there are no general techniques for finding such weightings. However, if weights are applied then of course the data must NOT then be renormalised.

2.16 Projects

A **Project** is the collection of all Experiments performed on a given data set. A given Project is determined initially once the data set for analysis is defined. At Project creation time the number of inputs and outputs to a time series have to be set and options to normalise or scale the data. There is also an option to generate a parallel moving average and/or difference series alongside a time series.

As successive Experiments are completed the parameter settings for each Experiment and the results are added to the Project which can be saved as a file and reloaded at a later date. Thus there is no need to repeat the same experiment- which may have taken a while to compute.

[8] All distance functions are equivalent to within a constant. But rescaling changes specific near neighbour relationships.

Handy Tip The Project file (*.gpr) contains the path of the data file used in the project, in fact the data file name and path are the first item in the Project file. Project files are in DOS ASCII format and can be edited (as long as the file is saved in the same format after editing). This can be useful to know if the data has been moved or renamed, or you have moved both the Project file and the data file to another system where the path is different. One way to handle this is to just edit the path in the Project file. However, if *winGamma* cannot find the data file associated with a project it will ask you to **Browse** for the file and you can indicate the new location. (It is important to select the right file - which must not have been altered.)

CHAPTER III Building and testing a model

3.1 Introduction

We now assume that you have analysed the data and decided which inputs and how much data to use. To actually build the model *winGamma* offers several techniques:

Local linear regression
Dynamic local linear regression
Neural networks using various types of learning algorithm.

Local linear regression models are fast to construct and quite fast to execute a query. Local linear regression models can also be easily updated as new training data becomes available, which is not the case with neural networks (where a prolonged extra period of training, or starting training all over again, may be required to modify the model on the basis of new data). Indeed *winGamma* also offers a dynamic local linear regression option which is exactly local linear regression with dynamic updating (this option is quite useful for time series prediction).

Usually local linear regression is extremely accurate in parts of the input space where the training data density is high. However, local linear regression will not generalise well to parts of the input space for which training data is sparse.

Neural network models take time to construct but in parts of the input space where data is sparse tend to generalise better than local linear regression. It is often quite hard to get a neural network to train down to a very small *Gamma statistic* (say 10^{-6} or 10^{-7} which can easily happen with zero noise dynamical system time series), i.e. it may take several attempts, each of which takes a long time. However, neural networks can make predictions at blinding speeds compared with local linear regression based algorithms, so for some applications it is well worth the time and effort to construct a neural model.

3.2 Local linear regression

To make a prediction for a given query point in input space **local linear regression** (LLR) first finds the *k* nearest neighbours of the query point from the given data set (where the number *k* is supplied by the user) and then builds a linear model using these *k* data points. Finally the model is applied to the query point thus producing a predicted output. Because of the way *winGamma* analyses the data to compute the *Gamma statistic* the *k* nearest neighbours of any point in input space can be found very rapidly. Consequently local linear regression using the *k* nearest neighbours (in the training data) of the query point can be accomplished quickly. Thus local linear regression is a very fast and capable predictive tool.

LLR is most effective in regions of the input space with a high density of data points. If data points are few and far between in the vicinity of the query point then LLR will not be very effective if the underlying function we are trying to model is truly non-linear.

It may seem odd that although *winGamma* is all about constructing *smooth* models the global function produced by patching together many LLR predictions in general is *not even continuous*! However, as the number of data points increases, the global function produced by LLR will converge rapidly to the unknown function generating the data, provided this is itself a smooth function. You can easily see this by using the *winGamma* **WhatIf** facility on LLR models built from increasing sized data sets, for example on the data from *Sin500.asc*.

LLR can produce very accurate predictions in regions of *high* data density in input space, but it is liable to produce unreliable results for non-linear functions in regions of low data density. In other words LLR does not generalise well but is a very good interpolative tool if we have large amounts of data.

There are three user settable parameters in LLR: the number of near neighbours, whether or not to include a constant term in the linear model, and a threshold value for filtering the local eigenvectors.

The choice of the number of near neighbours *k* in LLR is quite critical. If the noise level on the output (i.e. the asymptotic *Gamma statistic*) is low then some small multiple of the number of inputs should suffice. If the noise level on the output is high then *k* needs to be larger to obtain better noise cancellation. Unfortunately if the unknown function *f* we seek to model is highly non-linear (has regions of high curvature) then, unless we have a very large amount of data, setting *k* large may mean that in the region of these *k* points the assumption that the unknown function can be locally approximated by a linear model may be false. In this case the resulting predictions would be inaccurate. We have not yet developed rules of thumb for this situation but in practice it is not a major problem to optimise the choice of *k* using a test set.

There is also an option whether or not to include a constant term in the locally linear model. In general it is better to include this term.

The final user settable parameter is equivalent to a local principal components threshold filter on the eigenvectors of the local linear model. We are tying to predict along the tangent plane of the local flow and eigenvectors corresponding to relatively small eigenvalues probably represent noise and lie outside the tangent plane. The threshold decides which eigenvectors we should ignore. Setting it low or zero will essentially include all eigenvectors in the local model, the default is around 10^{-6}. Raising this threshold will filter out more and more eigenvectors. For noisy time series one often finds that 0.001 gives quite good results. Again the best approach is to experiment on a test set.

3.3 Dynamic local linear regression

This option is mainly designed for time series analysis. It is basically identical to LLR with the additional feature that as new data is seen for the first time it is incorporated into the model. You can see the effect of this by starting the model with very little training data and running a test on a large amount of data. As new test data is encountered (but *after* the attempt at prediction of course) dynamic LLR will make steadily better predictions. This is interesting to observe but is not actually the best way to use dynamic LLR. It is better to start with a reasonable training set size because then the initial *kd*-tree (a data structure used extensively by *winGamma*) will be more balanced and query times will be reduced.

Under the circumstance it is not surprising that if the same test data is presented to the model a *second* time the *MSError* will reduce dramatically.

3.3 Two layer back propagation

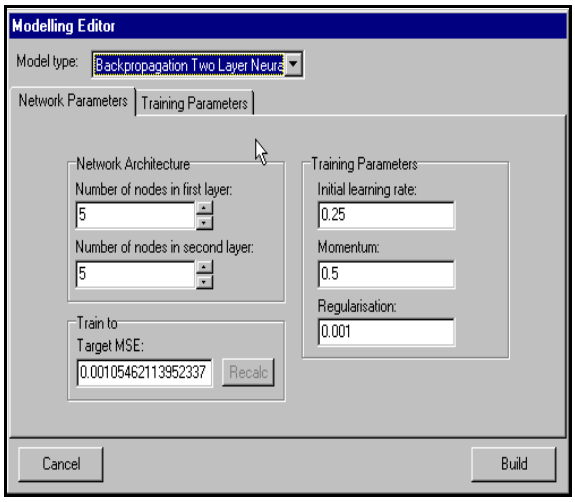


Figure 3-1 The first set-up menu for two layer backpropagation.

