Gamma test analysis tools for non-linear time series

by

Samuel Edward Kemp
Department of Computing & Mathematical Sciences
Faculty of Advanced Technology
University of Glamorgan
Wales
UK

A thesis submitted in partial fulfilment of the requirement
for the degree of Doctor of Philosophy

August 2006

Certificate of Research

This is to certify that, except where specific reference is made, the work described in this thesis is the result of the candidate's research. Neither this thesis, nor any part of it, has been presented, or is currently submitted, in candidature for any degree at any other University.

Signed	....	(Candidate)
Signed	....	(Director of Studies)
Date	....	

Abstract

This thesis is concerned with the development of new tools for non-linear time series modelling and prediction, and the subsequent application of these tools to the problems of crime prediction and global climate modelling.

We review the state of time series modelling for conventional stochastic time series. After describing the Gamma test and related background material, we extend the available tools by developing a new M-test based heuristic for Gamma test confidence intervals.

We go on to study and develop new results describing the effects of measurement error on modelling and predicting noisy time series. We then describe the application of new techniques, based around the Gamma test, that are capable of selecting predictively useful input variables for modelling and predicting non-linear time series. In the process of this work we developed a much sought after suite of R-tools which are now freely distributed on the R web-site (*http://www.r-project.org*).

Finally, we illustrate the use of these tools by examining two very different types of time series data. The first relates to the problem of crime prediction, where the data emerged as inadequate to support the aspirations of the original researchers. The second modelling exercise takes available data sets for climate modelling and develops predictive models for global surface temperature.

Acknowledgements

It has been a great privilege and delight to work with both of my supervisors: Hasan Al-Madfai and Antonia J. Jones. I owe them both a great deal of debt for their support and encouragement, even when I didn't really deserve it!

Thanks are also due to Dr. Dafydd Evans from the School of Computing at Cardiff University, and Dr. Ian Wilson and Dr. Paul Jarvis from the Department of Computing & Mathematical Sciences at Glamorgan, whom at various points during the project have made interesting and helpful comments.

I would also like to thank Prof. Andrew Ware for giving me the chance to study for a Ph.D., and who took me on amazing trips to Brazil and the United States.

Most importantly of all, I acknowledge with love and gratitude the unswerving support of my parents Richard and Alison, and my brother Ben. This thesis is dedicated to them.

Samuel Edward Kemp

August 2006

Contents

Abstract **3**

Acknowledgements **4**

Contents **5**

List of Figures **11**

List of Tables **16**

1 Introduction **20**

1.1 Limiting the diversity of models 20

1.2 Exploiting the past to predict the future 21

1.3 Linear versus non-linear 22

1.4 Thesis structure 23

2 Background **25**

2.1 Stationarity 27

2.2 ARMA modelling 28

2.3 Some linear multivariate time series models 30

2.3.1 Transfer functions 31

2.3.2 VARMA modelling 32

2.4 Some non-linear parametric time series models 34

2.4.1 Non-linear autoregression 35

2.4.2 Threshold autoregressive models 35

2.4.3 Multivariate threshold autoregressive models 36

2.5 Neural networks: A universal non-parametric modelling technique . . . 37
2.5.1 How a feedforward neural network works 39
2.5.2 Backpropagation 42
2.5.3 Conjugate gradient descent 45
2.5.4 BFGS (Quasi-Newton) method 47
2.5.5 Neural networks for prediction 49
2.6 Local linear regression: an alternative non-parametric modelling technique 50
2.6.1 Singular value decomposition 52
2.7 Chapter summary 53

3 The Gamma test 54
3.1 (ϵ, δ)-methods 55
3.1.1 Difference-based estimators for non-autonomous time series . . . 55
3.1.2 The δ-test 57
3.2 Applications of data-derived noise estimates 57
3.3 The Gamma test 59
3.3.1 Class of models f 60
3.3.2 The noise distribution Ψ 61
3.3.3 The sampling distribution Φ 61
3.4 Determining the number of near neighbours for a Gamma test 61
3.4.1 Background 62
3.4.2 Assessing the Gamma regression line 64
3.5 Data analysis tools based on the Gamma test 66
3.5.1 The M-test 66
3.5.2 Heuristic confidence intervals for the Gamma test 67
3.5.3 Full embedding search 72
3.6 Determining the embedding dimension of chaotic time series 76
3.6.1 False nearest neighbour algorithm 76
3.6.2 Increasing embedding search 78
3.6.3 An example: the Hénon Map 78
3.7 Chapter summary 81

4 The effect of measurement error on time series 82

4.1 Noisy time series . . . 83

4.2 Effective noise . . . 83

4.2.1 Model construction . . . 85

4.2.2 Estimates of effective noise . . . 85

4.3 Input/output data with noisy inputs . . . 85

4.3.1 Example . . . 88

4.4 Noisy time series . . . 89

4.4.1 An Example: The Hénon time series . . . 90

4.5 Chapter summary . . . 92

5 Key algorithms and their R implementation 94

5.1 The algorithms . . . 94

5.1.1 kd-trees . . . 95

5.1.2 Approximate nearest neighbour search in fixed dimensions . . . 102

5.1.3 The effect of approximate near neighbours on Γ . . . 106

5.2 The implementation . . . 107

5.2.1 Requirements definition & specification . . . 109

5.2.2 The Gamma test package for R . . . 109

5.3 Using the Gamma test package: An example R session . . . 112

5.3.1 The first experiment: A simple Gamma test . . . 113

5.3.2 Time series embedding functions . . . 115

5.3.3 M-test . . . 117

5.3.4 Increasing embedding search . . . 117

5.3.5 Full embedding search . . . 118

5.3.6 New embedding search functions . . . 121

5.3.7 Embedding search analysis functions . . . 121

5.4 Chapter summary . . . 122

6 Input variable selection using the Gamma test 123

6.1 Introduction 123

6.1.1 Δ-correlation 124

6.2 New input variable selection routines 126

6.2.1 Approximate nearest neighbour full embedding search 126

6.2.2 Random embedding search 127

6.2.3 Frequency combination method 128

6.3 Empirical Findings 131

6.3.1 Experimental design 131

6.3.2 Results 133

6.3.3 Discussion 133

6.4 Chapter summary 134

7 Applications I: Crime modelling 136

7.1 The study area 137

7.2 Crime data 138

7.2.1 Weather data 138

7.2.2 Criminal damage data 140

7.2.3 Burglary data 140

7.3 Criminal damage modelling 140

7.3.1 Input selection & data analysis 141

7.3.2 Model for criminal damage 145

7.4 Burglary modelling 146

7.4.1 Input selection & data analysis 146

7.4.2 Model for burglary 149

7.5 Chapter summary 150

8 Applications II: Climate modelling 151

8.1 Paleoclimate data and modelling 152

8.1.1 Vostok Ice core data 153

8.1.2 Lack of long-term solar irradiance data 154

8.1.3 Berger Insolation data 155

8.1.4 Input selection: paleo-models 156

8.1.5 Paleoclimate models for temperature 162
8.1.6 Regarding the Vostok data 164
8.2 Modelling temperature for the last millennium 165
8.2.1 Amalgamating data sets for temperature 166
8.2.2 Amalgamating data sets for CO_2 167
8.2.3 Input selection: millennium models 167
8.2.4 Reflections on the millennium models 174
8.3 Chapter summary 175

9 Conclusions 176

9.1 New data analysis tools 176
9.1.1 Heuristic confidence intervals for the Gamma test 176
9.1.2 New embedding search routines 177
9.1.3 Frequency Combination Method for input selection 177
9.2 Noisy time series 177
9.3 R-tools 178
9.4 Applications: crime data 178
9.5 Applications: climate data 179
9.6 Future work 179

Appendix A Simulated Datasets 182

A.1 Dataset A: Linear AR time series 182
A.2 Dataset B: SETAR models 182
A.3 Dataset C: Transfer functions 183
A.4 Dataset D: VAR time series 184
A.5 Dataset E: TARSO time series 184

Appendix B Data sets for climate modelling 189

B.1 The last millennium 189
B.1.1 Historical times: Mann proxy temperature data 189
B.1.2 Historical times: Law dome ice core data 190
B.1.3 Recent times: CRU Temperature data 190
B.1.4 Recent times: Mauna Loa CO_2 data 191
B.2 Paleodata: The last 422 kilo years 192
B.2.1 Vostok ice core data 192
B.2.2 Insolation data 193

Appendix C Analysis of CO_2 Δ-correlation lags 196

References 199

Index 206

List of Figures

2.1 Potential time series models. 27

2.2 The Box-Jenkins approach is an iterative modelling procedure. (Source: Box and Jenkins, 1970) 29

2.3 A realization of the SETAR(2;1,1) model given in (2.23) where $M = 100$ and e_t is taken from a normal distribution with mean zero and $\sigma^2 = 1$. 37

2.4 Neural network architecture ($3 \rightarrow 2 \rightarrow 1$) for forecasting using lags 1, 2 and 3 of a time series y_t as inputs. 39

2.5 The calculation between layers. Here a weight change between node i and node j only has an effect on the output of a single node in the output layer. 43

2.6 The calculation between layers. Now a weight change between node k and node i has an effect on *all* the outputs. 44

3.1 The noisy signal $y_t = 5\sin(8\pi t) + r_t$ is shown in green, the black line is the 'true' signal. In this example $M = 800$ and $\sigma^2 = 0.5$. 56

3.2 The same $y_t = 5\sin(8\pi t) + r_t$ noisy signal, but here $M = 100$. 56

3.3 The problem of selecting p. If M is small the Gamma test will not give a good estimate for σ^2, and if M is large the value p is substantially irrelevant. However, there is an intermediate range within which it is desirable to know the optimal value of p. 62

3.4 The Γ_{error} over increasing p for $y = x^2 + r$. 63

3.5 The regression fit between $(\delta_M(k), \gamma_M(k))$ pairs when $p = 149$ for $y = x^2 + r$. 63

3.6 The high frequency sine data (with large mean squared gradient over $[-1, 1]$), where $y = \sin(40\pi x) + r$. In fact $\mathcal{E}(|\nabla y|^2) = 800\pi^2 \approx 7895.68$. 65

3.7 A close up of the high frequency sine data, where x is taken over the range $[-0.1, 0.1]$. 65

3.8 The Γ_{error} (red) and $\text{MSE}_{\delta\gamma}$ (blue) for $y = \sin(40\pi x) + r$ on small, mid-range and large M. The dashed lines indicate the upper bound on p for the data as suggested by the $\text{MSE}_{\delta\gamma}$. 65

3.9 The M-test graph for $y = x^2 + r$, where $M = 500$ and $\sigma^2 = 0.075$ (dashed line). The Γ (green line) is a close estimate to $\hat{\sigma}_M^2$ (red line). . 67

3.10 The M-test graph for $y = x^2 + r$, where $M = 1000$ and $\sigma^2 = 0.075$. This time both Γ and $\hat{\sigma}_M^2$ are a close estimate to σ^2. 67

3.11 The Γ histogram of 1000 samples for $M = 500$ with an overlay plot of the corresponding normal distribution. The mean 0.075 and $\pm$ one standard deviation are shown. 68

3.12 The Γ histogram of 1000 samples for $M = 1000$ with an overlay plot of the corresponding normal distribution. The mean 0.075 and $\pm$ one standard deviation are shown. 68

3.13 The log(SD) plotted against $\log(M)$ with the regression line fit log(SD) $= -2.0605 - 0.5091 \log(M)$. 69

3.14 An M-test with the heuristic confidence intervals at the 90% level shown as red bars for $f(x) = \sin(x) + r$ with $\sigma^2 = 0.075$. The true noise variance (blue line) is almost always inside the confidence interval. 71

3.15 An M-test with the heuristic confidence intervals at the 99% level shown as red bars for $f(x) = \sin(x) + r$ with $\sigma^2 = 0.075$. The true noise variance (blue line) is *always* inside the confidence interval. 71

3.16 The correlation between the candidate input variables $x_1, \ldots, x_{10}$ and the output y for the linear dataset. A variable uncorrelated with the output is likely to fall between the dashed lines with probability 0.95. . 73

3.17 Linear example. The frequency of input inclusion in the embeddings from the lowest 10% of Γ statistics for $y = 5x_3 + 3x_8 + r$. 74

3.18 Linear example. The frequency of input exclusion in the embeddings from the highest 10% of Γ statistics for $y = 5x_3 + 3x_8 + r$. 74

3.19 Non-linear example. The correlation between the candidate input variables $x_1, \ldots, x_{10}$ and the output y for the nonlinear dataset. A variable uncorrelated with the output is likely to fall between the dashed lines with probability 0.95. Clearly no relevant input variables have been selected by this method. 75

3.20 Nonlinear example. The frequency of input inclusion in the embeddings from the lowest 10% of Γ statistics for $y = \sin(x_3\pi) + 2\cos(x_8) + r$. . . 75

3.21 Nonlinear example. The frequency of input exclusion in the embeddings from the highest 10% of Γ statistics for $y = \sin(x_3\pi) + 2\cos(x_8) + r$. . . 75

3.22 The Hénon map attractor (black dots) draped over a topographic rendering of the surface $f(u, v) = -u^2 + bv + a$, where u corresponds to z_{t-1} and v to z_{t-2}. 79

3.23 False nearest neighbour embedding for the 'clean' Hénon map time series. The graph shows that an embedding dimension $d = 2$ would be suitable. 80

3.24 The increasing embedding for the 'clean' Hénon map time series. The graph suggests that an embedding dimension of $d = 2$ would be suitable. 80

3.25 Results of FNN for the noisy Hénon map time series. The graph suggests $d = 7$ would be a suitable embedding dimension. 80

3.26 Increasing embedding for the noisy Hénon map. The graph suggests $d = 13$ would provide a minimised MSE. 80

4.1 Reality versus observation. 86

4.2 The data set having noisy inputs (x, y) and the function $y = w^2$. 88

4.3 The data set of noisy inputs and the 'best-fit' function. 88

4.4 First 100 points of the clean Hénon time series z_t. 91

4.5 First 100 points of the noise $r_t = \epsilon_t$. 91

4.6 The topographic neural network surface $\hat{g}$, produced by training on 600 points of the noisy data $(\boldsymbol{x}_t, y_t)$, where $\boldsymbol{x}_t = (y_{t-1}, y_{t-2})$. 92

5.1 Algorithm overview. The red dotted circle is the bounded error i.e. $dist(\boldsymbol{q}, \boldsymbol{s})/(1 + \epsilon)$. When the distance from $\boldsymbol{q}$ to the current leaf cell exceeds this bound the algorithm terminates. In this example, this occurs when the algorithm reaches cell 9 causing $\boldsymbol{s}$ to be returned as an $(1 + \epsilon)$-approximate nearest neighbour. 104

5.2 The average relative error for $k = 1$ of a uniformly distributed set of points $C \in [0, 1]^{16}$, where $M = 1000$. 105

5.3 The fraction of approximate nearest neighbours returned by the algorithm that are not exact near neighbours for a uniformly distributed point set $C \in [0, 1]^{16}$, where $M = 1000$. 105

5.4 The measured time (in seconds) to return the 10 nearest neighbours for different ϵ values on a set of 5000 uniformly distributed points, $C \in [0, 1]^d$ for $d = 1, \ldots, 20$. 106

5.5 The effect of approximate near neighbours on Γ. The Γ statistic is shown in blue, whilst its computation time (in seconds) is shown in red. The dashed line indicates the sample noise variance (0.0497). 107

5.6 The Gamma test regression line for $y = \sin(x_1) + 2\cos(x_2) + r$ where $\sigma^2 = 0.05$, $p = 10$ and $M = 1000$. 114

5.7 The M-test graph displayed on the R graphics device for $y = \sin(x) + r$, where $M = 1000$, $p = 10$ and $\sigma^2 = 0.075$. The heuristic confidence intervals at the 90% level are shown in red. 118

5.8 The increasing embedding search results for the Hénon map as displayed on the R graphics device. The plot shows the Gamma statistic over an increasing number of included lagged inputs. 119

5.9 The Gamma histogram returned by the full embedding search for the AR(1) time series, with $m = 10$. 120

5.10 Inspecting the input inclusion occurrence for the Γs in the range $[0, 1]$. 120

6.1 The computing times for a full embedding search on varied input dimensions and approximate near neighbour error bounds ϵ. 128

6.2 The results of the frequency combination method (described in section 6.2.3) on the $y = \sin(x_3\pi) + 2\cos(x_8) + r$ data where $m = 16$ and $\epsilon = 20$. Using this error bound halved the computing time of the full embedding search. 128

6.3 The Gamma histogram of the full embedding search ($M = 1000, m = 10$) on the SETAR time series. 131

6.4 FCM applied to the results of the random embedding search ($M = 1000, m = 10$). The blue dashed line is the upper quartile of the m combined frequencies, which we use as a guideline for selecting relevant inputs. 131

7.1 The Cardiff daily weather readings. 139

7.2 A plot of the daily number of criminal damage cases reported to the Police from 02-Sept-2001 to 30-Sept-2003. 140

7.3 A plot of the daily number of burglaries reported to the Police from 02-Sept-2001 to 30-Sept-2003. 141

7.4 The Δ-correlations for the criminal damage data. 142

7.5 Scatterplot and M-test with confidence intervals, for the 5-input/1-output data set, $M = 600$. 145

7.6 Model test results for one step temperature BFGS neural network ($5 \rightarrow 10 \rightarrow 5 \rightarrow 1$) on unseen test data from. Green: actual criminal damage. Blue: predicted criminal damage. Red: error. 145

7.7 The Δ-correlations for the burglary data. 146

7.8 Scatterplot and M-test with confidence intervals, for the 8-input/1-output data set, $M = 600$. 148

7.9 Model test results for one step temperature BFGS neural network ($8 \rightarrow 16 \rightarrow 8 \rightarrow 1$) on unseen test data from. Green: actual number of burglaries. Blue: predicted number of burglaries. Red: error. 149

8.1 Scaled proxy northern hemisphere surface temperature, 75 yr-smoothed (red) and scaled 75 yr-smoothed CO_2 (blue), for the last 1000 years. . . 152

8.2 Irradiance since 1978: composite satellite measurements (C.Fröhlich and Lean, 1998b,a). 154

8.3 Insolation. (Note that 7.5 kyr ago insolation was decreasing.) 155

8.4 Δ-correlation plots for temperature against itself and the other variables for different sampling intervals (lags). 158

8.5 Input selection methods: Durrant (examine frequency of each input in the high/low Gamma regions), FCM (combining frequencies of inputs from the high/low Gamma regions of the Gamma histogram from a random sample of masks). 161

8.6 Scatterplot and M-test with confidence intervals, for the 6-input/1-output data set (50 yr sampling rate), $M = 7000$. 162

8.7 Model test results for one step temperature BFGS neural network ($6 \rightarrow 12 \rightarrow 6 \rightarrow 1$) on unseen test data from -56.685 to -4.485 kyr (sampling rate is 50 years). Green: actual temperature. Blue: predicted temperature. Red: error. The inset zoom shows the test results for a 5 kyr interval from -19.235 kyr. 163

8.8 Combined temperature anomaly graphs 1000 AD to 2005 AD. 166

8.9 Combined CO_2 data plots 1010 AD to 2006 AD: Law Ice Dome (dark blue), Mauna Loa Instrumental CO_2 records (light blue). 167

8.10 Δ-correlation plots for temperature against CO_2 and itself. 168

8.11 Input selection methods: Durrant (examine frequency of each input in the high/low Gamma regions), FCM (combining frequencies of inputs from the high/low Gamma regions of the Gamma histogram from a random sample of masks). 169

8.12 Scatterplot and M-test with confidence intervals, for the 6-input/1-output data set (1 yr sampling rate), $M = 600$. 170

8.13 Millennium model: test results for one step 6-input/1-output temperature LLR prediction model on unseen test data from 1826 to 2005 (sampling rate is 1 yr). Green: actual temperature. Blue: predicted temperature. Red: error. 172

8.14 Two ways of future projection using the Millennium LLR model 173

8.15 Millennium model: iterating into the future beyond 2005. 173

8.16 Millennium model: predicting temperature on the various CO_2 hypotheses of Table 8.7. 174

9.1 The Carbon Cycle. An example of a complex dynamic system. (From: Wikipedia.) 180

B.1 Proxy annual N.hemisphere surface temperature anomaly 1000 AD to 1997 AD, after (Mann et al., 1998, 1999). 190

B.2 Law Dome Ice core data: annual CO_2 from 1010 AD to 1975 AD. . . . 191

B.3 The UEA Climate Research Unit Temperature data 1856 to 2005 AD. . 191

B.4 Mauna Loa and the monthly CO_2 data 1958 to 2006 AD. 192

B.5 Antarctic near surface temperature-trends 1971−2000. Courtesy: READER (**RE**ference **A**ntarctic **D**ata for **E**nvironmental **R**esearch). 193

B.6 The Vostok 422K year ice core data. 195

C.1 Δ-correlation plots for CO_2 against itself and the other variables for different sampling intervals (lags). 197

List of Tables

2.1 The expected behaviour of the autocorrelations and partial autocorrelations for different ARMA schemes. 30

4.1 Summary of Example 1 results ($M = 1000$). 89

4.2 Summary of Example 2 results ($M = 1000$). Here MSE($\hat{g}$) for the neural network is the weighted average of training and test data. 92

6.1 The possible input inclusion/exclusion combinations from the $lowestList$ and $highestList$. 130

6.2 The classifications of the models used in the study in terms of linearity and whether the model inputs include other time series. 132

6.3 For each input variable selection routine the number of time series where the inputs were successfully identified is shown as a proportion of the total number of the twenty time series generated per model class. . . . 133

6.4 The results of RES on the same time series, but this time $m = 20$ and the sample of random embeddings is 0.1%. 134

7.1 The two most recent lags and two lags based on the strongest Δ-correlations for criminal damage. 143

7.2 The inputs selected from a random embedding search using Durrant's method and the Frequency Combination Method to analyse the Γ-ordered input subsets. The candidate inputs were chosen using the Δ-correlation. 144

7.3 Key statistics for 5-input/1-output training set. 144

7.4 The performance of the one step predictive BFGS neural network ($5 \rightarrow 10 \rightarrow 5 \rightarrow 1$) on the *unseen* criminal damage test data. 146

7.5 The two most recent lags and two lags based on the strongest Δ-correlations for the burglary data. 147

7.6 The inputs selected from a random embedding search using Durrant's method and the Frequency Combination Method to analyse the Γ-ordered input subsets. The candidate inputs were chosen using the Δ-correlation. 147

7.7 Key statistics for 8-input/1-output training set. 148

7.8 The performance of the one step predictive BFGS neural network ($8 \rightarrow 16 \rightarrow 8 \rightarrow 1$) on the *unseen* burglary test data. 149

8.1 The three best lags (in kyr) based on the strongest Δ-correlations for temperature (sampling rate is 50 yr). 159

8.2 The three best lags (in kyr) based on the strongest Δ-correlations for temperature (sampling rate is 100 yr). 159

8.3 The three best lags (in kyr) based on the strongest Δ-correlations for temperature (sampling rate is 500 yr). 159

8.4 The inputs selected from a random embedding search using Durrant's method and the Frequency Combination Method to analyse the Γ-ordered input subsets. The candidate inputs were chosen using the Δ-correlation. 160

8.5 Key statistics for 6-input/1-output training set (50 year sampling rate). 162

8.6 The performance of the one step predictive BFGS neural network ($6 \rightarrow 12 \rightarrow 6 \rightarrow 1$) on different sections of *unseen* temperature test data (sampling rate is 50 yr). 164

8.7 Predicted CO_2 levels on various hypotheses. 166

8.8 The three lags (in years) based on the highest absolute Δ-correlations, and the three best peaks, for temperature against CO_2 and itself, over the period 1011AD-2005AD. 168

8.9 The inputs selected from a full embedding search using Durrant's method to analyse the Γ-ordered input subsets and the Frequency Count Method. The candidate inputs were chosen using the Δ-correlation. 169

8.10 Key statistics for the temperature data (1 year sampling rate) over the (initial) training set. 171

8.11 The performance of the one-year ahead predictions produced by the Millennium local linear regression model (using 100 nearest neighbours) on *unseen* temperature data. 172

8.12 The local linear regression predictions for global surface temperature at 2006, 2050, and 2100AD under the three different hypotheses for future CO_2 emissions. 173

A.1 The simulated datasets. 182

A.2 The linear autoregressive models used. 183

A.3 The SETAR models used. 185

A.4 The parameters used to generate the transfer function processes. 186

A.5 The linear multivariate time series models. 187

A.6 The TARSO models used. 188

C.1 The three best lags (in kyr) based on the strongest Δ-correlations for CO_2 (sampling rate is 50 yr). 198

C.2 The three best lags (in kyr) based on the strongest Δ-correlations for CO_2 (sampling rate is 100 yr). 198

C.3 The three best lags (in kyr) based on the strongest Δ-correlations for CO_2 (sampling rate is 500 yr). 198

List of Algorithms

1 A generalised algorithm for neural network training. 41
2 The conjugate descent algorithm for neural network training. 47
3 The BFGS algorithm for neural network training. 49
4 The *kd*-tree construction algorithm. 98
5 The *kd*-tree search initialisation. 98
6 The *kd*-tree search. 100
7 The *kd*-tree bounds overlap ball routine. 101
8 The Δ-correlation algorithm 125

Chapter 1

Introduction

A set of values measured at particular points in time can in general be considered a *time series*. Examples include rainfall, temperature, electricity demand, burglaries, company stock prices, exchange rates, etc. In general the goal is to construct, as far as possible, predictive models for such time series. Since the interactions which generate a particular time series can be very diverse, and in many cases involve stochastic[1] as well as deterministic processes, the range of potential techniques which can be applied to predictive modelling is vast.

Assuming that there exists some underlying causal structure, it is natural to suppose that past events to some degree play an indicative or causal role in determining future events. Even so the range of possible 'rules' which might 'transform' past events into present or future events can still vary from (say) logic functions to systems of differential equations. Apart from stochastic perturbations and potential measurement error, another major complication arises from *incomplete information*: we may not know, or be measuring, all the relevant factors which affect the evolution of the time series we seek to predict. We frequently model the effects of incomplete information as the introduction of a random variable into our model.

1.1 Limiting the diversity of models

In this thesis we narrow the range of possible models by making the assumption that, as well as stochastic perturbations, there is in fact some *underlying dynamics* we might seek to capture and which can then be exploited for the purposes of prediction.

[1] *Stochastic*: (Greek) $\sigma\tau o\chi\alpha\sigma\tau\iota\kappa o\zeta$ - to aim at, guess, conjecture. In this context a process with a probabilistic component.

Writing down the behaviour of the relevant components of the system in mathematical language, we attempt to combine all we know about their actions and interactions. Thus, the subject of *time series analysis* not only incorporates predictive model construction, but also methods for detecting these interactions in the underlying system. Using these approaches may allow one to build a simplified image of what happens in nature.

Conventional parametric model building techniques require that one postulates *a priori* the nature of the model, whereas non-parametric techniques make no such assumptions; instead they are often based on neighbourhood information elicited from the data. In this thesis the emphasis is on seeking improvements in non-parametric analysis.

1.2 Exploiting the past to predict the future

In modelling a time series non-parametrically we often attempt to reconstruct the dynamics using an embedding technique. Thus, following a theorem of Takens (Takens, 1981) we assume that one variable of a smooth *bounded* system y_t, which we seek to predict, and which evolves according to a system of differential equations, can be modelled in the form

$$y_t = f(\boldsymbol{x}_t) \tag{1.1}$$

where f is a smooth function, $y_t \in \mathbb{R}$ is the observed time series, and $\boldsymbol{x}_t \in \mathbb{R}^m$ is an *input vector* of dimension m. This will be the case provided the embedding dimension m is sufficiently large so as to unfold the dynamics. If the input vector only consists of previous values of y_t i.e. $\boldsymbol{x}_t = (y_{t-1}, \ldots, y_{t-m})$ then the time series is said to be *univariate*. On the other hand, if the input vector contains lagged values of another time series (say q_t) so that $\boldsymbol{x}_t = (y_{t-1}, \ldots, y_{t-m_1}, q_{t-1}, \ldots, q_{t-m_2})$ the time series is said to be *multivariate*.

This predictive technique, using past circumstances to predict future events, is very intuitive and has an illustrious history in forecasting, Lorenz called it the 'method of analogies'. The idea is that we make a prediction based on historical evidence by asking 'what happened in the past when we saw a *similar* sequence of events'? To implement this idea efficiently we simply recognise that finding sequences of historically similar events exactly corresponds to finding near neighbours in the embedding space. One scalable fast method of locating near neighbours is to construct a *kd*-tree, but we shall be considering alternative, very fast, scalable methods.

In general the function f could be so general (e.g. a logic function) as to defy practical analysis. Therefore we have to make some more specific assumptions.

- In this work we shall assume that $f : \mathbb{R}^m \to \mathbb{R}$ is *smooth* (with bounded partial derivatives).

1.3 Linear versus non-linear

Traditional time series analysis techniques largely rely on the assumption of *linearity*. Such techniques are strictly limited to time series where f is linear or can be transformed to be linear. However, considering that *a priori* f is unknown, if non-linearities are suspected such linearity assumptions become untenable. In such cases transforming the data to linearity becomes a rather perilous and frustrating task. One would be forgiven for thinking it odd that the subject of non-linear times series (of which linearity is a special case) has historically often been given a wide berth by the time series modelling community. This may be due to the overwhelming variety of non-linear models - von Neumann once referred to non-linear theory as ‘a theory of non-elephants’ - but it is also the case that non-linear modelling invariably requires much larger data sets. However, owing to increased computing power, automated means of data collection, and recent developments in non-linear data analysis, the subject has started to be recognised as a more practical option for the modeller.

One such development in non-linear data analysis is a fast non-parametric noise estimation algorithm called the *Gamma test* (Stefánsson et al., 1997). Using the available data this test estimates the residual error variance of the best predictive smooth model even though this model is unknown. Since its inception in 1995 the Gamma test has provided the foundation for a suite of new non-linear data analysis methods. Furthermore, during this time a mathematical proof of the algorithm (Evans et al., 2002; Evans and Jones, 2002) has been presented, thus giving it a strong theoretical underpinning[2]. The majority of the Gamma test research has been applied to input/output models with a limited exposure to chaotic time series in (Tsui, 1999) and Flood prediction (Durrant, 2002).

As we have already remarked time series are often interpreted as *stochastic*, i.e. where y_t is only partly determined by its own previous values and possibly also the previous

[2]Although it should be noted that the proof as it stands does not explicitly address the topic of time series.

values of some other time series. Because some relevant data may not be available, or because we may not fully understand the system under investigation, incomplete information can only be modelled as a random variable. Thus, there is inevitably a certain amount of error involved in calculating future values via a deterministic function.

The general form of a smoothly evolving time series can be written as

$$y_t = f(\boldsymbol{x}_t) + r_t \tag{1.2}$$

where f is a smooth function, and r_t is a random variable[3] with $\mathcal{E}(r_t) = 0$.

The defining feature of a *stochastic* time series is that the noise r_{t-1} *feeds through* to affect the next value of y_t. Our notation is then to replace r_t by e_t, in the interests of clarity. If the noise does *not* feed through to affect the next value we call the time series *noisy*. In parts of this thesis the distinction between stochastic time series and noisy time series is important[4]. The Gamma test can be used in either case. If there is no stochastic component then the time series will be *purely deterministic*.

In any event, when modelling time series the analyst will typically be faced by the following questions:

(a) Do I have enough data points to construct a smooth model?

(b) Is the data good enough to get a reliable model?

(c) What are the inputs?

(d) To what extent does the model capture the underlying dynamics of the system?

This thesis investigates empirically the extent to which the Gamma test can answer these questions over a wide class of time series.

1.4 Thesis structure

We begin with a brief study of both linear and non-linear time series analysis techniques in Chapter 2. We consider the current state of non-linear time series analysis and

[3]In conventional statistics the noise is often assumed to follow a Gaussian distribution. However, the Gamma test does not require this assumption.

[4]Of course, in reality many time series possess both qualities, but it is convenient to separate them conceptually.

introduce the techniques that are used throughout this thesis. This is followed by a history of non-parametric noise estimation and a detailed account of the Gamma test in Chapter 3.

It is generally known that measurement error has an adverse effect on our ability to effectively determine the underlying process f. In Chapter 4 using a Gamma test analysis we show how this effect can be *quantified* and the implications it has on modelling such data sets[5].

A consequence of this thesis has been the construction of an R Gamma test package[6] that provides powerful scripting capabilities to a data modeller. We have integrated the techniques developed around the Gamma test into a fast, user-friendly, reliable, and comprehensive toolkit for the analysis of smooth non-linear systems. The design, implementation, and use of the R package is discussed in Chapter 5.

One of the critical areas of time series analysis is to identify the explanatory variables (inputs) to the underlying time series model. Durrant's input selection method (Durrant, 2002), based on the Gamma test, has been successfully applied in the input/output modelling and in time series for the case of river level and flow prediction. Chapter 6 provides a number of extensions to Durrant's method that provide a faster and more robust set of techniques particularly suited to non-linear time series.

The interactions at play in criminal activity are arguably in part a smooth, albeit complex, dynamical system which might be amenable to analysis using the Gamma test. One of the attractive features of a Gamma test analysis is that it can tell us the extent to which smooth determinism actually holds. If the noise level is not so high as to mask the smooth deterministic component then we can apply smooth non-linear modelling techniques. We can state the simplified problem as follows: Can we determine the number of crime and disorder offences in a city centre area from past and present measurements of offences and current environmental factors such as rainfall?

In Chapter 7 we analyse actual crime data and environmental factors using the time series analysis methods described and developed in this thesis.

Finally, in Chapter 8 we apply the same tool-kit to the technically demanding issue of climate modelling. We treat two cases of interest: paleoclimate modelling over the last 500 kyr, and modelling for the last millennium, in an attempt to make some projections with regard to global temperature.

[5]This represents joint work with Antonia J. Jones and D. Evans.

[6]R is an open source language and statistical package. Web address: www.r-project.org

Chapter 2

Background

In time series analysis, we often hypothesize that the variable of interest is just one of a number variables of a complex dynamical system, described by a system of differential equations. An N^{th}-order *autonomous* continuous time system is defined as

$$\frac{dx_i}{dt} = g_i(x_1, \ldots, x_N) \quad (1 \leq i \leq N) \tag{2.1}$$

where t is time, $\boldsymbol{x} = (x_1, \ldots, x_N)$ is a vector in the N-dimensional state space $\mathbb{R}^N$, i.e. the x_i are dynamic variables, some of which are observable, and $g = (g_1, \ldots, g_N)$ is a vector field in the state space.

If t appears explicitly in one or more g_i then the system is called *non-autonomous.* In general an example of a non-autonomous system might be one in which there is a explicit time dependence creating a non-linear trend, or dependence on an extraneous, possibly cyclical or seasonal, time dependent input, such as 'rainfall' for example. Where possible, we shall mostly seek to eliminate such effects by pre-processing, for which there are well established techniques, and concentrate on modelling autonomous systems.

For a sequence of values of one of the observed variables, say $x_1(t), x_1(t+1), \ldots$, our task is to attempt to construct predictive models. When considering a single time series we often suppress the identifying index and write the sequence as $x_t, x_{t+1}, \ldots$.

Thus we might seek to predict the next value of an autonomous system (the output) based on a number d of previous values (the input). In this context, the input is called the *delay vector* and d is called the *embedding dimension*.

For a time series $\{z_t\}$, Taken's theorem (Takens, 1981) and its subsequent extensions ensure, under a broad range of circumstances, that there exists a *smooth* function f

with bounded partial derivatives such that

$$z_t = f(z_{t-1}, z_{t-2}, \ldots, z_{t-d}) \tag{2.2}$$

which, provided d is sufficiently large to unfold the dynamics, can be used as the basis for a recursive one-step prediction.

In (2.2) no noise or measurement error is present and as such $\{z_t\}$ is a 'perfect' *deterministic* time series i.e. if f is known we are able to exactly predict the next value of z_t. Throughout this thesis we explicitly reserve z_t for describing 'perfect' (noise free) deterministic time series.

However, in real-world situations time series are typically subjected to noise. In this context noise is any component of the output that is not accounted for by a smooth transformation of the inputs. Noise occurs for a variety of reasons:

(a) Not all the predictively useful variables that influence the output been captured in the input.

(b) The inputs and output are subjected to measurement error.

(c) The underlying relationship between input and output is not smooth, or there is *no* relationship.

There are two different ways that noise can affect a time series $\{y_t\}$. Firstly, noise can be *stochastic* so that

$$y_t = f(y_{t-1}, y_{t-2}, \ldots, y_{t-d}) + e_t \tag{2.3}$$

here the noise e_{t-1} feeds through to affect the output y_t. We might think of e_{t-1} as a *random perturbation* of the system which actually changes its state. If this is the case then $\{y_t\}$ is called a *stochastic time series*. The implication of stochastic noise is that we can only partly determine the next value of y_t and there will inevitably be an error associated with the prediction.

The second type of noise that can occur, does *not* feed through to affect the next value of y_t, thus

$$\begin{aligned} z_t &= f(z_{t-1}, z_{t-2}, \ldots, z_{t-d}) \\ y_t &= z_t + r_t \end{aligned} \tag{2.4}$$

We imagine that z_t is a perfect time series (i.e. deterministic) that has been corrupted by an independently and identically distributed noise variable r_t which we can think of as *measurement error*. This limits the precision of our perception, but does not affect the evolution of the system. We call such time series *noisy time series*.

The potential class of time series models is vast. Figure 2.1 attempts to classify some of the more popular models by their linearity and stochasticity.

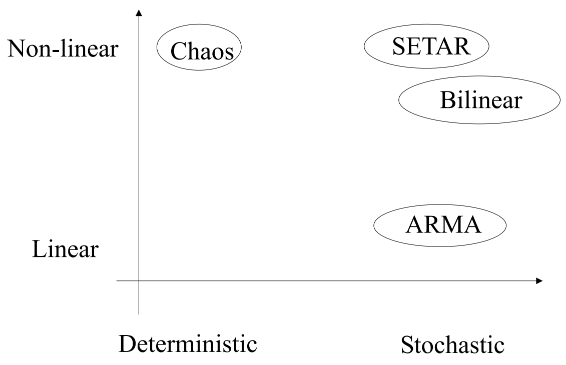


Figure 2.1: Potential time series models.

2.1 Stationarity

A time series $\{y_t\}$ is called *stationary* if, loosely speaking, its statistical properties (particularly the mean and variance) do not change with time. More precisely, $\{y_t\}$ is said to be *strictly stationary* if, for any set of times $t_1, t_2, \ldots, t_M$, and any integer k, the joint probability distribution of $\{y_{t_1}, y_{t_2}, \ldots, y_{t_M}\}$ is identical with the joint probability distribution of $\{y_{t_1+k}, y_{t_2+k}, \ldots, y_{t_M+k}\}$. Less stringently, we say that $\{y_t\}$ is *weakly stationary* (second-order stationary) if only the joint moments up to order 2 of the above probability distributions exist and are identical. A series which is weakly stationary is usually simply referred to as *stationary.* Thus, for a stationary series the mean and variance of y_t remain constant over time and the covariance between any two values, y_t, y_s, depends only on the separation between the time points t and s and *not* their individual locations.

For example, a time series that exhibits trend[1] cannot be stationary because the mean does not remain constant over time. Furthermore, a monthly time series that exhibits a seasonal pattern (e.g. increased retail sales over the Christmas period) also cannot

[1]A steady increase or decrease and rate of change in the time series, when allowances for other components have been made. A simple way of detecting trend in seasonal data is to take averages over a certain period. If these averages change with time we can say that there is evidence of trend in the series. There are also more formal tests to detect trend in a time series.

be stationary because the joint probability distribution of $\{y_{t_1}, y_{t_2}, \ldots, y_{t_M}\}$ will not be the same as the joint probability distribution when we 'move' the time series 1-step ahead i.e. $\{y_{t_2}, y_{t_3}, \ldots, y_{t_M+1}\}$.

2.2 ARMA modelling

Autoregressive Moving Average (ARMA) models have been a popular *linear stochastic* time series modelling technique for the past 30 years. A full account of ARMA modelling can be found in (Box and Jenkins, 1970), what follows here is a brief description.

If the *weak* stationary (Section 2.1) and linear conditions hold on a time series $\{y_t\}$ then we may consider using the stochastic difference equation

$$\Phi_p(B)y_t = \Theta_q(B)e_t \qquad (\max\{p,q\} < t \leq M) \tag{2.5}$$

to model the data, where

$$\begin{aligned}\Phi_p(B) &= 1 - \phi_1 B - \cdots - \phi_p B^p,\\ \Theta_q(B) &= 1 - \theta_1 B - \cdots - \theta_q B^q\end{aligned}$$

Here B is the *backward shift* operator such that $By_t = y_{t-1}$ for $t > \max\{p,q\}$, the values of the *autoregressive* parameters $\phi_i \in \mathbb{R}$, $1 \leq i \leq p$, and the values of the *moving average* parameters $\theta_i \in \mathbb{R}$, $1 \leq i \leq q$. For a stationary process the zeros of the polynomial $\Phi_p(B)$ must lie outside the unit circle (Box and Jenkins, 1970, Section 3.2). The noise variable e_t is an independent and identically distributed (iid) random variable with mean zero and variance $\mathrm{Var}(e_t)$. The stochastic difference equation in (2.5) is often referred to as an ARMA(p,q) model. The task of identifying, fitting and validating ARMA(p,q) models is often achieved by using the *Box-Jenkins approach* (Box and Jenkins, 1970). This iterative approach to ARMA modelling is shown in Figure 2.2. To identify the order of an ARMA model one typically examines the sample *autocorrelation* and *partial autocorrelation* structure of the time series y_t. Once a tentative order of the ARMA model is entertained; the $\Phi_p(B)$ and $\Theta_q(B)$ parameters of the model are estimated using either maximum likelihood or least squares. The residual error, e_t, of the fitted model is then examined. We expect the correlation structure of the time series to be captured by the estimated model, if this has occurred then the residual error should contain no correlation structure i.e. *white noise*. On the other hand if there is a correlation in the residual error then it may be that not all the

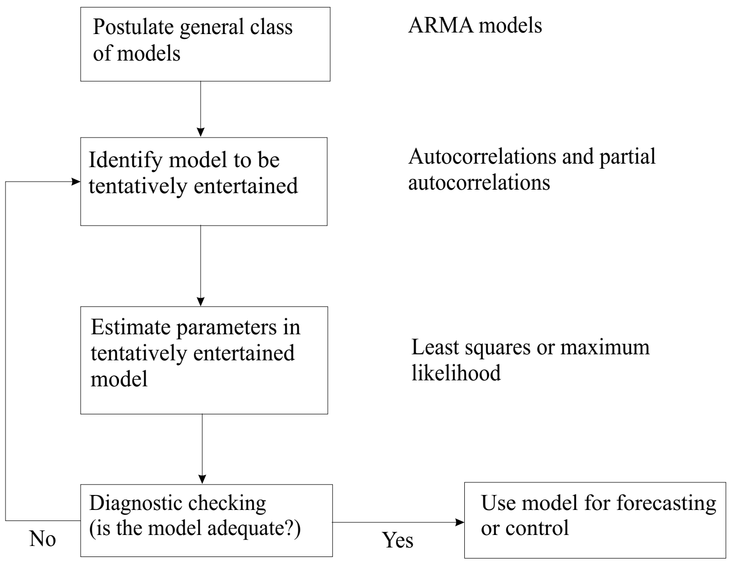

Postulate general class of models
ARMA models
Identify model to be tentatively entertained
Autocorrelations and partial autocorrelations
Estimate parameters in tentatively entertained model
Least squares or maximum likelihood
Diagnostic checking (is the model adequate?)
No
Yes
Use model for forecasting or control

Figure 2.2: The Box-Jenkins approach is an iterative modelling procedure. (Source: Box and Jenkins, 1970)

pattern has been accounted for in the estimated model. As such we need to go back to the identification stage and re-assess the inputs of the model.

The autocorrelation coefficients are the standardised set of autocovariance coefficients $\{\gamma_k\}$ that lie in the range [-1,1]. The sample autocorrelation $\hat{\rho}_k$ at lag k is calculated using

$$\hat{\rho}_k = \frac{\sum_{t=1}^{M-k}(y_t-\bar{y})(y_{t+k}-\bar{y})}{\sum_{t=1}^{M}(y_t-\bar{y})^2} \qquad (1 \le k \le m) \tag{2.6}$$

where y_t is the observed datum at time t, M is the length of the series, $\bar{y}$ is the mean of the sample, and m is the maximum lag we wish to consider, typically $m \approx 20$.

The partial autocorrelation is a measure of association between y_t and y_{t-k}, when the effects of other time lags $1, 2, \ldots, k-1$ are removed. Suppose that there is a significant autocorrelation between y_t and y_{t-1}. Then, there would also be a significant autocorrelation between y_t and y_{t-2} since y_t and y_{t-2} are both related to y_{t-1}. Thus, for example, to measure the real correlation between y_t and y_{t-2} the effects of y_{t-1} should therefore be removed. The *sample partial autocorrelation* is calculated by

$$\hat{\rho}_{kk} = \frac{\hat{\rho}_k - \sum_{j=1}^{k-1}\hat{\rho}_{[k-1,j]}\hat{\rho}_{k-j}}{1-\sum_{j=1}^{k-1}\hat{\rho}_{[k-1,j]}\hat{\rho}_j} \qquad (1 \le k \le m) \tag{2.7}$$

where $\hat{\rho}_k$ is the autocorrelation between y_t and y_{t-k}, $\hat{\rho}_{[k-1,j]}$ is the autocorrelation between y_{k-1} and y_j $(1 \le k \le m)(1 \le j \le k-1)$.

The guidelines provided by Box and Jenkins (Box and Jenkins, 1970) for using the autocorrelation and partial autocorrelations to identify the orders p and q in an ARMA model are shown in Table 2.1.

Model	**Autocorrelations**	**Partial autocorrelations**
White noise	no significant coefficients	no significant coefficients
AR(p)	decays exponentially and may contain damped oscillations	p significant coefficients
MA(q)	q significant coefficients	decays exponentially and may contain damped oscillations
ARMA(p,q)	both decays exponentially and may contain damped oscillations	

Table 2.1: The expected behaviour of the autocorrelations and partial autocorrelations for different ARMA schemes.

In Chapter 6 we shall use these models to generate time series processes in order to test the input variable selection routines proposed within that chapter. Similar remarks apply to the other models described in the remainder of this chapter.

2.3 Some linear multivariate time series models

The Box-Jenkins methodology is limited to univariate time series. However, in many cases two or more time series may be ‘linked’ in some way. For example changes in one time series may be influential is causing changes in another time series later in time; the first is said to be a *leading indicator* of the second. Thus using *multiple* time series in the inputs can sometimes produce better predictive models.

This is especially the case when modelling systems which are not ‘dynamically closed’. For example, rainfall in the Brecon Beacons is a leading indicator of water levels several hours later on the river Taff at Cardiff. Simply observing the time series of water levels at Cardiff, and using the lagged values for predicting the current value, can never produce predictions which would be as good as those which took account of the rainfall in the mountains some hours earlier. The system is not ‘dynamically closed’. It is driven by the stochastic input called ‘rainfall’.

2.3.1 Transfer functions

Thus a given time series $\{y_t\}$ may not only be 'influenced' by its own past values, but also by the past values of another time series, $\{x_t\}$, say. In this context, $\{x_t\}$ is the *input time series* and $\{y_t\}$ is the *output time series.* When the input series influences the output series, but the output series *does not* feedback to influence the input series we call the underlying system *open*.

A transfer function is a parsimonious representation of the transformation from the input series to the output series. The transfer function model can be written as

$$\Lambda_n(B)y_t = \Omega_v(B)x_{t-b} \tag{2.8}$$

where

$$\begin{aligned}\Lambda_n(B) &= (1 - \lambda_1 B - \cdots - \lambda_n B^n)\\ \Omega_v(B) &= (C - \omega_1 B - \cdots - \omega_v B^v)\end{aligned}$$

here the backward shift operator B is defined by $Bx_{t-b} = x_{t-b-1}$ for $t > \max(n, v)$, C is some constant, and b is the *time delay.* The time delay is the amount of time before an input time series begins to influence the output e.g. the time taken for rainfall to affect a river's level. Essentially, the transfer function representation is saying that a time series can be modelled using a linear combination of its own past values and a linear combination of another time series.

In practice the system under investigation will contain disturbances or noise, whose net effect is to corrupt the output predicted by the transfer function model by an amount N_t. Therefore, (Box and Jenkins, 1970) propose the *transfer function-noise model*, defined as

$$\Lambda_n(B)y_t = \Omega_v(B)x_{t-b} + N_t \tag{2.9}$$

Box and Jenkins make the assumption that the noise series $\{N_t\}$ can be modelled using an ARMA representation, and that the noise variable N_t is statistically independent of the input x_t. Writing N_t as an ARMA model, equation (2.9) now, explicitly, becomes

$$\Phi_p(B)\Lambda_n(B)y_t = \Phi_p(B)\Omega_v(B)x_{t-b} + \Lambda_n(B)\Theta_q(B)e_t \tag{2.10}$$

where e_t is a Gaussian white noise process.

Again Box and Jenkins (Box and Jenkins, 1970) provide a procedure for identifying, fitting and validating a transfer function model (see Bowerman and O'Connell, 1993, for an excellent account). However, it has not been as widely adopted as their ARMA modelling procedure. The major pitfall of their approach is that the guidelines for model selection using cross-correlations are unreliable. One is often better off setting the orders v and n to 2, and then adjust the orders until an acceptable 'model performance' is observed in the model diagnostic stage.

2.3.2 VARMA modelling

Some time series are a part of a *closed-looped* system because each series influences one another i.e. y_{1t} affects another time series y_{2t}, and y_{2t} *feedbacks* to affect y_{1t}. A famous example is the British Pound, American Dollar and Japanese Yen exchange rate system. Obviously, transfer functions will be a poor choice of model for closed-looped systems because they do not allow for such feedbacks. Thus, to deal with the notion of feedback one may consider using the *vector autoregressive moving average* (VARMA) model.

Suppose we are given k time series

$$\{y_{1t}\}, \ldots, \{y_{kt}\}, \qquad t = 0, 1, \ldots, M \tag{2.11}$$

each with a constant sampling rate. Writing

$$\boldsymbol{y}_t = (y_{1t}, \ldots, y_{kt})^T \tag{2.12}$$

we shall refer the k series as a k-dimensional vector of multiple time series. The general VARMA model can be expressed as

$$\boldsymbol{\Phi}_p(B)\boldsymbol{y}_t = \boldsymbol{\Theta}_q(B)\boldsymbol{e}_t \tag{2.13}$$

where

$$\begin{aligned} \boldsymbol{\Phi}_p(B) &= \boldsymbol{I} - \boldsymbol{\phi}_1 B - \cdots - \boldsymbol{\phi}_p B^p \\ \boldsymbol{\Theta}_q(B) &= \boldsymbol{I} - \boldsymbol{\theta}_1 B - \cdots - \boldsymbol{\theta}_q B^q \end{aligned}$$

are matrix polynomials in B, both $\boldsymbol{\Phi}$ and $\boldsymbol{\Theta}$ are $k \times k$ matrices, $\{\boldsymbol{e}_t\}$ with $\boldsymbol{e}_t = (e_{1t}, \ldots, e_{kt})^T$ is a sequence of random shock vectors identically independently and normally distributed, and $\boldsymbol{I}$ is the identity matrix. When $k = 1$ we notice from (2.13) that it would be an ARMA model i.e. univariate, thus when using the term VARMA we assume that $k \geq 2$. The series $\boldsymbol{y}_t$ will be stationary when the zeros of $|\boldsymbol{\Phi}_p(B)|$ are all outside the unit circle[2] (Tiao and Box, 1981).

[2]Explicitly all k time series must be stationary for VARMA models to be applied.

The identification of inputs for a VARMA model is achieved by using the sample *cross-correlation* and sample *partial autoregression* statistics. Tiao and Box (Tiao and Box, 1981) provide some useful rules of thumb that we shall now describe.

Given k time series, the sample $k \times k$ cross-correlation matrix at lag l is given by

$$\hat{\boldsymbol{\rho}}(l) = \begin{pmatrix} \hat{\rho}_{11} & & \\ & \ddots & \\ & & \hat{\rho}_{kk} \end{pmatrix} \tag{2.14}$$

where

$$\hat{\rho}_{ij}(l) = \frac{\sum (y_{it} - \bar{y}_i)(y_{j(t+l)} - \bar{y}_j)}{\sqrt{\operatorname{Var}(y_i)\operatorname{Var}(y_j)}} \qquad (1 \leq i \leq k)(1 \leq j \leq k) \tag{2.15}$$

In the end we should have m $\hat{\boldsymbol{\rho}}$ matrices, (typically $m = 12$) of cross-correlations. However, plotting such values would be cumbersome as k increases and reading the values in the matrices would be even more difficult. To overcome this (Tiao and Box, 1981) propose using $+, -, \cdot$ signs to indicate correlation strength. The $+$ sign is used to flag a value which is greater than the upper confidence limit (i.e. $2M^{-\frac{1}{2}}$), a $-$ sign if the value is less than the lower confidence limit (i.e. $-2M^{-\frac{1}{2}}$) and a $\cdot$ sign is used when the value of the correlation is not significantly different from zero.

As an example consider the following $\hat{\boldsymbol{\rho}}$ matrices for lags $1-3$ for a pair of time series y_{1t} and y_{2t} where $M = 250$

$$\begin{pmatrix} -0.28 & 0.37 \\ -0.21 & -0.19 \end{pmatrix} \begin{pmatrix} 0.03 & 0.08 \\ 0.02 & 0.01 \end{pmatrix} \begin{pmatrix} 0.04 & -0.03 \\ -0.01 & -0.08 \end{pmatrix}$$

Using the Tiao and Box indicator variables on the above matrices would give

$$\begin{pmatrix} - & + \\ - & - \end{pmatrix} \begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \end{pmatrix}$$

In general the pattern of indicator symbols makes it very easy to identify a low order moving average model. If $\hat{\boldsymbol{\rho}}(1)$ consists of indicator variables $+$ and/or $-$ and the other $\hat{\boldsymbol{\rho}}(l)$ $(2 \leq l \leq m)$ mainly contain a $\cdot$ indicator, then we may tentatively assume a VARMA$(0, 1)$ model. On the other hand, the persistence of large correlations suggests the possibility of autoregressive behaviour.

To identify any autoregressive behaviour we can use the partial autoregression matrices $\boldsymbol{\mathcal{P}}(l)$. Estimates for $\boldsymbol{\mathcal{P}}(l)$ and their standard errors can be obtained by fitting autoregressive models of successively higher order $l = 1, 2, \ldots$ by standard multivariate least squares. Again, the indicator variables $+, -, \cdot$ can be used to simplify the identification procedure. We assign a plus when a coefficient in $\hat{\boldsymbol{\mathcal{P}}}(l)$ is greater than 2 times its estimated standard error, a minus sign when it is less than -2 times its standard error, and a dot for values in between. If $\hat{\boldsymbol{\mathcal{P}}}(l)$ consists of $+$ and/or $-$ in the first one or two lags then we can tentatively assume an autoregressive model.

Once the tentative model has been specified we can use maximum likelihood or a least squares method to fit the model. The residual variance of this model, like ARMA models, should be *white noise*. To check this we calculate the cross correlation of the residual errors, if the errors are white noise then the $\hat{\boldsymbol{\rho}}(l)$ $(1 \leq l \leq m)$ matrices will largely consist of dot indicators. If they do not, then we may have to return to the model identification stage.

Using VARMA models is not without its difficulties:

- If all k time series are not stationary then time will have to be spent on transforming all k series to stationarity. Furthermore, there are no guarantees that this can be achieved.
- As k increases the model identification process becomes more cumbersome.
- If the underlying relationships between the k series are non-linear then VARMA modelling will only provide a crude estimate of the underlying dynamics. This is more problematic given that these relationships *a priori* are unknown.

However if time series are, or can be transformed to stationarity, k is relatively small $(k \leq 3)$, and we have enough evidence to be confident that the underlying model is approximately linear then VARMA models may be a good parsimonious representation to use.

2.4 Some non-linear parametric time series models

In nature many phenomena are fundamentally non-linear and using linear modelling techniques is likely to yield a rather crude estimate of the underlying dynamics. However, the class of potential non-linear models is enormous. Parametric methodology attempts to reduce the size of this class by only considering a particular subclass of non-linear model e.g. threshold models. The development of these subclasses is such that they have a general appeal i.e. they are capable of describing a large variety of non-linear phenomena.

Given the vastness of potential non-linear models, it is not surprising that a large number of parametric non-linear modelling techniques have been proposed (see Tong, 1990, for an up-to-date account). We briefly describe the more popular methods.

2.4.1 Non-linear autoregression

Motivated by dynamical systems, non-linear autoregression has become an important class of time series models. Specifically, a time series $\{y_t\}$ is said to follow a non-linear autoregressive model of order d with additive noise if there exists a function $f : \mathbb{R}^d \to \mathbb{R}$ such that

$$y_t = f(y_{t-1}, \ldots, y_{t-d}) + e_t \tag{2.18}$$

where $\{e_t\}$ is a sequence of i.i.d. variables and is typically independent of y_t. Frequently in the literature, the acronym NLAR(d) refers to the restricted class defined by (2.18).

As yet the functional form of f has not been specified. In a practical situation we may leave f unspecified and try to derive the form directly from the data; this leads to the so-called non-parametric autoregressive function approach which is detailed in section 2.5. Alternatively, we may parameterize f and this leads us to the focus of this section on the *finite parametric autoregressive approach.*

2.4.2 Threshold autoregressive models

Threshold autoregressive (TAR) models were introduced by Tong in a long sequence of papers culminating in (Tong and Lim, 1980). These models have jump discontinuities and so strictly speaking are outside our domain of discourse, but they are useful and it is interesting to observe that the Gamma test (although it will ‘see’ jump discontinuities as noise) can be used in such situations provided there are not too many points of discontinuity.

The basic idea is that we start with a linear model for the time series $\{y_t\}$ and then allow the parameters of the model to vary according to the values of a finite number of previous values of y_t. For example, a first-order threshold autoregressive model, with two regimes, would typically be of the form

$$y_t = \begin{cases} \phi_0^{(1)} + \phi^{(1)} y_{t-1} + e_t & \text{if } y_{t-1} < \tau \\ \phi_0^{(2)} + \phi^{(2)} y_{t-1} + e_t & \text{if } y_{t-1} \geq \tau \end{cases} \tag{2.19}$$

where label '(1)' means *regime* one, '(2)' is a label for regime two, $\{e_t\}$ is a white noise process, $\phi^{(1)}$, $\phi^{(2)}$ are constants, and the constant τ is called the *threshold.* This model was first introduced by Tong to analyse river flow data and can be readily generalised to a TAR(k) model, with l regimes, by

$$y_t = \phi_0^{(i)} + \phi^{(i)} y_{t-1} + \cdots + \phi^{(i)} y_{t-k} + e_t \quad (k+1 < t \le M)(1 \le i \le l) \tag{2.20}$$

if $(y_{t-1}, \ldots, y_{t-k}) \in \mathcal{C}_i \subseteq \mathbb{R}^k$, where $\mathcal{C}_i$ is a closed bounded subset of $\mathbb{R}^k$. The model (2.20) may now be regarded as a piecewise linear approximation to the general non-linear kth-order AR model,

$$y_t = f(y_{t-1}, \ldots, y_{t-k}) + e_t \quad (k+1 < t \le M) \tag{2.21}$$

In practice it would not really be feasible to consider fitting data to a model of the form (2.20) with k reasonably large since the determination of the threshold regions would involve a search of k-dimensional space. Tong therefore restricted attention to the case where the various sets of parameter values are determined by *just a single past value*, y_{t-1}, say. In general a k^{th} order threshold autoregressive model, with l regimes, can be written

$$y_t = \phi_0^{(i)} + \sum_{j=1}^{k} \phi_i^{(i)} y_{t-j} + e_t, \quad \text{if } y_{t-d} \in \mathcal{C}_i \qquad i = 1, \ldots, l \tag{2.22}$$

where $\mathcal{C}_i \subseteq \mathbb{R}^k$. Since, the indicator variable i is now a function of y_t itself we call the univariate times series $\{y_t\}$ given by (2.22) a *self-exciting threshold autoregressive model of order* $(l; k, \ldots, k)$ or SETAR$(l; k, \ldots, k)$, where k is repeated l times.

In (2.22) we have assumed a common order, k, for each of the autoregressions; in practice, however, it could well turn out that different threshold regions give rise to models of different orders. To deal with this we must take k to the largest order involved and set all the redundant coefficients to zero.

Example. Figure 2.3 shows a realization of the SETAR(2;1,1) model:

$$y_t = \begin{cases} 1 + 0.7 y_{t-1} + e_t & \text{if } y_{t-1} < 0 \\ -1 + 0.3 y_{t-1} + e_t & \text{if } y_{t-1} \ge 0 \end{cases} \tag{2.23}$$

where e_t is taken from a normal distribution with mean zero and $\sigma^2 = 1$.

2.4.3 Multivariate threshold autoregressive models

The threshold autoregressive model can be extended to the multivariate case. We call two series $\{y_t, x_t\}$ an *open-loop threshold autoregressive system* with $\{y_t\}$ as the

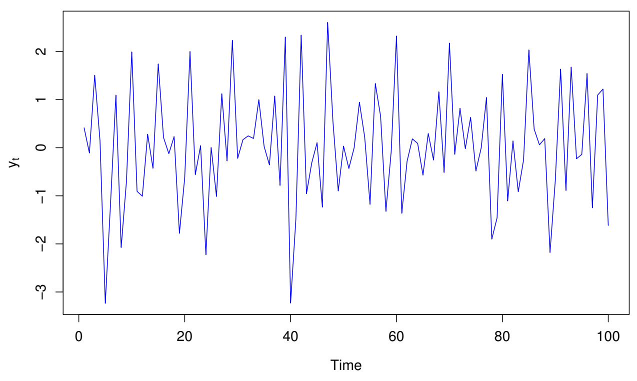


Figure 2.3: A realization of the SETAR(2;1,1) model given in (2.23) where $M = 100$ and e_t is taken from a normal distribution with mean zero and $\sigma^2 = 1$.

observable output and $\{x_t\}$ as the observable input, if

$$y_t = \phi_0^{(i)} + \sum_{j=1}^{p_i} \phi_j^{(i)} y_{t-j} + \sum_{j=1}^{q_i} \theta_j^{(i)} x_{t-j} + e_t^{(i)} \tag{2.24}$$

conditional on $x_{t-d} \in \mathcal{C}_i$ for $i = 1, \ldots, l$, where $\{e_t^{(i)}\}$ $(1 \leq i \leq l)$ are hetrogeneous white noise processes with zero mean and finite σ^2 and each being independent of $\{x_t\}$. These l sequences are assumed to be independent of one another. We denote this system by TARSO$(l; (p_i, q_i), \ldots, (p_l, q_l))$. Typically, we use the exogenous variable x_{t-d} to indicate which subsystem to activate.

The TARSO model can be used to describe a system where two time series are interrelated (or in a closed-loop). Hence, if (y_t, x_t) and (x_t, y_t) are both TARSO processes then $\{y_t, x_t\}$ is called a closed-looped threshold autoregressive system (TARSC).

2.5 Neural networks: A universal non-parametric modelling technique

The first work on the computational representation of a neuron can be found in (McCulloch and Pitts, 1943), with a device that took the weighted sum as the input and generated an output either 0 or 1 depending on whether the sum was above or below a given threshold.

Rosenblatt developed *perceptrons* (Rosenblatt, 1962), single layer feedforward networks of McCulloch-Pitts neurons, where then research focussed on how to to find suitable weights for a particular computational task. The 'perceptron-convergence' theorem justified an iterative training algorithm which was guaranteed to arrive at the correct classification of training data falling into one of two different classes, provided the data was 'linearly separable' i.e. all patterns in one class lay on one side of a hyperplane whilst all in the second class lay on the other.

The weaknesses of the single layer perceptrons were demonstrated in (Minsky and Papert, 1969). Here, they showed that many classes of problems that one might wish to solve were quite definitely *not* linearly-separable, e.g. counting the number of holes in a surface, or deciding whether a point was on the inside or the outside of a non-linear surface. The classic example of such a problem often used is the Exclusive-OR problem. They conjectured that finding a suitable training algorithm for multi-layered perceptrons would be difficult, if not impossible. Their conclusions effectively paused scientific interest in artificial neural networks for 15 years.

Choosing a set of weights that produces a global error-minimum in weight space is a straight forward, if rather demanding, mathematical calculation. But this somewhat misses the point of neural network research, in which we imagine that by being presented with a finite sequence of training data the network progressively adjusts it's weights *by local computations only* so as to reach a suitable error minimum.

As long as the neural output is categorical, e.g. restricted to '0' or '1' the potential range of training algorithms is somewhat limited. However, the '0/1' output, all-or-none, model of McCulloch-Pitts can be replaced by a model in which the output of a unit is a *real number* and this opens up new possibilities for training algorithms.

The interest in neural networks was regenerated in the mid 1980s with the re-discovery of Werbos' backpropagation (Werbos, 1974) in (Rumelhart et al., 1986), (Le Cun, 1986) and (Parker, 1985). Backpropagation is a local error-gradient-descent technique for training *multi-layer* perceptrons with continuous outputs by adjusting the connecting weights in successive layers. As was well understood by Minsky and Papert, the provision of hidden layers means that such networks can be used on problems which are not linearly separable. Their reservations concerned the practicality of providing effective training algorithms.

The fact that such local descent procedures could prove effective in a wide range of practical applications is by no means obvious. That the local error minimum so obtained is often a good approximation to the global minimum error is a reflection of the fact that good approximations are generally ubiquitous in weight-space, and Minsky and Papert can be forgiven for not anticipating such an unexpected result. It is also the case, as we shall discuss later, that for *noisy data* one should *not in any case* be attempting to minimise the error over the training set.

It was proved by (Hornik et al., 1989) and (Cybenko, 1989) that any feedforward neural network, with as few as one hidden layer, using fixed sigmoidal functions, can approximate to any desired degree of accuracy any continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ over a compact subset of $\mathbb{R}^n$ provided there are sufficiently many hidden units available. However, this does not guarantee that any particular training algorithm will converge to the required approximation, nor does it indicate the number of hidden units required.

Multi-layer perceptrons trained using backpropagation must have differentiable output functions, thus the threshold unit proposed in (McCulloch and Pitts, 1943) had to be replaced. The typical choice of output function is a sigmoid.

2.5.1 How a feedforward neural network works

An input is passed through the network in a feedforward fashion to produce an output. The input to each node in the first layer is calculated using (2.25), the processing at each node is performed and the output is fed forward to the next layer. This process continues until the outputs are produced by the network.

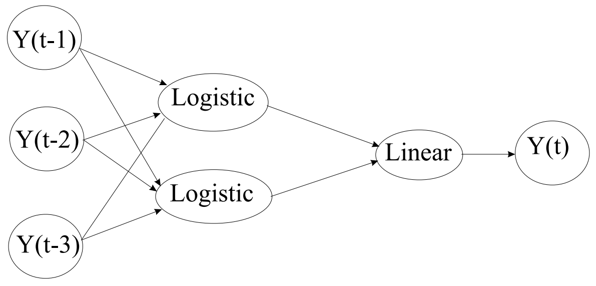


Figure 2.4: Neural network architecture ($3 \to 2 \to 1$) for forecasting using lags 1, 2 and 3 of a time series y_t as inputs.

A representation of a neural network is shown in Figure 2.4. The nodes in the input layer and the bias nodes do not perform any processing. However, the data can be pre-processed before it is presented to the network (e.g. we may remove the effects of trend in a time series). The other nodes in the hidden layer of the network all perform processing by using an activation function. This is where the input to the i^{th} node is processed as a sum of the outputs of the nodes in the previous layer multiplied by their connecting weights to the i^{th} node, defined as

$$net_i(y_i, \dots, y_m) = \sum_{j=1}^{m} w_{ij} \cdot y_j - \theta_i \tag{2.25}$$

where m is the number of nodes in the previous layer, w_{ij} is the weight from the j^{th} node to the i^{th} node, θ_i is the threshold for the i^{th} node and y_j is the output of the j^{th} node in the previous layer. In practice, the thresholds are implemented by adding a bias node to each layer, which is always on and is connected only to the nodes in the next layer. The connection weights can be adjusted during training without special treatment in the implementation to create the threshold.

Given the activation net_i, the output function determines the response of a node. The chosen function is typically differentiable, non-linear and monotonic to provide a smooth mapping between continuous variables. The conventional output functions are either the logistic sigmoidal function

$$h(net_i) = \frac{1}{1 + e^{-net_i}} \qquad 0 < h < 1 \tag{2.26}$$

or the *tanh* function

$$h(net_i) = \frac{e^{net_i} - e^{-net_i}}{e^{net_i} + e^{-net_i}} \qquad -1 < h < 1 \tag{2.27}$$

The goal of training a feedforward neural network

A neural network for non-linear regression is trained by example using input-output data. Since virtually all real data is subject to measurement error the target outputs can be assumed to contain noise. Estimating this noise is the subject of the next chapter. Anticipating slightly, we can see that it is pointless to train a network to such an extent that it begins to model the noise present in the training data. Such a procedure would result in a model with poor generalisation, it would suffer from 'overtraining', as the saying goes.

Thus the purpose of training is to reduce some measure of error on the training data by adjusting the model parameters until a point is reached where further reduction would result in overtraining. In particular

- The objective of training a feedforward network on noisy data is *not* to *minimise* the Mean-Squared-Error on the training data.

If the distribution variance of the noise on an output of the training data is σ^2, and the then the objective of training is to minimise some measure of the difference $\sigma^2 - \mathrm{MSE}$.

If the error function is a differentiable function of the outputs (e.g sum-of-squares), then the error becomes a differentiable function of the weights. Thus, the reduction of MSE, so that $D = (\sigma^2 - \mathrm{MSE})^2$ (say) is minimised, becomes an optimisation in weight space. The network weights are adjusted until D is at a minimum or a predefined limit has been reached. A simple algorithm for neural network training is described in Algorithm 1. Here, the training data is periodically shuffled in order to avoid repetitive cycles which may lead to the algorithm being stuck in a bad local minima, i.e. where D is high. Each vector (or pattern as it is referred to in some neural network literature) is passed through the network and the error is calculated between the expected model output and the actual model output and the weights in the network are adjusted accordingly. At each iteration the algorithm checks whether the stopping criterion has been reached.

```
{initialisation}
establish stopping criteria
determine the network architecture
initialise the weights

{training loop}
while the stopping criteria has not been reached do
  shuffle the data
  for $i = 1$ to $M$ do
    feedforward $\boldsymbol{x}_i$ through the network to calculate the error adjust the network
    weights to reduce the error
  end for
end while
```

Algorithm 1: A generalised algorithm for neural network training.

A considerable amount of insight into the weight optimisation problem, and the techniques for solving it, can be gleaned by considering the local quadratic approximation to the error function. Consider the second Taylor expansion of the error surface $E(\boldsymbol{w})$ around the minimum point $\boldsymbol{w}^*$

$$E(\boldsymbol{w}) = E(\boldsymbol{w}^*) + (\boldsymbol{w} - \boldsymbol{w}^*)^T \nabla E(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^*) \tag{2.28}$$

where $\boldsymbol{H}$ is the Hessian matrix and $\boldsymbol{w}$ is a vector of weights. At the minimum $\boldsymbol{w}^*$, the linear term is eliminated since $\nabla E(\boldsymbol{w}^*) = 0$, and as such the quadratic error function can be expressed as

$$E(\boldsymbol{w}) = E(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \tag{2.29}$$

The three types of quadratic error minimisation techniques considered in this thesis include

Steepest gradient descent: This is an iterative minimisation process which descends in the direction of the steepest local gradient. The rate of convergence is determined by the step size and other convergence enhancing techniques such as momentum.

Conjugate gradient descent: These procedures use previous gradient measures to improve the minimisation procedure. After a minimisation has occurred in one direction it is not considered for minimisation again.

Quasi-Newton methods: Here the location of the minimum of the quadratic surface $E(\boldsymbol{w}^*)$ from the Newton direction is found using the inverse Hessian. In practice, it is computationally expensive to compute the Hessian directly, so an approximation is calculated. Initially, the algorithm performs steepest gradient descent, but as the approximation of the Hessian improves, the convergence to the minima can be rapid.

Both the conjugate gradient descent and quasi-Newton methods calculate the Hessian matrix to perform the minimisation, whereas the gradient descent method does not. Furthermore, conjugate and quasi-Newton methods guarantee that the error function will not increase with a change to the weights.

2.5.2 Backpropagation

Let $y_1, \ldots, y_m$ be the outputs from the output layer units and let $t_1, \ldots, t_m$ be the corresponding target outputs for the given input. Then we can define the error function (for this input[3]) as

$$E(y_1, \ldots, y_m; t_1, \ldots, t_m) = \frac{1}{2}\sum_{j=1}^{m}(y_j - t_j)^2 \tag{2.30}$$

[3]In practice we average this over all inputs.

The popular backpropagation learning algorithm proposed in (Rumelhart et al., 1986) is the original steepest gradient descent error minimisation technique. Here, the initial weight vector (of the entire network) denoted $\boldsymbol{w}_1$ is typically chosen at random. The error E depends on the weight w_{ji} only via the summed input net_j to unit j, where the index j applies to the output layer nodes, and the index i to the preceding layer nodes.

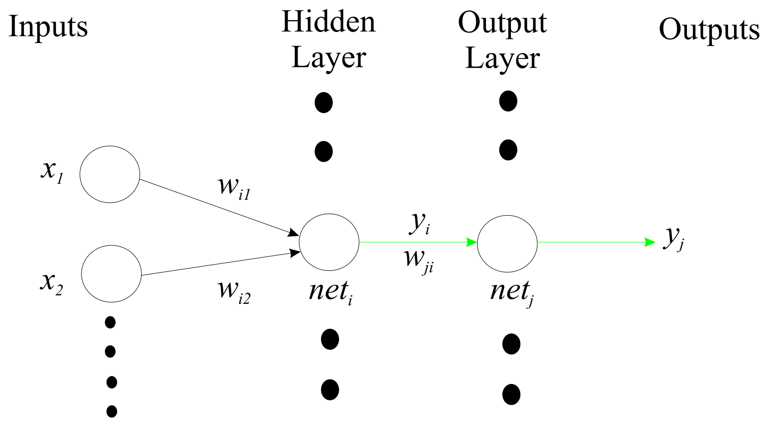


Figure 2.5: The calculation between layers. Here a weight change between node i and node j only has an effect on the output of a single node in the output layer.

Thus, the chain rule can be applied to give for the weight change Δw_{ji}

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -\eta \frac{\partial E}{\partial net_j} y_i \tag{2.31}$$

where the parameter $\eta > 0$ denotes the 'learning rate', i.e. a parameter used to control the step size in weight space, and the last equality follows from (2.25). We now estimate the partial derivatives in terms of known quantities. Again using the chain rule we write

$$\begin{aligned}\delta_j &= -\frac{\partial E}{\partial net_j} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j} \\ &= -h'(net_j)\frac{\partial E}{\partial y_j} = -h'(net_j)(y_j - t_j)\end{aligned} \tag{2.32}$$

from (2.30).

Thus for the output layer we have

$$\Delta w_{ji} = \eta \delta_j y_i \tag{2.33}$$

where δ_j is given in terms of known quantities by (2.32).

In a similar way we can compute a rule which adjusts the weights in the previous layers. We shall not go through the details of the derivation (which are quite straightforward), but the rule which emerges is, for node i in the *previous layer*,

$$\Delta w_{ik} = \eta \delta_i \frac{\partial net_i}{\partial w_{ik}} = \eta \delta_i y_k \tag{2.34}$$

where

$$\delta_i = -h'(net_i)\frac{\partial E}{\partial y_i} = h'(net_i)\sum_{j=1}^{n} \delta_j \frac{\partial net_j}{\partial y_i} = h'(net_i)\sum_{j=1}^{n} \delta_j w_{ji} \tag{2.35}$$

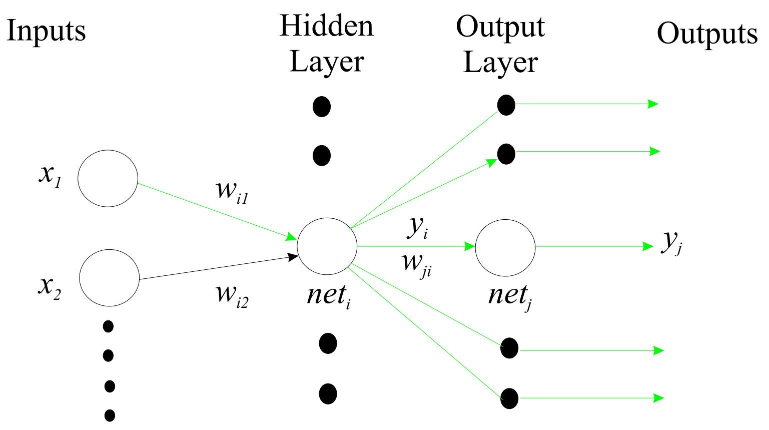


Figure 2.6: The calculation between layers. Now a weight change between node k and node i has an effect on *all* the outputs.

The backpropagation procedure can be summarized in the following steps:

1. Apply the input vector $\boldsymbol{x}$ to the network and forward propagate through the network using (2.25) and either (2.26) or (2.27) to find all the activations of the hidden and output nodes.
2. Evaluate and save the δ_j for all the output nodes using (2.32).
3. Compute the weight changes for the output layer units using (2.33).
4. Using the δ_j computed in step 2 compute and save the δ_i from (2.35).
5. Compute the weight changes for the previous layer using (2.34).
6. These steps are repeated for each layer.

There are many limitations to the backpropagation algorithm. If the learning rate is too large the algorithm may overshoot leading to an increase in the error. By contrast, if the value of the learning rate is too small then the algorithm will be slow to converge, leading to large computation times. Moreover, the algorithm may take many steps to reach a satisfactory local minimum because the local gradient in general does not point towards the direction of the global minimum. In practice, this problem can often be overcome by re-initialising the weights and re-starting training. However, it is interesting to note that, provided the network architecture is appropriate and training ceases at an appropriate stage, under normal circumstances a very good model can be constructed (Hornik et al., 1989; Cybenko, 1989). The rationale for this comes from the fact that there are a large number of error local minima at which the value of the error is close to the global error minima. If it were not for this rather fortuitous fact, Minsky and Papert's conjecture regarding the difficulty of training multi-layer perceptrons may have been realized.

2.5.3 Conjugate gradient descent

Conjugate gradient descent (CGD) has two distinct parts. First we must decide which direction to move in, and second, we must decide how far to move in that direction. By contrast, in steepest gradient descent the direction of each step is given by the local negative gradient of the error function and the step size is given by an arbitrary learning rate parameter. We shall not attempt a detailed justification of CGD here (see (Shewchuk, 1994)), but merely summarise the essential steps.

Starting at iterative step j, consider a minimisation along a search direction $\boldsymbol{d}_j$ in weight space, which is achieved when $\boldsymbol{w}_{j+1}$ is reached. A new direction $\boldsymbol{d}_{j+1}$ is called *conjugate* or *non-interfering* to the old direction $\boldsymbol{d}_j$ if

$$\boldsymbol{d}_{j+1}^T \boldsymbol{H} \boldsymbol{d}_j = 0 \tag{2.36}$$

Here H is the Hessian of second-order partial derivatives evaluated at w_{j+1}. This says the new direction is H-orthogonal to the old direction, i.e. the inner product $< \boldsymbol{d}_{j+1}, H\boldsymbol{d}_j > = 0$. For various reasons it is desirable to search in successively conjugate directions. To achieve this, the gradient $g = \nabla E(\boldsymbol{w})$ of the error surface at the next point must be a minimum in the current search direction $\boldsymbol{d}_j$.

Once the search direction has been found a line search is implemented to minimise the error along this direction. Suppose that at some iteration j of the algorithm we have a weight vector $\boldsymbol{w}_j$ and we wish to consider a particular search direction $\boldsymbol{d}_j$ through weight space. The next weight vector will be given by the minimum along the search direction:

$$\boldsymbol{w}_{j+1} = \boldsymbol{w}_j + \lambda_j \boldsymbol{d}_j \tag{2.37}$$

where the parameter λ_j is chosen so that

$$\boldsymbol{g}_{j+1} \equiv \nabla E(\boldsymbol{w}_j + \lambda_j \boldsymbol{d}_j) = 0 \tag{2.38}$$

is a minimum. This gives an automatic procedure for selecting the step size once the search direction is known. The line search represents a one-dimensional search problem and as such it would be viable to proceed along a search direction in small steps until the error starts to increase. However, it is possible to find much more efficient approaches. Each line proceeds in two stages. The first stage is to bracket the minimum for finding three points $a < b < c$ along the search direction such that $E(a) > E(b)$ and $E(c) > E(b)$. Since the error function is continuous this ensures that the minimum lies between (a, c) (Press et al., 1992). The second stage is to find the minimum itself. Since the error function is smooth and continuous, this can be achieved by parabolic interpolation. Here, a quadratic polynomial is fitted to the error function at three successive points and then moved to the minimum of the parabola, the process is then repeated at the new point. This algorithm was refined in (Brent, 2002) to produce a very robust line search algorithm.

The next problem is how to construct a set of mutually conjugate directions. This can be achieved by selecting the first direction to be the negative gradient $\boldsymbol{d}_1 = -\boldsymbol{g}_1$, and then choosing each successive direction to be a linear combination of the current gradient and the previous search direction

$$\boldsymbol{d}_{j+1} = -\boldsymbol{g}_{j+1} + \beta_j \boldsymbol{d}_j \tag{2.39}$$

where the β_j coefficients can be found using the conjugacy condition to give

$$\beta_j = \frac{\boldsymbol{g}_{j+1}^T(\boldsymbol{g}_{j+1} - \boldsymbol{g}_j)}{\boldsymbol{g}_j^T \boldsymbol{g}_j} \tag{2.40}$$

which is expressed in a Polak-Ribiere form. The Polak-Ribiere form is generally found to give slightly better results than other expressions such as the Hestenes-Stiefel expression and Fletcher-Reeves form (Bishop, 1996). Note that the conjugate gradient algorithm finds the minimum after at most w line minimisations, where w is the dimension of the weight space, without the need to explicitly calculate the Hessian matrix. This represents a significant improvement on the steepest gradient descent methods which can take a large number of steps to converge on a good approximation to the global minima. However, in practice the error function may not necessarily be quadratic and therefore the algorithm may need to run over many iterations until a small enough error is obtained.

The conjugate descent algorithm is summarised in Algorithm 2.

```
{initialisation}
$j \leftarrow 1$
choose an initial weight vector $\boldsymbol{w}_j$
compute the gradient vector $\boldsymbol{g}_j$ at $\boldsymbol{w}_j$
set the initial search direction $\boldsymbol{d}_j = -\boldsymbol{g}_j$
perform line search along $\boldsymbol{d}_j$
compute $\boldsymbol{w}_{j+1}$

{main loop}
**while** the stopping criteria has not been reached **do**
  $j \leftarrow j + 1$
  compute the new gradient vector $\boldsymbol{g}_j$ at $\boldsymbol{w}_j$
  compute the new search direction $\boldsymbol{d}_j = -\boldsymbol{g}_j$
  perform line search along new $\boldsymbol{d}_j$
  compute $\boldsymbol{w}_{j+1}$
**end while**
```

Algorithm 2: The conjugate descent algorithm for neural network training.

2.5.4 BFGS (Quasi-Newton) method

Using the local quadratic approximation, we can obtain directly an expression for the location of the error surface's minimum. From (2.29) the gradient at any point $\boldsymbol{w}$ is given by

$$\boldsymbol{g} \equiv \nabla E(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \tag{2.41}$$

and so the weight vector $\boldsymbol{w}^*$ corresponding to the minimum of the error function satisfies

$$\boldsymbol{w}^* = \boldsymbol{w} - \boldsymbol{H}^{-1}\boldsymbol{g}. \tag{2.42}$$

The vector $-\boldsymbol{H}^{-1}\boldsymbol{g}$ is known as the Newton direction or Newton step, which when evaluated at any $\boldsymbol{w}$ on a quadratic error surface will point directly to the minimum of the error function $\boldsymbol{w}^*$. From (2.42), it is becomes clear that the gradient descent procedure (2.33) is equivalent to one step of the Newton formula (2.42), with the inverse Hessian matrix approximated by the unit matrix times η, where η is the learning rate parameter.