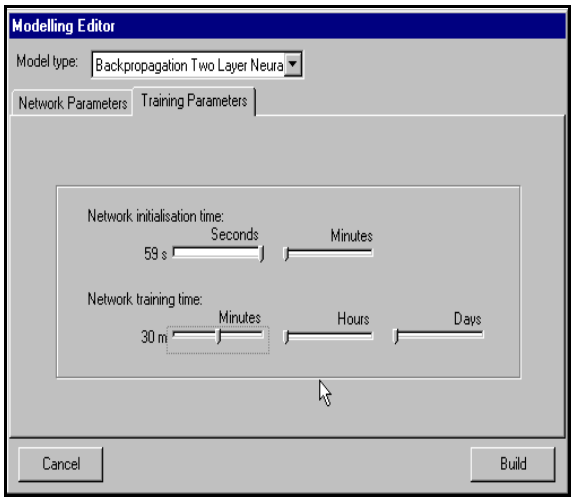


Figure 3-2 The second set-up menu for two layer backpropagation.

This option uses the standard backpropagation algorithm to produce a two-layer feedforward neural network.

With all the neural network training algorithms one should note the option to recalculate the target *MSError*. This is useful in the event that the partition of the data for training and testing has been altered. Clicking **'Recalc'** will cause a new *Gamma statistic* for that part of the data selected as training data to be calculated, and hence set a new target *MSError* for training.

Alternatively the user can set any target *MSError*. However, if the target *MSError* is much less than the *Gamma statistic* on the training data then (i) the network may end up being `overtrained' resulting in poor predictions, or (ii) the training algorithm may never be able to reach the (possibly) unrealistic target *MSError*.

User settable options:

For each of the neural training algorithms we shall need to specify the number of hidden units. Thus each neural network option needs

The *number of units* in the two hidden layers (default 5, 5)

The number of units required to achieve a good model will depend on the complexity of the unknown function we are trying to approximate. Unfortunately here there are few general rules to guide us. One useful guide is that if the *Gradient* value returned by the Gamma test is large then the unknown function has regions of high curvature and we shall require more hidden units to approximate it

accurately. The best approach is to try to train using relatively few units (the defaults are set quite low) and if training fails to converge to the target *MSError* progressively increase the numbers of hidden units in the two layers.

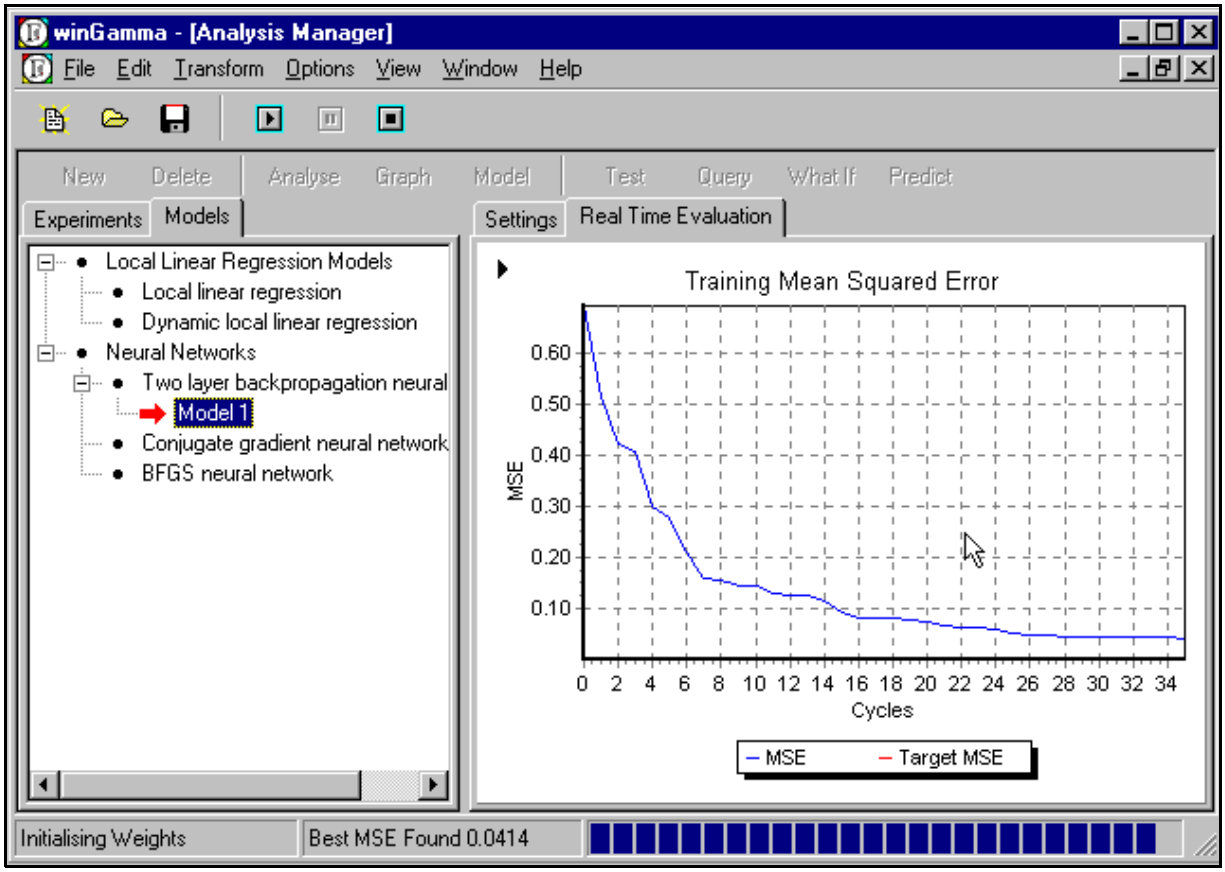


Figure 3-3 The **Analysis Manager** during backpropagation training.

Two layer backpropagation also requires:

> The *initial learning rate* (must be positive). This controls the initial step size in weight adjustment.
>
> *Momentum constant* (must be positive). This controls the extent to which the size and direction of the current step in weight space is influenced by the size and direction of the previous step. Setting this parameter to zero means there is no momentum term in the weight adjustment at each step.
>
> *Regularisation constant* (must be positive). This limits the size of weights. A zero here corresponds to no restriction on weight magnitude.

These options are configured using the set-up menu shown in Figure 3.1. There is a second tab which allows the user to specify the maximum amount of time to spend on trying to attain the *MSError* goal. This is shown in Figure 3.2.

Handy Tip. Backpropagation along with most other processes in *winGamma* can be paused/resumed or terminated using the buttons on the top level menu. Terminating an operation does not necessarily loose everything. Any results already calculated will be displayed and in the case of neural net training the model created so far will be retained.

Figure 3.3 shows the **Analysis Manager** during backpropagation training. Note that the graphical window can be zoomed and moved using the left and right mouse buttons.

Because the number of layers, the number of hidden units, and the slope of the sigmoidal are fixed, limiting the size of the weights also limits the magnitudes of the partial derivatives of the neural network as a function of its inputs. Thus if the unknown function to be approximated has regions of high curvature the training algorithm with regularisation may find it difficult to obtain the desired approximation. We can get some idea if this is likely to be the case by examining if the *Gradient* returned by the Gamma test is unusually large.

These parameters may be left at default until fine tuning of the model is required.

3.4 Conjugate gradient descent

This a variation and improvement on two-layer vanilla backpropagation, it is generally more effective but requires more memory. The procedures for set up are very similar.

3.5 BFGS neural network

Probably the fastest and most efficient neural network training algorithm offered by *winGamma* is a modified version of the Broyden-Fletcher-Goldfarb-Shanno learning algorithm. This algorithm uses second differences and is sometimes degraded by very noisy data, but generally it is probably best to use this option first when trying to produce a neural model.

3.6 Example model construction and testing for *solar.asc*

We return to the example solar panel data we analysed in 2.12.3. Using the first 8400 data and scaling we initially build a local linear regression model with $k = 20$. We then test this model on the remaining points in the data file.

3.6.1 Building and testing a LLR model

We initially build a LLR model using the first 8400 points (the results of our earlier analysis suggest that slightly more points are required for a really good model).

1. Load *solar.csv*. Do not normalise and use all the data for analysis. Execute a simple Gamma test. these steps are described in 1.3 and 2.12.3.

Handy Tip. *winGamma* requires that at least a simple Gamma test Experiment be conducted before any attempt to build a LLR model (a kd-tree is required).

2. After the Gamma test results appear click on **`Model'** in the **Analysis Manager**.

3. Select the training set as 1-8400.

4. In the **Modelling Editor** leave the model type set at *Local Linear Regression*,set the number of nearest neighbours at 20, leave the *Add constant* box checked, and leave the *Define local flow threshold* option at 1E-6. Then click on **`Build'**.

5. When the **Test**, **Query**, **WhatIf** and **Predict** buttons become active in the **Analysis Manager** the model is built and ready to be used. Click on **`Test'**.

6. In the **Select proportion of data set for model testing** window set the range of test data to 8400-10578 as shown in Figure 3.4.

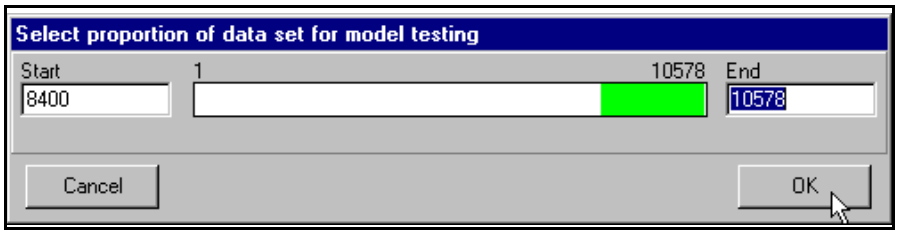


Figure 3-4 Selecting a proportion of the data for testing.

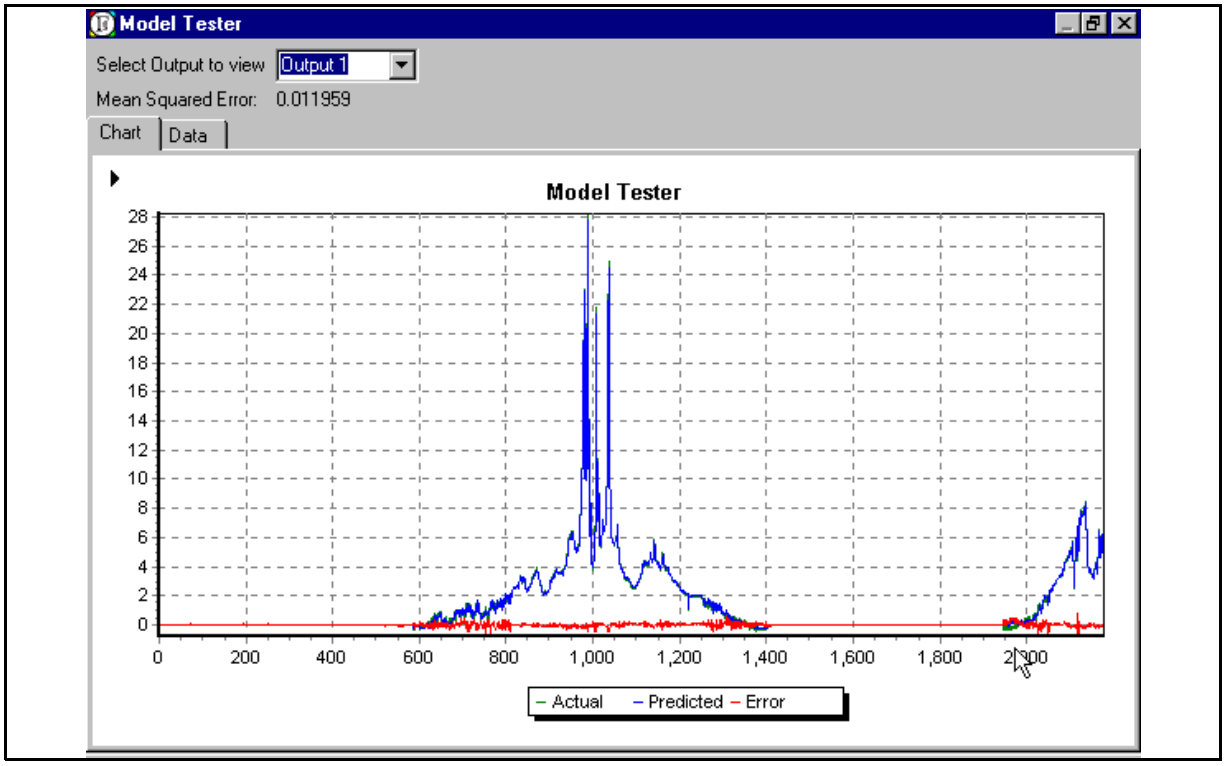


Figure 3-5 Result of LLR test for *solar.csv*.