The weight vector of the neural network can be updated by

$$\boldsymbol{w}_{j+1} = \boldsymbol{w}_j - \boldsymbol{H}^{-1}\boldsymbol{g}_j. \tag{2.43}$$

From the Newton formula (2.42) we can see that the weight vectors at steps j and $j+1$ are related to the corresponding gradients by

$$\boldsymbol{w}_{j+1} - \boldsymbol{w}_j = -\boldsymbol{H}^{-1}(\boldsymbol{g}_{j+1} - \boldsymbol{g}_j) \tag{2.44}$$

which is known as the quasi-Newton condition. Because calculating the exact inverse Hessian is computationally expensive, a quasi-Newton approach involves generating a sequence of matrices $\boldsymbol{G}_j$ which represent increasingly accurate approximations to the inverse Hessian $\boldsymbol{H}^{-1}$, using only information of the first derivatives of the error function. The approximation $\boldsymbol{G}$ of the inverse Hessian must be constructed as to satisfy the quasi-Newton condition (2.44).

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) procedure provides a method for iteratively calculating the approximate Hessian matrix $\boldsymbol{G}$ by

$$\boldsymbol{G}_{j+1} = \boldsymbol{G}_j + \frac{\boldsymbol{p}\boldsymbol{p}^T}{\boldsymbol{p}^T\boldsymbol{v}} - \frac{(\boldsymbol{G}_j\boldsymbol{v})\boldsymbol{v}^T\boldsymbol{G}_j}{\boldsymbol{v}^T\boldsymbol{G}_j\boldsymbol{v}} + (\boldsymbol{v}^T\boldsymbol{G}_j\boldsymbol{v})\boldsymbol{u}\boldsymbol{u}^T \tag{2.45}$$

where we have defined the following vectors:

$$\begin{aligned} \boldsymbol{p} &= \boldsymbol{w}_{j+1} - \boldsymbol{w}_j && (2.46)\\ \boldsymbol{v} &= \boldsymbol{g}_{j+1} - \boldsymbol{g}_j && (2.47)\\ \boldsymbol{u} &= \frac{\boldsymbol{p}}{\boldsymbol{p}^T\boldsymbol{v}} - \frac{\boldsymbol{G}_j\boldsymbol{v}}{\boldsymbol{v}^T\boldsymbol{G}_j\boldsymbol{v}} && (2.48) \end{aligned}$$

By using direct substitution of $\boldsymbol{p}$, $\boldsymbol{v}$ and $\boldsymbol{u}$ into (2.45) it can be seen that the BFGS algorithm does indeed satisfy the the quasi-Newton condition (2.44).

Initialising the procedure is achieved by setting $\boldsymbol{G}$ to the identity matrix as this corresponds to taking the first step in the direction of the negative gradient. At each iteration of the algorithm the direction $-\boldsymbol{G}\boldsymbol{g}$ will always be negative because the matrix $\boldsymbol{G}$ is positive definite. However, the full Newton step given in (2.42) may take the search outside the range of validity of the quadratic approximation. The solution is to use a line search (as described for conjugate descent methods) to find the minimum of the error function along a search direction. Therefore, the weight vector is updated using

$$\boldsymbol{w}_{j+1} = \boldsymbol{w}_j + \lambda_j\boldsymbol{G}_j\boldsymbol{g}_j \tag{2.49}$$

where λ_j is found by line minimisation. This guarantees that successive iterations of the algorithm will decrease the error $E(\boldsymbol{w})$.

The advantage the BFGS algorithm has over conjugate descent methods is that the line search does not need to be performed with the greatest accuracy because it does not play a critical role in the algorithm.

The BFGS method is summarised in Algorithm 3.

```
{initialisation}
$j \leftarrow 1$
set the inverse Hessian to the identity matrix $\boldsymbol{G}_j = I$
choose an initial weight vector $\boldsymbol{w}_j$
compute the gradient vector $\boldsymbol{g}_j$

{main loop}
while the stopping criteria has not been reached do
    compute the search direction $\boldsymbol{d}_j = \boldsymbol{G}_j \boldsymbol{g}_j$
    perform line search along $\boldsymbol{d}_j$ to compute $\boldsymbol{w}_{j+1}$
    compute the gradient vector $\boldsymbol{g}_{j+1}$
    update the inverse Hessian $\boldsymbol{G}_{j+1}$ using (2.45)
    $j \leftarrow j + 1$
end while
```

Algorithm 3: The BFGS algorithm for neural network training.

2.5.5 Neural networks for prediction

Within the past ten years there has been an explosion of research applying neural networks to prediction problems (Corcoran et al., 2003; Wilson et al., 2002; Durrant and Jones, 2003; Tsui, 1999; Tsui et al., 2002; Park et al., 1996; Balkin and Ord, 2000; Olson and Mossman, 2003). However, (Chatfield, 1993) argues that most of this research is carried out by computer scientists, who have little or no knowledge of forecasting. He goes on to state that most of the comparisons made between neural networks and traditional statistical approaches are unfairly biased towards the former. If there is a lack of understanding by computer scientists about traditional forecasting techniques, then the reverse is also true. It was shown in (Callen et al., 1996) that linear time series techniques produced better models than those by neural networks over a sample of 296 different time series. However, each series only contained 89 data points which is not enough to train, test and validate a neural network model. Given the small datasets used in this study, there seems little point in using neural networks as a basis for comparison. A comparison between neural networks and the Box-Jenkins approach can be found in (Faraway and Chatfield, 1998) where it is concluded that the Box-Jenkins approach gives a vastly superior model for the air passenger data. However, the BFGS training algorithm is not used to its full potential, something the authors themselves admit.

This is not to say that applying neural networks to time series prediction is without difficulties. One has to carefully consider values for:

- The size of the training set.
- Learning rate.
- Momentum.
- Number of nodes.
- Number of hidden layers.

Finding the optimum values for these parameters may take some degree of patience and luck. This leads many to believe that neural network based time series modelling is more subjective and heuristic than an exact science. However, *provided* the stopping criterion has been correctly identified, i.e. we have a good estimate for the noise as described in the next chapter, and the architecture is appropriate then training a neural network using the BFGS algorithm is normally an almost trivial procedure.

2.6 Local linear regression: an alternative non-parametric modelling technique

In this section, and the following subsection on SVD, we follow the treatment of Durrant (2002).

An alternative to neural networks is Local linear regression. Local linear regression (LLR) performs linear regression through the p_{max} nearest points to a query point to produce a linear model in the locality of that query point. This process is repeated across the training data to produce a piece-wise linear model. Although the function so constructed may not even be continuous (let alone smooth) nevertheless, if the data reflects an underlying smooth process f, then in the limit, as the number of well distributed input data points becomes large, the global function constructed by LLR will converge to f.

We use a fast nearest neighbour search algorithm called the *kd*-tree (described in section 5.1.1) to compute the p_{max} nearest points to a query point. Thus, given a neighbourhood of p_{max} points, we must solve the linear matrix equation

$$\mathbf{Xm} = \mathbf{y} \tag{2.50}$$

in the unknown $\boldsymbol{m}$, where $\mathbf{X}$ is a $p_{max} \times d$ matrix of the p_{max} input points in d-dimensions, $\mathbf{x}_i$ $(1 \leq i \leq p_{max})$ are the nearest neighbour points, $\mathbf{y}$ is a column vector of length p_{max} of the corresponding outputs, and $\mathbf{m}$ is a column vector of parameters that must be determined to provide the optimal mapping from $\mathbf{X}$ to $\mathbf{y}$, such that

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1d} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{p_{max}1} & x_{p_{max}2} & x_{p_{max}3} & \cdots & x_{p_{max}d} \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_d \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{p_{max}} \end{pmatrix} \tag{2.51}$$

The *rank* r of the matrix $\mathbf{X}$ is the number of linearly independent rows, which will affect the existence or uniqueness of solutions for $\mathbf{m}$.

If the matrix $\mathbf{X}$ is square and non-singular then the unique solution to (2.50) is $\mathbf{m} = \mathbf{X}^{-1}\mathbf{y}$. If $\mathbf{X}$ is not square or singular then we modify (2.50) and attempt to find a vector $\mathbf{m}$ which minimises

$$|\mathbf{Xm} - \mathbf{y}|^2 \tag{2.52}$$

As was proved by Penrose the unique solution to this problem is provided by $\mathbf{m} = \mathbf{X}^{\#}\mathbf{y}$ where $\mathbf{X}^{\#}$ is the pseudo-inverse matrix (Penrose, 1955, 1956).

For any given matrix $\mathbf{X} \in \mathbb{R}^{p_{max} \times d}$, the matrix $\mathbf{X}^{\#} \in \mathbb{R}^{d \times p_{max}}$ is said to be a *pseudo-inverse* of $\mathbf{X}$ if the following conditions are satisfied

$$\begin{aligned} \mathbf{XX}^{\#}\mathbf{X} &= \mathbf{X} \\ \mathbf{X}^{\#}\mathbf{XX}^{\#} &= \mathbf{X} \\ (\mathbf{XX}^{\#})^T &= \mathbf{XX}^{\#} \\ (\mathbf{X}^{\#}\mathbf{X})^T &= \mathbf{X}^{\#}\mathbf{X} \end{aligned} \tag{2.53}$$

where T denotes the transpose of the matrix. The terms *generalised inverse* or *Moorse-Penrose inverse* are also commonly used for such an $\mathbf{X}^{\#}$.

If $\mathbf{X}$ is square and non-singular then $\mathbf{X}^{\#}$ is just the inverse matrix $\mathbf{X}^{-1}$. In practice, the computation of $\mathbf{X}^{\#}$ is modestly demanding for large matrices. There are many algorithms for approximating pseudo-inverses (Kerr, 1985) and (Penrose, 1955).

A generalised technique to solve (2.50) for $\mathbf{m}$ is *singular value decomposition* (SVD), which is a computationally expensive but a widely accepted technique for its accuracy. Both (Press et al., 1992) and (Cherkassky and Mulier, 1998) provide good introductions to linear algebra and SVD, especially in the wider context of learning from data. For the purpose of this discussion of modelling techniques, we shall focus on introducing SVD for local-linear regression and side-step the more general subject of linear algebra.

2.6.1 Singular value decomposition

(Tsui, 1999) has taken the separate theories given in (Press et al., 1992)[4] to provide a unified account of SVD, where the context of the discussion is similar to this thesis. The detail contained within that thesis will not be replicated here.

SVD is based on a generalisation in linear algebra that any symmetric matrix can be diagonalised via an orthogonal transformation. This leads to a technique to obtain the inverse of a non-singular square matrix, in this case $\mathbf{X}^{-1}$. SVD also solves the linear least squares approximation (2.52) without requiring $\mathbf{X}$ to be non-singular or even square.

From (2.50), $\mathbf{X}$ is an $p_{max} \times d$ matrix that can be written, using a standard theorem of linear algebra, as

$$\mathbf{X} = \mathbf{U}\mathbf{W}\mathbf{V}^T \tag{2.54}$$

where $\mathbf{U}$ is a $p_{max} \times d$ orthogonal[5] matrix, $\mathbf{V}$ is a $d \times d$ orthogonal matrix, and $\mathbf{W}$ is a $d \times d$ matrix with positive or zero elements (singular values) w_j, such that $w_1 \geq w_2 \geq \ldots \geq w_r > 0$, where r is the rank of $\mathbf{X}$. Then rewriting $\mathbf{W}$

$$\mathbf{W} = \left(\begin{array}{cc} A & 0 \\ 0 & 0 \end{array} \right) \tag{2.55}$$

where

$$\mathbf{A} = diag(w_1, \ldots, w_r) \tag{2.56}$$

provides a definition for the pseudo-inverse $\mathbf{X}^{\#}$

$$\mathbf{X}^{\#} = \mathbf{V} \left(\begin{array}{cc} A^{-1} & 0 \\ 0 & 0 \end{array} \right) \mathbf{U}^T \tag{2.57}$$

[4](Press et al., 1992) focus on providing an algorithm rather than defining the theory behind SVD in a contained manner.

[5]A matrix is orthogonal if its inverse equals its transpose.

Then to find a solution for $\mathbf{m}$ that minimises (2.52), let

$$\begin{aligned} \mathbf{m} &= \mathbf{X}^{\#}\mathbf{y} \\ &= \mathbf{V}\begin{pmatrix} A^{-1} & 0 \\ 0 & 0 \end{pmatrix}\mathbf{U}^T\mathbf{y} \end{aligned} \tag{2.58}$$

where

$$\mathbf{A}^{-1} = diag(1/w_1, \ldots, 1/w_r) \tag{2.59}$$

The inverse of the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ are their own transpose (i.e. $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ and $\mathbf{V}\mathbf{V}^T = \mathbf{I}$). Hence the process of finding $\mathbf{m}$ using (2.58) is trivial once $\mathbf{X}$ is decomposed using (2.54). It is this decomposition of $\mathbf{X}$ to generate $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$ that now needs explanation.

The columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{X}\mathbf{X}^T$, the columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{X}^T\mathbf{X}$, and the singular values on the diagonal of $\mathbf{W}$ are the square roots of the eigenvalues of $\mathbf{X}\mathbf{X}^T$ or $\mathbf{X}^T\mathbf{X}$ (they have the same eigenvalues). The process of extracting the eigenvectors and eigenvalues is more difficult to explain and beyond the scope of this general introduction to modelling. The full computational process to perform the decomposition of $\mathbf{X}$ given in (2.54) is provided in (Tsui, 1999) and (Press et al., 1992) provide an algorithm written in C. SVD is also available in *Mathematica*.

2.7 Chapter summary

In this chapter we first introduced time series and some of the classical tools and classifications used in conventional parametric time series modelling. We then went on to discuss some appropriate tools for non-parametric modelling of non-linear time series, principally neural networks and local linear regression.

Overall, LLR or neural networks compared with traditional parametric prediction methods should not be seen as rivals, but rather as two different approaches in the time series analyst's toolbox.

If it is clear that a time series is linear, or can easily be reduced to a linear form, then traditional linear prediction methods may provide us with a good parsimonious model. However, when we suspect complex non-linearities are at play, it seems that neural networks or LLR may be our best hope of achieving a suitable model. We shall have cause to use both in later parts of this thesis.

Chapter 3

The Gamma test

In this chapter we give an overview of the Gamma test and associated techniques. Section 3.5.2 is new and represents joint work with Antonia J. Jones. Given an input/output dataset

$$\{x_1(i), ..., x_m(i), y_i\} = \{(\boldsymbol{x}_i, y_i) | 1 \leq i \leq M\} \tag{3.1}$$

where the vector $\boldsymbol{x} = (x_1, ..., x_m) \in \mathbb{R}^m$ contains the inputs and the corresponding scalar y_i is the output, M is the number of data points, and m denotes the number of input variables, we attempt to model the relationship between an input $\boldsymbol{x}$ and the corresponding output y by

$$y = f(x_1, \ldots, x_m) + r \tag{3.2}$$

where f is a *smooth* function, and r is a random variable representing the noise with a

- distribution variance σ^2,
- and sample variance $\hat{\sigma}^2_M$.

Given sufficient data, the Gamma test (Stefánsson et al., 1997) is a technique that can efficiently estimate the sample variance of the noise variable $\hat{\sigma}^2_M$. Since $\hat{\sigma}^2_M \to \sigma^2$ in probability as $M \to \infty$, it follows that the Gamma test asymptotically estimates σ^2. Moreover, it is capable of making this estimation *directly from the data* without making any assumptions regarding the parametric form of the equations governing the system. To this end, the only requirement is that the system is *smooth*[1].

[1]If the rule determining y from $\boldsymbol{x}$ is not smooth the Gamma test will return an estimate for σ^2 close to the variance of the signal y.

We begin this Chapter by outlining some previous work into data-derived noise estimates in Section 3.1. In Section 3.2 we discuss the applications of having an efficient noise estimate before detailing the Gamma test in Section 3.3.

The Gamma test algorithm[2] calculates the noise variance using a p number of nearest neighbours in the input space. In the majority of situations $p = 10$ usually gives a good estimate of the noise variance, however, there is no *guarantee* that $p = 10$ will give the best estimate. Section 3.4 is devoted to discussing this problem and introduces a method that can help determine a suitable range of p that provides a good Gamma test estimate of the noise variance.

Finally, Sections 3.5 and 3.6 discuss some non-linear data analysis tools based on the Gamma test.

3.1 (ϵ, δ)-methods

Before looking at the Gamma test we briefly look over some other data-derived noise estimation techniques, which we classify as (ϵ, δ)-methods.

3.1.1 Difference-based estimators for non-autonomous time series

We may write a *non-autonomous* time series with additive noise as

$$y_t = f(t) + r_t \quad (0 \leq t \leq M) \tag{3.3}$$

where f is a smooth function and the r_t's are IID with mean zero and variance σ^2. Techniques used to estimate the variance of r in these non-autonomous time series exploit differences to remove trend in the mean function, an idea that originated from time series analysis. Here we assume that t is univariate and $0 \leq t_1 \leq \cdots \leq t_M$. Rice (Rice, 1984) proposes a first order difference-based estimator

$$\hat{\sigma}^2_{Rice} = \frac{1}{2(M-1)} \sum_{t=2}^{M} (y_t - y_{t-1})^2 \tag{3.4}$$

[2]A simple introductory tutorial can be found in (Kemp et al., 2004).

However, by taking expectation to the Rice estimator,

$$\begin{aligned}\mathcal{E}(\hat{\sigma}^2_{Rice}) &= \frac{1}{2(M-1)}\sum_{t=2}^{M}\mathcal{E}((y_t - y_{t-1})^2) \\ &= \sigma^2 + \frac{1}{2(M-1)}\sum_{t=2}^{M}\mathcal{E}((f(t) - f(t-1))^2) \qquad (3.5)\end{aligned}$$

we find that it is always positively biased. In contrast the Gamma test exploits the bias of several estimators to yield a single *unbiased* estimate.

Example. Consider $t = i/800$ for $1 \leq i \leq 800$ and $y_t = 5\sin(8\pi t) + r_t$ where $\sigma^2 = 0.5$, and $\hat{\sigma}^2 = 0.49855$, a plot of which is shown in Figure 3.1. The Rice estimator for $\hat{\sigma}^2$ is 0.503276 i.e. very close to the true noise variance. However, by reducing the sampling rate to $t = i/100$ for $1 \leq i \leq 100$ (see Figure 3.2), so that $\hat{\sigma}^2_{100} = 0.46422$, we find the Rice estimator for σ^2 is 0.7672061 i.e. the estimator is positively biased by the variance of function differences.

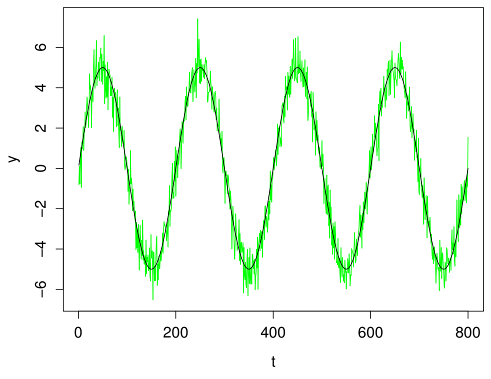


Figure 3.1: The noisy signal $y_t = 5\sin(8\pi t) + r_t$ is shown in green, the black line is the 'true' signal. In this example $M = 800$ and $\sigma^2 = 0.5$.

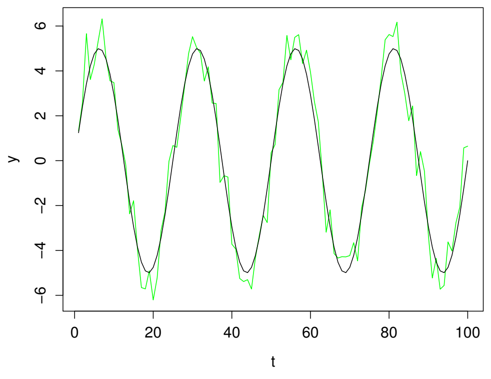


Figure 3.2: The same $y_t = 5\sin(8\pi t) + r_t$ noisy signal, but here $M = 100$.

A similar method called the GSJ estimator (Gasser et al., 1986) proposed the following second order differenced based estimator

$$\hat{\sigma}^2_{GSJ} = \frac{2}{3(M-2)}\sum_{i=2}^{n-1}\left(\frac{1}{2}y_{i-1} - y_i + \frac{1}{2}y_{i+1}\right)^2 \qquad (3.6)$$

which aims is to minimize the effect of bias from the function. In the above example where $M = 100$, the GSJ estimator for σ^2 is 0.524, which is much better than 0.7672061 given by the Rice estimator.

3.1.2 The δ-test

The δ-test (Pi and Peterson, 1994) is a noise estimation approach based on calculating conditional probabilities from vector component distances. Given any two points $\boldsymbol{x}'$ and $\boldsymbol{x}$ in the input space, and their associated outputs y' and y the conditional expected value of $\frac{1}{2}(y'-y)^2$, on the hypothesis that the associated input points $\boldsymbol{x}'$ and $\boldsymbol{x}$ are located within a distance δ of each other, converges to σ^2 as δ approaches zero, i.e.

$$\mathcal{E}\left(\frac{1}{2}(y'-y)^2 \middle| |\boldsymbol{x}'-\boldsymbol{x}| < \delta\right) \to \sigma^2 \text{ as } \delta \to 0 \tag{3.7}$$

The δ-test is based on the fact that for any particular value of δ, the expectation in (3.7) can be estimated by the sample mean

$$E(\delta) = \frac{1}{|I(\delta)|} \sum_{(i,j)\in I(\delta)} \frac{1}{2}|y_j - y_i|^2 \tag{3.8}$$

where

$$I(\delta) = \{(i,j) | |\boldsymbol{x}_j - \boldsymbol{x}_i| < \delta, 1 \le i \ne j \le M\} \tag{3.9}$$

is the set of index pairs (i,j) for which the associated points $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ are located within δ distance of one another.

At first glance, (3.8) suggests that computing the sample mean $E(\delta)$ for δ sufficiently small will provide a good estimate for σ^2. However, given finitely many data points, the number of pairs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ satisfying $|\boldsymbol{x}_j - \boldsymbol{x}_i| < \delta$ decreases as δ decreases. Choosing a small δ value implies that the sample mean $E(\delta)$ is computed over a small number of y_i and y_j pairs and therefore $E(\delta)$ is subjected to significant *sampling error*. Therefore, any δ must be chosen sufficiently large, say $\delta > \delta_0$, to ensure that the sample mean $E(\delta)$ provides a good expectation in (3.8). Clearly, if δ is chosen too large this will restrict the effectiveness of $E(\delta)$ as an estimate for σ^2.

A major problem with the δ-test, is that the computational cost of finding the index list (3.9) is of order $O(M^2)$. This means that using the δ-test for anything beyond that of a noise estimator (e.g. input variable selection) is infeasible in a reasonable amount of computing time.

3.2 Applications of data-derived noise estimates

Before describing the Gamma test in detail we outline some important applications of techniques capable of estimating σ^2.

- These techniques provide a method for assessing the 'quality' of the data.

If the estimated noise variance is small compared to the variance of the output y then it is likely that the output is determined by the inputs by some smooth model. Data derived noise estimation techniques can also tell us whether or not we have sufficient data to form a smooth non-linear model, and also indicate how good that model is likely to be. If the error of prediction is too high regardless of the number of data points, we may consider increasing the accuracy of our measurements or alternatively question whether we have included all the inputs that are likely to affect the output.

- These techniques also provide a method for determining a suitable stage at which to stop adapting a model to fit the data.

One of the major problems of non-parametric model construction is to determine when to stop adapting the model to fit the data. The *mean squared error* between the actual output values and those predicted by the model provide one indicator of how well the model fits the training data. If we fit a model beyond the point at which the mean squared error falls significantly below the noise level, we will have incorporated some of the noise into the model itself. This is highly undesirable since the model will perform poorly on previously unseen inputs despite the fact that its performance on the training set may have been perfect. Such models have memorised the training data and are consequently said to suffer from *overfitting.* If we stop adapting the model when the mean squared error of the training set reaches the estimate of the noise variance then we should obtain a smooth model having the best possible mean squared error.

- Data-derived estimates provide a criterion for determining the most relevant input variables.

In this context, the goal of *input variable selection* is to choose a selection of input variables from a number of candidate variables that best describes the model. That data-derived methods may prove useful in this respect derives from the insight that low noise levels can only be encountered when all the relevant predictive[3] factors that determine the output have been included in the input. Some input variables may be irrelevant, while others may be subject to high measurement error so that incorporating them into our model may be counter-productive, i.e. they may lead to a higher *effective* noise level on the output.

[3]Such factors may or may not be causal. In an ideal world we would be measuring all causal variables, but in practice this is often not possible.

3.3 The Gamma test

Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ be any two points in the input space C and r, r' the associated noise on the corresponding outputs. By (3.2) we have $y = f(\boldsymbol{x}) + r$ and $y' = f(\boldsymbol{x}') + r'$, and therefore

$$\frac{1}{2}(y - y')^2 = \frac{1}{2}\left((r - r') + (f(\boldsymbol{x}) - f(\boldsymbol{x}'))\right)^2 \tag{3.10}$$

The continuity of f implies that

$$|f(\boldsymbol{x}) - f(\boldsymbol{x}')| \to 0 \qquad \text{as} \qquad |\boldsymbol{x} - \boldsymbol{x}'| \to 0 \tag{3.11}$$

therefore, by (3.10) we obtain

$$\frac{1}{2}(y - y')^2 \to \frac{1}{2}(r - r')^2 \qquad \text{as} \qquad |\boldsymbol{x} - \boldsymbol{x}'| \to 0 \tag{3.12}$$

Since the expectation of r is zero and r and r' are assumed to be independent and identically distributed we have

$$\mathcal{E}\left(\frac{1}{2}(r - r')^2\right) = \sigma^2 \tag{3.13}$$

Hence, taking expectations of both sides in (3.12) it follows that

$$\mathcal{E}\left(\frac{1}{2}(y - y')^2\right) \to \sigma^2 \qquad \text{as} \qquad |\boldsymbol{x} - \boldsymbol{x}'| \to 0 \tag{3.14}$$

In the limit, if $\boldsymbol{x} = \boldsymbol{x}'$ then (3.14) implies $\mathcal{E}(\frac{1}{2}(y - y')^2) = \sigma^2$. So we look for points $\boldsymbol{x}'$ close to $\boldsymbol{x}$ and use the associated y's in order to estimate σ^2.

The Gamma test algorithm firstly constructs the kth $(1 \leq k \leq p)$ nearest neighbour lists $\boldsymbol{x}_{N[i,k]}(1 \leq i \leq M)$ of the input vectors $\boldsymbol{x}_i(1 \leq i \leq M)$. Here, p is set before the algorithm begins, typically $p \approx 10$. The construction of the near neighbour lists can be achieved in $O(M \log M)$ time using a kd-tree (Bentley, 1975). The algorithm then computes the following statistics

$$\delta_M(k) = \frac{1}{M}\sum_{i=1}^{M} |\boldsymbol{x}_{N[i,k]} - \boldsymbol{x}_i|^2 \qquad\qquad (1 \leq k \leq p) \tag{3.15}$$

where $|.|$ is the Euclidean distance, and

$$\gamma_M(k) = \frac{1}{2M}\sum_{i=1}^{M} (y_{N[i,k]} - y_i)^2 \qquad\qquad (1 \leq k \leq p) \tag{3.16}$$

where $y_{N[i,k]}$ is the output value associated with $\boldsymbol{x}_{N[i,k]}$. The relationship between $\delta_M(k)$ and $\gamma_M(k)$ $(1 \leq k \leq p)$ is *approximately linear* (Evans and Jones, 2002) for sufficiently small p. Therefore, regressing $\gamma_M(k)$ on $\delta_M(k)$ $(1 \leq k \leq p)$ gives an estimate for the variance of the noise, σ^2, at the intercept $\delta = 0$. This estimate is denoted by Γ. The main result of (Evans and Jones, 2002) is that Γ converges in probability to σ^2 as $M \to \infty$.

If Γ is large (compared to the variance of the output, y) then it is not likely that the outputs are determined by the inputs from a smooth model, whereas if the Γ is small or close to zero this becomes more likely (Jones, 2004). The V_{ratio}, is a useful statistic which compares the (estimated) variance of the noise to the variance of the output. Hence, we have

$$V_{ratio} = \frac{\Gamma}{\mathrm{Var}\,(y)} \tag{3.17}$$

If the $V_{ratio} \approx 1$ then this indicates that the underlying model is not smooth. Whereas, $V_{ratio} \approx 0$ indicates the data is derived from a smooth model. The benefit of such a statistic is that it is scale-independent.

The mathematical proof of the Gamma test algorithm (Evans and Jones, 2002; Evans et al., 2002) only holds under certain conditions. These conditions fall into three categories; (i) conditions of the class of models f; (ii) conditions of the noise distribution Ψ and (iii) the conditions of the sampling distribution Φ.

The conditions below were required to formulate the proof of the Gamma test algorithm (Evans and Jones, 2002; Evans et al., 2002). However, (Evans, 2002) suggests that the algorithm continues to work under less restrictive conditions.

3.3.1 Class of models f

The class of possible models $f : \mathbb{R}^m \to \mathbb{R}$ is restricted to the class of continuous functions having bounded first and second partial derivatives over the input space $C \subset \mathbb{R}^m$. In particular the Gamma test cannot readily be applied to problems involving categorical data.

More formally, let $\nabla f(\boldsymbol{x}) = (\partial f/\partial x_1, \ldots \partial f/\partial x_m)$ be the gradient vector and $Hf(\boldsymbol{x}) = (\partial^2 f/\partial x_i \partial x_j)$ denote the Hessian matrix of second order partial derivatives. The Gamma test analysis requires that f be well behaved and continuous, the partial derivatives exist, and that ∇f and Hf are bounded over a convex, closed bounded set $C \subset \mathbb{R}^m$.

3.3.2 The noise distribution Ψ

The noise r is defined to be a random variable representing the component of the output which cannot be accounted for by a smooth transformation of the inputs. It is assumed that the noise distribution, Ψ, has a mean zero. As the aim of the Gamma test is to estimate the variance of the noise, r, we obviously require that this variance is finite. For technical reasons outlined in (Evans, 2002) we also require the third and fourth moments of the noise to be finite. Moreover, the noise has to be *independent* of the input corresponding to the output on which it is measured. It is usually further assumed that the noise is homogeneous over the input space. Finally, the noise values r_i and r_j on the two outputs y_i and y_j $(i \neq j)$ are independent and in particular uncorrelated, so that $\text{cov}(r_i, r_j) = 0 \, \forall \, i \neq j$ and $\text{cov}(r_i, r_i) = \sigma^2 \, \forall \, i$, where σ^2 is the variance of r.

3.3.3 The sampling distribution Φ

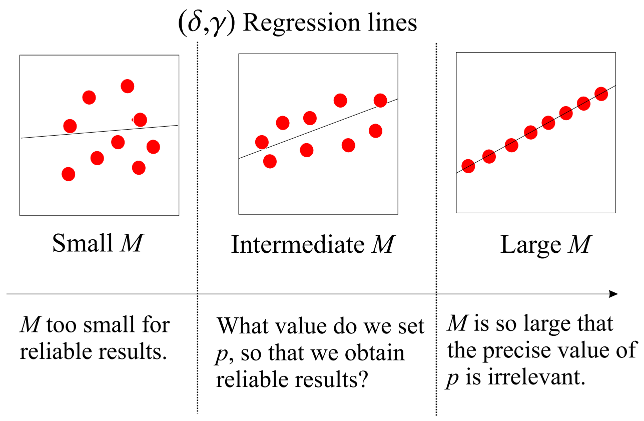
It is assumed that the inputs are selected from the set of real values $\mathbb{R}^m$ by some sampling distribution Φ, and we denote the corresponding density function by ϕ. The input space is then defined to be the set

$$C = \{\boldsymbol{x} \in \mathbb{R}^m | \phi(\boldsymbol{x}) > 0\} \tag{3.18}$$

The formal analysis assumes that the sampling density ϕ over the convex closed bounded set C is a smooth positive function, which implies the near neighbour distances become progressively smaller as the number of data points increases.

3.4 Determining the number of near neighbours for a Gamma test

The literature suggests that in a typical Gamma test the number of near neighbours used p, should be set to ten (Jones et al., 2002; Wilson et al., 2004; Stefánsson et al., 1997; Jones et al., 1998). This is fine if we have enough data with respect to the curvature of the underlying function, but this will not always be the case. The problem is we simply do not know whether we have a suitable amount of data to run a Gamma test on $p = 10$. Figure 3.3 shows this problem graphically.



(δ,γ) Regression lines
Small M
Intermediate M
Large M
M too small for reliable results.
What value do we set p, so that we obtain reliable results?
M is so large that the precise value of p is irrelevant.

Figure 3.3: The problem of selecting p. If M is small the Gamma test will not give a good estimate for σ^2, and if M is large the value p is substantially irrelevant. However, there is an intermediate range within which it is desirable to know the optimal value of p.

3.4.1 Background

If the underlying function f is non-linear then selecting a 'large' p will prove counter productive because the linear regression of the $(\delta_M(k), \gamma_M(k))$ $(1 \leq k \leq p)$ points may suffer from *non-linearity error*. Mathematically, this nonlinearity error arises from equation (3.10), i.e.

$$\mathcal{E}\left(\frac{1}{2}(y' - y)^2\right) = \sigma^2 + \mathcal{E}\left(\frac{1}{2}(f(\boldsymbol{x}') - f(\boldsymbol{x}))^2\right) \tag{3.19}$$

where

$$f(\boldsymbol{x}') - f(\boldsymbol{x}) \approx (\boldsymbol{x}' - \boldsymbol{x}).\nabla f(\boldsymbol{x}) + (\boldsymbol{x}' - \boldsymbol{x})^T.\boldsymbol{H}f(\boldsymbol{x}).(\boldsymbol{x}' - \boldsymbol{x}) \tag{3.20}$$

to second order in $(\boldsymbol{x}' - \boldsymbol{x})$. Because the Gamma test involves a linear regression on $\delta_M(k)$, essentially deriving from the first term on the RHS of (3.20), the 'nonlinearity error' involved in a Gamma test estimate of σ^2 clearly depends on:

(a) The number of data points M and/or;

(b) the nature of f.

Given reasonable assumptions on the sampling density, since the set C is a closed and bounded, M controls the local density of data points. If M is very large, the local linearity of the $(\delta_M(k), \gamma_M(k))$ regression line is essentially trivial and p can be taken quite large without degrading the estimate. However, p must stay bounded otherwise the run times can become too long.

As M is decreased the local density of points decreases and the non-linearities of f become an issue. This is where the first and second derivatives[4] show that the higher the average local curvature of the function $f = f(x_1, ..., x_m)$, the higher M needs to be in order to ensure that the local density of points is sufficient to get a good (δ, γ) regression line. However, we often do not have the luxury of increasing M whenever we have a very non-linear function, therefore the best alternative is to decrease p so that the neighbours $(\boldsymbol{x}', \boldsymbol{x}'', \ldots, \boldsymbol{x}^{(p)})$, in question are not too far away from the point $\boldsymbol{x}$ and the function f does not change too much in the locality.

Example. Consider $y = f(x) = x^2 + r$, where x is taken from a uniform distribution in the range $[-1, 1]$, $\sigma^2 = 0.05$, $M = 150$, and $\hat{\sigma}^2_{150} = 0.0478$. We define the Gamma error, Γ_{error}, to be:

$$\Gamma_{error} = \Gamma(p, M) - \hat{\sigma}^2_M \tag{3.21}$$

where $\Gamma(p, M)$ is the Gamma statistic return for a specified p and $\hat{\sigma}^2_M$ is the sample noise variance. For $M = 150$ Figure 3.4 shows the Γ_{error} over increasing p. For $p < 40$ the error is relatively small, but as p increases beyond that point the non-linearity effects cause the error to progressively increase. Indeed, viewing the regression plot in Figure 3.5 of the Gamma test for $p = 149$ it is clear that the relationship between $(\delta_M(k), \gamma_M(k))$ pairs for relatively large p is *not* linear.

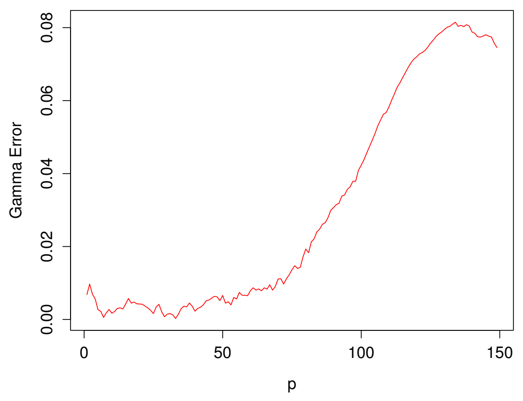


Figure 3.4: The Γ_{error} over increasing p for $y = x^2 + r$.

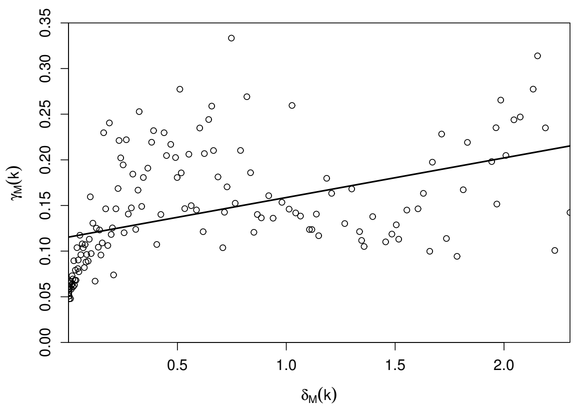


Figure 3.5: The regression fit between $(\delta_M(k), \gamma_M(k))$ pairs when $p = 149$ for $y = x^2 + r$.

Other factors that influence the optimal value of p will be:

(a) The noise level σ^2.

[4]The partial derivatives in question are the partials w.r.t to the input variables.

(b) The precise nature of the noise distribution.

In a Gamma test analysis the minimum we take can p to be is 2, since this is the minimum required number of points to calculate a regression line. However, one could use $\frac{1}{2M}(y' - y)^2$ i.e. $p = 1$. Indeed, if the underlying function is very 'curvy' or M is too small to get a good (δ, γ) regression fit then the best estimate we can hope to achieve is when $p = 1$.

3.4.2 Assessing the Gamma regression line

For brevity we now surpress the use of the indexing on $(\delta_M(k), \gamma_M(k))$ $(1 \leq k \leq p)$ points and simply refer to them as (δ, γ) pairs.

The mean squared error of the estimated regression line should provide an intuitive means to measure the quality of linear fit between the (δ, γ) pairs i.e.

$$\text{MSE}_{\delta\gamma} = \frac{1}{(p-2)} \sum_{k=1}^{p} (\gamma_M(k) - (\Gamma + A\delta_M(k)))^2 \tag{3.22}$$

We aim to 'track' Γ_{error} using $\text{MSE}_{\delta\gamma}$, so that a suitable range of p for a given dataset is provided. Our hypothesis is that the p at which the $\text{MSE}_{\delta\gamma}$ begins to rise will provide us with the upper-bound for p, going beyond this point will be counter-productive since the non-linearity errors will become evident in the (δ, γ) regression fit, in turn leading to a poor estimate for σ^2.

Example. Consider $y = \sin(40\pi x) + r$ where x is taken from uniform distribution over the range $[-1, 1]$, and r is Gaussian with $\sigma^2 = 0.05$, $M = 5000$, and $\hat{\sigma}^2_{5000} = 0.0493$. Figure 3.6 shows the input and output.

We compute Γ_{error} for $M = 100$, $M = 1000$ and $M = 5000$ along with the $\text{MSE}_{\delta\gamma}$ for increasing p. Figure 3.8 shows the results.

As can be seen the $\text{MSE}_{\delta\gamma}$ does a good job of 'tracking' the Γ_{error}. When $M = 100$ we see that $\text{MSE}_{\delta\gamma}$ rises very quickly, which indicates that there is not really enough data for a Gamma test to provide a reliable estimate for $\hat{\sigma}^2_{100}$. Here the best estimate for $\hat{\sigma}^2_{100}$ is obtained from $p = 1$. In the case where $M = 1000$ the $\text{MSE}_{\delta\gamma}$ is fairly flat for $p \in [3, 12]$ and the Γ_{error} is also small within this range. Finally when $M = 5000$, the $\text{MSE}_{\delta\gamma}$ suggests we can afford to take p quite large (i.e. $3 < p < 45$) to estimate $\hat{\sigma}^2_{5000}$. We also notice from the graphs in Figure 3.8 that as we increase M the Γ_{error} decreases.

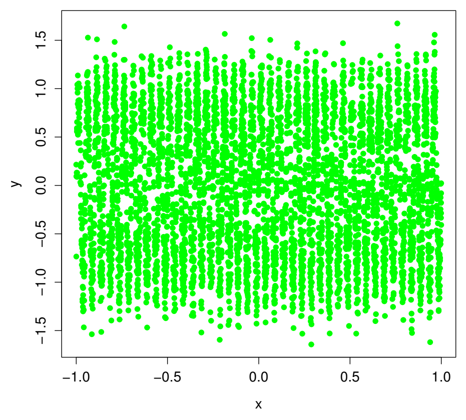


Figure 3.6: The high frequency sine data (with large mean squared gradient over $[-1, 1]$), where $y = \sin(40\pi x) + r$. In fact $\mathcal{E}(|\nabla y|^2) = 800\pi^2 \approx 7895.68$.