We have used the points from 8400 - 10578 for testing. A sequence of such experiments for the number of near neighbours k = 10, 15, 20, 25 shows that $k \approx 20$ seems to give the smallest *MSError* ≈ 0.011959 on the test set. This is somewhat better than the Gamma test result led us to believe but very much in the same ball park. The results of the test with $k = 20$ are shown in Figure 3.5. Here we can see that the agreement between the predicted (blue trace) and the actual test data (green trace) is very close. The red trace indicates the error.

3.6.1.1 Using the **WhatIf** and **Query** options on the LLR model

The **WhatIf** option allows us to see what happens if we set values for all of the inputs except one and vary the remaining input over some range. This is a very useful tool in a variety of contexts.

For example, in a sales and marketing campaign we may be able to answer the question "If I spend X on advertising on TV and Y on advertising in newspapers how will the sales of the soft drink vary with the mean day time temperature"?

Similarly the **Query** option allows a particular selection of all inputs to be queried. The use of **Predict** is discussed in section 3.8.

Having analysed the data, built and tested a model, we can now ask some interesting questions regarding the *solar.csv* data. For example, using the **WhatIf** options we can answer the question:

- How does the power output vary when the temperature is fixed at 7 degrees and the Irradiance varies from 0 to 30?

The answer is given in Figure 3.6 As expected at a fixed temperature the power output is almost linear with the Irradiance.

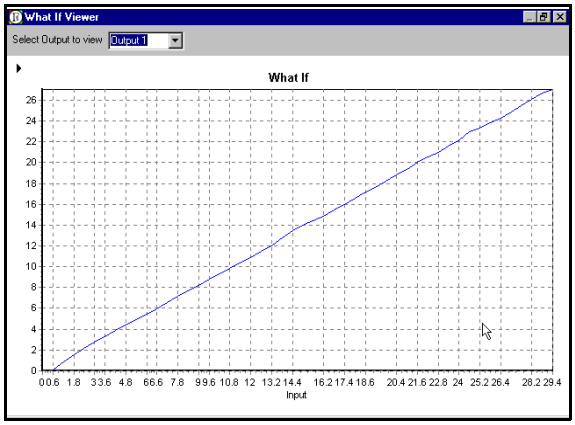


Figure 3-6 The variation of output power as Irradiance varies from 0 to 30 and temperature is 7 degrees.

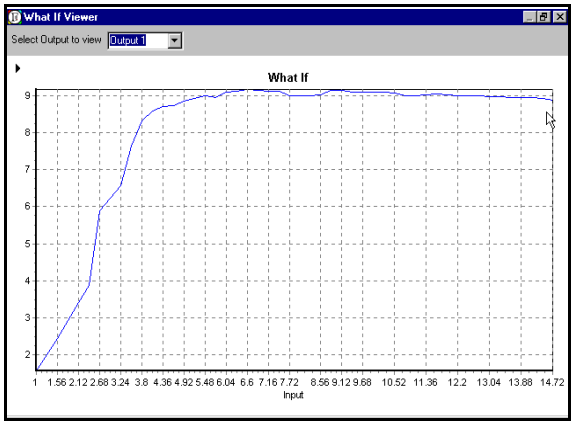


Figure 3-7 The variation of output power as Temperature varies from 1 to 15 and Irradiance is 10.

Similarly we can ask

- How does the power output vary when the Irradiance is fixed at 10 and the Temperature from 1 to 15?

The answer is given in Figure 3.7. Here the result is somewhat different. One interesting feature is the slight rolloff of power output with increasing temperature. This is a real effect and is a consequence of the physics of solar cells.

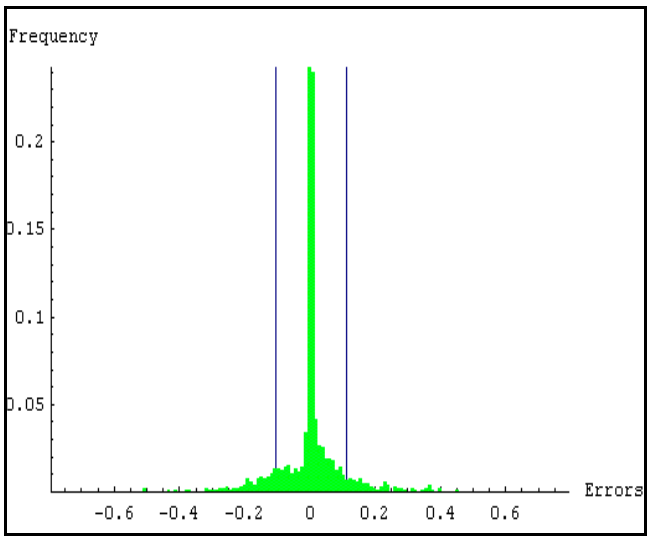


Figure 3-8 An error histogram for the LLR test.

3.6.1.2 A histogram of prediction errors for LLR model

If we save the data produced by the results of the LLR test then we can examine an error histogram for the predictions. This is shown in Figure 3.8 The vertical gridlines are one standard deviation either side of the mean, which is close to zero. This is the final test of our model.

3.6.2 Building and testing a neural model

We can now repeat the model building process using a neural model.

3.6.3 Visualising the data.

For a 2-Input/1-Output data set we can visualise the model as a 2-dimensional surface and using suitable software plot this surface directly from the data. Of course in higher dimensional spaces such graphical realisations are not possible. Moreover, if the data is very noisy such a surface will be very jagged and not much use as a model.

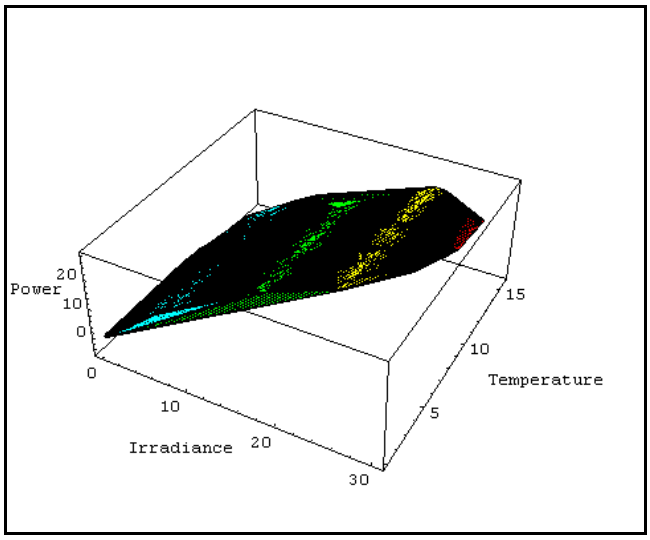


Figure 3-9 A topographic plot of the *solar.csv* data.

Nevertheless now that we have finished studying *solar.csv* it would be interesting to see what the surface constructed from the data actually looks like. Figure 3.9. This is a topographic plot of the surface in which lower power outputs are blue and higher power outputs are red. We could regard our **WhatIf** graphs as cross sections (using the model) of a surface which is very similar to this plot.

3.7 Example model construction and testing for *DH(34)5000.asc*

In Chapter II we analysed this data and concluded that it represented a low or zero noise time series for which the current sample could be predicted accurately on the basis of between 4 and 7 previous samples. The mask 1101 was identified as a good mask.

1. Load the file *DH(34)5000.asc*.

2. In the Time Series options specify 4 inputs and 50 outputs. This gives 4952 samples for analysis.

We are going to build a local linear regression model and this needs a kd-tree so it is necessary first run a Gamma test.

3. In the **Experiments tab** highlight *Gamma tes*t. Leave the number of near neighbours set at the default of 10. In the *Mask tab* enter the mask as 1101. Now click **`Execute'**.

Result. We specified 50 outputs and so are trying to predict a maximum of 50 steps into the future. If we just look at the one step prediction then the result for the first output is a *Gamma statistic* of -6.089E-5 with a *SE* of 4.3118E-5.

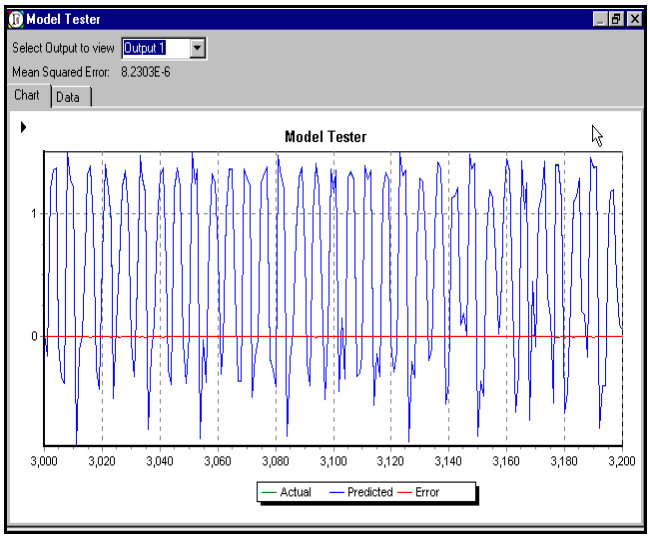


Figure 3-10 A test of the LLR model on the data set *DH(34)5000.asc* (blue predicted, green actual, red error).

4 For output 1 select **`Model'**. Our previous experiments suggest that about 3000 data points are need to obtain a good model (a fact confirmed by a *M*-test for this embedding - which curiously gives small negative results increasing towards zero!) Select a local linear regression model with 10 nearest neighbours and set the Mask to 1101.

Results The *MSError* over the test set 3000 - The *MSError* of the test set 3000-3200 is 8.2303E-6. The graph of predictions, actual values and errors is shown in Figure 3.10.

3.7.1 How the prediction quality degrades into the future.

Results. From the result of Step 3 in last experiment we actually got 50 output *Gamma statistic* results. The graph of Gamma against the number of steps ahead is shown in Figure 3.11. Here we see an exponential rise in the error of prediction, which is typical of a chaotic process.

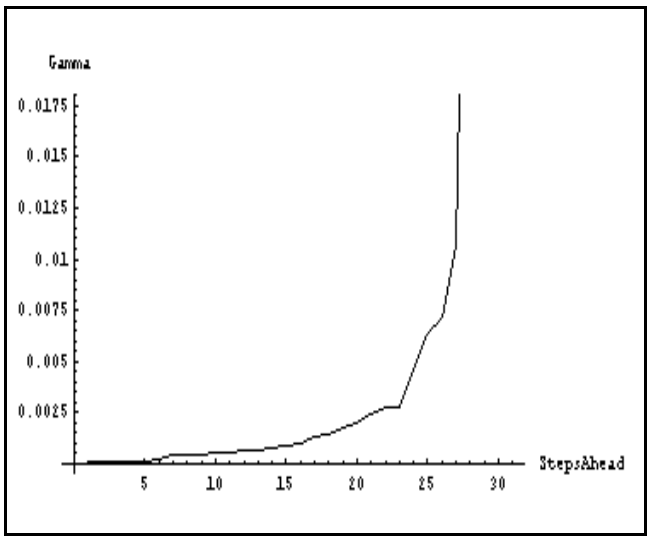


Figure 3-11 How the Gamma statistic varies against the number of steps ahead for *DH(34)5000.asc*.

We conclude that the Time Series data is of a low/zero noise smooth process which is chaotic and that we can make accurate short term predictions but that long term prediction becomes exponentially more difficult.

3.8 Using a prediction file

Building and testing models when you know the outputs for a corresponding set of inputs is quite interesting but it is purely an academic exercise. Sooner or later you will want to make predictions that matter and where the outcomes are not known. Perhaps from some large quantity of input data.

To accomplish this it is first necessary to have a `prediction' file, i.e. the input data is placed in a file without the corresponding inputs.

3.8.1 Using a prediction file on *Input/Output* data.

Load the data file EXAMPLE NEEDED HERE

3.8.2 Using a prediction file on *Time Series* data.

Load the data file EXAMPLE NEEDED HERE

3.9 Using the neural networks outside of *winGamma*

If the neural models are used outside of *winGamma* (i.e. in other software) it is necessary to know some technical details of the implementation.

3.9.1 The activation function and the sigmoidal.

The activation function used by the neural networks is

$$act(x) = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} x_j$$

where w_{ij} is the weight of the connection *from* unit j *to* unit i and x_j is the output of unit j.

The sigmoidal used by each neural node as an output function is

$$sigmoidal(act) = scaleFactor.\left(\frac{2}{1 + e^{-act/Temperature}} - 1 \right)$$

where *act* is the activation (weighted sum of inputs), *scaleFactor* = 1.5, and *Temperature* = 0.8333

To speed up neural computations this function is implemented in *winGamma* as a fine grained look-up table, whereas for feedforward computations when the weights are loaded into other software it can be implemented directly as a function. This may cause very small differences in neural output calculations using the same weights outside *winGamma*.

3.9.2 NetReader.

NetReader.nb is a *Mathematica* program supplied with *winGamma* which can read the neural network weights saved from *winGamma* and implement the neural network for feedforward testing. Which type of network training was used in the creation of the weights is automatically identified from the weights file.

3.9.3 Exporting and using Neural network models in *Excel*.

After *winGamma* has built a neural network model it may be exported as an *Excel Macro* and used directly in *Excel* (This facility is not currently available for LLR models.). We illustrate this process using an example.

Step 1. Build a model.