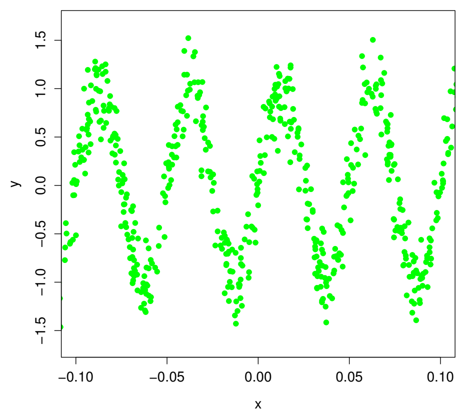


Figure 3.7: A close up of the high frequency sine data, where x is taken over the range $[-0.1, 0.1]$.

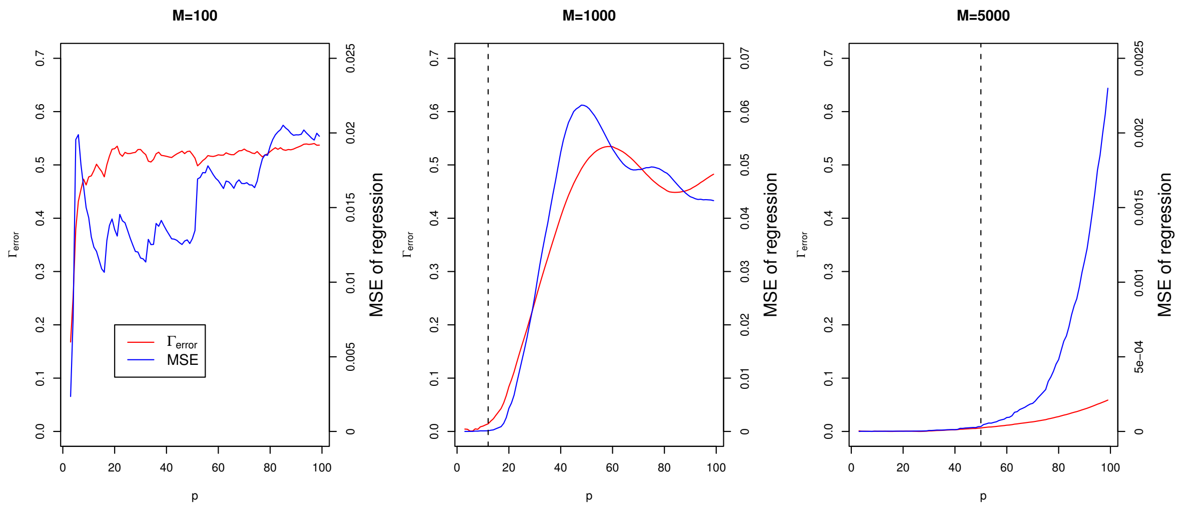


Figure 3.8: The Γ_{error} (red) and $\text{MSE}_{\delta\gamma}$ (blue) for $y = \sin(40\pi x) + r$ on small, mid-range and large M. The dashed lines indicate the upper bound on p for the data as suggested by the $\text{MSE}_{\delta\gamma}$.

3.5 Data analysis tools based on the Gamma test

Using a Gamma test to gain an efficient estimate of the noise variance has led to the development of a collection of more sophisticated non-linear data analysis routines including

- Finding the minimum number of data points required to construct a smooth model (the *M-test*).
- Methods for identifying predictively useful variables (e.g. the *full embedding* technique).

3.5.1 The M-test

The proof of the Gamma test (Evans et al., 2002; Evans and Jones, 2002) shows that as $M \to \infty$ the estimate Γ will converge in probability on the distribution noise variance, σ^2.

However, we really need to know how *quickly* the estimate returned by the algorithm will stabilise to a close approximation of the noise variance. One simple method for accomplishing this is to compute the Γ statistic for increasing M. By plotting the Γ values over M it can be seen whether the graph appears to be approaching a stable asymptote[5].

It is worth noting that previous literature has tended to stress that the Gamma test was estimating the distribution noise variance σ^2. However, close examination of the relevant equations (Evans and Jones, 2002) shows that in the first instance the Gamma test is estimating the sample variance $\hat{\sigma}_M^2$, which in turn approaches σ^2 in probability as $M \to \infty$.

Example. Take $y = x^2 + r$, where x is a random variable drawn from a uniform distribution over the range $[-1, 1]$, r is a Gaussian random variable with mean zero and $\sigma^2 = 0.075$. We take $M = 500$ and obtain $\hat{\sigma}_{500}^2 = 0.06404472$. From the M-test graph in Figure 3.9 on the (x, y) dataset, we see that the Gamma test is primarily estimating $\hat{\sigma}_M^2$ as opposed to σ^2 (the black dashed line in the figure). Indeed, the estimate $\Gamma = 0.06448444$ is close to $\hat{\sigma}_{500}^2$.

[5]It is easier to do this by eye than to define a reliable and efficient algorithm.

However, as M increases we expect $\hat{\sigma}^2_M \to \sigma^2$. Using the same fixed process[6] as above, but this time setting $M = 1000$, the sample noise variance is 0.07503981 i.e. close to $\sigma^2 = 0.075$. The M-test graph in Figure 3.10 shows that both $\hat{\sigma}^2_M$ and Γ are converging on σ^2; the returned estimate $\Gamma = 0.07530238$ is now close to σ^2.

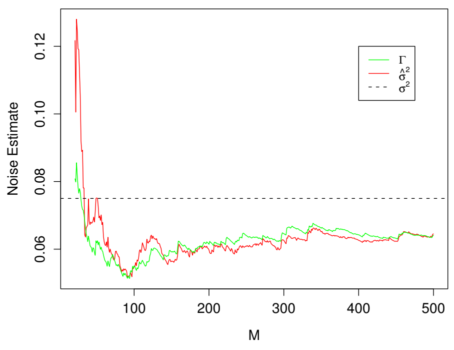


Figure 3.9: The M-test graph for $y = x^2 + r$, where $M = 500$ and $\sigma^2 = 0.075$ (dashed line). The Γ (green line) is a close estimate to $\hat{\sigma}^2_M$ (red line).

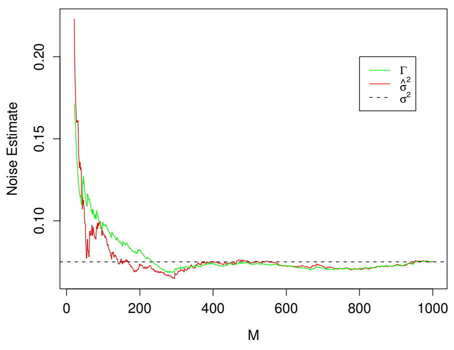


Figure 3.10: The M-test graph for $y = x^2 + r$, where $M = 1000$ and $\sigma^2 = 0.075$. This time both Γ and $\hat{\sigma}^2_M$ are a close estimate to σ^2.

3.5.2 Heuristic confidence intervals for the Gamma test

In the previous section we showed that in the first instance the Gamma test estimates the sample noise variance $\hat{\sigma}^2_M$. In this section we introduce a heuristic technique for estimating confidence intervals for the Gamma test i.e. for a given confidence level a technique for estimating the interval in which the true value of $\hat{\sigma}^2_M$ lies.

We first empirically examine the distribution of Gamma values evaluated for the same input/output function and noise distribution, but where M is fixed and the data set of M samples is varied.

The Gamma distribution for fixed M

We imagine that for a fixed process and fixed M we are able to construct many input/output data sets of size M. For each data set we compute a Γ statistic. Now we ask: *What is the distribution of* Γ *over the set of all possible data sets of size*

[6]i.e. the same f and the same noise distribution.

M? Although this distribution is theoretically somewhat inaccessible we can easily construct an experiment to get an approximate histogram for artificial data.

Consider the process $y = \sin(x) + r$, where x is uniform on $[0, 2\pi]$, r is a Gaussian random variable with mean zero and $\sigma^2 = 0.075$, and $M = 1000$. We generate 1000 different datasets of this fixed process and compute a Gamma statistic for each. A plot of the Γ histogram for $M = 1000$ is shown in Figure 3.12, as a comparison the histogram for $M = 500$ is shown in Figure 3.11. For $M = 1000$, the mean Gamma statistic over the 1000 samples is 0.07493139 with a standard deviation of 0.003740105. From Figures 3.11 and 3.12 it seems reasonable to assume that the Γ distribution is tending to normality as M increases. Based on this assumption we can apply the Student's t-test to give a confidence interval of (0.07473666, 0.07512611) at the 90% level. The interval becomes larger as the confidence level increases e.g. at the 99% level the interval is (0.07462616, 0.07523662).

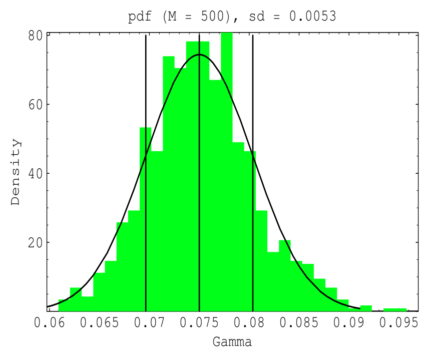


Figure 3.11: The Γ histogram of 1000 samples for $M = 500$ with an overlay plot of the corresponding normal distribution. The mean 0.075 and ± one standard deviation are shown.

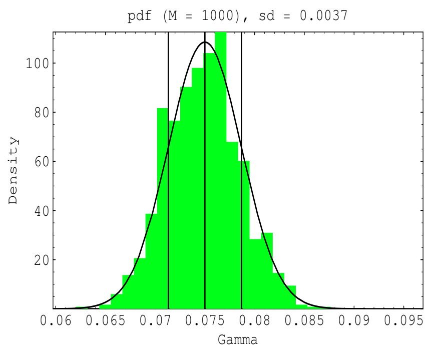


Figure 3.12: The Γ histogram of 1000 samples for $M = 1000$ with an overlay plot of the corresponding normal distribution. The mean 0.075 and ± one standard deviation are shown.

By the Central Limit theorem we might expect that the standard deviation, SD, of the Γ distribution for fixed M scales like

$$\mathrm{SD} = c/\sqrt{M} \quad \text{i.e.} \quad \log(\mathrm{SD}) = \log(c) - \frac{1}{2}\log(M) \tag{3.23}$$

for some $c > 0$, as M becomes large. To check whether SD does scale according to (3.23) we used the same fixed process as before, but this time we vary M i.e. $M = 500$ to $M = 1000$ in steps of $\Delta M = 50$. Figure 3.13 shows the plot of log(SD) against $\log(M)$ with the regression line calculated as

$$\log(\mathrm{SD}) = -2.0605 - 0.5091 \log(M) \tag{3.24}$$

The slope of (3.24) is close to the theoretical value, $-\frac{1}{2}$, from (3.23). By taking the exponential of the intercept in (3.24) we find that in this particular case the SD scales like $0.1274/\sqrt{M}$.

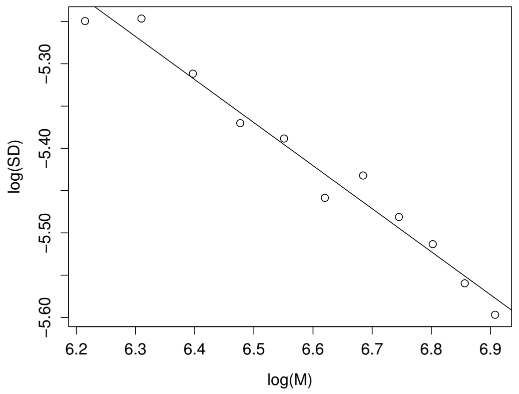


Figure 3.13: The log(SD) plotted against $\log(M)$ with the regression line fit log(SD) $= -2.0605 - 0.5091 \log(M)$.

Computing confidence intervals

Given L samples from a fixed, approximately normal, distribution the standard small sample Student t-test used to generate confidence intervals can be considered as a procedure `StudentTCI[mean, se, dof]`, where `mean` and `se` are the sample mean and standard error. The standard error is computed from the sample standard deviation `sd` as `sd`$/\sqrt{L}$. The number of degrees of freedom `dof` in this context is the number of data samples minus one, i.e. $L - 1$. The procedure should take an option which specifies the confidence level and return the left and right endpoints of the confidence interval. Thus

```
StudentTCI[mean, se, dof, ConfidenceLevel → 0.9]
```

would return the 90% confidence interval.

For example on the data sets for $M = 1000$ generated in the previous section this procedure returns (0.07473666, 0.07512611) at the 90% confidence level. This is a precise estimate for the confidence interval, but it is based on 1000 Gamma tests on independent data sets of size $M = 1000$, i.e. we needed 10^6 input/output data points to arrive at this figure. With this amount of data we could almost certainly derive an extremely accurate estimate for the Gamma statistic. Plainly we need a more practical method for computing confidence intervals.

Our heuristic method of estimating confidence intervals, derives from the M-test (*i.e.* computing a Γ over an increasing number of data points) and is based on some assumptions. We shall assume that the Gamma distribution for fixed M is approximately normal. This is manifestly not the case at the tails of the distribution but, as illustrated in the previous sub-section, may serve for the purposes of the present section. We saw that the standard deviation of the Gamma distribution scales approximately like $c/\sqrt{M}$, for some $c > 0$, as M becomes large. We can exploit this observation to construct a heuristic algorithm to compute confidence intervals based on an M-test.

Student's t-test is based on the idea of taking independent samples from a *fixed* distribution. In our case the Gamma distribution varies as M increases in the M-test; the mean remains fixed at the variance σ^2 of the noise, but `sd` scales like $c/\sqrt{M}$ as M increases.

In order to use `StudentTCI` to compute confidence intervals for an M-test we need to have estimates for the mean and standard deviation of the Gamma distribution derived from a limited L number of sample Gamma test calculations in which M varies. Suppose for increasing $M = M_1, M_2, \ldots, M_L$ we have computed (M_i, Γ_i) $(1 \leq i \leq L)$.

The problem we have is that our samples $\Gamma_1, \ldots, \Gamma_L$ are each computed on data sets of *different* sizes, and so are actually drawn from *different* Gamma distributions. These distributions have the same underlying mean (the true noise variance σ^2), but *different* standard deviations, which we now *assume* are of the form $c/\sqrt{M_i}$ $(1 \leq i \leq L)$ for some unknown but fixed[7] $c > 0$.

To correct for this we linearly 'normalise' each sample value Γ_i so that it can be considered to come from *the same* distribution. We do this by multiplying by $\sqrt{M_i}$ for calculating the *normalised* mean and standard deviation, and then *re-normalising* to the distribution for M_L by multiplying by $M_L^{-1/2}$. In this way we endeavour to ensure

[7]Of course, the value of c might be expected to be problem dependent.

that we are averaging comparable quantities. Thus we compute

$$\begin{aligned}
\texttt{mean} &= \frac{1}{M_L^{1/2}\,L}\sum_{i=1}^{L}\sqrt{M_i}\ \Gamma_i \\
\texttt{sd}^2 &= \frac{1}{M_L(L-1)}\sum_{i=1}^{L}(\sqrt{M_i}\ \Gamma_i - \sqrt{M_i}\ \texttt{mean})^2 \\
\texttt{se} &= \frac{\texttt{sd}}{\sqrt{L}} \\
\texttt{CI} &= \texttt{StudentTCI[}\Gamma_L\texttt{, se, } L-1\texttt{, ConfidenceLevel } \rightarrow 0.9\texttt{]}
\end{aligned}$$

which returns the confidence interval `CI` associated with the L^{th} Gamma statistic Γ_L in an M-test which computes Γ_i for $M = M_1, M_2, \ldots, M_L$. Note that Γ_L is given as the `mean` in `StudentTCI` because this is the best estimate we have of the distribution noise variance σ^2.

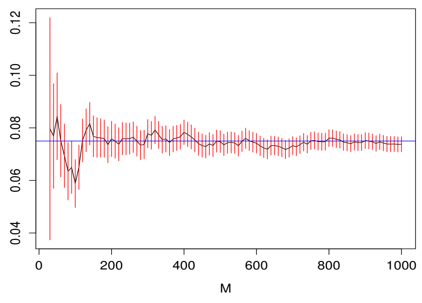


Figure 3.14: An M-test with the heuristic confidence intervals at the 90% level shown as red bars for $f(x) = \sin(x) + r$ with $\sigma^2 = 0.075$. The true noise variance (blue line) is almost always inside the confidence interval.

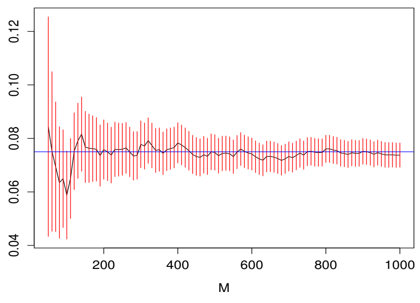


Figure 3.15: An M-test with the heuristic confidence intervals at the 99% level shown as red bars for $f(x) = \sin(x) + r$ with $\sigma^2 = 0.075$. The true noise variance (blue line) is *always* inside the confidence interval.

Figure 3.14 shows the heuristic confidence intervals at the 90% level for $M_L = 20$ to $M_L = 1000$ in steps of 10. The confidence interval returned by the heuristic for $M_L = 1000$ is (0.07089257, 0.07665589), which compares with our earlier accurate confidence interval of (0.07473666, 0.07512611). The latter was computed using 10^6 data samples and 1000 Gamma tests, whereas the former, our heuristic, used $M_L = 1000$ data samples and $L = 99$ Gamma tests.

For comparison, Figure 3.15 shows the intervals at the 99% level for $M_L = 20$ to $M_L = 1000$ in steps of 10. The confidence interval returned by the heuristic for

$M_L = 1000$ is (0.06921554, 0.07833291) compared with (0.07462616, 0.07523662) for the accurate intervals.

Because, in the first instance, the Gamma test estimates $\hat{\sigma}^2_M$, not all M-test graphs with confidence intervals will necessarily produce intervals nested around the true noise variance (as these figures tend to do). For example we should compare Figure 3.14 and Figure 3.15 with Figure 7.5(b) or Figure 8.6(b).

As we should expect from a heuristic confidence interval, our method is a *conservative estimate* because it contains as a subset the more accurate interval.

3.5.3 Full embedding search

Given m candidate inputs there are $2^m - 1$ possible non-trivial input subsets (or embeddings). A binary string of length m is one way to represent each possible input subset where '1' and '0' denotes input inclusion and exclusion respectively. We often call this binary string a *mask*.

A full embedding is an exhaustive search that computes a Gamma statistic for each subset of possible inputs and then orders the input subsets in ascending Gamma order. A useful way to represent a full embedding search is in the form of a *Gamma histogram* where the range of Γ values are divided into bins and the frequency of the Γ values per-bin are plotted vertically. The input subsets which produced 'small' Γ values are said to be in the *low-*Γ *region* and those that produced 'large' Γ values are in the *high-*Γ *region*.

We can exploit the full embedding search to find a suitable subset of inputs for the unknown model using *Durrant's method* (Durrant, 2002). This is where a *frequency count analysis* is conducted on the Γ-ordered input subsets i.e.

1. Define an $n_L \times m$ matrix $\boldsymbol{L}$, where n_L is the number of input subsets taken from the low-Γ region and m is the number of candidate inputs. Each row in $\boldsymbol{L}$ is a mask representing an input subset.
2. Define an $n_H \times m$ matrix $\boldsymbol{H}$, where n_H is the number of input subsets taken from the high-Γ region and m is the number of candidate inputs. Each row in $\boldsymbol{H}$ is a mask representing an input subset.
3. For each column (input co-ordinate) in $\boldsymbol{L}$ count the number of input *inclusions* over n_L i.e. the number of 1's.

4. For each column (input co-ordinate) in $\boldsymbol{H}$ count the number of input *exclusions* over n_H i.e. the number of 0's.

From the frequency count analysis we can find which inputs adhere to the the following principle:

"An input is relevant to the model if it is included in the majority of embeddings with a small Gamma statistic and excluded in the majority of embeddings with a large Gamma statistic."

We consider two experiments; one linear and one non-linear. In each case we compare the results of the full embedding search with those gained from ordinary correlations.

Example One. (A linear function f) We define the input vector as $\boldsymbol{x} = (x_1, x_2, \ldots, x_{10})$, where each variable is taken from a uniform distribution over the range $[-2, 2]$. The output y is defined by

$$y = 5x_3 + 3x_8 + r \tag{3.25}$$

where $\sigma^2 = 0.5$, $\mathrm{Var}(y) = 47.17$, and $M = 1000$. Therefore, any input variable selection technique should flag x_3 and x_8 as the inputs to this model.

Figure 3.16 displays the correlation between the inputs $x_1, \ldots, x_{10}$ and the output y. Both x_3 and x_8 have been flagged as significant variables, which would lead us to correctly assume that they are the only relevant inputs.

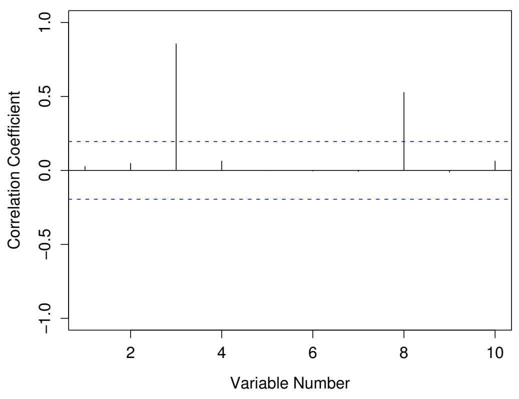


Figure 3.16: The correlation between the candidate input variables $x_1, \ldots, x_{10}$ and the output y for the linear dataset. A variable uncorrelated with the output is likely to fall between the dashed lines with probability 0.95.

Figure 3.17 shows that variables x_3 and x_8 have been *included* in all the embeddings that produced the lowest 10% of Gamma statistics from the full embedding search[8]. Furthermore, Figure 3.18 shows that variables x_3 and x_8 have been *excluded* from all the embeddings that produced the highest 10% of Gamma statistics returned by the full embedding search. Following our input variable selection principle we can concluded (correctly) that both x_3 and x_8 are relevant inputs to the model.

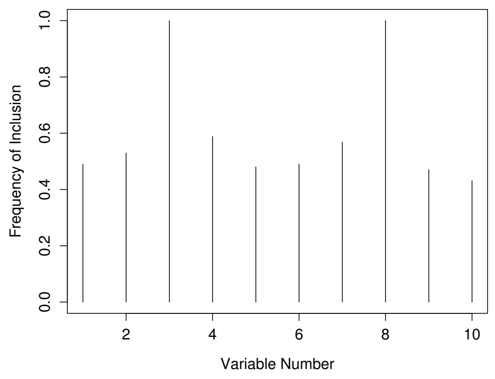


Figure 3.17: Linear example. The frequency of input inclusion in the embeddings from the lowest 10% of Γ statistics for $y = 5x_3 + 3x_8 + r$.

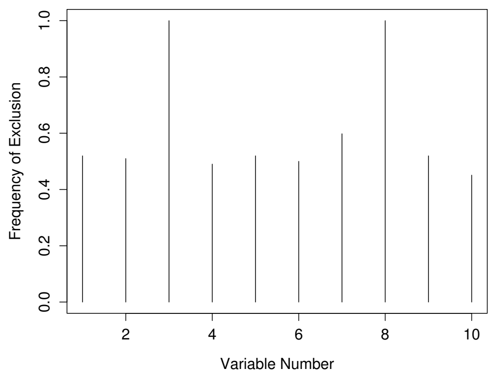


Figure 3.18: Linear example. The frequency of input exclusion in the embeddings from the highest 10% of Γ statistics for $y = 5x_3 + 3x_8 + r$.

Thus in this linear example both the correlation and Gamma test methods of input variable selection work well.

Example Two. (A non-linear function f) Using the same input and noise variables as example one, we change the output so that the mapping from inputs to output is now *non-linear.* The output y is determined by

$$y = \sin(x_3\pi) + 2\cos(x_8) + r \tag{3.26}$$

where $\mathrm{Var}(y) = 1.92$.

Figure 3.19 displays the correlation between the inputs $x_1, \ldots, x_{10}$ and the output y. As can be seen this method has not managed to identify either x_3 or x_8 as significant inputs to the model.

Figure 3.20 shows that variables x_3 and x_8 were *included* in all the embeddings that produced the lowest 10% of Gamma statistics in the full embedding search. Furthermore, Figure 3.21 shows that variables x_3 and x_8 were *excluded* from all the embeddings

[8]10% was somewhat arbitrarily chosen for these examples.

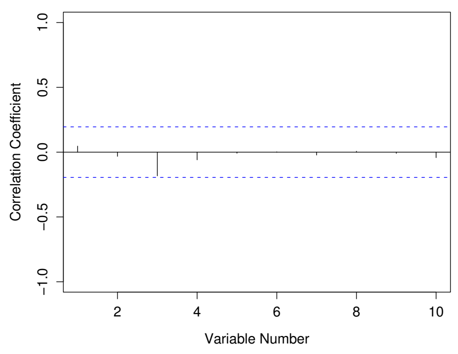
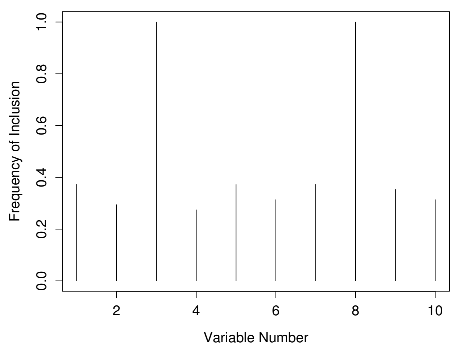


Figure 3.19: Non-linear example. The correlation between the candidate input variables $x_1, \ldots, x_{10}$ and the output y for the nonlinear dataset. A variable uncorrelated with the output is likely to fall between the dashed lines with probability 0.95. Clearly no relevant input variables have been selected by this method.

that produced the highest 10% of Gamma statistics in the full embedding search. In both cases we see that the correct input variables x_3 and x_8 are selected.



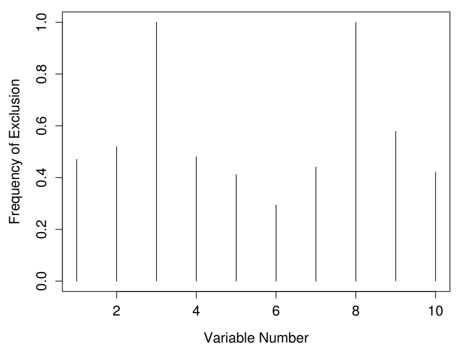


Figure 3.20: Nonlinear example. The frequency of input inclusion in the embeddings from the lowest 10% of Γ statistics for $y = \sin(x_3\pi) + 2\cos(x_8) + r$.

Figure 3.21: Nonlinear example. The frequency of input exclusion in the embeddings from the highest 10% of Γ statistics for $y = \sin(x_3\pi) + 2\cos(x_8) + r$.

Taken together these two examples illustrate that correlations are substantially less effective for non-linear data sets. Given that in most practical circumstances f will be unknown, and quite likely non-linear, a pure correlation analysis has the potential to mislead us regarding the relationship between inputs and output.

On the other hand, the Gamma test has the potential to identify the relevant inputs of a model *regardless of linearity/non-linearity.* However, one of its major drawbacks is the computing time required for the full embedding search. If the number of candidate inputs becomes too large, then the time taken to search all the input subsets becomes infeasible[9], typically when $m \approx 20$. In cases where a faster analysis in required one can use a heuristic search, e.g. a genetic algorithm (Holland, 1975), or a small random sample of input subsets.

In Chapter 6 we propose using approximate near neighbours in the Gamma test calculation, which typically halves the full embedding search time without noticeably harming the effectiveness of the input variable selection process. Another fast algorithm we propose using is a *random embedding search* which computes Gamma statistics for a small sample of embeddings. To make efficient use of the information provided by the embedding search routines, Chapter 6 also provides some significant improvements to the frequency count analysis.

3.6 Determining the embedding dimension of chaotic time series

3.6.1 False nearest neighbour algorithm

The *false nearest neighbour* (FNN) algorithm (Kennel et al., 1992) is a technique to determine the embedding dimension for phase-space reconstruction. A chaotic attractor is typically a compact object in phase-space, such that points of an orbit on the attractor acquire neighbours. It has been suggested that the evolution of phase-space neighbourhoods can determine how points on or near the attractor will evolve, and also provide a way to accurately compute the Lyapunov exponents. However, in this restricted discussion we are purely concerned with identifying the correct embedding dimension.

If the embedding dimension of an attractor is sufficient there will be a one-to-one mapping from the delay-space (the time series) to the original phase-space of the attractor such that the topological properties of the attractor will be maintained. The assumed smoothness of the function means that neighbourhoods of points in delay-space will map to neighbourhoods of points in phase-space. An embedding dimension that is too small will not preserve the topological structure of the attractor, so that points that are neighbours in one embedding dimension, d, will not necessarily be neighbours in the

[9]i.e. it can take a number of days to calculate.

next higher embedding dimension, $d+1$, because the attractor has not been completely unfolded. It is these points that are classified as false nearest neighbours and the number present for a particular embedding dimension determine whether that embedding dimension, d, sufficiently describes the attractor. The FNN algorithm identifies these points for a range of embedding dimensions and (in theory) the optimal embedding dimension has the minimum number of false nearest neighbours.

In order to describe the FNN algorithm we define the delay-space points as y_t and the corresponding phase-space points as $\boldsymbol{y}_t$, where

$$\boldsymbol{y}_t = (y_t, y_{t-1}, \ldots, y_{t-(d-1)}) \tag{3.27}$$

If we are in d dimensions and we denote the k^{th} nearest neighbour of $\boldsymbol{y}_t$ by $\boldsymbol{y}'_t$ then the square of the Euclidean distance between the point $\boldsymbol{y}_t$ and this neighbour is

$$R_d^2(t,k) = \sum_{j=0}^{d-1} (y_{t-j} - y'_{t-j})^2 \tag{3.28}$$

In going from dimension d to dimension $d+1$ by time delay embedding we add a $(d+1)^{th}$ coordinate onto each of the vectors $\boldsymbol{y}_t$. This new coordinate is just y_{t-d}. We now ask what is the Euclidean distance *as measured in dimension* $d+1$, between $\boldsymbol{y}_t$ and the *same* k^{th} neighbour as determined in dimension d? After the addition of the new $(d+1)^{th}$ coordinate the distance between $\boldsymbol{y}_t$ and the same k^{th} nearest neighbour we determine in d dimensions is

$$R_{d+1}^2(t,k) = R_d^2(t,k) + (y_{t-d} - y'_{t-d})^2 \tag{3.29}$$

A natural criterion for catching embedding errors is that the increase in distance between $\boldsymbol{y}_t$ and $\boldsymbol{y}'_t$ is large when going from dimension d to dimension $d+1$. We state this criterion by designating as a *false neighbour* any neighbour for which

$$\sqrt{\frac{R_{d+1}^2(t,k) - R_d^2(t,k)}{R_d^2(t,k)}} > R_{tol} \tag{3.30}$$

where R_{tol} is some threshold. According to (Kennel et al., 1992) the false nearest neighbours are clearly identified for $R_{tol} \geq 10$. Furthermore, it is sufficient to consider only nearest neighbours ($k = 1$) and interrogate every point in the orbit ($t = 1, 2, \ldots, M$) to establish how many of the M nearest neighbours are false.

A second criterion is also applied to deal with the issue of limited data. Let R_A be the size of the attractor

$$R_A = \sqrt{\mathrm{Var}(y_t)} \tag{3.31}$$

we can then write the second criterion as

$$\frac{R_{d+1}(t)}{R_A} > A_{tol} \tag{3.32}$$

where A_{tol} is another threshold, typically $1 < A_{tol} < 2$. This ensures that distant near neighbours, which are stretched to the extremities of the attractor as the embedding dimension increases, are classed as false nearest neighbours.

As explained in (Kennel et al., 1992), the utility of the second criterion is to distinguish between low dimensional chaos and high dimensional chaos or noise. The algorithm computes the total proportion of false nearest neighbours in the data set as determined by the two criteria (3.30) and (3.32).

3.6.2 Increasing embedding search

The increasing embedding search (Durrant, 2002) is a Gamma test based method that can be used to determine the suitable embedding dimension for chaotic time series.

The technique works by computing successive Gamma statistics from an increasing number of lags in the input vector up to lag m. Thus, the input vectors for each Γ_i $(1 \le i \le m)$ take the following form

$$\begin{array}{rcl} \Gamma_1 & : & \boldsymbol{x} = (y_{t-1}) \\ \Gamma_2 & : & \boldsymbol{x} = (y_{t-1}, y_{t-2}) \\ \vdots & & \\ \Gamma_m & : & \boldsymbol{x} = (y_{t-1}, \ldots, y_{t-m}) \end{array}$$

By plotting Γ_i $(1 \le i \le m)$ it can be seen at which lag, d, the Gamma statistic is minimized.

3.6.3 An example: the Hénon Map

The Hénon time series is generated iteratively using the equation

$$z_t = f(z_{t-1}, z_{t-2}) = -z_{t-1}^2 + bz_{t-2} + a \tag{3.33}$$

where $z_0 = 0$, $z_1 = 0$, $a = 1.4$ and $b = 0.3$. The points (z_{t-1}, z_{t-2}) of the map ergodically sample the attractor of the system, which is a set of zero measure but positive Hausdorff

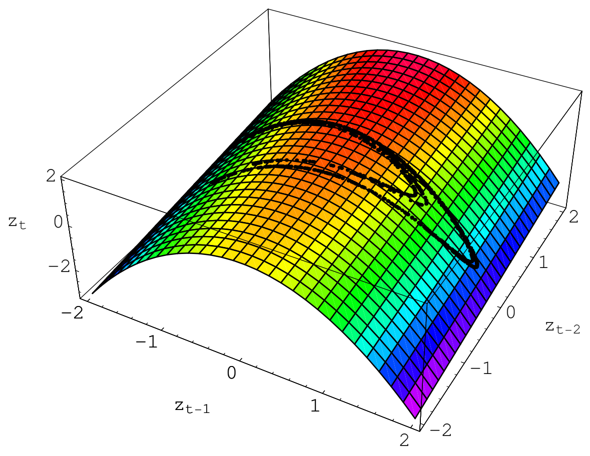


Figure 3.22: The Hénon map attractor (black dots) draped over a topographic rendering of the surface $f(u, v) = -u^2 + bv + a$, where u corresponds to z_{t-1} and v to z_{t-2}.

dimension. This can be extracted from the time series data and visualized by simply plotting the inputs (z_{t-1}, z_{t-2}) against the corresponding output z_t, as shown in Figure 3.22. Figure 3.23 shows the results from the FNN algorithm (where $R_{tol} = 10$ and $A_{tol} = 1.5$) and Figure 3.24 shows the results from the increasing embedding search on the time series data generated from the Hénon map. The optimal embedding dimension selected by both methods uses the first two lags in the embedding, which is what we would hope for given equation (3.33).

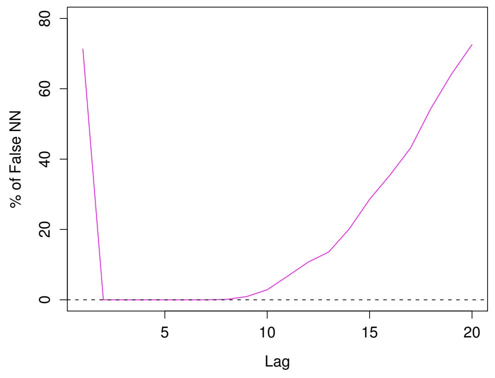


Figure 3.23: False nearest neighbour embedding for the 'clean' Hénon map time series. The graph shows that an embedding dimension $d = 2$ would be suitable.

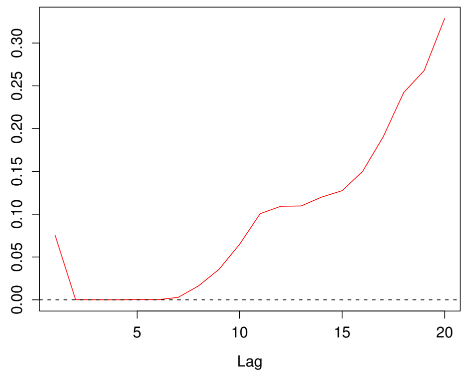


Figure 3.24: The increasing embedding for the 'clean' Hénon map time series. The graph suggests that an embedding dimension of $d = 2$ would be suitable.

However, this is not to say that the FNN algorithm and increasing embedding are providing the same information. The FNN algorithm suggests the minimum regular embedding dimension required to unfold the dynamics of the system, whereas the increasing embedding search suggests a regular embedding dimension that will provide us with the best possible predictive model. To demonstrate we add noise to the Hénon map so that it is a *noisy time series* i.e.

$$y_t = z_t + r_t \tag{3.34}$$

where $\sigma^2 = 0.075$. Figure 3.25 shows the results of the FNN algorithm (where $R_{tol} = 10$ and $A_{tol} = 1.5$) and Figure 3.26 shows the results of the increasing embedding search. The FNN algorithm suggests that the minimum embedding dimension required to unfold the dynamics of the system should be $d = 7$. The increasing embedding search, on the other hand, states that for the *best possible model* ($\Gamma = 0.0754$) an embedding dimension of 13 is required. It is worth noting that a *irregular* embedding may provide a better model than using a regular embedding of the previous 13 values of the time series.

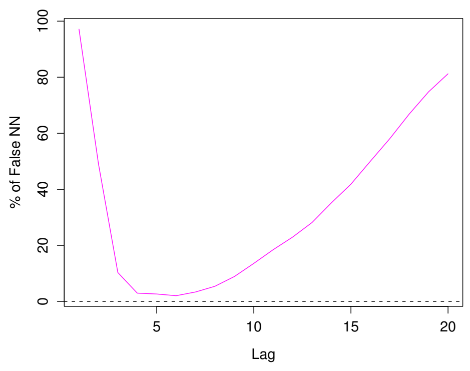


Figure 3.25: Results of FNN for the noisy Hénon map time series. The graph suggests $d = 7$ would be a suitable embedding dimension.

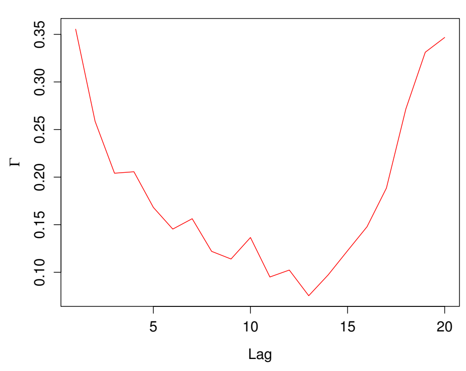


Figure 3.26: Increasing embedding for the noisy Hénon map. The graph suggests $d = 13$ would provide a minimised MSE.

At first glance one may think that the FNN and increasing search should identify $d = 2$ as a suitable embedding dimension of the noisy Hénon map time series. However, as shown in Chapter 4 when the inputs of a model are *themselves* subject to noise (measurement error) we no longer seek to model the original function f, but rather a different unknown model g which best describes the *noisy data*.

3.7 Chapter summary

In this Chapter we have seen how a Gamma test can be used to estimate the sample noise variance in an input/output dataset and *autonomous* time series, provided that the functional mapping from input to output is smooth. We have shown how a simple Gamma test can be exploited to reveal the number of data points we need to construct a smooth model, and how it can be used to find the relevant inputs to the model be it linear or non-linear.

Furthermore, this Chapter has introduced two new methods to add to the Gamma test data analysis toolkit, including:

- A technique to determine the maximum number of near neighbours, p, one can use in a Gamma test analysis to get a good estimate of the noise variance.
- A heuristic method based on the M-test for constructing confidence intervals for the Gamma test.

These existing and new data analysis methods based on the Gamma test will be used in Chapter 7, when we model crime data and in Chapter 8 on climate modelling.

All the papers and software referring to the Gamma test and its associated techniques are freely available to download from the Gamma archive web-site:

http://users.cs.cf.ac.uk/Antonia.J.Jones/GammaArchive/IndexPage.htm

Chapter 4

The effect of measurement error on time series

So far this thesis has assumed the noise to be confined to the output, however this need not be the case. The inputs may also be corrupted by noise, for example due to the effects of *measurement error*. In a smooth input/output process $y = f(\boldsymbol{x})$, if the input data $\boldsymbol{x} \in \mathbb{R}^d$ is noise free and only the output data y is corrupted by noise, then a near optimal smooth model $\hat{g}$ will be a close approximation to the original function f. However, as previously observed, for example in (Kantz and Schreiber, 2004), if the input data is also corrupted by noise then this is no longer the case. With noise on the inputs, the best predictive smooth model based on noisy data need not be an approximation to the actual underlying process; rather, the best predictive model depends on both the underlying process *and* the noise. A consequence of this observation is that one cannot readily infer the *nature* of a process from noisy data. Since almost all data has associated noise the implications of this observation are somewhat unsettling.

In this chapter[1] we show how these effects can be *quantified* using the Gamma test. In particular we examine the Gamma test analysis of noisy time series data. We show that the noise level on the best predictive smooth model (based on the noisy data) can be much higher than the noise level on individual time series measurements, and we give an upper bound for the first in terms of the second. Furthermore, we show how the results of a Gamma test analysis should be interpreted first when applied to input/output data with noisy inputs, and second when applied to noisy time series data.

[1]Joint work with Antonia J. Jones and D. Evans.

4.1 Noisy time series

We direct our attention to the subject of *noisy time series*, which are subtly different from *stochastic time series*. We consider the case where a perfect time series $\{z_t\}$ is observed under additive noise, i.e.

$$y_t = z_t + r_t \tag{4.1}$$

where the true values $\{z_t\}$ are subject to independent and identically distributed random perturbations r_t having expectation zero.

We assume that the noise-free value z_t is determined by a smooth function $f : \mathbb{R}^d \to \mathbb{R}$ of some number d of the previous noise-free values $z_{t-1}, \ldots, z_{t-d}$,

$$z_t = f(z_{t-1}, \ldots, z_{t-d}) \tag{4.2}$$

Therefore we conceptualize that the time series z_t is evolving according to some unknown smooth rule such as (4.2). However, the time series that we are given is comprised of the corrupted values $y_t = z_t + r_t$, where $\mathcal{E}(r_t) = 0$ and typically the noise arises from measurement error. Importantly, and in contrast with stochastic time series, the noise associated with previous values such as y_{t-1} does *not* feed through to affect the value y_t.

Of course, in many real world situations a time series may be *both* stochastic and noisy. However, here we seek to examine just those features specifically relating to noisy time series.