In *winGamma* load data file Sun280.asc this is a single time series file.
Transform the data to 3 inputs 1 output
Export transformed data as *test.csv*
Perform Gamma analysis.
Train neural network model on the transformed data

Step 2. Export the model

Right click on '**Model**'
Select '**Export'**
Choose '**Save as type**'

Set to '**Excel Macro**' (*.mac)
Enter directory and file name
Export model

Step 3. Setting up the data and model in Excel.

Start up *Excel*
Load data from *test.csv*
Save the file as an *Excel* Workbook
Right click on worksheet tab '**test**' at the bottom left corner.
Select '**Insert**'
Select '**MS Excel 4.0 Macro**'. Hit OK
This now opens a macro sheet
Load '**test.mac**' into *Notepad*, select all text, and copy.
Paste text into macro sheet in *Excel* in cell A1
Highlight column A
Do **Insert\Name\Define**
In the macro box set to '**Function**'
Set name to '*model*'. Hit OK
Now when in the macro sheet with column A highlighted you should see 'model' in the top left name box.
Switch back (using the tabs at the bottom) to the '**test**' worksheet.
Enter heading 'model' in cell F1
In cell F2 type "= model(A2:C2)" (no quotes) and press '**Return**'
You should now see the model output value in cell F2 as compared to the actual output in cell D2.
Select cell F2 and copy
Highlight the range of cells from F3 to F278 and paste.
You should now have all the model predicted values for each row.

APPENDIX I General Information

Shipping list

1. Compact disc
2. This manual.
3. The gamma test and how to use it: a practitioners guide.

Hardware requirements

This software is PC based and normal minimum requirements are:

Pentium processor 133 MHz or preferably faster.

RAM 32-64 Mbytes. The amount of memory you will need to run *winGamma* is not really constrained by the program so much as the size of the data sets that you wish to analyse. With the possible exception of the neural network training algorithms the theoretical average case computation times of the main algorithms in *winGamma* scale like O(Mlog$M$), where M is the number of rows in the data file. However, under some conditions some algorithms in *winGamma* may require quite a lot of memory to achieve the theoretical scaling.

An example is **Increasing Near Neighbours** when pmax is large. Suppose we consider *solar.csv* sith 10578 rows of three numbers each and set pmax = 100. This demand will require approximately 0.25 Mgbytes for the data, 0.25 Mgbytes for the kd-tree but more than 4 Mgbytes for the 10^6 numbers which constitute the list of 100 nearest neighbours for each of the 10578 input vectors in the data file. To perform a Gamma test each of the near neighbour indices must be instantly available and they could be anywhere in the range 1-10578. If the system has less than 4Mgbytes of available RAM then it will have to keep paging data in and out from the hard disk. This will dramatically slow the algorithm and may in fact render the entire computation infeasible. If you observe a large amount of continuous paging disk activity then (a) Close down all other applications (b) Consider if it is feasible to perform the analysis on a subset of the data. If the problems continue you need more RAM. In most cases 64 Mgbytes is sufficient for any reasonable data set.

At least 50 Mbytes remaining hard disk space.

Operating system: Windows 95 or 98, or Windows NT4.0 were the original development targets but we have so far observed no problems with later versions of Windows operating systems. Licenses for a script file driven UNIX version of the Gamma Test software may be available by special arrangement.

Installation

Beta release: At present simply copy all files in the *winGamma* directory into a convenient directory on your hard disk. If you experience problems getting the help system to work you may have an older version of Explorer. To update run the file hhupd.exe.

V 1 release: Place CD in drive. Follow install instructions from screen.

List of files and directory structure after installation

Program and associated files
Directory of C:\WinGamma

11/20/98	12:28p	<DIR>	Data
10/30/98	02:37p	<DIR>	TestFiles
02/09/98	02:00a	29,952	BORLNDMM.DLL
02/09/98	02:00a	996,872	CP3240MT.DLL
11/05/98	06:28p	471,840	hhupd.exe (Run to update HTML if problems with help occur)
10/24/98	04:01a	420,864	Tee4C.bpl
02/09/98	02:00a	1,455,736	vcl35.bpl
02/22/99	02:31p	107,677	winGamma.chm
03/02/99	04:08p	1,228,288	winGamma.exe
03/02/99	04:20p	968,704	winGammaBaseComponents.bpl
02/17/99	03:32p	35,328	winGammaComponents.bpl

Real data files

<DIR>	Data			
	02/04/99	02:35p	<DIR>	Solar
	11/21/98	11:51a	430,515	Solar.csv
	12/11/98	03:37p	<DIR>	Sunspot
	03/19/98	07:52p	2,240	Sun280.asc
	04/20/98	02:15p	24,543	SunPairs.asc

Artificially generated test data files

<DIR>	TestFiles			
	01/04/99	02:18p	<DIR>	Noise
	04/16/98	12:21p	50,183	Ran500.asc
	03/27/98	03:04p	20,539	Sin500.asc
	03/02/99	12:43p	<DIR>	NoNoise
	09/15/98	04:58p	1,958	Hen100.asc
	09/15/98	04:59p	9,830	Hen500.asc
	09/15/98	05:00p	19,666	Hen1000.asc
	10/29/98	01:26p	983,909	Hen50000.asc
	04/22/98	12:51p	9,617	MGls500.asc

10/22/98	05:09p	96,097	MGls5000.asc
02/24/99	02:44p	205,286	ModSin5000.asc
12/09/98	03:02p	98,966	DH(34)5000.asc

Mathematica™ 3.01 files

12/19/98	04:25p	2,317	DataAnaly.m
12/19/98	04:25p	2,812,869	DataAnaly.nb
03/02/99	07:13p	8,831	DataGen.m
03/02/99	07:13p	946,827	DataGen.nb
10/01/98	2:07p	38,062	mathlinkGamma.nb
01/28/99	04:36p	253,440	GammaTestProject.exe
10/29/98	4:18p	50,773	NetReader.nb

Problems reported

Graphics files saved are in billions of colours and cause `out of memory' errors when attempts are made to load them into some software including WPCorelV8 and Graphics Workshop.

APPENDIX II Data file formats

All data files are in plain ASCII and have the file name suffix *.asc. Data files may be created using *Excel*™ as *.csv files and imported into *winGamma*. Data files for *winGamma* are in two basic formats.

- **Times series data**.

Example: a single time Series.

0.0262
0.0575
0.0837
0.1203
0.1883
0.3033
0.1517

etc.

Each number followed by a carriage return/linefeed.

Example: multiple time Series. it is the responsibility of the user to prepare the data so that fields referring to the same time are on the correct line (most recent data is last).

0.0262 1000.26
0.0575 1031.78
0.0837 1037.86
0.1203 1038.567
0.1883 1040.810
0.3033 1100.721
0.1517 1027.851

Each number followed by one or more spaces. The last number on a line followed by a carriage return/linefeed. There must be the same number of data fields on each row.

- **Input/Output data**.

Example: a 4-input/1-output file.

0.36368593157164 0.3304959949667 -0.21811098544356 -0.20933961443087, -0.0220710621963
-0.00591105325917 -0.9085902611647 0.19548859472561 -0.34015487882487, -0.0064356217878
0.86221883819100 -0.5929180658183 -0.36843151702318 -0.89277930056707, 0.6617039028787
0.59877814813365 0.9562473549851 0.25582643936911 0.97996127233012, 0.4810764303063
0.13712162278232 0.9035299186427 0.29916358157799 -0.22014139763247, 0.7734356912106
-0.42696607632396 -0.4827254329784 0.98919821679839 -0.20449324659299, 0.5789449769352

etc.

Each number followed by one or more spaces. The end of the input vector is signified by a comma. What follows the comma is one or more outputs separated by one or more spaces. The last number on a line should be followed by a carriage return/linefeed. There must be the same number of data fields before and after the comma on each row.

- **Prediction file data**.

APPENDIX III Using the *Mathematica* 4.0 files

A number of *Mathematica* files for data generation/manipulation, data analysis, and model testing are supplied with *winGamma*. There is also a C-code executable *MathLink* file which can be used to execute the Gamma test from within *Mathematica*. To use these files you will need to have *Mathematica* installed and be familiar with *Mathematica* notebooks.

At a later stage it is hoped to supply equivalent files in *Matlab*.

DataGen.nb (Data Generator)

This file enables the creation of *Input/Output* files and *Time Series* files of data with our without added noise. In includes a large number of examples and shows how every test file used in this manual was created.

DataAnal.nb (Data Analyser)

This file is useful for producing graphics such as histograms and performing various types of supplementary data analysis.

GammaTestProject.exe

This is a C-code executable which communicates with *Mathematica* via *MathLink* and enables a variety of Gamma test computations to be called directly from *Mathematica*. It cannot be executed as a standalone program.

mathlinkGamma.nb

This shows how to load and use the file *GammaTestProject.exe* and gives examples of each function that can be called.

NetReader.nb

This notebook can read in any neural network created and saved by *winGamma*. The program identifies the network type and can then run the network. There may be very small differences in the results owing to the fact that this notebook uses a pure form of the sigmoidal function whereas *winGamma* uses a fine grained discrete lookup table for speed in training.

APPENDIX IV Generating test files

Generating your own data files.

Data files may be generated using a wide variety of software tools. All data files used by *winGamma* are in plain ASCII format. One convenient method of generating data files is to use *Excel* to manipulate your data into the required rows and columns and then save the data in *.csv format. Another convenient method for creating data files is to use *Mathematica*. *winGamma* is supplied along with a number of useful *Mathematica* programs for generating, manipulating and saving data in the correct formats. These are described in Appendix III.

Data is generally divided into four types: **analysis**, **training**, **testing**, and **prediction**. Prediction files are different in that they contain no output values but otherwise use the same formatting conventions. We use prediction files when we genuinely do not know the corresponding output values and want to generate predictions. For the prediction file the output fields are empty because it is assumed that the outputs are unknown. The use of prediction files is discussed in section 3.10.

In general data files can be divided into two main categories: **input/output** files and **time series** files.

Creating data files using *Excel*

If data is prepared in a spreadsheet it can be exported to *winGamma* in the *.csv format. Make sure that the numbers exported are in pure decimal format. At present *winGamma* may read numbers in the xEy format incorrectly.

When a *.csv file is loaded the user will be automatically prompted to nominate particular columns as inputs or outputs by selecting with the mouse or using up/down cursor keys and the Enter (or Return) key. The mouse may also be used to select then double clicking will change an input to an output an vice versa.

APPENDIX V Definitions

Model. A *smooth data model* is a differentiable function from inputs $\boldsymbol{x} = (x_1, ..., x_m)$ to each output y. It is assumed that the data can be represented by an unknown model f so that

$$y = f(x_1, ..., x_m) + r$$

where r is a stochastic variable which represents noise.

Gamma test. An algorithm to estimate the variance of the noise Var(r) associated with a particular output. Not to be confused with the variance of the output..

Gamma statistic. Often referred to as a `Gamma value'. It is the vertical intercept of the regression line plot and represents our best estimate for Var(r).

Embedding. A selection of past values of a time series used to predict the current value.

Mean squared error (*MSError*). If $y(i)$ $(1 \le i \le M)$ is a set of values of an output and $y^*(i)$ is a set of predictions for y(i) then the *MSError* of the predictions is given by

$$MSError = \frac{1}{M} \sum_{i=1}^{M} (y^*(i) - y(i))^2$$

Standard Error (*SE*) This is the standard error about a regression line and is calculated as

$$SE(\Gamma) = \sqrt{\frac{1}{n-2} \sum_{i=1}^{pmax} (\Gamma(i) - \bar{\Gamma})^2}$$

where $\Gamma(i)$ is the i*th* Gamma regression point value and $\bar{\Gamma}$ is their mean.

Over-training describes the effect when we attempt to produce a model by exactly following the training data. Consider the effect of trying to produce a model by drawing a line through every point in the noisy sine data in Figure 1.8. It would look nothing like a sine curve and if we asked this model to predict y for a particular value of x we should little faith in the prediction. One of the main advantages of *winGamma* is that it gives us the necessary information to prevent over-training *before* we begin to build a smooth model such as a neural network.

GA Fitness. In order to better control the GA search it is useful to know how the GA fitness is calculated. The overall fitness of a mask is composed of three parts, corresponding to the fitness due to the intercept (i.e. the actually *Gamma statistic*) because mainly we want masks with small Gamma,

the fitness due to the *Gradient* because if we have enough data to estimate the *Gradient* accurately it might in model construction to choose a mask with a low *Gradient* which will correspond to a simpler model, and the fitness due to the number of 1's in the mask because shorter masks also mean simpler models. The contribution of each of these terms is controlled by three weights $W_{intercept}$, $W_{gradient}$, and W_{length} according to the formula

$$\begin{aligned} fitness(mask) = {} & W_{intercept}.interceptFitness(mask) + \\ & W_{gradient}.gradientFitness(mask) + \\ & W_{length}lengthFitness(mask) \end{aligned}$$

The component fitness calculations are described below, where *Vratio*(*mask*) and *Gradient*(*mask*) return the *Vratio* and *Gradient* as calculated by the Gamma test on the data set for *mask*, *outputrange* is the range of the output and |.| denotes absolute value.

$$\begin{aligned} interceptFitness(mask) &= 1 - (1 - 10.Vratio(mask))^{-1}, \ if Vratio(mask) < 0 \\ &= 2 - 2(1 + Vratio(mask))^{-1}, \ otherwise \end{aligned}$$

$$gradientFitness(mask) = 1 - (1 + |gradient(mask)|/outputRange)^{-1}$$

$$lengthFitness(mask) = \frac{numofones(mask)}{length(mask)}$$

APPENDIX VI Frequently asked questions

Why is the Gamma statistic sometimes negative?