4.2 Effective noise

For $d \in \mathbb{N}$, let $\boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_M$ denote the noisy delay vectors:

$$\boldsymbol{x}_t = (y_{t-1}, \ldots, y_{t-d}) \in \mathbb{R}^d \tag{4.3}$$

Using only the noisy time series data $(\boldsymbol{x}_t, y_t)$, we seek to identify a smooth function $g : \mathbb{R}^d \to \mathbb{R}$ that 'best explains' the observed behaviour of the time series. We first clarify what is meant by 'best explains', i.e. what is an optimal smooth model in this context.

Let $S = \{h : \mathbb{R}^d \to \mathbb{R} \mid h \text{ smooth}, |\nabla h|^2 \leq B\}$, i.e. S is the class of smooth functions in the sense described earlier. For each h consider the mean squared error

$$\text{MSE}(h) = \mathcal{E}\big((y - h(\boldsymbol{x}))^2\big) \tag{4.4}$$

where the expectation is taken over all realisations of the input/output pair $(\boldsymbol{x}, y)$. The set of *optimal* predictive smooth data models is defined to be

$$S_{\text{opt}} = \{g \in S : \text{MSE}(g) \leq \text{MSE}(h) \text{ for all } h \in S\} \tag{4.5}$$

Let $g \in S_{\text{opt}}$. We write

$$y = g(\boldsymbol{x}) + R \tag{4.6}$$

where R is a zero-mean random variable, called the *effective noise* on the output, which accounts for all variation in the output that cannot be accounted for by *any* smooth transformation of the input.

Note that

$$\mathcal{E}(R^2) = \mathcal{E}\big((y - g(\boldsymbol{x}))^2\big) = \text{MSE}(g) \tag{4.7}$$

so the variance of the effective noise coincides with the minimum achievable mean-squared error by a smooth data model based on the given selection of inputs.

To optimally model the time series data, we need to identify/construct a function $\hat{g} \in S$ which is as close as possible to an optimal data model $g \in S_{opt}$. Such a model will have close to minimal $\mathcal{E}\big((y - \hat{g}(\boldsymbol{x}))^2\big)$ and will not change significantly as more and more data is used in the model construction, i.e. as $M \to \infty$. We describe such a model as 'asymptotically stable'.

By (4.4) and (4.6) the MSE of $\hat{g}$ will include both the effective noise and the modelling error, i.e.

$$\text{MSE}(\hat{g}) = \mathcal{E}(R^2) + \mathcal{E}\big((g(\boldsymbol{x}) - \hat{g}(\boldsymbol{x}))^2\big) \tag{4.8}$$

Once the model selection process has been completed, it is tempting to assume that $g = f$, and hence that $\hat{g}$ is an approximation to the original function f that generated the noise-free data z_t. However, as observed in (Kantz and Schreiber, 2004) and as we illustrate here, this is not necessarily the case. The main contribution of this chapter is to illustrate how these differences can be quantified using the Gamma test.

4.2.1 Model construction

In practice, given a noisy time series (y_t), we seek to construct an asymptotically stable model $\hat{g}$ for which the *empirical* mean squared error, defined by

$$\text{MSE}_{\text{emp}}(\hat{g}) = \frac{1}{M-d} \sum_{t=d+1}^{M} (y_t - \hat{g}(\boldsymbol{x}_t))^2 \tag{4.9}$$

is as close as possible to $\mathcal{E}(R^2)$, the variance of the effective noise. By (4.8), this ensures that $\hat{g}$ is as close as possible to an optimal predictive data model $g \in S_{\text{opt}}$ (in a mean-squared sense).

4.2.2 Estimates of effective noise

The notion of effective noise is not the same as residual error measured against a particular model. Residual errors against a particular model can occur simply because the model is poorly chosen. However, even with the best possible model one cannot regularly produce an error variance lower than the effective noise variance imposed by the constraints of the overall situation, e.g. measurement error, incorrectly chosen embedding dimension in equation (4.6) etc. As remarked in (Jones, 2004), even with no measurement error whatsoever, effective noise may still be present, for example as a consequence of choosing non-optimal input variables for the model[2].

4.3 Input/output data with noisy inputs

For brevity, we suppress the dependence on the index t, and consider a 'typical' data point $(\boldsymbol{x}, y)$. If the input components $x_1, \dots, x_d$ of $\boldsymbol{x}$ are themselves subject to noise, this will increase the effective noise variance on the output. The Gamma test can still be used to estimate the variance of this effective noise (modulo the optimal smooth model).

As illustrated in Figure 4.1, we define

$$\boldsymbol{x} = \boldsymbol{w} + \boldsymbol{\epsilon} \quad \text{and} \quad y = f(\boldsymbol{w}) + r \tag{4.10}$$

where $\boldsymbol{w}$ and $f(\boldsymbol{w})$ represent the noise-free input and output, while $\boldsymbol{\epsilon}$ and r represent the input noise and output noise, respectively.

[2]In such cases the effective noise distribution is partly determined by the input distribution.

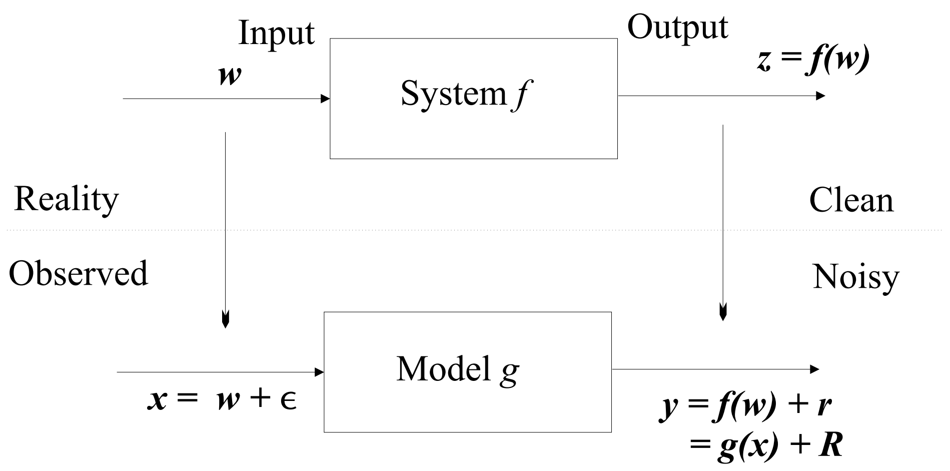


Figure 4.1: Reality versus observation.

We assume that each input noise component ϵ_j has mean zero and bounded variance $\mathrm{Var}(\epsilon_j)$, and is independent of every other noise component ϵ_k $(j \neq k)$ and of r. In particular, because the ϵ_j are zero mean, this implies that $\mathcal{E}(\epsilon_j \epsilon_k) = 0$ whenever $j \neq k$.

Proposition 4.1. *Provided the input noise* $\boldsymbol{\epsilon}$ *is 'small' in some sense, the effective noise variance* $\mathrm{Var}(R)$ *satisfies*

$$\sigma^2 \leq \mathrm{Var}(R) \leq \sigma^2 + \mathcal{E}(|\nabla f|^2)\ max_j\{\mathrm{Var}(\epsilon_j)\} \tag{4.11}$$

where $\mathcal{E}(|\nabla f|^2)$ *is taken with respect to the* $\boldsymbol{w}$ *input sampling distribution,* $\mathrm{Var}(\epsilon_j)$ *is the variance of the noise distribution on the* j*th input coordinate, and* σ^2 *is the variance of the output noise distribution.*

Remark Here we seek to explain the practical consequences of our observations, rather than supply a formal mathematical analysis. What is interesting is that these conclusions are largely independent of the exact nature of the distributions involved.

Proof. As we have seen, the variance of the effective noise is the minimum mean squared error achievable by any asymptotically stable smooth data model. In particular, because the underlying function f is smooth,

$$\mathcal{E}(R^2) \leq \mathrm{MSE}(f) = \mathcal{E}\big((y - f(\boldsymbol{x}))^2\big) \tag{4.12}$$

Furthermore, $y = f(\boldsymbol{w}) + r$ where $\boldsymbol{w}$ is the clean input and r is the output noise. Hence by Taylor's theorem, for small $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_d)$,

$$\begin{aligned} y - f(\boldsymbol{x}) &= r + f(\boldsymbol{w}) - f(\boldsymbol{x}) = r + f(\boldsymbol{w}) - f(\boldsymbol{w} + \boldsymbol{\epsilon}) \\ &\approx r - (\nabla f(\boldsymbol{w})).\boldsymbol{\epsilon} \end{aligned}$$

Squaring and taking expected values, because the output noise r is independent of both the noise-free input $\boldsymbol{w}$ and the input noise $\boldsymbol{\epsilon}$, and because $\mathcal{E}(r) = 0$,

$$\mathrm{MSE}(f) \approx \mathcal{E}(r^2) + \mathcal{E}((\nabla f(\boldsymbol{w}).\boldsymbol{\epsilon})^2) \tag{4.13}$$

If we assume that the input noise $\boldsymbol{\epsilon}$ is independent of $\nabla f(\boldsymbol{w})$, for the $\boldsymbol{w}$ on which it is measured, and writing

$$\nabla f = (f_{x_1}, \ldots, f_{x_d}) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d}\right)$$

then

$$\begin{aligned}\mathcal{E}((\nabla f.\boldsymbol{\epsilon})^2) &= \mathcal{E}\left((\sum_{j=1}^{d} f_{x_j}\epsilon_j)^2\right) = \sum_{j=1}^{d} \mathcal{E}(f_{x_j}^2 \epsilon_j^2) + \sum_{j \neq k} \mathcal{E}(f_{x_j}\epsilon_j f_{x_k}\epsilon_k) \\ &= \sum_{j=1}^{d} \mathcal{E}(f_{x_j}^2)\mathcal{E}(\epsilon_j^2) + \sum_{j \neq k} \mathcal{E}(f_{x_j} f_{x_k})\mathcal{E}(\epsilon_j \epsilon_k)\end{aligned}$$

which, since $\mathcal{E}(\epsilon_j \epsilon_k) = 0$ for all $j \neq k$, gives

$$\mathcal{E}((\nabla f.\boldsymbol{\epsilon})^2) = \sum_{j=1}^{d} \mathcal{E}(f_{x_j}^2)\mathcal{E}(\epsilon_j^2) \leq \mathcal{E}(|\nabla f|^2)\max_j\{\mathcal{E}(\epsilon_j^2)\}$$

i.e.

$$\mathcal{E}((\nabla f.\boldsymbol{\epsilon})^2) \leq \mathcal{E}(|\nabla f|^2)\max_j\{\mathrm{Var}(\epsilon_j)\} \tag{4.16}$$

Finally, since $\mathrm{Var}(R) \leq \mathrm{MSE}(f)$, we conclude from (4.12), (4.13) and (4.16) that

$$\sigma^2 \leq \mathrm{Var}(R) \leq \sigma^2 + \mathcal{E}(|\nabla f|^2)\max_j\{\mathrm{Var}(\epsilon_j)\} \qquad \square$$

As an upper bound for $\mathrm{Var}(R)$, this is a worst case analysis in that f itself certainly provides one *particular* smooth data model, although it may not be the function that best predictively models the data. The effective noise variance (corresponding to the best data model g) is therefore bounded above by $\mathrm{MSE}(f)$, which in turn is bounded above by $\sigma^2 + \mathcal{E}(|\nabla f|^2)\max_j\{\mathrm{Var}(\epsilon_j)\}$ for small ϵ_j. In particular, note that $\mathrm{Var}(R) \to \sigma^2$ as $\max_j\{\mathrm{Var}(\epsilon_j)\}) \to 0$, i.e. as the noise on the inputs vanishes the effective noise variance approaches the variance of the noise on the output.

4.3.1 Example

Consider the function $f : \mathbb{R} \to \mathbb{R}$ defined by $f(w) = w^2$. We generate $M = 1000$ clean samples (w, z) where $w \in [1, 2]$ and $z = f(w) = w^2$. Then we add normally distributed[3] noise ϵ, having mean zero and variance 0.075, to the clean inputs w, yielding the noisy inputs $x = w + \epsilon$. In this example, we set the output noise to zero, i.e. $\sigma^2 = 0$, so that $y = w^2$. Figure 4.2 shows the spread of data and the $f(w) = w^2$ curve.

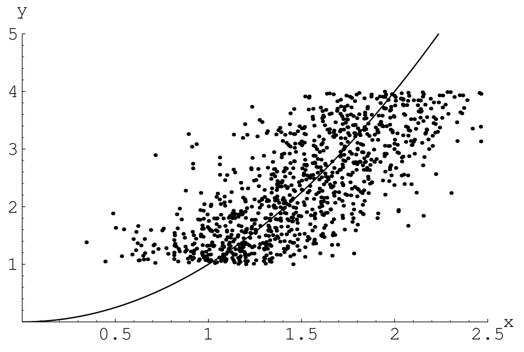


Figure 4.2: The data set having noisy inputs (x, y) and the function $y = w^2$.

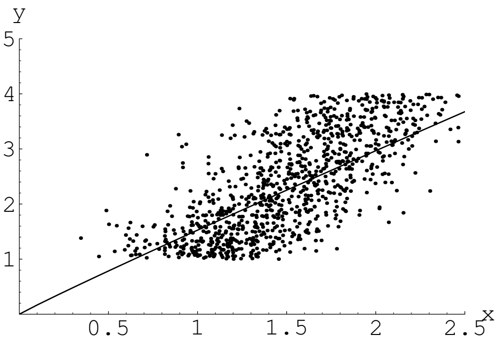


Figure 4.3: The data set of noisy inputs and the 'best-fit' function.

Next we run the Gamma test on the resulting input/output pairs (x, y) and obtain $\Gamma = 0.336593$ as an estimate for the effective noise variance $\mathrm{Var}(R)$, which indicates the best possible modelling error achievable by a smooth data model. For comparison, we also compute[4]

$$\mathrm{MSE}(f) \approx \frac{1}{M} \sum_{i=1}^{M} (y_i - f(x_i))^2 = 0.713487$$

This is the residual mean-squared deviation of the data with respect to the original function $f(w) = w^2$ used to generate the data, and is more than twice the Gamma statistic. By comparison, direct numerical computation of $\mathcal{E}(|\nabla f|^2)\,\mathrm{Var}(\epsilon)$ which, since $\sigma^2 = 0$, is the upper bound given by Proposition 4.1, yields 0.7 which is in close agreement with $\mathrm{MSE}(f)$. This demonstrates that Γ may be significantly less than $\sigma^2 + \mathcal{E}(|\nabla f|^2)\,\mathrm{Var}(\epsilon)$.

[3]We have added normally distributed noise as this is typical of measurement error, but the results herein are (subject to some reasonable conditions) largely independent of the precise nature of the noise distribution.

[4]In practice, we do not know this value because the underlying function f is unknown. In fact, because $f(w) = w^2$ and the (clean) inputs w are uniformly distributed in $[1, 2]$, it is easily shown that $\mathcal{E}(|\nabla f|^2) = 28/3 \approx 9.33$ and hence, because $\mathrm{Var}(\epsilon) = 0.075$, that the upper bound of Proposition 4.1 is equal to 0.7.

The difference between the estimate $\Gamma = 0.336593$ and $\text{MSE}(f) = 0.713487$ prompts us to ask whether we can find a function g that better fits the data?

Indeed, transforming the data to $(\log(x), \log(y))$ and performing a least-squares linear fit on this data returns the relation $\log y = 0.959745 \log x + 0.422462$. Exponentiating the data in the form $y = \alpha x^\beta$ then leads to the model $\hat{g}(x) = 1.52501x^{0.959745}$. Figure 4.3 shows the spread of data and the $\hat{g}(x)$ curve.

Adding more data does not essentially change this best-fit model, so the model is asymptotically stable. For example, repeating the exercise for $10,000$ data points, we obtain $y = 1.51946x^{0.958342}$, so the values 1.52 and 0.959 were essentially determined by the input noise (and of course the original function f).

Finally the residual mean squared deviation from the model $\hat{g}$ is computed:

$$\text{MSE}(\hat{g}) = \frac{1}{M}\sum_{i=1}^{M}(y_i - 1.52501x_i^{0.959745})^2 \approx 0.352164$$

This is close to the best possible modelling error estimated by $\Gamma = 0.336593$, which indicates that $\hat{g}$ is a close approximation to the optimal data model g. We summarize the numerical results in Table 4.1.

Function	g	$\hat{g}$	f
Noise	$\text{Var}(R)$	$\text{MSE}(\hat{g})$	$\mathcal{E}(\lvert\nabla f\rvert^2)\text{Var}(\epsilon) = 0.7$
Experiment	$\Gamma = 0.33659$	0.35216	$\text{MSE}(f) = 0.71348$

Table 4.1: Summary of Example 1 results ($M = 1000$).

One can use any suitable construction that produces a smooth model having a residual error variance close to the Γ value, although Occam's razor suggests that, given several models with similar error variance, one should strive to choose the *simplest*.

4.4 Noisy time series

The clean (noise-free) time series is represented by the sequence $z_1, \ldots, z_M$. As in (4.2), we assume that there is some smooth function f and a number d such that

$$z_t = f(z_{t-1}, \ldots, z_{t-d}) \tag{4.19}$$

The underlying clean time series is corrupted by independent and identically distributed additive noise $r_1, \ldots, r_M$, so we observe the noisy time series $y_1, \ldots, y_M$,

$$y_t = z_t + r_t \tag{4.20}$$

For $t = d+1, \ldots, M$ we construct a set of (noisy) input points $\boldsymbol{x}_t \in \mathbb{R}^d$ using *delay vectors*:

$$\boldsymbol{x}_t = \begin{pmatrix} y_{t-1} \\ \vdots \\ y_{t-d} \end{pmatrix} = \begin{pmatrix} z_{t-1} \\ \vdots \\ z_{t-d} \end{pmatrix} + \begin{pmatrix} r_{t-1} \\ \vdots \\ r_{t-d} \end{pmatrix} \tag{4.21}$$

We write $\boldsymbol{x}_t = \boldsymbol{w}_t + \boldsymbol{\epsilon}_t$ where

$$\boldsymbol{w}_t = \begin{pmatrix} z_{t-1} \\ \vdots \\ z_{t-d} \end{pmatrix} \quad \text{and} \quad \boldsymbol{\epsilon}_t = \begin{pmatrix} r_{t-1} \\ \vdots \\ r_{t-d} \end{pmatrix} \tag{4.22}$$

represent the clean input and the input noise respectively.

Applying Proposition 4.1, because $\mathrm{Var}(\epsilon_j) = \sigma^2$ for all j we obtain the following:

Corollary 4.1. *For noisy time series, provided the noise r is 'small' in some sense,*

$$\sigma^2 \leq \mathrm{Var}(R) \leq (1 + \mathcal{E}(|\nabla f|^2))\sigma^2 \tag{4.23}$$

where the expectation $\mathcal{E}(|\nabla f|^2)$ is taken with respect to the distribution of the noise-free delay vectors $\boldsymbol{w}$.

4.4.1 An Example: The Hénon time series

We remember from Section 3.6.3 that the Hénon time series is generated iteratively using the equation

$$z_t = f(z_{t-1}, z_{t-2}) = -z_{t-1}^2 + bz_{t-2} + a \tag{4.24}$$

where $z_0 = 0$, $z_1 = 0$, $a = 1.4$ and $b = 0.3$.

We generate $M = 1000$ points z_t of the Hénon map. The first 100 points are plotted in Figure 4.4. Next we add normally distributed noise r_t, having mean zero and variance $\sigma^2 = 0.075$, to each z_t, which yields a noisy time series $y_t = z_t + r_t$. Figure 4.5 illustrates the noise component.

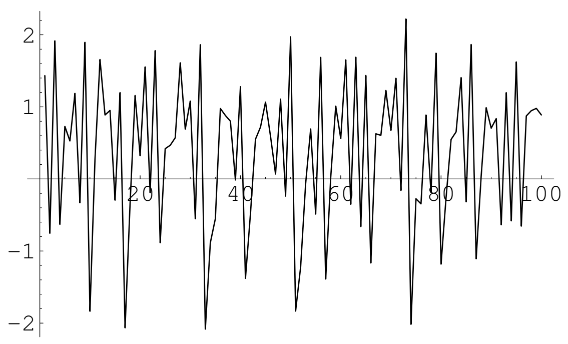


Figure 4.4: First 100 points of the clean Hénon time series z_t.

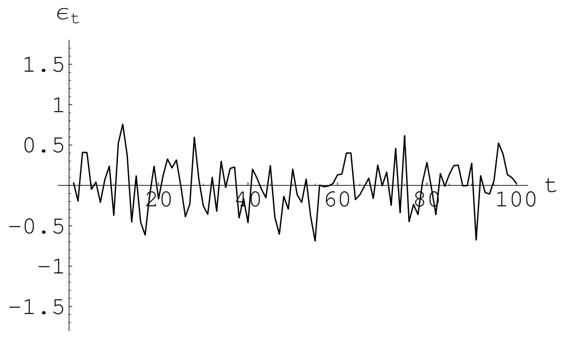


Figure 4.5: First 100 points of the noise $r_t = \epsilon_t$.

Next we form a set of 2-input/1-output data points $(\boldsymbol{x}_t, y_t)$ where $\boldsymbol{x}_t = (y_{t-1}, y_{t-2})$ and $2 \leq t \leq M$, from which we compute an estimate $\Gamma = 0.27713$ for the effective noise variance $\mathrm{Var}(R)$. This is an estimate for the best mean squared error we are likely to achieve for a smooth predictive model using the noisy data, based on the *assumed*[5] embedding dimension $d = 2$. For $10,000$ data points, the Gamma statistic becomes 0.26702, so does not change significantly.

Estimating the upper-bound given by Corollary 4.1, we obtain $(1 + 4.68073) \times 0.075 = 0.426055$, i.e. about twice the Gamma statistic. Thus we are in a situation analogous to that of Example 4.3.1: the Gamma statistic is suggesting that there is a function g, different from the surface f, which, using the noisy data as inputs, should produce a better prediction model for the noisy output.

Finally we constructed an approximation $\hat{g}$ to the surface g using a $2 \rightarrow 5 \rightarrow 5 \rightarrow 1$ feed-forward neural network, trained on the first 600 data points, using the BFGS algorithm (Fletcher, 1987). The network was trained to a mean squared error of 0.264986, which was the Gamma statistic computed using these training points. The mean squared error on the remaining 400 points was 0.32407, somewhat worse than the Gamma statistic.

The trained network surface $\hat{g}$ is shown in Figure 4.6. At first sight we might suppose this to be a crude approximation to the surface f of Figure 3.22, but using considerably more data (e.g. 5000 points) to train the network does not significantly change

[5]Given the recursive definition of the time series in (3.33), at first glance an embedding dimension of $d = 2$ seems reasonable. However, there are many recurrence relations which derive from (4.24) and can be used to generate the time series (for example in the RHS of (4.24) substitute $z_{t-2}^2 + bz_{t-3} + a$ for z_{t-1} etc.). Moreover, the effect of introducing noise, and the fact that the best modelling function g actually *depends* on the noise, means we have almost no *a priori* knowledge of the optimal embedding dimension for modelling purposes.

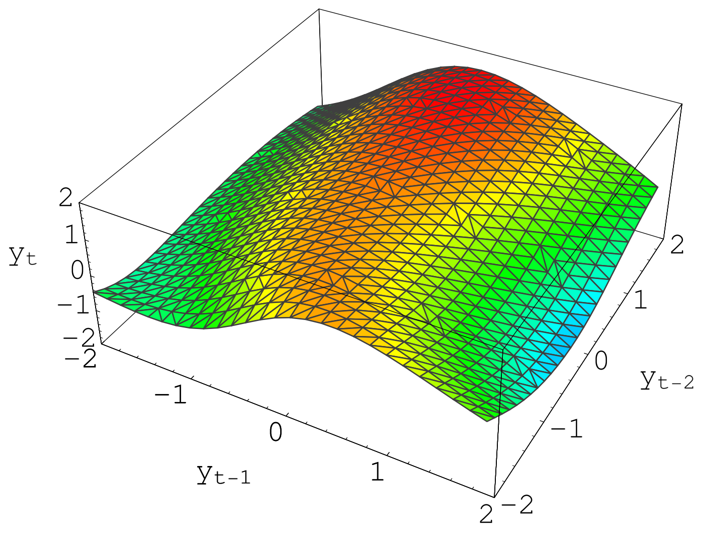


Figure 4.6: The topographic neural network surface $\hat{g}$, produced by training on 600 points of the noisy data $(\boldsymbol{x}_t, y_t)$, where $\boldsymbol{x}_t = (y_{t-1}, y_{t-2})$.

Function	g	$\hat{g}$	f
Noise	$\mathrm{Var}(R)$	$\mathrm{MSE}(\hat{g})$	$(1 + \mathcal{E}(\lvert\nabla f\rvert^2))\sigma^2$
Experiment	$\Gamma = 0.27713$	0.30662	$\mathrm{MSE}(f) \approx 0.42605$

Table 4.2: Summary of Example 2 results ($M = 1000$). Here $\mathrm{MSE}(\hat{g})$ for the neural network is the weighted average of training and test data.

the surface, i.e. the model is asymptotically stable. The essential fact is that the g approximated by $\hat{g}$ is a *different* surface from the f of (4.24).

4.5 Chapter summary

We have illustrated that a Gamma test analysis on embeddings of noisy time series data

$$y_t = z_t + r_t$$

does not return an estimate for $\mathrm{Var}(r_t) = \sigma^2$, but rather returns an estimate for the *effective* noise variance $\mathrm{Var}(R)$, where R is defined by (4.6) and g is an unknown but optimal smooth data model. Moreover, g may or may not be an approximation to the original smooth function f which generated the underlying clean time series.

- This has the interesting consequence that, in these circumstances, an appropriate embedding dimension for an optimal predictive model *depends on the noise* and not just on the intrinsic dynamics.

This fact often seems to be not well understood in the literature.

Our first example highlights the danger of inferring a *process law* using a *model* constructed from noisy data. The process was actually quadratic, but one might easily be tempted to infer that it was linear from a near optimal predictive model.

It is an unfortunate fact that an optimal modelling function g may not bear a close relationship to f (Kantz and Schreiber, 2004). However, this does not reduce the efficiency of a $\hat{g} \approx g$ in producing the best possible predictions using unseen noisy data drawn from the same process.

For time series we have seen that the estimate for the effective noise variance $\mathrm{Var}(R)$ may be much larger than the variance of the noise on individual measurements σ^2. For the purposes of predictive model building, knowing σ^2 is not so critical. Rather, we need to know $\mathrm{Var}(R)$, e.g. so as to know when to stop training a neural network, and this is exactly what the Gamma test estimates.

What is of concern is that even if the measurement error variance on the original time series is quite modest, this can lead to a much larger error variance $\mathrm{Var}(R)$ for the predictions, because the fluctuations in the input can be magnified by the the transformation f. Therefore when using such predictive techniques, it is essential to reduce the measurement error on time series data as much as possible.

In practice, $\mathrm{Var}(R)$ is likely to increase with $\mathcal{E}(|\nabla f|^2)$, which is itself *a priori* unknown, and will certainly be different for different embeddings. Nevertheless it is still valid to use the Gamma statistic Γ to estimate $\mathrm{Var}(R)$ for different embeddings, and to select, based on a 'minimum Γ' criterion, an appropriate embedding dimension and a suitable irregular embedding for a noisy time series model.

Chapter 5

Key algorithms and their R implementation

R is an open source software project for data analysis that is similar in many ways to Matlab, S-plus, Maple and Octave i.e. it has its own language, can import and export C programs and has a powerful graphics engine. A consequence of our research was to synthesize the majority of analytical and modelling tools used in this thesis into an R package. The development of these tools was specified to provide an easy-to-use, flexible, cross-platform application for use by researchers and time series analysts.

Some of the algorithms contained within our R package had been previously implemented within *winGamma* (Durrant, 2002); a Microsoft Windows application complete with graphical interface. This project developed those algorithms further and allowed the user the flexibility to use the algorithms within their own R programs and scripts.

In this Chapter we detail the algorithms used within our R package and also how they were implemented. Furthermore, section 5.3 provides examples of our package within an R session.

5.1 The algorithms

The majority of computing time for a Gamma test is taken up by finding the p nearest neighbours for each point in the input dataset. A simple brute force approach can be programmed in a couple of hours but the computing time scales to $O(M^2)$. Such an implementation is not feasible for the algorithms considered in this thesis and we therefore need a more sophisticated nearest neighbour algorithm.

Therefore, we decided to use the kd-tree ("k dimensional tree") algorithm (Bentley, 1975). Here, for points in $\mathbb{R}^d$ (we use d for dimension), a tree structure is constructed which enables fast lookup of near neighbours. The tree construction can be accomplished in time $O(dM \log M)$ and a near neighbour lookup in $O(\log M)$. Thus, provided p remains bounded, the entire p^{th} nearest neighbour list for every point can be constructed in $O(pdM \log M)$ time.

Problems associated with the *kd*-tree algorithm are; (a) it is much more difficult to implement; and (b) in high dimensions the computing time is not much better than brute force (typically when the dimension exceeds 15).

To overcome these two problems we exploited an open source C++ library[1] of fast nearest neighbour implementations called *the approximate nearest neighbour library* (ANNLIB). Using this library conferred the additional advantage that we did not have to implement the tricky *kd*-tree code, and we could optionally initiate *approximate* nearest neighbour searching, which is a much faster routine in high dimensions than the classical *kd*-tree search outlined in (Bentley, 1975).

In this section we firstly describe the Bentley *kd*-tree construction and search algorithms, and then we go on to describe and examine approximate nearest neighbour search algorithms.

5.1.1 kd-trees

In this sub-section we lean heavily on the account of *kd*-tree construction given in (Margetts, 2001, Appendix B).

A *kd*-tree is a data structure for storing a finite set of points from a k-dimensional space. There are two components to a *kd*-tree:

1. The construction of the *kd*-tree data structure.
2. The search algorithm for finding the nearest neighbours from the *kd*-tree.

We begin by describing the technique to build the data structure (the *kd*-tree) and then describe the search method to find the nearest neighbours of a query point.

[1]A collection of C++ functions.

kd-tree construction

The *kd*-tree is a generalisation of the binary tree where the search space $\mathbb{R}^k$ is divided into two parts at each node. The root node represents the whole dataset and each sub-node represents a subset of the parent's data. Maximally efficient information storage is encapsulated when the tree is balanced, such that each child node has an equal chance of being selected.

The partitioning of the search space occurs for the variable $K \in \{1, \ldots, k\}$ with the greatest range, where K is the *partitioning key*. The median value of the variable given by the partitioning key K provides the *partitioning value* V. Any data point $\boldsymbol{x}_i$ can then be located either to the left sub-tree L or the right subtree R with respect to the partitioning value V and the partitioning key K, such that $\boldsymbol{x}_i \in L$ if $\boldsymbol{x}_{i,K} \leq V$, otherwise $\boldsymbol{x}_i \in R$ (where $\boldsymbol{x}_{i,K}$ is the K component of $\boldsymbol{x}_i$).

This process of partitioning the data into sub-trees continues until no more than B points are stored at each node. These nodes are terminal and are called *buckets* and contain 1 or more data points up to a maximum *bucket size* B. Empirical evidence provided by (Friedman and Bentley, 1977) suggests that between 4 and 32 points per bucket provides optimal performance. A bucket size of 4 was arbitrarily chosen for the R package implementation.

The whole dataset must be scanned at each level of the tree to calculate the partitioning keys and the median values for each node. The calculation of the median for a list of numbers can be achieved with time complexity $O(M)$ (Press et al., 1992). Given that this operation must be performed for each of the $\log M$ levels, the total time complexity for the *kd*-tree construction is $O(dM \log M)$. The storage requirements for a *kd*-tree is $O(M)$. In addition to storing the dataset, very little extra information needs to be stored. Provided the calculation of the median value is achieved by sorting the data in situ, the storage requirements are: for each non-terminal node the location of the data subset, the partitioning key K, the partitioning value V, and the links to the child nodes to be stored, and for each terminal node (i.e. bucket) the number of data points and the location of the bucket data needs to be stored.

The *kd*-tree construction algorithm is given in Algorithm 4. The algorithm accepts data as input and returns the root node to the tree. If the number of data points in the current node does not exceed the bucket size, the node is made terminal and the algorithm finishes. If more data is available than can be accommodated in a single bucket, the data set is partitioned into two data sets according to the partitioning key and partitioning value. The `BuildTree` function is then called for each of these two left and right data sets. This recursive process continues partitioning the data until all of the branches of the tree end with terminal nodes. As the recursive process unwinds the connections from the parent nodes to their immediate child nodes are made and stored. The `CalcSpread` routine should be implemented to return the range of the data for a particular variable. The `Median` function returns the median of the data for a particular variable (Press et al., 1992, describes one method to perform this with time complexity $O(M)$).

The `LeftSubSet` and `RightSubSet` functions create the subsets of the data for the left and right child nodes according to the partitioning key (given by the variable with the greatest spread) and the partitioning value (given by the median of the variable with the greatest spread). `MakeNonTerminalNode` and `MakeTerminalNode` create the data structures for the nodes and store the appropriate supporting variables, i.e. the location of the data subset, partitioning key, partitioning value, and links to the child nodes for non-terminal nodes, and the number of points in the bucket and location of the bucket data for the terminal nodes.

Searching the kd-tree for nearest neighbours

If, for a given dataset, the associated *kd*-tree is constructed in an optimal configuration then the number of records required to be searched should be minimal. The *kd*-tree data structure enables the search to consider only those records closet to the query record, thereby reducing the overall search time.

Some initialisation has to be done prior to the tree being searched and this is shown in Algorithm 5. Once, initialised, the search, described in Algorithm 6 can begin.

The search is made for the `pmax` points closest to the `queryPoint` in the data set. A list of the near neighbours and their associated distances are maintained, called `NN` and `NNDistance` respectively. Initially the near neighbour distances are set to infinity (in practice we set them to a very large number e.g. 9999999.99) to enable nearer points to enter the list as the search progresses. A list of upper and lower bounds are maintained for each dimension (`upperBound` and `lowerBound` respectively). These describe the current bounds of the search space and are used to eliminate searching branches of the *kd*-tree that lie outside of the search space. Finally, a call to `SearchTree` starts the search. The search algorithm is shown in Algorithm 6. The initial call to SearchTree is made from the root node. The algorithm then performs a depth-first recursive search

```
node function BuildTree(data)
begin
if size(data) $\leq$ bucketSize then
  return (MakeTerminalNode(data))
end if
maxSpread = 0
for $j = 1$ to $k$ do
  if CalcSpread(data, j) $>$ maxSpread then
    maxSpread = CalcSpread(data,j)
    partitioningKey = j
  end if
  $j = j + 1$
end for
partitioningValue = Median(data, partitioningKey)
return(MakeNonTerminalNode(partitioningKey,partitioningValue,
BuildTree(LeftSubSet(partitioningKey,partitioningValue,
data)),BuildTree(RightSubSet(partitioningKey,partitioningValue,
data)))
end


root = BuildTree(data)
```

Algorithm 4: The *kd*-tree construction algorithm.

```
{initialisation}
set queryPoint and pmax
NN[1:pmax]
NNDistance[1:pmax] = $\infty$
upperBound[1:k] = $\infty$
lowerBound[1:k] = $\infty$

{search the tree from the root node}
SearchTree(root)
```

Algorithm 5: The *kd*-tree search initialisation.

through the tree. The algorithm first checks whether the current node is terminal. If the node is terminal, all of the points in the bucket are checked against the current nearest neighbour distances list `NNDistance` to see if any points from the bucket are closer than those found so far. If a closer near neighbour is found, `NNDistance` is updated and the point is inserted into the `NN` list, displacing the furthest near neighbour point found so far. The algorithm then returns because there are no further branches to enter.

If the current node is non-terminal then the partitioning value (i.e. the median) and the partitioning key for that node are extracted. These are then used to determine which branch of the tree to examine from the current node. If `queryPoint[partitioningKey]` is less than or equal to the median then the search proceeds down the left-hand branch, otherwise the right-hand branch is searched. The algorithm proceeds by descending the chosen branch, temporarily updating the upper and lower search boundaries for that branch, as determined by the median, and recursing down the tree to the bucket nodes. As the recursion unwinds, the bounds prior to the descent are reinstated followed by a call to `BoundsOverlapBall`. This is used to determine whether the opposite branch needs to be searched because it contains points closer to the query point than the furthest near neighbour found so far. If this is the case, the search bounds for the new branch are temporarily recorded and the descent is made. Algorithm 7 describes the `BoundsOverlapBall` routine.

A *ball* can be imagined to surround the nearest neighbour points whose extent in each dimension is determined by the minimum and maximum values of the nearest neighbours in that dimension. If the extent of the ball lies outside of the search boundaries then there is no need to continue searching. If, however, the search boundaries overlap the ball (i.e. fully or partially contain it) then the search must continue down the opposite branch. This is because points in the opposite branch from the one searched already might be closer than the nearest neighbours found so far.

```
function SearchTree(node)
if node is terminal then
  Examine each point in bucket and update NN and NNDistance
  return
end if
{traverse the tree}
median = median(node)
partitioningKey = PartitioningKey(node)
if queryPoint[partitioningKey] ≤ median then
  {recurse on nearest child}
  temp = upperBound[partitioningKey]
  upperBound[partitioningKey] = median
  SearchTree(LeftChild(node))
  upperBound[partitioningKey] = temp

  {recurse on furthest child}
  temp = lowerBound[partitioningKey]
  lowerBound[partitioningKey] = median
  if BoundsOverlapBall then
    SearchTree(RightChild(node))
  end if
  lowerBound[partitioningKey] = temp
else
  {recurse on nearest child}
  temp = lowerBound[partitioningKey]
  lowerBound[partitioningKey] = median
  SearchTree(RightChild(node))
  lowerBound[partitioningKey] = temp

  {recurse on furthest child
  temp = upperBound[partitioningKey]
  upperBound[partitioningKey] = median
  if BoundsOverlapBall then
    SearchTree(LeftChild(node))
  end if
  upperBound[partitioningKey] = temp
end if
```

Algorithm 6: The *kd*-tree search.

```
boolean function BoundsOverlapBall
sum = 0
for $d = 1$ to $k$ do
  if queryPoint[d] < lowerBound[d] then
    sum = sum + Distance(queryPoint[d], lowerBound[d])
    if Dissem(sum) > NNdistance(furthest) then
      return false
    end if
  else if queryPoint[d] > upperbound[d] then
    sum = sum + Distance(queryPoint[d], upperBound[d])
    if Dissem(sum) > NNdistance(furthest) then
      return false
    end if
  end if
end for
return true

{Euclidean Distance Function}
real function Distance(x1,x2)
begin
 return $(x1 - x2)^2$
end

{Euclidean Dissimilarity Function}
real function Dissim(x)
begin
 return $sqrt(x)$
end
```

Algorithm 7: The *kd*-tree bounds overlap ball routine.

5.1.2 Approximate nearest neighbour search in fixed dimensions

In the previous section we showed that by using kd-trees, we could construct a data structure in $O(dM \log M)$ time such that a nearest neighbour query can be achieved with expected time complexity $O(\log M)$ and space complexity $O(M)$. However, the implied constant in the query time tends to hide the fact that the *dimensional* scaling of the classical *kd*-tree search algorithms are exponentially poor both in theory and practice. In dimension d the constant factors hidden in the asymptotic running time grow at least as fast as 2^d (depending on the metric).

Arya et al. (1998) remarked that, because of the apparent difficulty in high dimensions of obtaining search algorithms that are efficient in the worst case with respect to both space and query time, it might be worth considering an alternative approach based on finding *approximate* nearest neighbours. For many applications it is often sufficient to know that a point is *near* the query point, we don't necessarily need to know that it is exactly the k^{th} nearest neighbour.

We suppose that we have a sample of M data points taken from a suitably distributed set of points confined to a closed bound set C in $\mathbb{R}^d$.

For $1 \leq k \leq M$ we define a k^{th} $(1+\epsilon)$-approximate nearest neighbour $\boldsymbol{s}$ of a query point $\boldsymbol{q}$ to be a data point such that

$$\frac{dist(\boldsymbol{s}, \boldsymbol{q})}{dist(\boldsymbol{s}^*, \boldsymbol{q})} \leq (1+\epsilon) \tag{5.1}$$

where $\boldsymbol{s}^*$ is the (exact) k^{th} nearest neighbour of $\boldsymbol{q}$. For $1 \leq k \leq M$ one then defines a sequence of k approximate nearest neighbours of query point $\boldsymbol{q}$ to be a sequence of k distinct data points, such that the i^{th} point in the sequence is an approximation to the i^{th} nearest neighbour of $\boldsymbol{q}$.

The main result in (Arya et al., 1998) is

Theorem 5.1. *Consider a set C of M data points in $\mathbb{R}^d$. There is a constant $c_{d,\epsilon} \leq d\lceil 1+6d/\epsilon \rceil^d$, such that in $O(dM \log M)$ time it is possible to construct a data structure of size $O(dM)$, such that for any Minkowski metric:*

(i) Given any $\epsilon > 0$ and $\boldsymbol{q} \in \mathbb{R}^d$, a $(1+\epsilon)$-approximate nearest neighbour of $\boldsymbol{q}$ in C can be reported in $O(c_{d,\epsilon} \log M)$ time.

(ii) More generally, given $\epsilon > 0$, $\boldsymbol{q} \in \mathbb{R}^d$, and any k, $1 \leq k \leq M$, a sequence of k $(1+\epsilon)$-approximate nearest neighbours of $\boldsymbol{q}$ in C can be computed in $O((c_{d,\epsilon}+kd)\log M)$ time.

Using these ideas it has subsequently been shown that by computing approximate nearest neighbours, it is possible to achieve significantly faster running times often with relatively small actual errors. The maximum approximation error bound, ϵ, allows us to control the tradeoff between accuracy and running time.