Sometimes the Standard Error (the error obtained from the (δ, γ) regression which is always stated when a Gamma result is obtained) is large enough to account for a negative intercept by the regression line. This is most likely to occur when the true asymptotic *Gamma statistic* is close to zero. It can also happen when the data fails to fulfill the basic requirement that inputs and outputs are drawn from a continuous range. If many inputs are categorical it is also possible to get a negative Gamma statistic.

How should I choose the right number of inputs for a Time Series?

Initially set the number of inputs large (but reasonable in the context of the data). Then do an **'Increasing embedding'**. This will compute successive Gamma statistics based on one input (the historically most recent sample of the time series (rightmost on the mask), then on two inputs (the two most recent samples) and so on up to the maximum number of inputs you have selected.

The minimum *Gamma statisti*c obtained will determine an upper bound for the maximum number of inputs it is useful to consider.

An optimum for the number of near neighbours used in the Gamma test should now be obtained. Then the maximum number of inputs can be checked again using that number of near neighbours in the Gamma test. (If the maximum number of inputs changes then the optimum number of near neighbours should be checked again). Finally using the best maximal number of inputs a check for the best embedding can be run, this may cause some inputs to be discarded.

How should I choose a method for establishing an optimal embedding (mask)?

The best method for choosing a mask on the inputs is 'Full embedding'. The problems come with this method when the run times required become too long. Runtime is a function of the input dimensionality (the number of inputs, *m*), the number of nearest neighbours (pmax) and the length of the data (*M*). If run times are just too long then the Genetic Algorithm (GA) can be used with 'Hill climbing' and a 'Sequential embedding' embedding to do a small search around the candidates offered by the GA.

How should I choose the optimal number of near neighbours (pmax) in the Gamma test?

See section 2.4 of the manual.

How should I choose the optimal number of nearest neighbours (*k*) in Local linear regression?

By experiment with a test set.

What is "the best Gamma" and what does it mean?

The `best Gamma' in the context of a Gamma test is the closest approximation to the asymptotic *Gamma statistic*, which should approach the true noise variance.

The `best Gamma' in the context where we have a number of Gamma estimates for different selections of inputs (essentially different models), assuming these estimates are accurate, is the *Gamma statistic* closest to zero - because that suggests the model which should have smallest *MSError* when predicting outputs from inputs not used in the model construction process.

Note that if the true noise variance is *actually zero* (and the data is of arbitrarily high precision) there is no limit to how accurately we can model the unknown function, provided only that we have more and more data.

In most real life situations there is a *positive noise variance* remaining even after optimising the selection of inputs (because real measurements are subject to error) and there is *no point* in building more and more accurate models (for example by using the noise cancelling features of local linear regression) because the predictions of the model will never agree with our measured data unless the measurement error is decreased (for example).

An exception to this might be if are trying to get some idea about an underlying *theoretical* model and *winGamma* can help in this respect but determining a theoretical model (as opposed to an accurate numerical model) lies outside the competence of *winGamma*.

How should I choose between a local linear regression (LLR) method and a neural net method of model building?

Nets take a long time to train but may generalise better than LLR in regions of the input space where data is sparse. A high *Gamma statistic* on the training data may make neural network training even more difficult. If data is densely distributed over the input space then LLR may be a better choice in this situation.

The *particular applicatio*n also has an influence on which may be the best modelling tool to select. For example, to learn new data it may be necessary to retrain a neural network from scratch which is time consuming, whereas dynamic LLR can easily accommodate new training data.

Local linear regression models are very fast to build, but take relatively longer to query because a *kd*-tree is used to find the near neighbours of the query point. If the final target application is a real time system neural networks offer the advantage that they can be implemented in hardware.

How should I choose between local linear regression and dynamic local linear regression?

For a model to adapt it must be dynamic. Every data row (vector) "seen" by a dynamic LLR model will be added to the model, but of course eventually the model becomes memory hungry and starts to slow down. At this point the model will have to be pruned. If the phenomena that you are trying

to model is likely to be fixed then a static model is best. If the underlying dynamics themselves might be changing (e.g. the stock market) then a dynamic model is more sensible.

How should I choose between the Backpropagation, Conjugate Gradient Descent and BFGS neural net algorithms?

Backpropagation is the original feedforward neural network training algorithm. It is reasonably effective on simple problems but only makes use of the first differentials of the error surface in weight space. Therefore backpropagation can take longer to train than other more sophisticated neural training algorithms and may fail to converge to the target *MSError* derived by the Gamma test at all. But compared to more recent algorithms backpropagation is inexpensive on memory.

CGD offer some improvements over BP at the cost of extra memory.

BFGS uses the second differences of the error surface in weight space which in most cases gives faster convergence at the expensive of a more complicated algorithm and more memory.

What do all the fields associated with a Gamma Result mean?

See section 1.3.1 of this manual.

What does a high gradient suggest?

If there is enough data to give a stable *Gradient* asymptote then a high *Gradient* (computed values on artificial test sets can come out as high as 20,000) suggests a complicated unknown function with on average regions of high curvature.

Why is the *Vratio* useful?

It provides a standardised estimate of the noise which is independent of the output variable range.

What is the use of the Standard Error?

It tells us how reliable the *Gamma statistic* is as an estimate of the variance of the noise on the output.

What file formats are permitted for data to be analysed by *winGamma*?

See Appendix II.

How much data should I use for training?

If the *Gamma statistic* is asymptoting to zero you can use as much data as is practical and models with MSErrors of order 10^{-7} are quite feasible.

If the *Gamma statistic* is asymptoting to a positive value a good rule of thumb is to use as much data as will give a standard deviation (the square root of the variance) of the Gamma values about the asymptote, of around 10% of the asymptote value on the last 10% of the data.

When should I use external files of data for testing?

When the analysis data set doesn't include the test data or further tests need to be made on a model.

When should I use the moving average option?

Usually when you have plenty of data and want to determine if the number of data samples used to estimate the Gamma statistic gives a stable value over a range of different sample sets of the same size.

This test is also useful to investigate if the underlying dynamics is itself varying.

When should I use the differential option?

It may improve the *MSError* for difficult time series.

Which input is the differential (or moving average) input?

When these options are activated the new data column is placed in the highest numbered positions with differential first and moving average last. This can be confirmed by placing the cursor over the vertical column and dragging it wider thus revealing the applicable legend.

INDEX