There are two important practical aspects of Theorem 5.1. First, space requirements are completely independent of ϵ and are asymptotically optimal for all parameter settings, since dM storage is needed just to store the data points. Second, preprocessing is independent of ϵ and the metric, implying that once the data structure has been built, queries can be answered for any error bound and for any Minkowski metric. Setting $\epsilon = 0$ will cause the algorithm to compute the true nearest neighbour, in which case the running time is no worse than the classical *kd*-tree lookup. Unfortunately, exponential factors in query time do imply that the approximate near neighbour search algorithm is still not practical for very large values of d. However, empirical evidence shows that the constant factors are much smaller than the bound given in Theorem 5.1. Indeed, David Mount (one of the authors in Arya et al. (1998)) writes[2]

> "Note that our query time has exponential factors. The factor grows as something like $\max(2, 1/\epsilon)^d$. In practice, this worst case scenario rarely occurs. This is why the approximate near neighbor data structure is quite popular in practice."

Here is an intuitive overview of the approximate nearest neighbour query algorithm. Given the query point $\boldsymbol{q}$, we begin by locating the leaf cell containing the query point in $O(\log M)$ time by a simple descent through the tree. Next, we begin enumerating the leaf cells in increasing order of distance from the query point. We call this *priority search*. When a cell is visited, the distance from $\boldsymbol{q}$ to the point associated with this cell is computed. We keep track of the closest point seen so far. For example, Figure 5.1 shows the cells of such a subdivision. Each cell has been numbered according to its distance from the query point.

Let $\boldsymbol{s}$ denote the closest point seen so far. As soon as the distance from $\boldsymbol{q}$ to the current leaf cell exceeds $dist(\boldsymbol{q}, \boldsymbol{s})/(1+\epsilon)$ (illustrated by the red dashed circle in Figure 5.1), it follows that the search can be terminated, and $\boldsymbol{s}$ can be reported as an approximate

[2]Personal email communication; quoted here by permission.

nearest neighbour to $\boldsymbol{q}$. The reason is that any point located in a subsequently visited cell cannot be close enough to $\boldsymbol{q}$ to violate $\boldsymbol{p}$ being an approximate nearest neighbour. (In the example shown in the figure, the search terminates just prior to visiting cell 9. In this case $\boldsymbol{s}$ is not the true nearest neighbour, since that point belongs to cell 9, which was never visited.)

When ϵ increases the red dashed circle in the figure will *contract*, which speeds up the query time because essentially a fewer number of cells will be visited. However, the exact nearest neighbour can never be closer to $\boldsymbol{q}$ than the red dashed line. In essence the higher value assigned to ϵ the quicker the search will terminate.

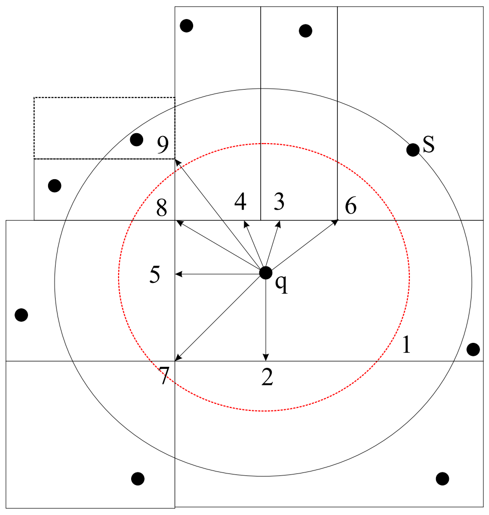


Figure 5.1: Algorithm overview. The red dotted circle is the bounded error i.e. $dist(\boldsymbol{q}, \boldsymbol{s})/(1+\epsilon)$. When the distance from $\boldsymbol{q}$ to the current leaf cell exceeds this bound the algorithm terminates. In this example, this occurs when the algorithm reaches cell 9 causing $\boldsymbol{s}$ to be returned as an $(1+\epsilon)$-approximate nearest neighbour.

Performance

It is desirable to measure the performance of the algorithm with respect to *distance errors*. The user supplies an upper bound ϵ on the allowable distance error, but the algorithm may find points that are closer. We find the exact first nearest neighbour off-line and then calculate the relative error, namely the ratio between the distance to the point reported by the algorithm and the exact nearest neighbour minus 1. The resulting quantity averaged over all query points is called the *average relative error*. Figure 5.2 shows the average relative error over increasing ϵ for a set of uniformly distributed points $C \in [0, 1]^{16}$ where $M = 1000$. The results show that even for large values of ϵ, the average relative error committed is typically very small.

A related statistic is how often the algorithm succeeds in finding the exact first nearest neighbour as a function of ϵ. Using the same set of uniformly distributed points as above, we calculated the fraction of times the algorithm missed the exact nearest neighbour for increasing ϵ. The results in Figure 5.3 show that for $\epsilon \geq 10$ the fraction tails off at 0.8 and that for $\epsilon \in [1, 3]$ the fraction remains below 1/2. Therefore, the algorithm is capable of returning a high proportion of exact nearest neighbours for quite modest values of ϵ.

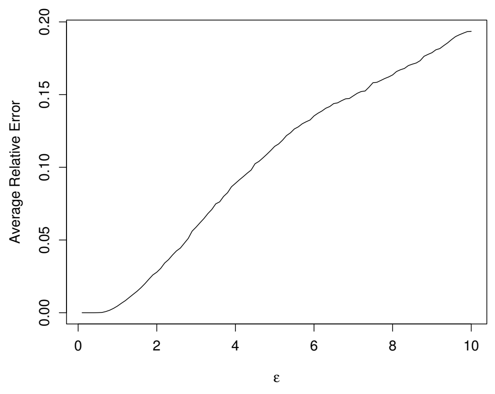


Figure 5.2: The average relative error for $k = 1$ of a uniformly distributed set of points $C \in [0, 1]^{16}$, where $M = 1000$.

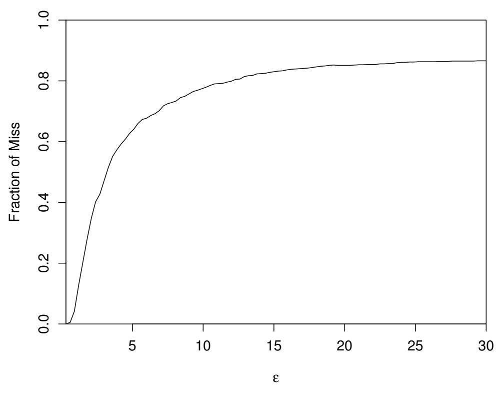


Figure 5.3: The fraction of approximate nearest neighbours returned by the algorithm that are not exact near neighbours for a uniformly distributed point set $C \in [0, 1]^{16}$, where $M = 1000$.

Speed comparison

The speed of the approximate nearest neighbour search algorithm is likely to depend on a number of factors, as David Mount remarks[3]:

> "There is a very subtle relationship between ϵ, dimension, and various aspects of the point and query distribution, all of which go into to affecting the algorithm's performance. My experience has shown that as long as the effective dimension of the data set is not too high (say less than 20), and ϵ is not too small (say at least 1/10), approximate near neighbors are quite efficient. But, the data structure sometimes surprises even me. In short, don't give too much credence to theoretically derived running times. Empirical performance varies widely."

[3]Personal email communication; quoted here by permission.

To compare the speed of the approximate nearest neighbour query algorithm to the classical *kd*-tree query algorithms, as the dimension increases, we conduct the following experiment.

We generate a set of 5000 uniformly distributed points, $C \in [0,1]^d$ for $d = 1, \ldots, 20$. In each dimension we construct a *kd*-tree and then measure the time taken to compute the 10 nearest neighbours for each data point using $\epsilon = 0$ (i.e. the original *kd*-tree search algorithm), $\epsilon = 0.5$, $\epsilon = 1$, $\epsilon = 5$ and $\epsilon = 10$. The results are shown in Figure 5.4. Over the range of our experiments, the original search method has an experimental polynomial query time of $O(d^{2.202})$, for $\epsilon = 1$ the query time is reduced to $O(d^{1.58})$ and if we are prepared to take a large error bound, $\epsilon = 10$, we are rewarded with a $O(d^{0.4241})$ query time.

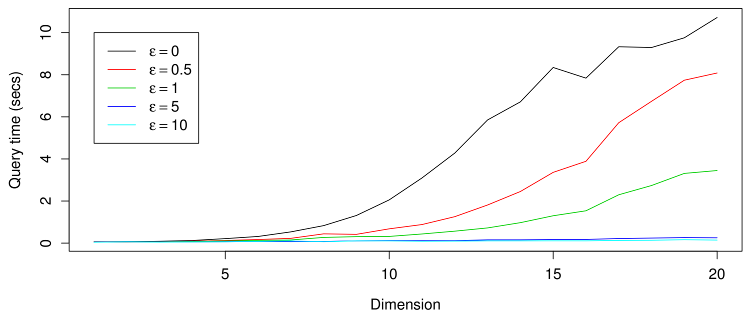


Figure 5.4: The measured time (in seconds) to return the 10 nearest neighbours for different ϵ values on a set of 5000 uniformly distributed points, $C \in [0,1]^d$ for $d = 1, \ldots, 20$.

5.1.3 The effect of approximate near neighbours on Γ

We expect that as we increase ϵ on the near neighbour search the Gamma statistic will become less accurate as an estimate for σ^2. However, the benefit of such an action will be to decrease the computing time of a Gamma test.

Example. Let's take two inputs x_1 and x_2 from a uniform distribution over the range $[-1, 1]$ and define the output y by

$$y = x_1^2 + 6x_2 + r \tag{5.2}$$

where $M = 10000$, r is a Gaussian random variable with $\sigma^2 = 0.05$, and $\mathrm{Var}(y) = 12.34$. We progressively increase the value of the error bound, $\epsilon = i/20$, for $0 \leq i \leq 200$

and record the Gamma statistic and the time taken for its computation. Figure 5.5 shows the results of this experiment. As we can see the results support our intuitive assumption that as ϵ is increased Γ becomes faster to compute but also becomes less accurate as an estimate for σ^2. However, closer inspection of the graph reveals that whilst the time taken to compute $\Gamma(\epsilon = 10)$ is 38% quicker than $\Gamma(\epsilon = 0)$, the $\Gamma(\epsilon = 10)$ is only 0.0002 less than the Γ for exact near neighbours. This shows that we are making significant speed gains with little impact on the estimate of the noise variance.

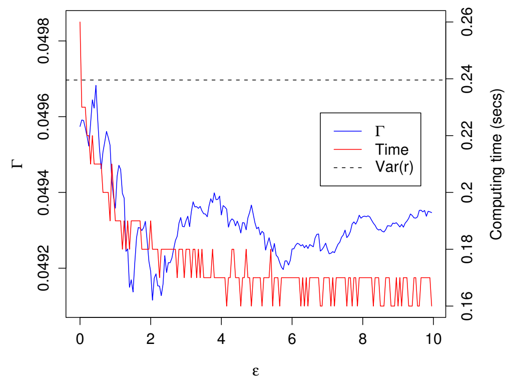


Figure 5.5: The effect of approximate near neighbours on Γ. The Γ statistic is shown in blue, whilst its computation time (in seconds) is shown in red. The dashed line indicates the sample noise variance (0.0497).

In practice there is not much need to use approximate near neighbours for a *single* Gamma test, since this is a fairly fast procedure in most cases[4]. When we want to compute many Gamma tests, e.g. a full embedding search on high-dimensional input data, where speed is important then using approximate nearest neighbour search algorithms may provide a reasonable solution.

5.2 The implementation

The R environment is an integrated suite of software facilities for data manipulation, calculation and graphical display that can run on all the major operating systems (Windows, Linux, Unix and Mac OS X). Among other things it has

- an effective data handling and storage facility

[4]Although, one may require approximate near neighbours when $M > 10^6$

- a suite of operators for calculations on arrays, in particular matrices
- a large, coherent, integrated collection of intermediate tools for data analysis
- graphical facilities for data analysis and display either directly at the computer or on hardcopy
- a well developed, simple and effective programming language (called S) which includes conditionals, loops, user defined recursive functions and input and output facilities. (Indeed most of the system supplied functions are themselves written in the S language.)
- the ability to interface with the C++ programming language. Thus, a programmed C++ function can be called into an R script and conversely R functions can be called into a C++ program.

The term *environment* is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software. R is very much a vehicle for newly developing methods of interactive data analysis. It has developed rapidly, and has been extended by a large collection of *packages*. However, most programs written in R are essentially ephemeral, written for a single piece of data analysis.

One of the prime reasons for R being so successful is the notion of a *package*. All R functions, datasets and documentation are stored in packages and only when a package is loaded are its contents made available to the user. A *base package* contains the basic functions that allow R to work, including example datasets, standard statistical functions (e.g. mean, standard deviation, etc) and graphical functions. They should be automatically available in any R installation. Furthermore, there are hundreds of *contributed packages* for R, written by many different authors. Some of these packages implement specialised statistical methods, others give access to data or hardware, and others are designed to complement textbooks.

The vast majority of contributed packages are made publicly available over the internet via the *comprehensive R archive network* (CRAN) and can be downloaded directly from the R environment. Before a package is made available on CRAN it has to pass a number of tests conducted by members of the CRAN team. These tests usually include checks for syntax errors, compilation errors/warnings and documentation errors.

5.2.1 Requirements definition & specification

Our implementation of the Gamma test into an R package focuses on code re-use and making an easy-to-use and unambiguous group of R functions. This broad starting point enabled the requirements to become more formally defined.

Requirements Definition. The R package must provide a means of nonlinear data and time series analysis using the Gamma test. The software must be of a quality to be made publicly available over CRAN and compatible with the Windows, Unix, Linux and Mac OS X operating systems.

The requirements definition was developed into a flexible *requirements specification*, with the final scope of the project limited by the ability to develop the algorithms using the tools available and within the time-limits imposed by the research.

Requirements Specification.

1. Develop a package for running Gamma test experiments.
2. Make the package of a quality such that it can be made publicly available over CRAN for Microsoft Windows, Linux, Unix and Mac OS X systems.
3. Re-use existing algorithms where possible.

5.2.2 The Gamma test package for R

The Gamma test R function uses a C++ program to rapidly compute the $\delta_M(k)$ and $\gamma_M(k)$ values for $(1 \leq k \leq p)$, R is then used to derive the Γ statistic by using its in-built regression function. Once this is done, all the routines that require a Gamma test noise estimate can be created in R by making calls to this function.

The C++ Module in detail

The C++ module carries out the following five tasks in the order presented:

1. Put the data 'received' from R into the suitable ANNLIB data structure.
2. Build the *kd*-tree.

3. For each data point search the *kd*-tree for its p nearest neighbours.

4. Compute the $\delta_M(k)$ and $\gamma_M(k)$ values for $k = 1, \ldots, p$.

5. Conduct *memory management* tasks.

One problem with the interface between R and C++ is that multi dimensional data structures cannot be passed between one another. For example, the input/output data structure in R is a matrix, but when it is passed to C++, the matrix is flattened into a one dimensional array. To solve this problem the C++ program takes the one dimensional array and transforms it into an ANNLIB two-dimensional array data structure called `ANNpointArray` for the input data. Similarly, the outputs are placed into a different `ANNpointArray` structure called `output_pts`. Once the data is in the correct ANNLIB format, we build the *kd*-tree by simply using the following C++ code

```
the_tree = new ANNkd_tree(
               data_pts,   // The input data points
               M,          // Number of points
               *dim);      // Dimension of space
```

where `the_tree` is an ANNLIB `ANNkd_tree` structure. Now that the input points have been pre-processed into a *kd*-tree, we can search the tree using ANNLIB's `annkSearch` function which will provide us with the input nearest neighbour distances and indexes. This member function is given each point from the data set i.e. `data_pts[i]`, a non-negative integer `p`, an array of point indices, `nn_idx`, and an array of distances, `dists`. Both arrays are assumed to contain at least p elements. The procedure computes the p nearest neighbours to each point in the dataset, and stores the indices of the nearest neighbours (relative to the point array given in the constructor). The nearest neighbour is stored in `nn_idx[0]`, the second nearest in `nn_idx[1]`, and so on. The squared distances to the corresponding points are stored in the array `dists`. The indexes are used to locate the nearest neighbours corresponding outputs, which are required to calculate $\gamma_M(k)$ for $k = 1, \ldots, p$. The C++ code to search the *kd*-tree and compute the $\delta_M(k)$ and $\gamma_M(k)$ $(1 \leq k \leq p)$ values is shown below.

```
for(int i = 0; i < M; i++)
{
        the_tree -> annkSearch(
                   data_pts[i],    // query point
                   p,              // number of near neighbors
                   nn_idx,         // nearest neighbor indexes (returned)
                   dists,          // nearest neighbor distances (returned)
                   error_bound);   // error bound
        for (int j = 0; j < numNN; j++)
        {
            outDist[j] = pow((output_pts[nn_idx[j]] - output_pts[i]),2)
            sumInputDists[j] += dists[j];
            sumOutDists[j]   += outDist[j];
        }
}
// Calculate the deltas and gammas....
for(int i = 0; i < numNN; i++)
{
       delta[i] = double(sumInputDists[i] / M);
       gamma[i] = double(sumOutDists[i] / (2*M));
}
```

The one remaining task is memory management. The arrays `data_pts`, `nn_idx`, `dists` and the *kd*-tree stucture `the_tree` are all allocated to memory at runtime. Once the program has finished using those objects, memory must be allowed to re-allocate that space, which is achieved using the `delete` operator. Failing to delete these objects from the heap would cause an undesirable *memory leak.* This is where the computer's RAM (Random Access Memory) is taken up by these objects until there is no available RAM and the program (in this case R) crashes.

The R module in detail

We created a function in R called `gammatest` that conducts the following tasks:

1. Run data integrity checks
2. Call the C++ module to compute the (δ, γ) pairs.
3. Use R's inbuilt linear regression function on the (δ, γ) pairs to compute the noise estimate Γ.

The user of `gammatest` must provide an input-output dataset in either a `data.frame` or `matrix` data structure with the output in the *last column.* Optional `gammatest` parameters include a mask, p and the error bound for approximate nearest neighbour search. If the user does not specify any of these optional parameters then all the inputs will be included, p defaults to 10 and the error bound is zero i.e. exact nearest neighbours are computed.

Data integrity checks are designed to capture problems in the data and parameters the user has given and abort the function immediately before any calculation is done. Example problems include missing values, computing a number of nearest neighbours that exceeds the number of data points and data that is not a real number. If the data and user parameters pass these tests we calculate from the data the number of inputs, M and the dimension of the whole input-output dataset before calling the C++ module (called `Gamma_Test_Main`) using

```
results <- .C("Gamma_Test_Main",
           as.matrix(data),
           as.integer(mask),
           as.integer(num.inputs),
           as.integer(p),
           as.integer(dimension),
           as.integer(M),
           as.double(eps),
           deltas = double(p),
           gammas = double(p))
```

The parameters of this `.C` call match those of the C++ `Gamma_Test_Main` function header, where `deltas` is an empty array that the C++ module will store the calculated $\delta_M(k)$ values and `gammas` is an empty array where the $\gamma_M(k)$ values will be stored.

Once the C++ code has finished its calculation of the $(\delta_M(k), \gamma_M(k))$ pairs we use the R's inbuilt linear regression function `lm` to conduct the Gamma regression fit and return the Gamma statistic to the user.

The Gamma statistic returned by the finished R routine was tested against the results returned by *winGamma* and the original *Mathematica* module on a variety of test datasets. We found that the results returned by the R routine matched those of these established implementations.

5.3 Using the Gamma test package: An example R session

In this section we demonstrate through a number of examples how to use the R Gamma test package[5].

[5]The figures contained within this section are the graphical output presented to the user in the R graphics device.

Firstly, to load the Gamma test package into an R session simply type at the R console:

```
> library(GammaTest)
```

All the functions, data and help files contained within the package can now ready to be used.

5.3.1 The first experiment: A simple Gamma test

Consider the following

$$y = \sin(x_1) + 2\cos(x_2) + r \tag{5.3}$$

Here x_1 and x_2 are randomly generated numbers from a uniform distribution in the range $[-2\pi, 2\pi]$ of length $M = 1000$ and r is additive Gaussian noise with $\sigma^2 = 0.05$.

The R code required to generate this dataset is

```
> x1 <- runif(1000, min=-2*pi, max=2*pi) # input 1
> x2 <- runif(1000, min=-2*pi, max=2*pi) # input 2
> r <- rnorm(1000, sd=sqrt(0.05)) # generate Gaussian noise
> y <- sin(x1) + 2*cos(x2) + r # create output
```

The `gammatest` function requires that the data is in an R `data.frame()` structure with the output in the *last* column. The code snippet below shows how we put the inputs and output in the required data frame structure, along with a printout of the first 4 points.

```
> xy <- data.frame(x1,x2,y)
> xy[1:4,]
          x1         x2          y
1  0.8610366  4.5561854  0.7462817
2 -5.5191346 -3.7596503 -0.8676516
3  3.9789590 -0.9767907  0.1983273
4  4.4693790  4.3073137 -1.9935077
```

The next code snippet demonstrates the usage and output of the `gammatest` function using the `xy` data generated above. The function also displays the Gamma regression line between $(\delta_M(k), \gamma_M(k))$ $(1 \leq k \leq 10)$ on the graphics device (Figure 5.6).

```
> gt.results <- gammatest(xy)
==========================================
     Gamma Test: Summary of Results
==========================================

Mask:                 1 1
Gamma Statistic:    0.04801128
Gradient:           0.617718
V Ratio:            0.01952209
```

In the above example both x_1 and x_2 have been included in the analysis, which the `Mask` denotes with a '1' for each input[6]. The `Gamma statistic` is the estimate of the noise variance, the `Gradient` is the gradient of the Gamma regression line, and `V Ratio` is $\Gamma/\operatorname{Var}(y)$.

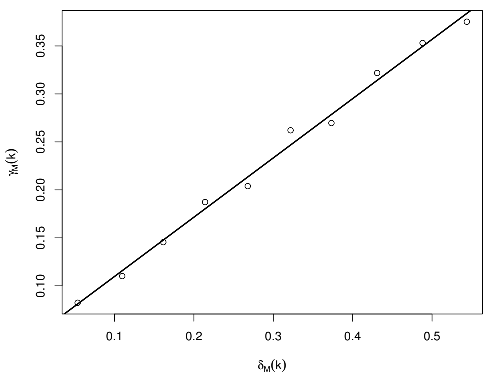


Figure 5.6: The Gamma test regression line for $y = \sin(x_1) + 2\cos(x_2) + r$ where $\sigma^2 = 0.05$, $p = 10$ and $M = 1000$.

To demonstrate the usefulness of the mask consider that we wish to *exclude* x_2 from our Gamma test analysis of the `xy` dataset. To achieve this we simply provide the `gammatest` function with a vector of binary numbers. In the code snippet below we have given the `gammatest` mask parameter a vector $(1, 0)$, which means x_1 is included but x_2 is excluded from the Gamma analysis.

```
> gt.results <- gammatest(xy, mask=c(1,0))
==========================================
     Gamma Test: Summary of Results
==========================================
```

[6]By default the `gammatest` function includes all the inputs.

```
Mask:              1 0
Gamma Statistic:   2.072668
Gradient:          -0.9626753
Standard Error:    0.04271852
V Ratio:           0.7792305
```

The noise has now increased because not all the causal inputs were included in the analysis.

5.3.2 Time series embedding functions

The `gammatest` function can be applied to time series, if the series are in the appropriate embedding (or delay) vector form. To create the embedding vectors we have provided three functions:

- The `dvec` function: Creates *regular* embedding vectors for a univariate time series.
- The `mdvec` function: Provides the embedding vectors for multivariate time series.
- The `mask2input` function. Using a mask the user can create an *irregular* embedding for a time series.

Below is an example of `dvec` on an AR(1) time series (created using R's inbuilt function `arima.sim`) with $\sigma^2 = 1$, $M = 300$ and the parameter of the model is $\phi_1 = 0.67$.

```
> ar1 <- arima.sim(300, model=list(ar=.67), sd=sqrt(1))
> plot.ts(ar1)
> dv.ar1 <- dvec(ar1, lag=1)
> dv.ar1[1:4,]
       lag.1     output
1  1.6294943  0.1153132
2  0.1153132 -1.7529210
3 -1.7529210 -0.1646481
4 -0.1646481 -1.0348422
```

The parameter `lag` of `dvec` sets the *embedding dimension* and the returned `data.frame` contains the *regular* embeddings. We are now in a position use the `gammatest` function on this 1-input/1-output dataset using

```
> gt.ar1 <- gammatest(dv.ar1)
==========================================
     Gamma Test: Summary of Results
==========================================

Mask:              1
Gamma Statistic:   0.9939498
Gradient:          -0.1034196
Standard Error:    0.05763946
V Ratio:           0.5748393
```

To use the `mdvec` function, the user provides a `data.frame` with the input time series and the output me series in the last column. An embedding dimension is also required, which the function will apply to *all* the input series. An example of this function is shown below, where from a 2-input time series/1-output time series the `mdvec` function has created a 4-input/1-output dataset.

```
> # generate nonsense data, just want to show how the function
works
> x1 <- runif(100)
> x2 <- runif(100)
> y <- runif(100)
> M <- data.frame(x1,x2,y)
> Mdv <- mdvec(M, lag=2)
> Mdv[1:2,] # display the first two data points
x1.lag1    x1.lag2    x2.lag1   x2.lag2   output
0.08040647 0.02447269 0.7532350 0.7993703 0.6996914
0.81628133 0.08040647 0.1423423 0.7532350 0.4082858
```

The `mask2input` function allows us to create an input/output dataset based on an irregular embedding. Example usage is shown below, where we have provided the function with an AR(1) time series and asked it to create a 2-input/1-output dataset with the inputs being lags 1 and 3.

```
> ar1 <- arima.sim(n=200, model=list(ar=c(.5)))
> ar1dv <- mask2input(mask=c(1,0,1), timeseries=ar1)
> ar1dv[1:2,] # display first two points
lag.1      lag.3       output
-1.1200064 0.005599498  0.2227308
0.2227308  0.261024854 -0.1845786
```

5.3.3 M-test

The M-test function calculates the confidence intervals for the Gamma statistic using the heuristic method established in section 3.5.2, and displays the M-test graph with the confidence intervals on the R graphics device.

We apply the `Mtest` routine to $y = \sin(x) + r$ where x is a random variable drawn from a uniform distribution over the range $[0, 2\pi]$, r is a random Gaussian variable representing the noise with $\sigma^2 = 0.075$, and $M = 1000$. The first 4 lines of the code snippet below show how we create this data in R, the fifth line shows how the `Mtest` function is applied, and lines 6-13 show some summary statistics including the calculated confidence intervals for σ^2.

```
> x <- runif(1000, min=0, max=2*pi) # input
> r <- rnorm(1000, sd=sqrt(0.075)) # noise
> y <- sin(x) + r # output
> xy <- data.frame(x,y)
> Mtest <- (xy, step=10, start=20, cl=0.9)
===========================
M-Test: Summary of Results
===========================
Starting point :   20
Step:              10
Conf.  Level:      90%
Conf.  Interval:   (0.07213442,0.7780483)
```

Figure 5.7 shows the graph that is plotted on the graphics device. The heuristic confidence intervals are drawn as red error bars. In this example the M-test graph looks as though it has reached a stable asymptote and therefore we can be confident that Γ is a good estimate of σ^2.

5.3.4 Increasing embedding search

In this section we show how the increasing embedding search function, `iesearch`, in the Gamma test package can be used to find the minimum embedding dimension for a chaotic time series.

As an example, consider the Hénon map, which can be loaded into the R session using

```
> data(HenonMap)
```

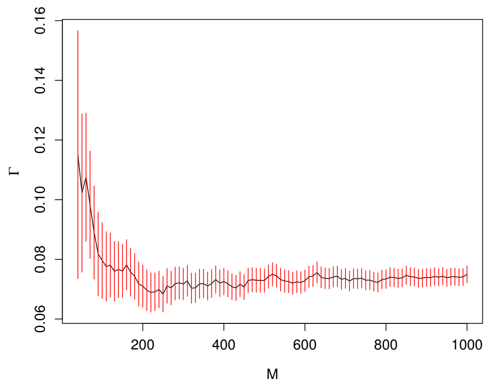


Figure 5.7: The M-test graph displayed on the R graphics device for $y = \sin(x) + r$, where $M = 1000$, $p = 10$ and $\sigma^2 = 0.075$. The heuristic confidence intervals at the 90% level are shown in red.

The `iesearch` function only allows input/output datasets to be used and therefore delay vectors of the Hénon map are created using

```
> henon.dvec <- dvec(HenonMap, lag=20)
```

The `iesearch` function can now be applied to the 20-input/1-output `henon.dvec` dataset using

```
> henon.ie <- iesearch(henon.dvec)
```

Figure 5.8 shows the plot of the Γ values over the increasing number of included lags for the Hénon map data. The Gamma statistic reaches a minimum after 2 lags, stabilises, then rises after 6 lags. We therefore conclude that the minimum embedding dimension is 2, which on referring back to the recursive rule of the Hénon map in (3.33), is correct.

5.3.5 Full embedding search

This section details the full embedding search function, `fesearch`, contained within the Gamma test package. The `fesearch` function computes Γ for each possible mask (a binary number representing the input subset), and returns a list of the masks in ascending Γ order. The information contained within the list of masks can then be used by one of the input variable selection routines (see section 6.2) to flag relevant inputs.

We generate an AR(1) time series using the `arima.sim` function. The AR(1) model specification is $y_t = 0.9y_{t-1} + r_t$, where $M = 200$ and $\sigma^2 = 1$. The `fesearch` function

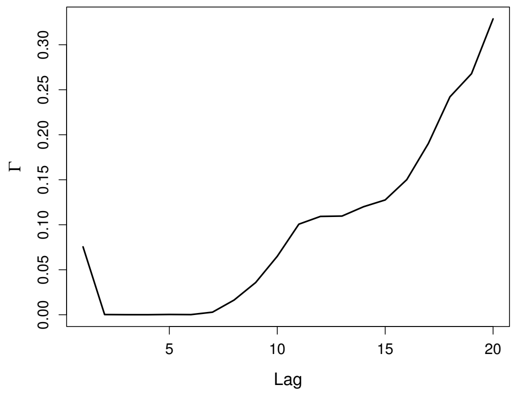


Figure 5.8: The increasing embedding search results for the Hénon map as displayed on the R graphics device. The plot shows the Gamma statistic over an increasing number of included lagged inputs.

requires the time series to be in an input/output format and therefore we employ `dvec` to create the delay vectors $\boldsymbol{x}_i = (y_{t-1}, \ldots, y_{t-10})$ for $i = 1, \ldots, (M-10)$. The function `fesearch` is then used on the 10-input/1-output dataset created by `dvec`. This example is shown in the R code below:

```
> ar1 <- arima.sim(300, model=list(ar=.9))
> ar1.dvec <- dvec(ar1, 10)
> fe.ar1 <- fesearch(ar1.dvec)

===================================================
            Full Embedding Statistics
===================================================
Full embedding search time: 17 secs
Number of Gamma tests:      1023
```

The 4 lowest Γ producing masks are shown below, where each column represents an input - the left most being lag1 and the right most lag 10.

```
> fe.ar1$mask.array[1:4,] # print out the list of masks

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    0    1    1    1    1    1     1
[2,]    1    1    1    0    1    0    1    1    1     1
[3,]    1    0    1    1    1    0    1    1    1     1
[4,]    1    1    1    1    1    1    0    0    1     1
```

A *Gamma histogram* is also plotted on the R graphics device as shown in Figure 5.9. This is where the range of Γ values are divided into bins and plotted on the horizontal axis and the occurrence of each Γ bin is plotted vertically. Figure 5.9 shows that a large proportion of masks (input subsets) caused Γ to be in the range $[0, 1]$, and therefore may contain useful information which requires further investigation to find the relevant inputs of the model.

The next chapter introduces a number of new input variable routines that flag the relevant inputs. Here we use a manual method of looking at the Gamma histogram and deciding which Γ range to conduct a *input frequency count analysis* using the `inspectgammahist` function:

```
> freq <- inspectgammahist(fe.ar1, gamma.range=c(0,1), ones=TRUE)
```

The `inspectgammahist` function requires the full embedding search results, the Γ range we wish to investigate and a logical indicating whether to count the input inclusions or exclusions. The function will return the number of input inclusions (or exclusions if `ones` is set to `FALSE`) for each input in the mask that falls within the defined Γ range. The (normalised) occurrence of each input for the above example is plotted in Figure 5.10, where it can be seen that lag 1 has the highest input occurrence within $\Gamma = [0, 1]$ which would, correctly, lead us to conclude that lag 1 is the relevant input to the underlying model.

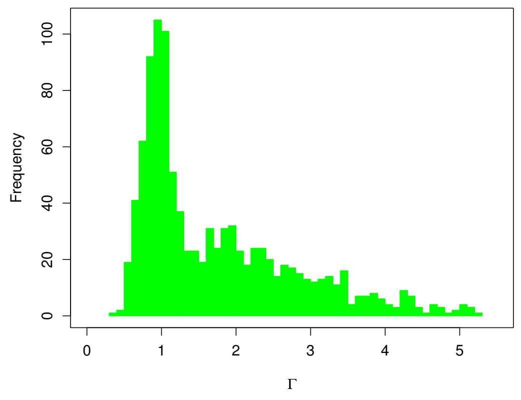


Figure 5.9: The Gamma histogram returned by the full embedding search for the AR(1) time series, with $m = 10$.

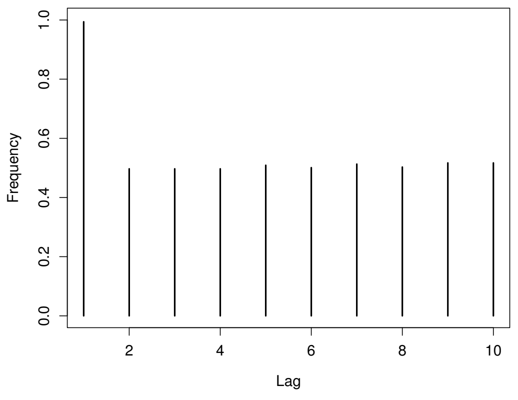


Figure 5.10: Inspecting the input inclusion occurrence for the Γs in the range $[0, 1]$

5.3.6 New embedding search functions

Here we briefly describe how to use the R functions `annfes` and `res`, that are the new *faster* embedding search routines - approximate nearest neighbour full embedding search (ANNFES) and random embedding search (RES) introduced in the next chapter.

The usage of `annfes` is as follows

```
> args(annfes)
function(data, error=1, plot=TRUE, p=10)
```

where `data` is an input/output dataset, `error` is the error bound for the approximate nearest neighbours for $m = 1$ (the algorithm adjusts the error bound as m changes), `plot` is a logical indicating whether to plot the Gamma histogram on the R graphics device (the default is true), and `p` is the fixed number of near neighbours to use in the Gamma test calculation. The function returns the list of masks (input subsets) in ascending Γ order and the Γ values themselves.

The usage of `res` is as follows

```
> args(res)
function(data, percentage=10, plot=TRUE)
```

where again `data` is the input/output dataset in a `data.frame` structure, `percentage` is the percentage of total embeddings to take at random (the default being 10%), and `plot` is a logical indicating whether to plot the Gamma histogram.

5.3.7 Embedding search analysis functions

Here we briefly describe how to use the input variable selection functions, FCM and Durrant's method, which are the focus of the next chapter.

Both FCM and Durrant's method are routines designed to analyse the Γ ordered input subsets with the aim of identifying the relevant inputs. These routines are coded into the `FCM` and `durrantsmethod` R functions. The parameters a user can provide for `FCM` are

```
> args(FCM)
function (mask.array, percentage = 12.5, ...)
```

where `mask.array` is the list of Γ ordered masks returned by one of the embedding searches, `percentage` is the proportion of the embeddings one wishes to examine (the default is 12.5%). The function plots a bar chart similar to 5.10 showing which inputs are likely relevant inputs.

For `durrantsmethod` the parameters a user can provide include:

```
> args(durrantsmethod)
function (mask.array, percentage=c(10,10))
```

where `mask.array` is again the Γ ordered masks (input subsets) returned by one of the embedding searches, `percentage` is a vector of size 2 where the first value is the proportion of embeddings to analyse that produced the lowest Γ values and the second value is the proportion of embeddings to analyse that produced the highest Γ values.

5.4 Chapter summary

In this Chapter we have described the key algorithms required for the Gamma test, most notably nearest neighbour algorithms. The trade-off between speed and accuracy of finding approximate nearest neighbours in a *kd*-tree was then examined. We concluded that computing approximate nearest neighbours does not have a considerable effect on Γ and can speed up the calculation significantly even with a small error bound, ϵ.

Furthermore, we showed how the C++ Approximate Nearest Neighbour Library (ANNLIB) could be implemented into R to provide the backbone of the routines described and developed in this thesis. These R routines provide a robust, fast and flexible implementation of the Gamma test and associated tools, that are now freely available to download from the R web-site (www.r-project.org).

Chapter 6

Input variable selection using the Gamma test

In this Chapter we develop some of the algorithms presented in (Durrant, 2002) into a suite of faster and more robust input variable selection tools.

These enhanced algorithms are applied to a wide range of popular time series models. In each case we compare and contrast the results of our algorithms with established techniques such as correlation.

From the outset of this Chapter we make it clear that our algorithms require any trend and preferably seasonality to be removed from the data *prior to analysis*. One of the pre-conditions of the Gamma test requires the input set, C, to be closed and bounded. Obviously, a consistent trend component in the data implies a potentially unbounded input variable, and so breaks the closed bounded condition. Moreover, a seasonal component may mean that the system is non-autonomous, effectively non-stationary. In this case the model would depend explicitly on time, which is potentially unbounded, and so again the closed bounded requirement for the domain of the input variables is violated. Thus we should should seek to remove such dependencies by preprocessing the data.

6.1 Introduction

The aim of input variable selection is to find the inputs that can be best used to predictively model the output y, i.e. to find the most *relevant* inputs from m *candidate* inputs. In the case of a time series, y_t, we may wish to query previous values of y_t, or previous values of another series as possible inputs.

If f is linear, then one may consider using correlations to find the suitable inputs of the series. However, if we suspect f to be nonlinear then correlations, as shown in Section 3.5.3, are not suitable for identifying predictively useful inputs. In such cases we may consider using a *input variable selection algorithm*, where there are two main components: a *criterion function* and a *search strategy* (Scherf and Brauer, 1997). The criterion function determines how good a particular set of inputs is, while the search strategy decides which set to try next. The goal is to find a set of inputs which enables the best possible model to be built. Whichever search strategy we choose, we clearly need an efficient criterion function. (Pfleger et al., 1994) distinguishes between two main types: *filters* and *wrappers*. Wrapper methods use a model to judge the worth of the input selection: we construct a model using the chosen inputs then use the performances of the model to rate the input set (e.g. Judd and Mees, 1998). The main problem with this approach is the repeated model building on the data. Not only does this limit us to models with low computational cost, but also we run the risk of finding inputs which are fitted to the vagaries of the test set not the data as a whole. The filter method is generally thought to be superior in this regard, as it does not rely on model building for the evaluation of a set of inputs. Instead, it uses statistics derived directly from the data to rate a candidate input set. There are many different statistics to use (e.g. Cherkauer and Shavlik, 1995) for this task, but for our purposes we will use the *Gamma test*.

6.1.1 Δ-correlation

One method of finding relevant lagged inputs for an embedding is to vary the delay on a set of input time series to determine how *changes* on the input times series correlate with *changes* on the target (output) time series. Durrant (Durrant, 2002) calls this technique Delta correlation. We proceed to give a synopsis of Durrant's account of Δ correlation.

From vector algebra we have

$$\cos\alpha = \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} \tag{6.1}$$

Given an input time series $\mathbf{a}$ and an output time series $\mathbf{b}$, with $\mathcal{E}(\mathbf{a}) = \mathcal{E}(\mathbf{b}) = 0$, the correlation is determined by $\cos\alpha$, where α is the angle between the two vectors. Strongly correlated time series will have $\cos\alpha \approx 1$ for a positive correlation and $\cos\alpha \approx -1$ for a negative correlation.

For example, in the context of river flows, the correlations for changes (Δ-correlations) will be positive, since the expectation is that as an upstream river rises (or falls) then the downstream river will correspondingly rise (or fall) at a later point in time. The time taken for the flow to reach the downstream point is the delay time (or lag). For other types of time series it may be that negative Δ-correlations are as significant as positive ones. The Δ-correlation algorithm is shown in Algorithm 8.

```
pre-conditions:
$input\_time\_series$ contains one or more time series
$output\_time\_series$ contains one time series

pre-processing (optional):
convert input and output time series to difference time series

{main algorithm}
vector $lag\_correlations$
for $lag = 1$ to $maximum\_lag$ do
  $lag\_correlations[lag]$ = Correlation($input\_time\_series$,$output\_time\_series$,$lag$)
end for
{end of main algorithm}

Correlation($input\_time\_series$,$output\_time\_series$,$lag$)
vector $correlation$
$M$ = number of elements in time series
$number\_items = M - 1 - lag$
$\mathbf{b}$ = last $number\_items$ in $output\_series$
for $input = 1$ to number of $input\_series$ do
  $\mathbf{a}$ = first $number\_items$ in $input\_series[input]$
  $correlation[input] = \mathbf{a} \cdot \mathbf{b}/(|\mathbf{a}||\mathbf{b}|)$
end for
return ($correlation$)
```

Algorithm 8: The Δ-correlation algorithm

Note: As stated we do not subtract means when computing the correlation. This is because for a stationary time series $\{y_t\}$ the expectation of the differences series is necessarily zero, since $\mathcal{E}(y_{t+1} - y_t) = \mathcal{E}(y_{t+1}) - \mathcal{E}(y_t) = 0$.

The attraction of this algorithm for input selection in time series is that it is very fast, compared with other alternatives, so we can look for important inputs having very large lags.

6.2 New input variable selection routines

Being able to gain an efficient insight into the noise variance via a Gamma test allows us to search different input subsets and observe the impact of including or excluding an input in the (unknown) model.

In this section we describe two new *faster* input subset searchs based on the Gamma test:

- Approximate Nearest Neighbour Full Embedding Search (ANNFES). In large dimensions the performance of a *kd*-tree is no faster than a brute force approach i.e. $O(M^2)$. Hence we use approximate near neighbours to speed up the search time.
- Random Embedding Search (RES). Instead of computing 2^m-1 Gamma statistics we select a small sample of embeddings at random for a Gamma analysis.

In section 3.5.3 we described some initial work by Durrant (Durrant, 2002) that showed how a Gamma test based embedding search can be analysed to find relevant input variables. Here we communicate a new technique to analyse the results of these embedding searches called the Frequency Combination Method (FCM). This technique extends the work of Durrant to provide a more robust and objective input selection tool.

In section 6.3 we compare and contrast these new faster embedding searches with the established full embedding search on different classes of time series. In doing so, we also make the comparison between FCM and Durrant's method to find which technique is more efficient for analysing the results of the embedding searches.

6.2.1 Approximate nearest neighbour full embedding search

In section 5.1.3 it was shown that, with a fixed input dimension, approximate near neighbours could speed up a single Gamma test whilst not seriously affecting Γ as an estimate for σ^2. In this section we propose using approximate near neighbours to calculate Γ for the large number of Gamma tests required by a full embedding search in high input dimensions. The idea being that the full embedding search time should be significantly reduced without harming the effectiveness of Durrant's method or FCM to identify the relevant inputs.

A major question we may ask is: *"What is a suitable value for the approximate nearest neighbour error bound ϵ?"* Obviously, we want ϵ to be high enough so that we can make the neccessary speed gains, but if we make it too high we may risk computing unacceptable estimates for σ^2, which in turn may harm the effectiveness of the input variable selection methods.

The optimal ϵ may depend on many factors, particularly the dimension of the input space. However, we can easily construct an experiment that may be able to provide a rule-of-thumb for selecting a suitable ϵ.

Example. Consider

$$y = \sin(x_3\pi) + 2\cos(x_8) + r \tag{6.2}$$

where x_3 and x_8 are drawn from a uniform distribution over the range $[0, 2]$, $M = 1000$, r is a Gaussian random variable with mean zero, and $\sigma^2 = 0.05$, representing the noise.

We do a number of full embedding searches for varied m and varied ϵ. We begin by computing the full embedding search using *exact* nearest neighbours i.e. $\epsilon = 0$ in the Gamma test analysis for $m = (8, 10, 12, 14, 16)$. This is achieved by adding into the input vector a number of "dummy" (irrelevant) variables. We then repeat the same experiment, but where ϵ is varied. The graph of full embedding computing times[1] for varied m and ϵ is shown in Figure 6.1. This graph shows that approximate nearest neighbours do not greatly improve the computing times in low dimensions $m \leq 14$. However, the speed up becomes more profound in higher dimensions e.g. when $m = 20$ the computing time is halved. Furthermore, using the approximate near neighbours has not adversely affected our ability to identify the relevant inputs of the model using FCM (see section 6.2.3) as Figure 6.2 demonstrates for $\epsilon = 20$ with 16 inputs.

In this example we found that computing the full embedding search with $\epsilon > 20$ did not make any further significant speed gains.

6.2.2 Random embedding search

The random embedding search (RES) works similarly to the full embedding search, only rather than computing a Gamma statistic for every possible embedding we only

[1]These search times are based on an Intel Pentium 4 2Ghz processor with 512RAM running Windows XP.

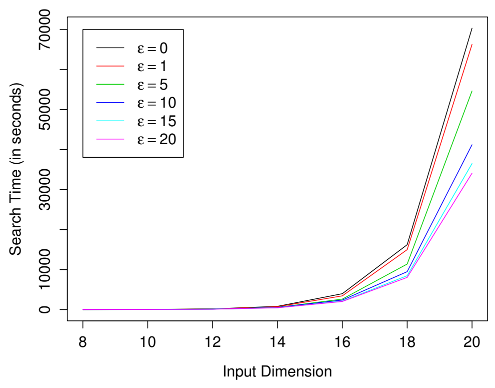


Figure 6.1: The computing times for a full embedding search on varied input dimensions and approximate near neighbour error bounds ϵ.

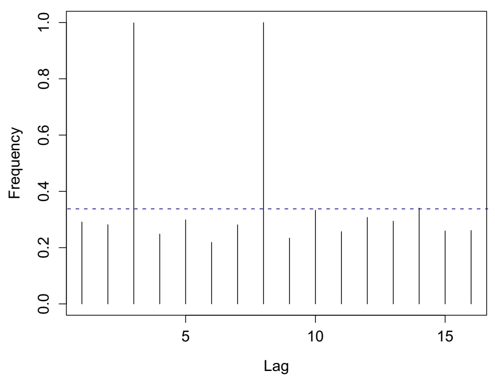


Figure 6.2: The results of the frequency combination method (described in section 6.2.3) on the $y = \sin(x_3\pi) + 2\cos(x_8) + r$ data where $m = 16$ and $\epsilon = 20$. Using this error bound halved the computing time of the full embedding search.

do so for a small random sample, s, of embeddings.

The size of the sample very much depends on the number of candidate inputs m. If $m \approx 15$ we can afford to take a fairly large sample, say $s = 10\%$. On the other hand, if $m = 20$, then the population size will be over a million embeddings and as such taking $s = 10\%$ would still require a large amount of computing time, therefore we may wish to reduce the sample to 0.1% for a quick search time.

We then apply FCM to these random embeddings to flag the relevant inputs of the model.

6.2.3 Frequency combination method

We have provided two notable extensions to Durrant's input selection method. Firstly, we provide a method for combining information in the low-Γ and high-Γ regions of a full embedding search and secondly, we have constructed confidence limits to objectively determine relevant inputs.

Combining low and high Gamma regions

The problem with examining the low-Γ and high-Γ regions of the Gamma histogram derived from a full embedding search is that they may contain conflicting information. For example, if an input has a high rate of inclusion in the low-Γ region, but a low rate of exclusion in the high-Γ region, which region do we trust the most to give a reliable answer?

Essentially we want inputs to be flagged as relevant if they have a high rate of *inclusion* in the low-Γ region, and a high rate of *exclusion* in the high-Γ region. Suppose we have a set of masks and their associated Γ values. We first sort this set into ascending order of the Γ values. We then choose from this sorted list the first n entries, we call this *lowestList*, and the last n entries, *highestList*, where n is the number of values we wish to consider in the high/low Γ-regions of the Gamma histogram. Note $1 \leq n \leq \frac{1}{2}(2^m - 1)$, and $n = \frac{1}{2}(2^m - 1)$ precisely when we have a full-embedding and consider every possible mask. Finally, for convenience we reverse the order of *highestList*, so that now it is in descending order of Γ values.

Now *lowestList* contains the half of the masks that have the lowest *Gamma* values in *ascending order*. Similarly, *highestList* contains the half of the masks that have the highest *Gamma* values, but this time sorted into *descending order*.

- The indicator variable $\mathcal{U}_i[j]$ $(1 \leq j \leq m)$, $(1 \leq i \leq n)$ is assigned a '1' if and only if input j has been included in *lowestList* at position i and also excluded in the input subset from *highestList* again at position i.

We denote the proportion of $\mathcal{U}_i[j]$ that are assigned a '1' by $\mathcal{F}_j$ for $1 \leq j \leq m$.

Guidelines for selecting relevant inputs

An input either affects the output or it doesn't. However, whether it does or not is masked by noise. Moreover, for each input/output data set of size M (randomly selected by the sampling distribution) we will get an associated set of Γ values for the masks of a full embedding. So the content of *lowestList* and *highestList* can be expected to be different each time we do a full embedding.

Let us take as our null hypothesis that an input is completely irrelevant, i.e. it does not affect the output value in the underlying process. Then it might be expected to appear or not appear with equal probability in any given mask[2]. Thus on the null hypothesis, for an *irrelevant input* j we have

$$\mathcal{E}(\mathcal{F}_j) = \frac{1}{2} \quad (1 \le j \le m) \tag{6.3}$$

where the expectation is taken over all input/output data sets of size M.

Now, the whole point of the Gamma test is that a relevant input is more likely to appear in a mask in *lowestList* and less likely to appear in *highestList*. Let us ask: what is the chance that an *irrelevant* input will appear in *lowestList* and not appear in *highestList*? Since there are four possibilities, as in Table 6.1, we might argue, all things being equal, that the probability that an irrelevant input j will appear in both lists is

$$\mathcal{E}(\mathcal{F}_j) = \frac{1}{4} \tag{6.4}$$

Combination:	(a)	(b)	(c)	(d)
lowestList	EXCLUDED	INCLUDED	INCLUDED	EXCLUDED
highestList	EXCLUDED	INCLUDED	EXCLUDED	INCLUDED

Table 6.1: The possible input inclusion/exclusion combinations from the *lowestList* and *highestList*.

One simple method of constructing guidelines would be to compute the mean and standard deviation of the m combined frequencies. One could then select an input based on whether it has a frequency greater than the mean plus[3] some multiple of the standard deviation. However, the mean is sensitive to outliers and we therefore found that this method gave unreliable results in practice.

Another simple method that we *did* find to be extremely useful for selecting relevant inputs is the upper (3^{rd}) quartile of the m combined frequencies, which is much more robust to outliers.

Example. Consider a time series generated from the following SETAR model[4]

$$y_t = \begin{cases} 1 + 0.3y_{t-1} + 0.4y_{t-2} + e_t & \text{if } y_{t-1} < 0 \\ -1 + 0.5y_{t-1} + 0.2y_{t-2} + e_t & \text{if } y_{t-1} \ge 0 \end{cases} \tag{6.5}$$

[2]In fact experience shows that this is not quite true, but we shall ignore this.

[3]We are only interested in inputs that have a high combined frequency, since small frequencies suggest the input has little relevance to the output.

[4]In fact this example is of a function f which has a single jump discontinuity and so is even more testing than it might appear. However, as mentioned in (Jones, 2004), the Gamma test is reasonably robust to a small number of discontinuities.

where $M = 1000$ and e_t is taken from a normal distribution with mean zero and $\sigma^2 = 1$. Figure 6.3 shows the Gamma histogram of the full embedding search and Figure 6.4 shows the results of FCM, where lags 1 and 2 both have a higher frequency than the upper quartile, which would lead us to include these inputs in the model.

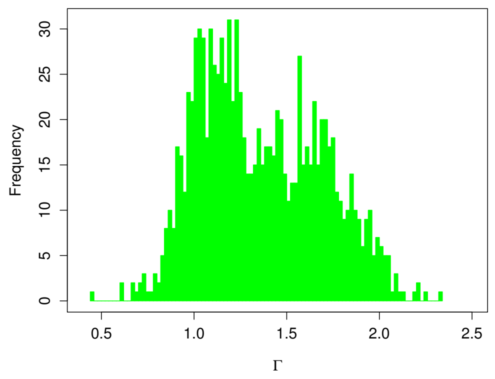


Figure 6.3: The Gamma histogram of the full embedding search ($M = 1000, m = 10$) on the SETAR time series.

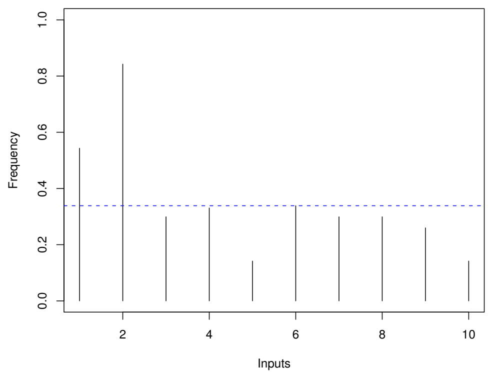


Figure 6.4: FCM applied to the results of the random embedding search ($M = 1000, m = 10$). The blue dashed line is the upper quartile of the m combined frequencies, which we use as a guideline for selecting relevant inputs.

6.3 Empirical Findings

In this section we examine the application of the new input variable selection routines of the previous section to identifying the inputs for a wide range of *simulated* time series.

6.3.1 Experimental design

The spectrum of possible time series model classes is much too vast to conduct an empirical study on each. Therefore, in our study the classes of time serie models were narrowed down to five: Autoregressive (AR), Self-exciting Threshold autoregressive (SETAR), transfer function (TF), Vector autoregressive (VAR), and open system threshold autoregressive (TARSO) models. Table 6.2 displays these models in terms of their linearity and whether of not they depend on external time series inputs i.e.

multivariate. In the case of multivariate time series we limited the number of external time series to 1 i.e. so that the series were dependent on their own past and the past of another time series. The rationale for this limitation was soley based on the computing times that would be required to compute the full embedding search.

Model Class	Linear	Nonlinear	Univariate	Multivariate
AR	X	-	X	-
SETAR	-	X	X	-
TF	X	-	-	X
VAR	X	-	-	X
TARSO	-	X	-	X

Table 6.2: The classifications of the models used in the study in terms of linearity and whether the model inputs include other time series.

For each model class between eighteen and twenty time series were simulated using different model parameters. The exact model specifications i.e. the number of observations M, model parameters, the σ^2 used, etc. are detailed in Appendix A.

Takens embedding technique was used on each of the simulated time series; for the univariate series 10 previous lags were used, and for the multivariate series, 5 previous lags from each input time series were used. On each 10-input/1-output dataset a

- full embedding search (FES),
- approximate nearest neighbour full embedding search (ANNFES) with $\epsilon = 20$,
- random embedding search (RES) with $s = 20\%$.

were conducted and the results analysed by Durrant's method and FCM to identify the relevant input variables, giving 6 possible input selection "strategies". As a comparison, autocorrelations and partial autocorrelations were applied to the univariate time series and cross-correlations for multivariate series[5].

[5]Using the guidelines proposed by (Box and Jenkins, 1970) in Table 2.1 for the univariate time series, and in the multivariate case the guidelines proposed by Tiao and Box, which are detailed in section 2.3.

6.3.2 Results

Table 6.3 shows for each input variable selection routine the number of time series where the inputs were successfully identified is shown as a percentage of the total number of time series generated per model class.

Input selection strategy $m = 10$	AR Linear Univariate $M = 300$	SETAR Non-linear Univariate $M = 500$	TF Linear Multivariate $M = 500$	VAR Linear Multivariate $M = 500$	TARSO Non-linear Multivariate $M = 500$
FES-DM	60%	75%	77%	60%	75%
FES-FCM	75%	**80%**	**94%**	75%	**85%**
ANNFES($\epsilon = 20$)-DM	55%	70%	72%	60%	75%
ANNFES($\epsilon = 20$)-FCM	75%	**80%**	**94%**	75%	80%
RES($s = 20\%$)-DM	70%	75%	77%	35%	65%
RES($s = 20\%$)-FCM	75%	**80%**	**94%**	65%	70%
Correlations	**85%**	45%	85%	**90%**	35%

Table 6.3: For each input variable selection routine the number of time series where the inputs were successfully identified is shown as a proportion of the total number of the twenty time series generated per model class.

As can be seen from Table 6.3 FCM outperforms Durrant's method as a technique for analysing the Γ-ordered embeddings in every model class.

- The most reliable input selection strategy was the full embedding search routine with FCM applied to the results.

6.3.3 Discussion

Somewhat surprisingly the simplest search routine, random embedding search (RES), has provided reliable results even though it does not compute a Γ for every possible non-trivial input subset. Provided we take a small enough sample size in high dimensions, RES is the fastest search strategy on offer. Given these encouraging results when the sample was 20% of $2^{10} - 1$ embeddings, we have tested RES when m is much higher.

Using the same time series as previously, we increase m to 20 and take 1049 random samples (0.1%) of the total number of input subsets. In terms of successful input identifications, the results of this enquiry are shown in Table 6.4. The results show that even with the relatively small sample size, the method can still be a useful tool for identifying relevant inputs in high dimensions. Moreover, the time taken to compute 1049 Gamma statistics for each time series when $M \approx 500$ is roughly 40 seconds on a machine with a 2GHz Intel processor and 512 RAM running Windows XP. This compares to 19 hours for a full embedding search on the same machine, which is a speed up of 1710%.

$m = 20$	AR	SETAR	TF	VAR	TARSO
RES($s = 0.1\%$)-DM	45%	75%	77%	30%	65%
RES($s = 0.1\%$)-FCM	75%	80%	94%	65%	70%

Table 6.4: The results of RES on the same time series, but this time $m = 20$ and the sample of random embeddings is 0.1%.

6.4 Chapter summary

In this chapter we have introduced two fast embedding search routines based on the Gamma test, and a new more robust technique for analysing the results of these searches so that predictively useful inputs can be identified to model the observed time series y_t.

Although, using approximate near neighbours speeds up the search time without affecting the ability FCM to identify the relevant inputs, the speed is still disappointing (taking a few hours) in high dimensions. The best strategy to use very much depends on the amount of time an analyst has to solve a particular predictive problem. In low dimensions ($m \leq 14$) the strategy of FES-FCM should be used as it can be computed in a few minutes and will give the most reliable results. From our empirical analysis the best strategy to use when time is of the essence is RES-FCM; it provides a high speed up without significantly affecting the reliability to identify the relevant inputs.

Auto, partial auto and cross correlations are only an effective input variable selection tool when the underlying model is linear, but as our results show, they are clearly (as expected) ineffective for non-linear models. What makes our methods appealing is that we do not assume the linearity of the underlying model. But, we cannot have things all ways; an adverse consequence of this is that our methods do not limit the class of models, they simply state "for the observed output these are the predictively useful inputs", how we model the the unknown input/output relationship is another matter entirely. However, we are aided in our search for an appropriate model by the application of a Gamma test i.e. the best model should have an MSE close to Γ.

In the next two Chapters we apply these new input variable selection routines to aid the construction of predictive models for crime rates in Cardiff city centre and global surface temperature.

Chapter 7

Applications I: Crime modelling

Crime prediction is not routinely conducted by the police. While there are numerous econometric studies, one is hard-pressed to find police departments or other police organisations making regular use of predicting - econometric or extrapolative - for deployment of *limited resources*. There are several explanations, but mainly we think that crime prediction was simply thought not to be useful or feasible until recently.

The major target of police tactics had been persons and their criminality; for example, analysing the *modus operandi* of serial criminals, and apprehending them. Of course, conventional predictive methods are of little use in forecasting the behaviour of individual serial criminals. Crime prediction, using conventional methods did not become relevant until two things occurred. First, the *criminality of places* was established i.e. spatial clusters of crime. Second, within the past 5-10 years, police began regularly *mapping* crimes using geographic information systems (GIS). The targets are still criminals, but the focus of crime prevention and law enforcement is on places where crime tends to take place. The datum are frequency counts of crimes by *time period* and *geographic area*, and it these frequency counts that are potentially predictable.

There are two aspects to crime prediction. The first is concerned with building predictive models for crime frequency that has taken place across a city area, and the second is to employ *spatial disaggregation* to answer *where* in the city the predicted crime is likely to occur.

In this Chapter we shall use the non-linear analysis tools developed in the previous chapters to aid the construction of predictive models for the frequency of *reported* criminal damage incidence and the number of reported burglaries. The spatial disaggregation of the predictions is the subject of another thesis (Ivaha, 2006) and is therefore not employed in our research.

7.1 The study area

Cardiff[1] is the capital and largest city of Wales. Located on the South Wales coast it is administered as a unitary authority. It was a small town until the early nineteenth century and came to prominence following the arrival of industry in the region and the use of Cardiff as a major port for the transport of coal. Cardiff was made a city in 1905 and proclaimed capital of Wales in 1955. In the Census 2001 the population of Cardiff was 305,340, making it the 16th largest settlement in the United Kingdom.

The industrial development and growth of Cardiff was initially based on the transportation of coal, where coal mined from the Rhondda Valley was sent to the port by barge along the valley of the River Taff, initially by canal and later by the Taff Vale Railway. A logical extension of the coal business was the development of an iron and steel industry, based largely on the port and the coal of the South Wales valleys. The 1980s brought closures to the industry in the entire region, and thousands of local workers were made redundant as the steel industry moved out of Cardiff.

Cardiff's port, known as Tiger Bay, was once one of the busiest ports in the world. After a long period of neglect as Cardiff Bay, it is now being regenerated as a popular area for arts, entertainment and nightlife.

The city's central region, extending from the Hayes is now full of attractive modern buildings. This area of Cardiff will also shortly be redeveloped, as part of St. Davids Centre - Phase 2 project. The affected area is bounded by The Hayes, Mill Lane, Mary Ann St and Bute Terrace.

The city has a professional association football team (Cardiff City F.C), a professional rugby union football team (Cardiff Blues R.F.C) and the Cardiff Devils ice hockey team. The city also features a 75,000 capacity international sporting venue: The Millennium Stadium. Between 2000-2006 the Millennium stadium hosted all the UK's premier sporting events in-place of the national Wembley stadium (London) which was undertaking a massive reconstruction. Events included the FA cup final, the League cup final, the football divisional playoffs, the FA vase trophy, the LD vans trophy, the 2002 and 2006 Heineken European rugby union cup finals and the rugby league challenge cup final. Furthermore, the Millennium stadium is the home of the Wales rugby union team, whom play on average 6 games a year at the stadium, and the Wales association football team that play a similar number of games.

[1]The name Cardiff is an Anglicisation of the Welsh name *Caerdydd.*

Cardiff is home to Cardiff Castle, the National Assembly for Wales, St. David's Hall, the National Museum and Gallery, and Cathays Park (municipal buildings), and the Cardiff Metropolitan Cathedral. The Welsh National Opera moved into the Wales Millennium Centre in November 2004.

Cardiff's centre has a number of parks with Bute park extending northwards from the top of the Cardiff's main shopping street (Queen Street); when combined with the adjacent Llandaff fields to the northwest it produces a massive open space skirting the river Taff. Other popular parks include Roath Park in the north, which also has a very popular boating lake, Victoria Park, and Thompson's Park.

The city has its own university, Cardiff University, as well as two University of Wales colleges, the University of Wales Institute, Cardiff and the Royal Welsh College of Music and Drama. The number of students living in Cardiff is approximately 30,000 (Census 2001), that accounts for approximately 10% of the total population.

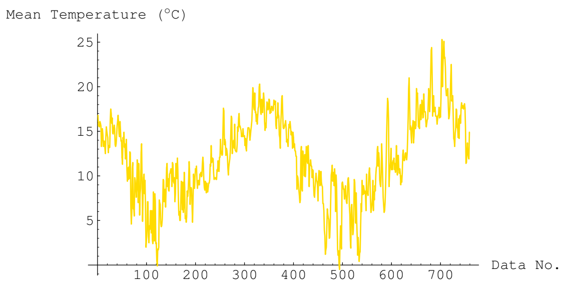
7.2 Crime data

In this section we shall describe the daily equisampled crime and weather data sources that have been made available to us, over the period 2-Sept-01 to 30-Sept-03 i.e. 759 data points.

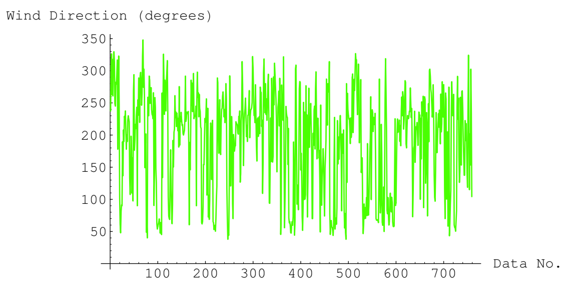
7.2.1 Weather data

It has been demonstrated that crime correlates with weather conditions (Cohn, 1993, 1996; Cohn and Rotton, 1997, 2000, 2003). If this result is also true of the Cardiff crime data, then it is reasonable to assume that weather information could be a useful input variable for our predictive models. Hence, we obtained the daily weather readings shown in Figure 7.1 that were taken from the St. Mary's street weather station[2] over the same 759 day period as the crime data. It is highly likely that these weather readings will contain some degree of measurement error, and as we observed in Chapter 4 this can increase the error variance for the predictions. However, we hope to minimize this error by employing the input variable selection routines described in Chapter 6.

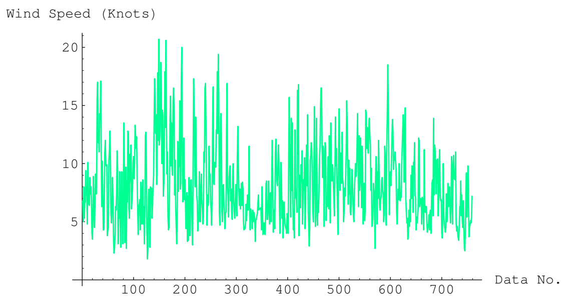
[2]i.e. positioned in the city centre at national grid reference (3182E, 1761N) and at an altitude of 52 metres above mean sea level.



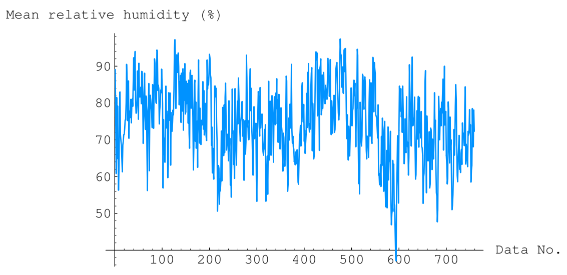
(a) Mean daily temperature.



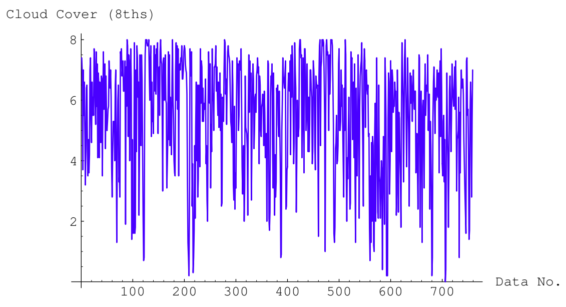
(b) Wind direction (degrees).



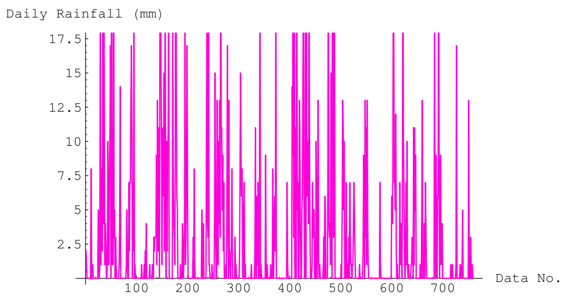
(c) Wind speed (knots).



(d) Mean relative humidity (%).



(e) Cloud cover (8ths).



(f) Rainfall (mm).

Figure 7.1: The Cardiff daily weather readings.

7.2.2 Criminal damage data

The daily number of criminal damage cases reported to the police over the period 2-Sept-01 to 30-Sept-03 is shown in Figure 7.2. As the figure shows, there are a number of spikes in criminal damage throughout this period. Some of which are associated with a calender event (particularly Halloween) and sporting events.

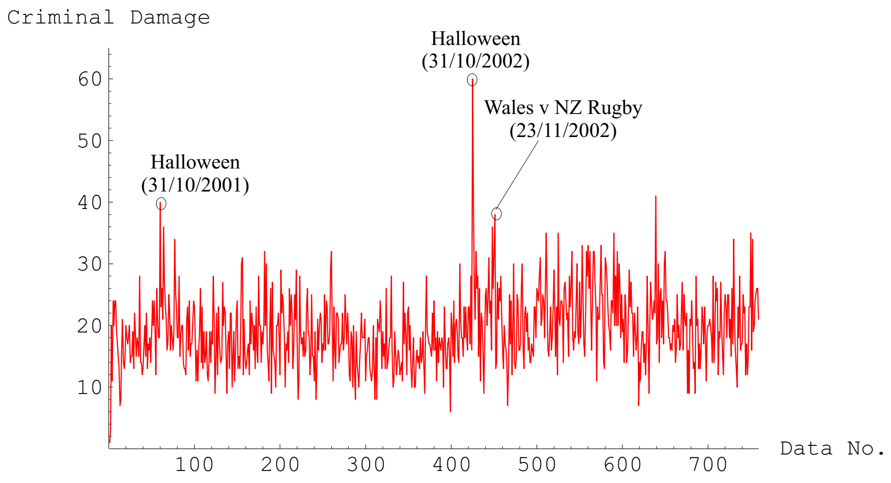


Figure 7.2: A plot of the daily number of criminal damage cases reported to the Police from 02-Sept-2001 to 30-Sept-2003.

7.2.3 Burglary data

The daily number of burglary cases reported to the police over the period 2-Sept-01 to 30-Sept-03 is shown in Figure 7.3. Unlike, the criminal damage data where spikes could reasonably be attributed to events, we cannot presume the same for the burglary data.

7.3 Criminal damage modelling

In this section we firstly analyse the criminal damage data using the tools we have developed in previous chapters, and then using the information from this analysis we go on to build a one-step ahead predictive model.

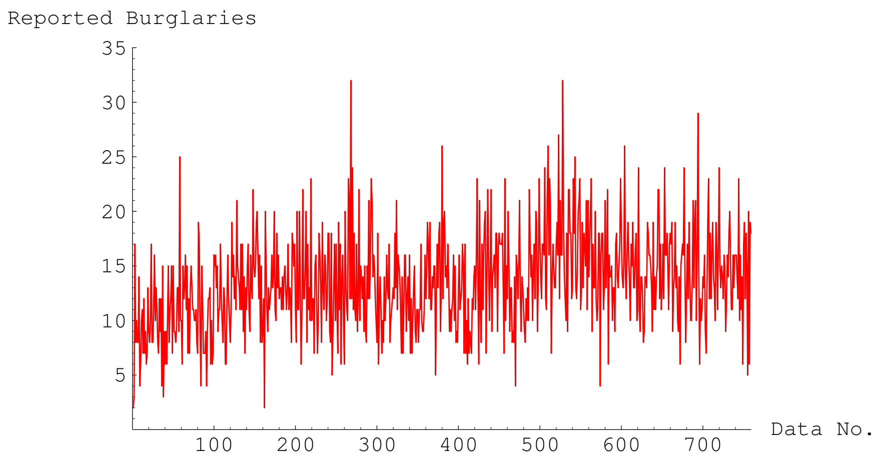


Figure 7.3: A plot of the daily number of burglaries reported to the Police from 02-Sept-2001 to 30-Sept-2003.

7.3.1 Input selection & data analysis

It is reasonable to assume that the number of criminal damage incidents will depend on the weather that occurs at the *same time*. Given that weather conditions can be accurately predicted 1-day ahead, we shall use as a candidate input the weather data at lag 0. Furthermore, criminal damage may depend on periodicities that go back over the past month, the appropriate lags for such a predictive model might also be expected to be large. Clearly there is no hope of doing a full embedding search with so many input variables. Therefore we adopt a two stage filtering process for input selection.

- First we look at the Δ-correlations between the target time series (criminal damage) and the potential input time series. In this case the seven potential input time series are: Temperature, Wind Direction, Wind Speed, Humidity, Cloud Cover, Rainfall, and Criminal Damage.

Because it is computationally feasible to do this over very large time scales we can use Δ-correlation to select the right orders of magnitudes for the different input lags. Having used Δ-correlation to select appropriate lags:

- Depending on the number of inputs we next use an appropriate embedding selection routine (either full embedding or a statistical sample of random embeddings) together with the analysis routines discussed earlier (Durrant's method and our own frequency combination method).

In this way we refine the selection and reduce the number of inputs to a manageable quantity.

Δ-correlation for criminal damage

We begin the input selection procedure by examining the Δ-correlation plots for the different data sets. In Figure 7.4 we see the Δ-correlation plots for criminal damage against the other variables of interest with lags going back 40 days.

Whilst small scale periodic maxima and minima for the Δ-correlations can be seen for most variables, it is clear from these graphs that the largest Δ-correlations for Criminal damage is with itself, rainfall, and cloud cover.

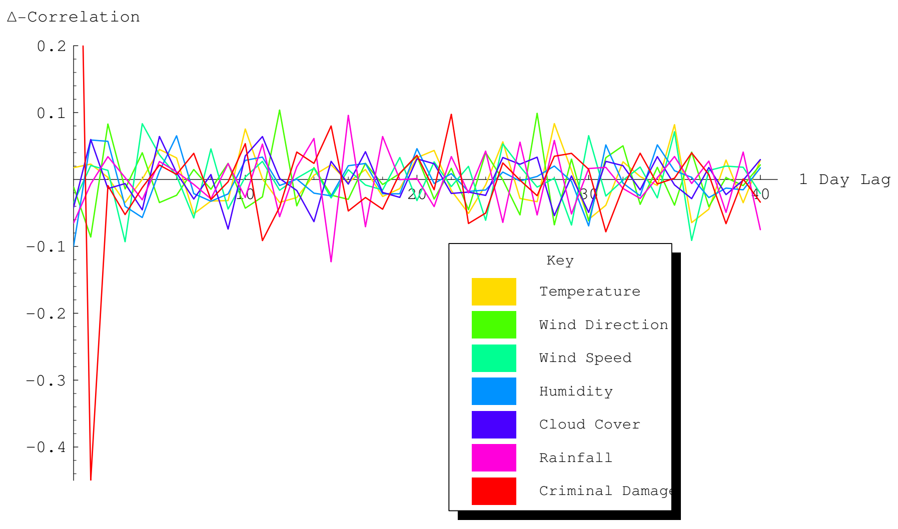


Figure 7.4: The Δ-correlations for the criminal damage data.

From the Δ-correlation results illustrated in Figure 7.4(a) we construct Table 7.1.

In the actual lag selection we will take the two most recent, which for the weather data means lags 0 and 1 and for the criminal damage data lags 1 and 2, we also take the two lags corresponding to the largest absolute Δ-correlations as shown in the table. This will give four inputs per variable, i.e. a total of 28 inputs for a one-step prediction of the single output criminal damage. These 28 inputs will then be examined by the Gamma test techniques developed earlier.

Points of interest arising from Table 7.1 are that, as one might expect, the Δ-correlations with rainfall are negative at lag 0, and the Δ-correlations with temperature are all positive.

Although the negative correlation between humidity and criminal damage makes sense in so far that one would expect criminal damage to go down in close weather conditions, the strength of the correlation is quite surprising.

	Lag	Δ	Lag	Δ	Lag	Δ	Lag	Δ
Temperature	0	0.018	1	0.023	10	0.076	28	0.084
Wind Direction	0	-0.011	1	-0.086	12	0.104	27	0.099
Wind Speed	0	-0.035	1	0.021	3	-0.093	4	0.083
Humidity	0	-0.100	1	0.059	6	0.065	30	-0.069
Cloud Cover	0	-0.042	1	0.060	5	0.064	9	-0.074
Rainfall	0	-0.065	1	-0.007	15	-0.123	16	0.096
Criminal Damage	1	-0.450	2	-0.009	11	-0.091	22	0.080

Table 7.1: The two most recent lags and two lags based on the strongest Δ-correlations for criminal damage.

Embedding searches for criminal damage data

Having made a preliminary selection of 28 inputs using Δ-correlation we next proceed to use the tools described in Chapter 6 to refine the selection. The results are summarized in Table 7.2.

The five inputs selected by FCM produce a Gamma statistic of 30.875. What is worrying here is the $V_{ratio} = 0.85599$, which suggests that our chances of building a good smooth predictive model for criminal damage are slim. The eight inputs selected by Durrant's method produced a slightly higher Gamma statistic at 33.047 ($V_{ratio} = 0.916$). Thus the final selection of inputs emerges as the five given in Table 7.3.

M-test and scatter plot analysis

The M-test graph for the 5-input/1-output criminal damage data set is shown in Figure 7.5.

The statistics are summarised in Table 7.3.

Number of candidate inputs	28
Number of masks	268,435,455
Sample size	8053
Durrant's method	Temp Lags 0 and 28
	Wind direction Lag 0
	Wind speed Lag 4
	Cloud cover Lag 0
	Rainfall Lag 16
	Criminal damage Lags 1 and 2
FCM Selected inputs	Temp Lag 28
	Wind direction Lag 1
	Cloud cover Lag 1
	Criminal damage Lags 1, 2

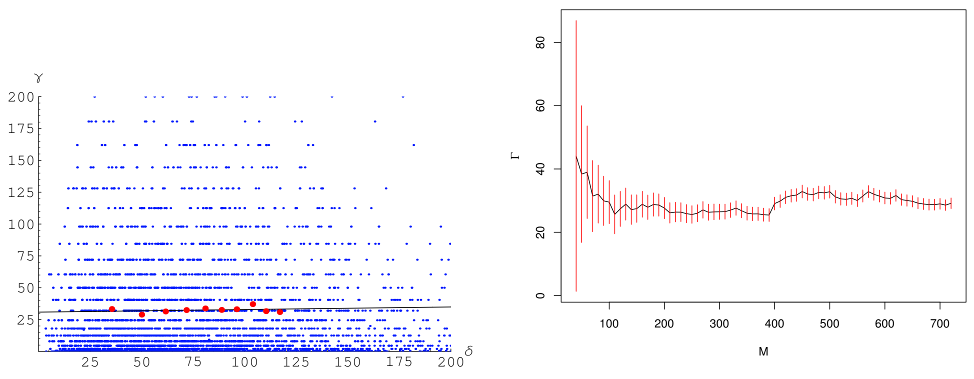
Table 7.2: The inputs selected from a random embedding search using Durrant's method and the Frequency Combination Method to analyse the Γ-ordered input subsets. The candidate inputs were chosen using the Δ-correlation.

		Training set	
Inputs used		Temperature Lag 28	
		Wind direction Lag 1	
		Cloud cover Lag 1	
		Criminal damage Lags 1 and 2	
M		600	
Γ	V_{ratio}	30.875	0.85599
Heuristic confidence intervals		(28.96204, 32.69663) at 95% level	
$\text{MSE}_{\delta\gamma}$ of Γ regression		4.292753	
MSE reached in training		30.8719	

Table 7.3: Key statistics for 5-input/1-output training set.

From the scatterplot of Figure 7.5 we notice a horizontal *banding effect.* As (Jones, 2004) remarked, this is indicative of low precision data since there are many pairs of $(\boldsymbol{x}_{N[i,k]}, y_{N[i,k]})$, $(\boldsymbol{x}_i, y_i)$ such that $\gamma = \frac{1}{2}(y_{N[i,k]} - y_i)^2$ take the same value. In our case, both criminal damage and burglary incidence are integer values, that are repeated multiple times in the data set.

The M-test graph suggests that the Gamma value is beginning to stabilise around 600 data points and therefore we decided to train the model on these first 600 points.



(a) Scatter Plot: $\Gamma = 30.875$. (b) M-test

Figure 7.5: Scatterplot and M-test with confidence intervals, for the 5-input/1-output data set, $M = 600$.

7.3.2 Model for criminal damage

With the five inputs selected (as in Table) we built a one step predictive BFGS neural network ($5 \rightarrow 10 \rightarrow 5 \rightarrow 1$) using the first 600 data points. The results of testing on unseen data can be seen in Figure 7.6.

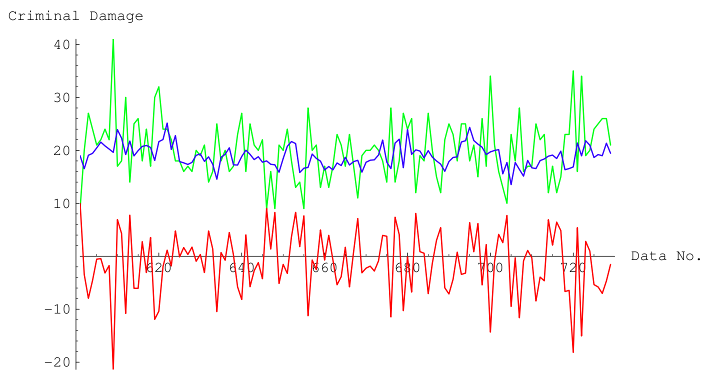


Figure 7.6: Model test results for one step temperature BFGS neural network ($5 \rightarrow 10 \rightarrow 5 \rightarrow 1$) on unseen test data from. Green: actual criminal damage. Blue: predicted criminal damage. Red: error.

The summary of performance statistics are displayed in Table 7.4. Although the model prediction error seems reasonable when compared with the mean change in criminal damage, the figure clearly shows that the model predictions are inadequate to be of any real practical use.

Section of test set	MSE	RMSE	Mean change in criminal damage	Successfully predicted turning points (%)
601 to 729	35.977	5.998	5.730	34.4

Table 7.4: The performance of the one step predictive BFGS neural network ($5 \rightarrow 10 \rightarrow 5 \rightarrow 1$) on the *unseen* criminal damage test data.

7.4 Burglary modelling

We now attempt to build a predictive model for the burglary dataset.

7.4.1 Input selection & data analysis

We now apply the same two-stage input selection methodology described in Section 7.3.1 to the burglary data.

Δ-correlation for the burglary data

In Figure 7.7 we see the Δ-correlation plots for burglary against the other variables of interest with lags going back 40 days. It is clear from these graphs that the largest Δ-correlations for burglaries is with itself, rainfall, and temperature.

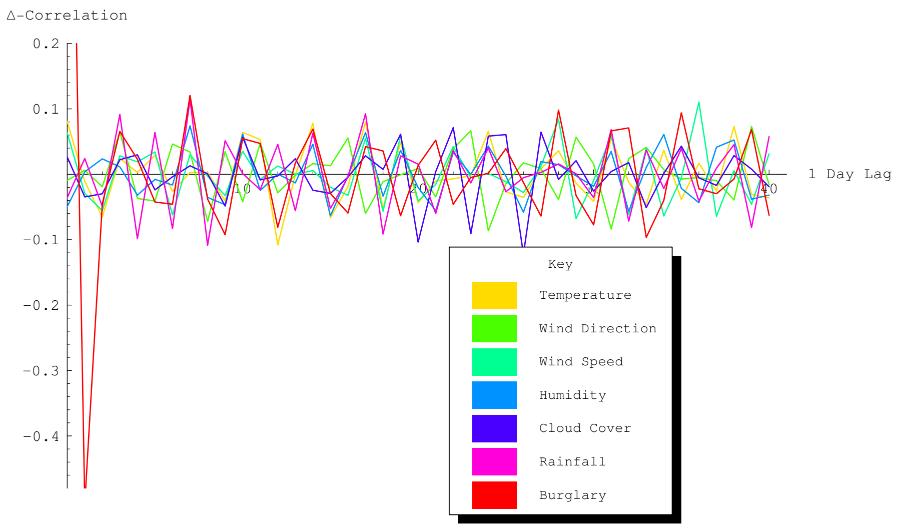


Figure 7.7: The Δ-correlations for the burglary data.

From the Δ-correlation results illustrated in Figure 7.7(a) we construct Table 7.5.

Using the same principle adopted for the lag selection in the criminal damage data we take the two most recent lags, and the two lags corresponding to the largest absolute Δ-correlations as shown in the table. This will give four inputs per variable, i.e. a total of 28 inputs for a one-step prediction of the single output criminal damage. These 28 inputs will also be examined by the Gamma test techniques developed earlier.

	Lag	Δ	Lag	Δ	Lag	Δ	Lag	Δ
Temperature	0	0.081	1	-0.011	12	-0.108	14	0.078
Wind Direction	0	-0.009	1	0.006	8	-0.072	23	0.067
Wind Speed	0	0.063	1	-0.029	28	0.084	29	-0.068
Humidity	0	-0.049	1	0.004	7	0.074	10	0.061
Cloud Cover	0	0.027	1	-0.035	20	-0.104	27	-0.122
Rainfall	0	-0.036	1	0.024	7	0.115	8	-0.108
Burglary	1	-0.504	2	-0.045	7	0.121	28	0.098

Table 7.5: The two most recent lags and two lags based on the strongest Δ-correlations for the burglary data.

Embedding searches for burglary data

Having made a preliminary selection of 28 inputs using Δ-correlation we again use the tools described in Chapter 6 to refine the selection. The results are summarized in Table 7.6.

Number of candidate inputs	28
Number of masks	268,435,455
Sample size	8053
Durrant's method	Wind direction Lag 0
	Wind speed Lags 28 and 29
	Burglary Lag 7
FCM Selected inputs	Temp Lag 1
	Wind direction Lag 0
	Wind speed Lags 28 and 29
	Cloud cover Lag 22
	Rainfall Lag 4
	Burglary Lags 7 and 28

Table 7.6: The inputs selected from a random embedding search using Durrant's method and the Frequency Combination Method to analyse the Γ-ordered input subsets. The candidate inputs were chosen using the Δ-correlation.

The eight inputs selected by FCM produce a Gamma statistic of 17.00. As in the

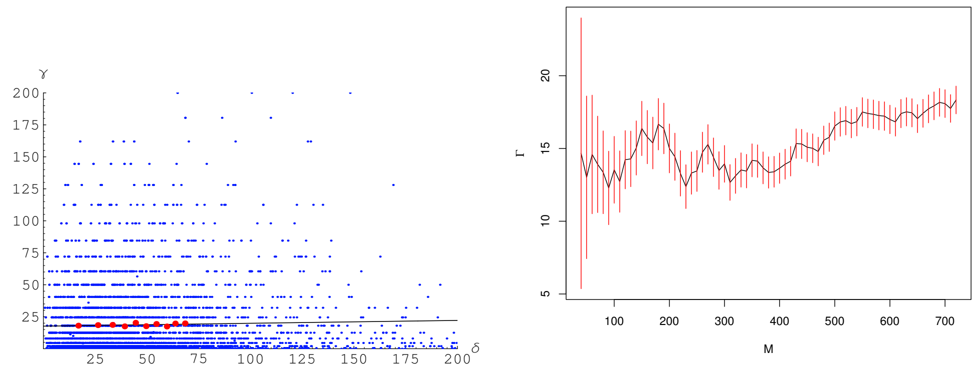
	Training set
Inputs used	Temperature Lag 1
	Wind direction Lag 0
	Wind speed Lags 28 and 29
	Cloud cover Lag 22
	Rainfall Lag 4
	Burglary Lags 7 and 28
M	600
Γ	17.00
V_{ratio}	0.843
Heuristic confidence intervals	(16.01485, 17.97538) at 95% level
MSE$_{\delta\gamma}$ of Γ regression	0.869
MSE reached in training	17.0233

Table 7.7: Key statistics for 8-input/1-output training set.

criminal damage data, the V_{ratio} is very high at 0.843, which suggests that our chances of building a good smooth model are slim due to a high level of noise. The 4 inputs selected by Durrant's method produced a slightly higher Gamma statistic at 17.64 ($V_{ratio} = 0.875$). Thus the final selection of inputs emerges as the eight given in Table 7.7.

M-test and scatter plot analysis

The M-test graph for the 8-input/1-output burglary data set is shown in Figure 7.8.



(a) Scatter Plot: $\Gamma = 17.00$. (b) M-test

Figure 7.8: Scatterplot and M-test with confidence intervals, for the 8-input/1-output data set, $M = 600$.

The statistics are summarised in Table 7.7.

From the scatterplot of Figure 7.8 we notice that the burglary data also suffers from horizontal banding effect. The M-test graph suggests that the Gamma value is beginning to stablise around 600 data points and therefore we decided to train the model on these first 600 points.

7.4.2 Model for burglary

With the eight inputs selected (as in Table) we built a one step predictive BFGS neural network ($8 \to 16 \to 8 \to 1$) using the first 600 data points. The results of testing the model on unseen data are shown in Figure 7.9.

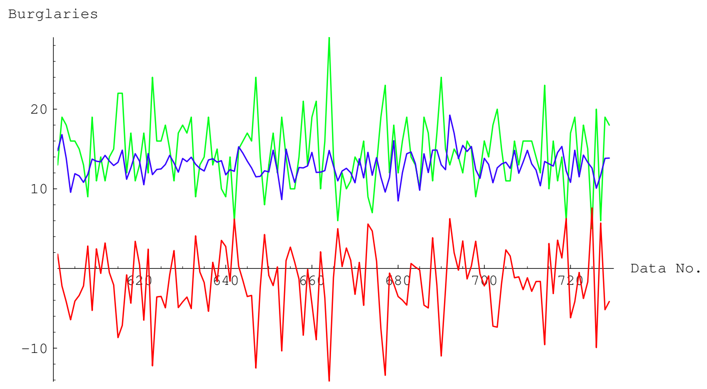


Figure 7.9: Model test results for one step temperature BFGS neural network ($8 \to 16 \to 8 \to 1$) on unseen test data from. Green: actual number of burglaries. Blue: predicted number of burglaries. Red: error.

The summary of performance statistics in Table 7.8, show that the error in the model predictions are within the mean change of burglary incidence. Although this may lead us to conclude that the model has done well, we must not lose sight of the fact that Figure 7.9 clearly shows the model predictions are inadequate to be of practical use.

Section of test set	MSE	RMSE	Mean change in burglary cases	Successfully predicted turning points (%)
601 to 729	21.462	4.633	4.756	50

Table 7.8: The performance of the one step predictive BFGS neural network ($8 \to 16 \to 8 \to 1$) on the *unseen* burglary test data.

7.5 Chapter summary

In this Chapter we have attempted to used the analysis tools developed in this thesis to help aid the construction of smooth predictive models for both criminal damage and burglary incidence in Cardiff. The one-step ahead predictive models that were built performed very badly on unseen data. It did not matter how much machinery we focussed on the data made available to us, the results did not match our original aspirations. The fact is the available data is insufficient to build *meaningful* smooth predictive models.

This is not to say that there is no practical merit in the tools we have considered, as we shall see in the next Chapter, our tools have been instrumental in helping to develop predictive models of global surface temperature.

Chapter 8

Applications II: Climate modelling

In this chapter we shall attempt to consolidate various environmental time series, and then use techniques developed earlier in the thesis to aid the construction of predictive models for global temperature.

For relatively recent times Figure 8.1 shows the 75 yr-smoothed, scaled[1] proxy[2] N. hemisphere surface temperature from the much criticised (Mann et al., 1998, 1999) (see Appendix B.1.1) in red, and a similar graph for CO_2 using data from the Law Dome Ice core data (see Appendix B.1.2) in blue. Points of interest on this graph are the 'Little Ice Age' (roughly 1425 – 1900) and 1816 'The year with no summer'. The most likely cause for the latter was volcanic influences, principally the largest eruption in modern times - Tambora in Indonesia during 1815.

Tropospheric concentration of gases such as CO_2, CH_4 and N_2O are known to be significant in affecting climate. Increased concentrations of greenhouse gases in the atmosphere due to human activities are most likely the underlying cause of warming in the 20th century, although this is still a matter of dispute by some. Other important factors are insolation (**In**coming **Sol**ar Radi**ation**) and atmospheric dust, which affects the Earth's albedo.

Thus a potentially interesting area of application for the techniques developed in this work is to the general subject area of greenhouse gases and global warming. We first

[1]Where, for comparison, we wish to overlay graphs with different vertical scales we shall normalize them to have mean zero and standard deviation 0.5.

[2]Since instrumental measurements are not available for earlier times often 'proxy' values are estimated from other measurable quantities. Using a variety of tree ring and ice core proxy records, along with the 20th Century instrumentation record, Mann *et al.* reconstructed temperatures in the Northern Hemisphere over the last one thousand years.

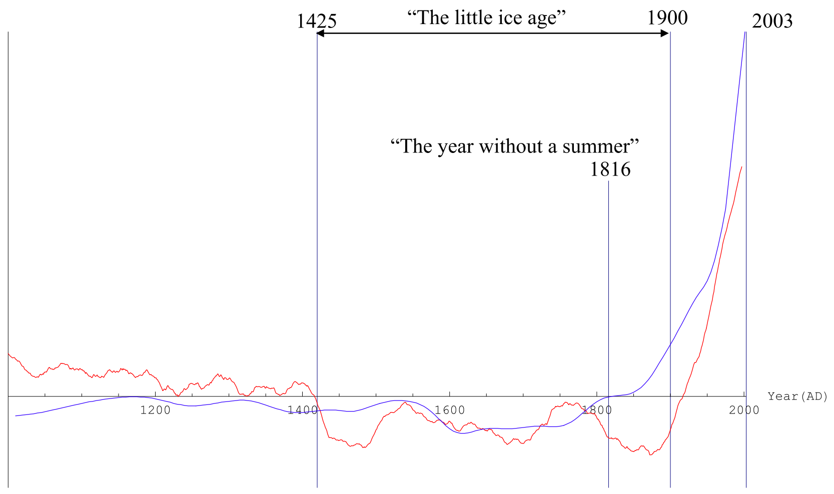


Figure 8.1: Scaled proxy northern hemisphere surface temperature, 75 yr-smoothed (red) and scaled 75 yr-smoothed CO_2 (blue), for the last 1000 years.

attempt to collect and consolidate suitable data from a variety of sources, and then investigate the construction of predictive models.

Although, in this initial study, we should not expect to produce convincing predictive models, nevertheless it might be interesting to compare *predictions* of models constructed on pre-historical data with *actual* current data. Our preliminary analysis may, in some small measure, contribute to the ongoing debate regarding the rôle played by the activities of mankind in promoting global warming.

We shall treat two separate cases of interest. First, we use available paleoclimate data to construct long-scale models for temperature covering roughly the last half-million years. Second, we shall take the available data sets for the last two thousand years, construct appropriate models for temperature, and see what these models tell us (if anything) regarding the relatively recent future - the next hundred years.

8.1 Paleoclimate data and modelling

In this section we explore and describe the paleoclimate data sources available and create consolidated 50, 100 and 500 year spacing equisampled time series data sets covering a 422 kyr interval.

8.1.1 Vostok Ice core data

One very useful source of environmental time series over very large time scales is the 'Vostok Ice Core Data for 420,000 Years' (Petit et al., 1999) (see Appendix B.2.1).

These data files provide CO_2, CH_4, Surface temperature changes, and dust concentrations over a 422kyr interval. Because surface temperatures obviously depend on location and altitude, in order to facilitate comparisons of different records climatologists use the average temperature over the period $1961 - 1990$, taken at the given location, and then measure temperature *changes* called the '*temperature anomaly*' at the surface relative to this baseline.

There are some repetition pairs for the same time point (which we have averaged) and the samples are not equispaced in time, nor do they always cover the same time interval. For the modelling techniques we use it is helpful to have a constant sampling rate. Therefore we constructed a *Mathematica* file which reads in the raw data and constructs interpolation functions for each data set (we have used linear interpolation but could equally have used quadratic or cubic interpolation). We are then able to write out data sets with a constant (user selectable) sampling rate, over the largest common interval.

We have thus reconstructed all the data sets for 50, 100 and 500 year sampling rates. The consolidated data sets then run over the interval $[-414.085, -4.509]$ kyr.

One significant problem for the modeller is that the time axis in these data sets is to some degree undetermined, at least to within a monotone transformation. For example CO_2 and CH_4 are calibrated against 'Gas Age', temperature against 'Depth Corrected Age' and Dust against 'Ice Age'. Whilst for the cognoscenti of climatology the relationships between these various estimates are no doubt well understood, for the data-modeller they pose a real issue. As Eric W. Wolff remarked

> "Getting the correct timescale is one of the most difficult aspects of what we do. The Greenland cores are better dated because there have been attempts at counting annual layers (but even there you will notice that the two cores, the European and US ones), drilled a few km apart, have timescales that differ quite a lot in the glacial period."

In the meantime we have just accepted that the time axis of the data sets is imprecise and may not be well correlated for different data sets. We are, in any event, dealing with noisy time series, as described in Chapter 4.

8.1.2 Lack of long-term solar irradiance data

The *solar 'constant'* is defined as the net solar energy per unit area arriving at the earth's orbit (1 Au from the Sun): the value is approximately 1366 Wm^{-2}. It cannot readily be measured from the ground and so was one of the first measurements consistently taken by satellites (using thermocouples). Figure 8.2 represents a composite of several satellite measurements and shows the value of the solar constant since 1978. It is apparent that the solar constant has a small periodic variation, in fact highly correlated with the 11 yr sunspot cycle.

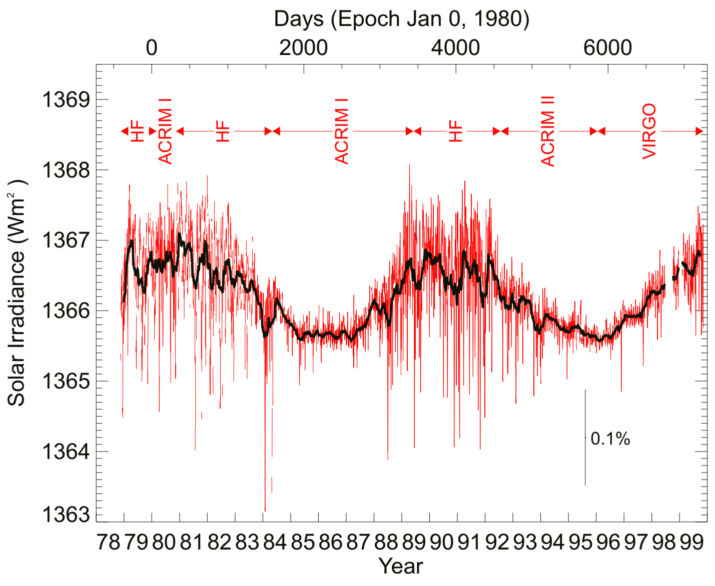


Figure 8.2: Irradiance since 1978: composite satellite measurements (C.Fröhlich and Lean, 1998b,a).

The relationship between Insolation (the fraction of arriving solar energy that is attributable to unit area on the planetary surface), also measured in Wm^{-2}, to the solar constant is quite simple. Approximating the Earth as a sphere, the amount of energy intercepted by the Earth is proportional to its cross-sectional area, i.e. πR^2, where R is the radius of the Earth. Over a 24 hr rotation period this incident energy must be shared across the whole of the Earth's surface, and so is divided by $4\pi R^2$. Thus to obtain insolation from the solar constant we merely divide by 4, which gives 341.5 Wm^{-2} at 1AU.

Unfortunately our understanding of long-term solar irradiance is restricted to generalisations regarding main sequences stars. At present there are no models or historical data extraction techniques which enable us to determine variations of solar irradiance over the last 500 kyr (say). There is some work on reverse engineering solar irradiance for the last few hundred years using historical reconstructions of the Earth's temperature. Clearly such data would be of little use to us in the present context, carrying as it does the danger of engaging in circular argument.

As it is obviously an important factor in determining global temperature, the lack of adequate solar irradiance data is a potential problem for long term climate modelling. However, on a scale of 500 kyr, it is not unreasonable to assume that the radiative output of the Sun has been approximately constant, and rely largely on insolation models to estimate solar thermal input to the system.

8.1.3 Berger Insolation data

Finally we have used the Berger (Berger and Loutre, 1991) Insolation data set (see Appendix B.2.2). This *tour de force* of astronomical calculation produces a file containing data on changes in the Earth's orbital parameters, and the resulting variations in insolation in units of Watts per square meter, over a 50 myr interval. Figure 8.3 shows the Berger data for the 500 kyr interval of interest.

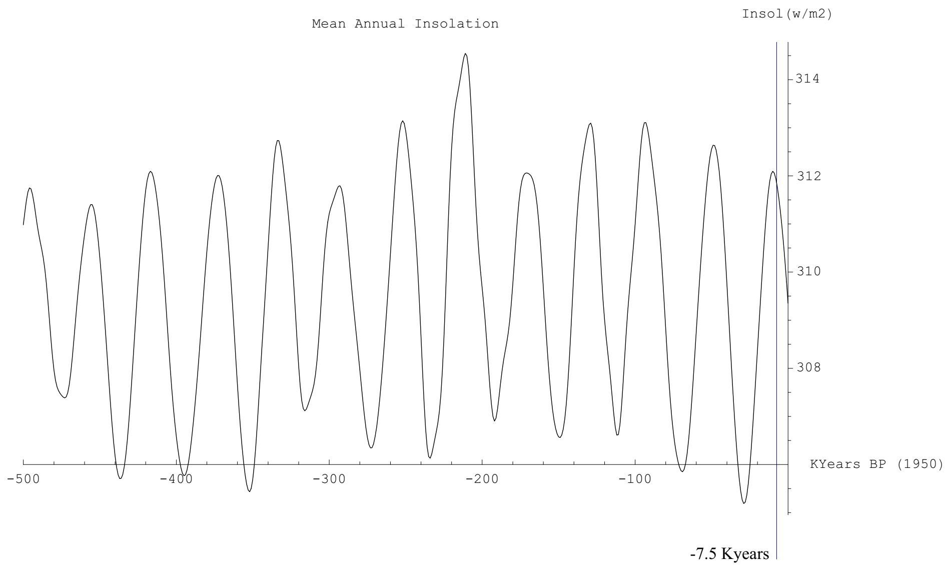


Figure 8.3: Insolation. (Note that 7.5 kyr ago insolation was decreasing.)

From the table supplied by Berger, using *Mathematica*, we built a first order interpolation function and produced time-equispaced data in steps of 50, 100, 500 years over the 450 kyr interval which covers the Vostok data. For the 500 year data we obtain 820 points. The mean annual isolation averaged over all latitudes is 309.47 Wm^{-2} with a minimum of 305.18 and a maximum of 314.45. An FFT on the 500 year data determines that insolation has an approximate 40.95 kyr principal period, which can be seen in Figure 8.3.

8.1.4 Input selection: paleo-models

As the dynamical system we seek to describe evolves over very large time scales (having periodicities of the order of 100 kyr) the appropriate lags for a predictive model might also be expected to be large. Clearly there is no hope of doing a full embedding search on many thousands of potential inputs for the model. Therefore we adopt a two stage filtering process for input selection.

- First we look at the Δ-correlations between the target time series (temperature) and the potential input time series. In this case the five potential input time series are: temperature, CO_2, CH_4, Insolation and Dust.

Because it is computationally feasible to do this over very large time scales we can use Δ-correlation to select the right orders of magnitudes for the different input lags. For example, a large amount of dust expelled into the atmosphere by volcanic activity might be expected to have a very short term lag (of the order of a year) with global temperature, whereas changes in Insolation might be expected to have a much longer lag before significantly affecting temperature, because of the large thermal capacity of the Earth.

Having used Δ-correlation to select appropriate lags:

- Depending on the number of inputs we next use an appropriate embedding selection routine (either full embedding or a statistical sample of random embeddings) together with the analysis routines discussed earlier: Durrant's method and our own frequency combination method (FCM).

In this way we refine the selection and reduce the number of inputs to a manageable quantity.

Δ-correlation for temperature

We begin the input selection procedure by examining the Δ-correlation plots for the different data sets. In Figure 8.4 we see the Δ-correlation plots for surface temperature against the other variables of interest for the 50, 100 and 500 year sampling rates, with lags going back $10,000$, $20,000$ and $20,000$ years respectively.

Whilst small scale periodic maxima and minima for the Δ-correlations can be seen for most variables, it is clear from these graphs that the largest Δ-correlation for temperature is with Insolation (radiative input to the surface). Whilst, of itself, not particularly surprising one interesting observation is that all three data sets consistently show that:

- The lag between changes in Insolation and temperature is around $7,300 - 7,500$ yr.

Examining Figure 8.3 we can see that 7.5 kyr ago Insolation has just begun to decrease. Thus the Δ-correlation results might suggest:

- Insofar as solar input is having any effect on current global temperature trends it is tending to *decrease not increase* temperature.

However, high correlations do not imply causality, and here we are just performing the standard analysis for predictive non-linear model building with noisy data.

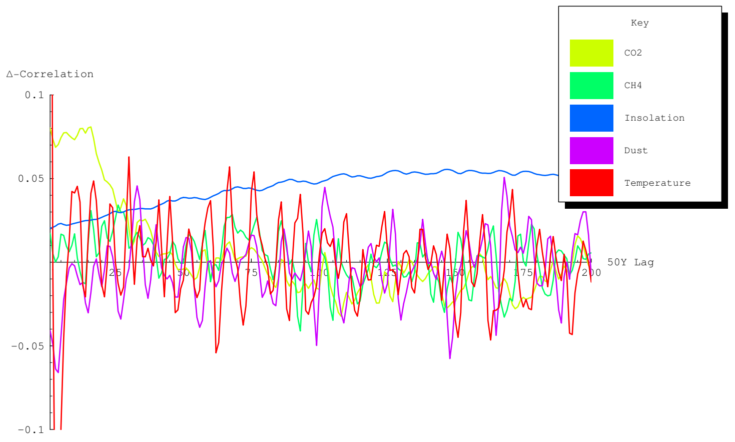
From the Δ-correlation results illustrated in Figure 8.4(a) we construct Table 8.1.

In the actual lag selection we will take the two most recent and then the three lags corresponding to the largest absolute Δ-correlations as shown in the table. This will give five inputs per variable, i.e. a total of 25 inputs for a one-step prediction of the single output temperature. These 25 inputs will then be examined by the Gamma test techniques developed earlier.

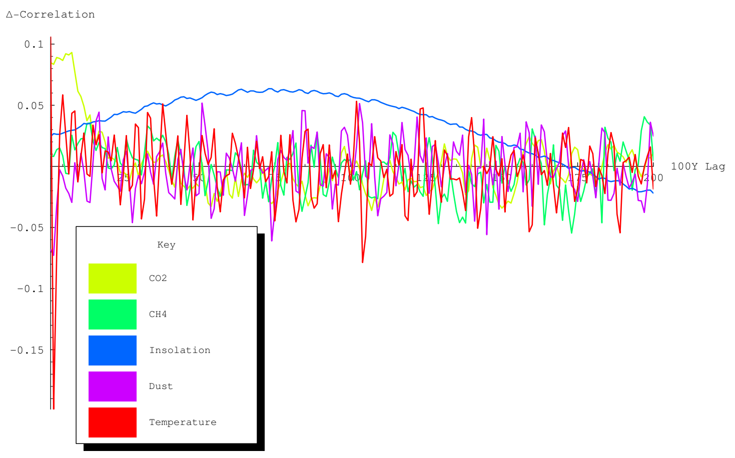
Points of interest arising from Table 8.1 are that, as one might expect, the Δ-correlations with CO_2 are all positive, and the Δ-correlations with Dust are all negative.

In fact the relationship between temperature and CO_2 appears to be quite subtle. It is thought that when insolation increases there is an approximate 800 ± 200, e.g. (Caillon et al., 2002), year period over which CO_2 is released by warming[3]. As levels of CO_2 increase, it is conjectured there is a positive feedback effect which then causes temperature to rise. We shall offer a few observations on our analysis of the Vostok data in section 8.1.6.

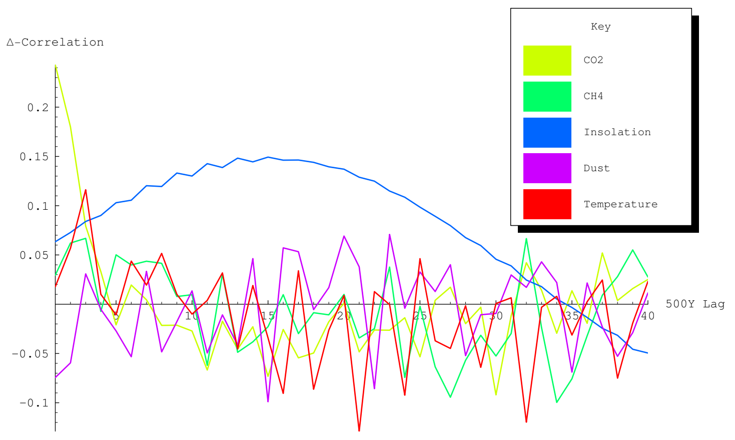
[3]Quite how this relates to our own observations of a 7.5 kyr lag between insolation and temperature is not clear.



(a) 50 year lags over 10, 000 years



(b) 100 year lags over 20, 000 years



(c) 500 year lags over 20, 000 years

Figure 8.4: Δ-correlation plots for temperature against itself and the other variables for different sampling intervals (lags).

	Lag (kyr)	Δ	Lag (kyr)	Δ	Lag (kyr)	Δ
CO_2	0.65	0.079	0.75	0.080	0.80	0.081
CH_4	1.30	0.034	4.65	-0.041	5.25	-0.035
Insolation	7.30	0.055	7.35	0.056	7.40	0.055
Dust	0.15	-0.064	0.20	-0.066	4.95	-0.058
temperature	0.15	-0.193	0.20	-0.170	0.25	-0.099

Table 8.1: The three best lags (in kyr) based on the strongest Δ-correlations for temperature (sampling rate is 50 yr).

	Lag (kyr)	Δ	Lag (kyr)	Δ	Lag (kyr)	Δ
CO_2	0.60	0.092	0.70	0.091	0.80	0.093
CH_4	12.90	-0.047	16.40	-0.048	17.30	-0.055
Insolation	6.40	0.063	7.30	0.064	7.40	0.063
Dust	0.10	-0.068	0.20	-0.073	7.4	-0.061
temperature	0.1	0.106	0.20	-0.199	10.4	-0.079

Table 8.2: The three best lags (in kyr) based on the strongest Δ-correlations for temperature (sampling rate is 100 yr).

	Lag (kyr)	Δ	Lag (kyr)	Δ	Lag (kyr)	Δ
CO_2	0.50	0.243	1.00	0.180	15.00	-0.092
CH_4	13.50	-0.094	17.00	-0.10	17.50	-0.076
Insolation	6.50	0.146	7.50	0.149	8.50	0.146
Dust	0.50	-0.074	7.50	-0.099	11.00	-0.086
temperature	1.50	0.116	10.50	-0.129	16.00	-0.120

Table 8.3: The three best lags (in kyr) based on the strongest Δ-correlations for temperature (sampling rate is 500 yr).

We perform a similar exercise for Tables 8.2 and 8.3. These tables show a reasonable degree of consensus.

We see that the lags for the three absolute highest Δ-correlations for Dust are small. Since the smallest sampling interval in the three data sets is 50 years, and dust from volcanic activity tends to makes its appearance quite rapidly, and will generally disappear over the span of at most a decade or two, it seems likely that Dust will not emerge as a useful input for any of the three predictive models.

Embedding searches (50kyr sample rate)

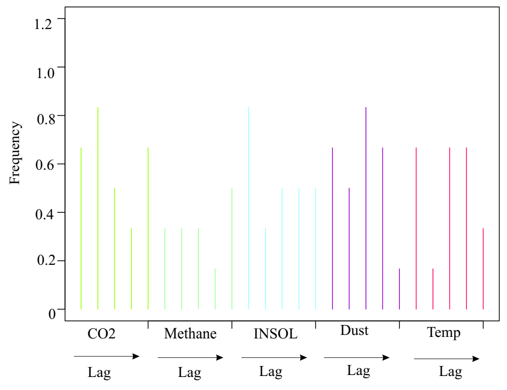
Number of candidate inputs	25
Number of masks	33,554,431
Sample size	3355
Durrant's method	CO_2 Lags 1, 2
	CH_4 Lags 7, 8, 10
	INSOL Lag 146
	Temp Lags 1, 2, 3, 5
FCM Selected inputs	CO_2 Lags 1, 2
	CH_4 Lag 105
	INSOL Lag 1
	Temp Lags 1, 2, 3, 5

Table 8.4: The inputs selected from a random embedding search using Durrant's method and the Frequency Combination Method to analyse the Γ-ordered input subsets. The candidate inputs were chosen using the Δ-correlation.

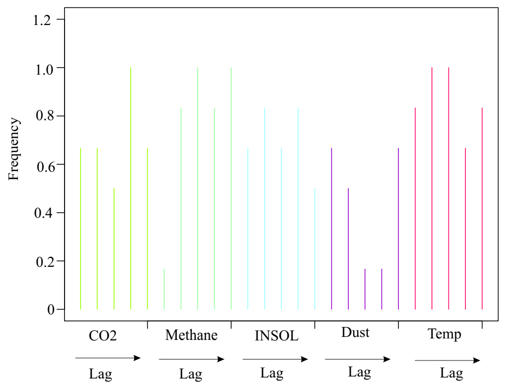
Having made a preliminary selection of 25 inputs using Δ-correlation we next proceed to use the tools described in Chapter 6 to refine the selection. The results are summarized in Table 8.4.

The eight inputs selected by FCM produce an absolute Gamma statistic of 0.0229 ($|V_{ratio}| = 0.0035$). As is clear from the graphs in Figure B.6, there is a strong correlation between CO_2 and CH_4 (in fact the correlation over the whole data set is 0.25). Just why this is so is not clear, but for our purposes it suggests that the CO_2 and CH_4 time series might have very similar predictive utility, so that CO_2 can act as a proxy for CH_4. By leaving out the methane input we obtain a lower Gamma statistic 0.00128 ($V_{ratio} = 0.000198$).

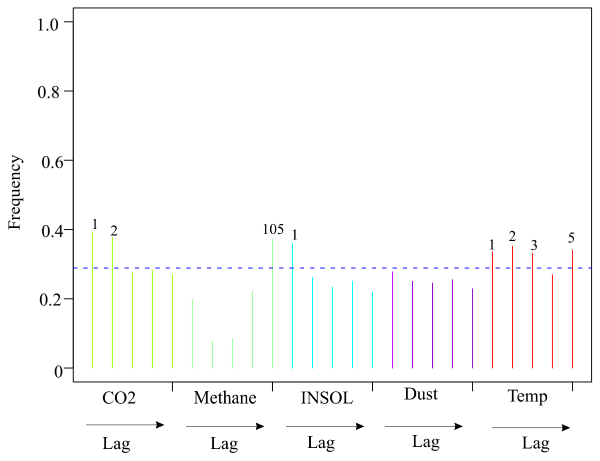
Insolation was also flagged as a relevant input by FCM. Leaving out the 50yr lagged insolation input causes the Gamma statistic to increase to 0.00927 ($V_{ratio} = 0.00143$).



(a) Durrant: Low Gamma region.



(b) Durrant: High Gamma region



(c) FCM analysis.

Figure 8.5: Input selection methods: Durrant (examine frequency of each input in the high/low Gamma regions), FCM (combining frequencies of inputs from the high/low Gamma regions of the Gamma histogram from a random sample of masks).

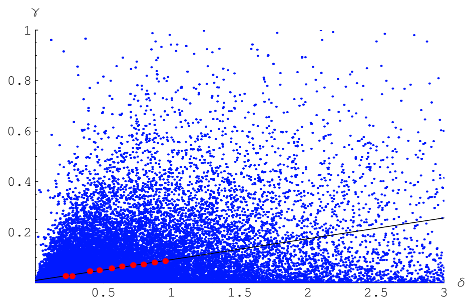
Although the model results are better if insolation is included as an input, the improvements are marginal. It seems as if more recent temperature measurements with relatively short term lags can act as proxies for long-lagged insolation.

Thus, in the interests of simplifying the model, the final selection of inputs emerges as the six given in Table 8.5.

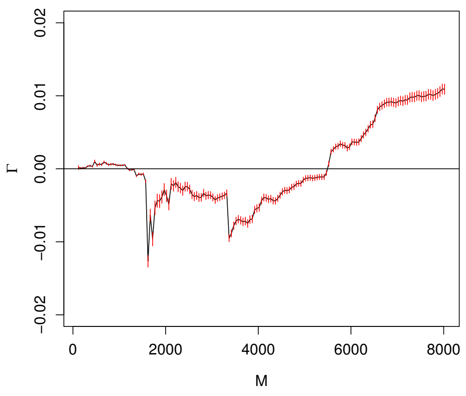
M-test and scatter plot analysis (50kyr sample rate)

The M-test graph for the 6-input/1-output data set for temperature show in Figure 8.6.

Examining the M-test graph of Figure 8.6 suggests that we should take $M = 7000$ for



(a) Scatter Plot: $\Gamma = 0.009218$.



(b) M-test

Figure 8.6: Scatterplot and M-test with confidence intervals, for the 6-input/1-output data set (50 yr sampling rate), $M = 7000$.

		Training set	
Inputs used		CO_2 Lags: 1 (50yrs) and 2 (100yrs)	
		Temp Lags: 1 (50yrs), 2 (100yrs), 3 (150yrs) and 5(250yrs)	
Start point (kyr)		-414.085	
End point (kyr)		-64.085	
M		7000	
Γ	V_{ratio}	0.009218	0.001423
Heuristic confidence intervals		(0.008421, 0.009609) at 95% level	
MSE$_{\delta\gamma}$ of Γ regression		8.0050×10^{-6}	
MSE reached in training		0.01112	

Table 8.5: Key statistics for 6-input/1-output training set (50 year sampling rate).

the training set.

The statistics are summarised in Table 8.5. As can be seen from the scatterplot of Figure 8.6 we have an excellent (δ, γ)-regression line, indeed from Table 8.5 we see that the MSE of the regression line is 8.0050×10^{-6}. The scatterplot also exhibits the 'empty wedge' in the left upper quadrant, characteristic of good data.

8.1.5 Paleoclimate models for temperature

We repeated the feature selection exercises for the 100 and 500 yr sampling rates. The results were broadly comparable, but frankly somewhat less impressive. So we have elected to only present the results of the 50 yr. sampling rate experiments.

Temperature model for 50 year sample rate

With the six inputs selected (as in Table 8.5) we built a one step predictive BFGS neural network ($6 \to 12 \to 6 \to 1$) using the first 7000 data points covering the period $[-414.085, -56.685]$ kyr. The results of testing on unseen data can be seen in Figure 8.7. As can be seen by the inset zoom these results are very good. The details are

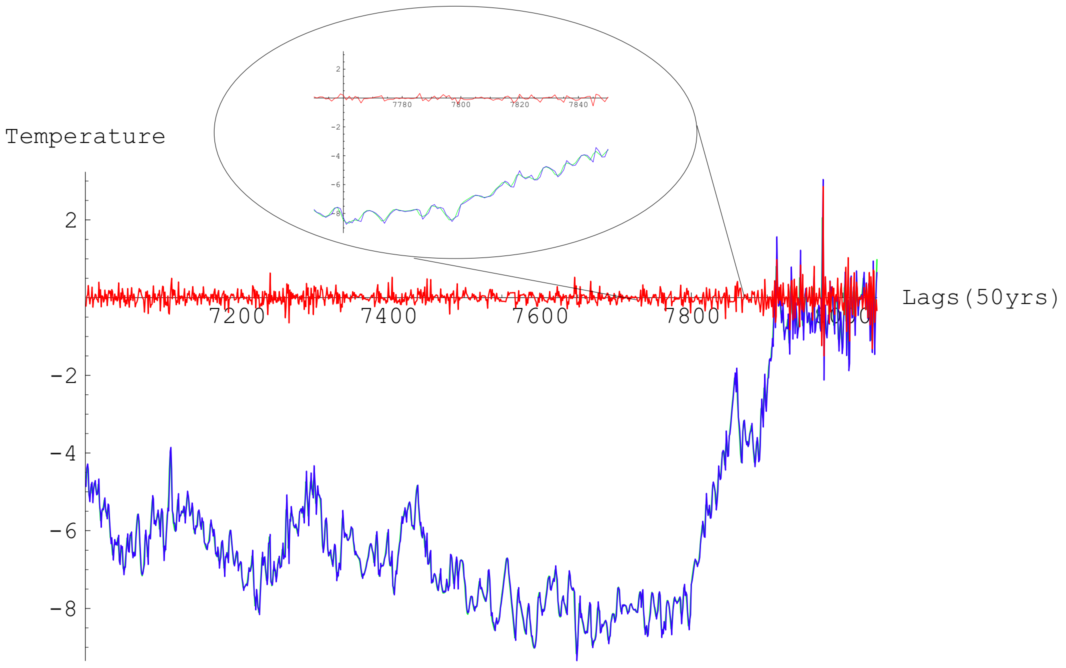


Figure 8.7: Model test results for one step temperature BFGS neural network ($6 \to 12 \to 6 \to 1$) on unseen test data from -56.685 to -4.485 kyr (sampling rate is 50 years). Green: actual temperature. Blue: predicted temperature. Red: error. The inset zoom shows the test results for a 5 kyr interval from -19.235 kyr.

summarized in Table 8.6.

One point of interest can be noted from Figure 8.7. Whilst over most of the test set the model performance is excellent, beginning approximately 11.685 kyr BP the, model performance begins to progressively degrade. Indeed, this observation carries across all the analysis we have completed on this data.

- We infer that approximately 11.685 kyr ago the environment started to become less predictable.

This is further illustrated in Table 8.6 where the first row relates to the whole test set, whilst the lower two rows correspond to the period up to -11.735 kyr, when the MSE was low, and the period after -11.685 kyr, when the MSE became much larger.

It is tempting to conjecture that this increase of unpredictability is caused by the rise of human civilisation, i.e. an anthropogenic signal (the time-scales are approximately right). However, it is hard to see how this hypothesis could be verified and there are other possible explanations, such as variations of solar irradiance for example.

Section of test set (in kyr)	MSE	RMSE	Mean absolute temperature change per step	Successfully predicted turning points (%)
-56.685 to -4.485	0.0506	0.2250	0.1830	76.3
-56.685 to -11.735	0.0234	0.1529	0.1614	76.3
-11.685 to -4.485	0.2269	0.4763	0.3180	71.5

Table 8.6: The performance of the one step predictive BFGS neural network ($6 \rightarrow 12 \rightarrow 6 \rightarrow 1$) on different sections of *unseen* temperature test data (sampling rate is 50 yr).

8.1.6 Regarding the Vostok data

It is immediately apparent, on examining the Vostok data in Figure B.6, that for nearly 500 kyr the variables temperature, CO_2, CH_4, and Dust were locked into a stable limit cycle. Indeed, an FFT on each of the equi-sampled data sets shows the periods are all around 102.4 kyr. The very fact that we are able to build such accurate 50 yr predictive models for most of this huge time interval is indicative of stable dynamics.

We have already observed that the principal period for Insolation is 40.95 kyr. This period does not match the observed Vostok periods. However, note that the range of variation of Insolation is 9.35 Wm^{-2}, i.e. about 3%, and compared with current estimates for the 'forcing effect' of CO_2 doubling (around 4 Wm^{-2}) this is significant. As illustrated in Figure 8.2, ignoring long-term trends[4], the 11 yr variation in the solar 'constant' is around 1.5 Wm^{-2}, which equates to an Insolation variation of 0.375 Wm^{-2} and is almost certainly inconsequential. Still, on balance, because of the period disparity, it seems likely that variations of radiative input to the Earth do not fully explain the Vostok cycle.

One question that pre-occupied the Global Warming debate is whether temperature leads CO_2 or vice-versa. In fact the current consensus is that temperature leads CO_2 by around 800 ± 200 yr, see for example (Caillon et al., 2002). Examining the correlations of changes in current temperature with changes in past CO_2 given in Table 8.1 we might be tempted to say CO_2 leads temperature by 600 years. On the other hand, in

[4]Probably not significant over 500 kyr.

Appendix C we have included for completeness the corresponding graphs and Tables for the correlations of changes in current CO_2 with changes in past temperature. From these we might equally be tempted to conclude that temperature leads CO_2 by 1200 years.

The truth is that all the Vostok time series of Figure B.6 are part of a single dynamic system that displays a limit cycle. In a Predator-Prey model, governed by the Lotka-Volterra equations, we might say *for convenience* that Prey leads Predator, but really one cannot divorce the two. The differential equations capture a higher level explanation, which is that, given unlimited grass, rabbits breed rapidly. Faced with an abundance of rabbits, the fox population rises until the rabbit supply starts to fail (a fox eats a lot of rabbits). Such cycles can arise from purely physical dynamics[5], but are often more typical of *living systems*.

Looked at in this way, the debate about whether temperature leads CO_2 or vice-versa becomes somewhat beside the point. The essential question is:

- "As temperature and CO_2 reach their maxima in the Vostok cycle *what mechanism causes CO_2 (or temperature etc.) to decline*?"

Just as a shortage of rabbits causes the fox population to crash, so there must be some powerful mechanism which comes into play that reduces atmospheric CO_2 and CH_4. It *must* be there because the Vostok data shows it[6]. What we lack at present is a convincing high level explanation - the mechanism (although a number of theories have been proposed). Given the known estimates for carbon reservoirs, the most plausible origins of such a mechanism are likely to be found in the deep oceans, but it remains unknown whether the mechanism is biological or physical.

8.2 Modelling temperature for the last millennium

From what we have seen for the paleoclimate model, it should be possible to produce reasonable relatively short term models for (say) up to 100 yr temperature prediction, provided one can estimate the increase in CO_2 ppm over the corresponding period. Obviously, this factor depends more on politics than on deterministic physical processes.

[5]Such as the variation in radiative input, which we have put to one side for the present.

[6]What's more, other data shows this CO_2 reduction occurs without excessive acidification of the oceans.

In 'Carbon Dioxide Sink 1970-2000 and Model Projections to 2100: a Statistical Mass Transfer Analysis' J. Ahlbeck writes:

> "In 1992, the Intergovernmental Panel on Climate Change (IPCC), presented a group of emission scenarios for different greenhouse gases. A 'mid-range' emission scenario was called IS92a. However, due to limited fossil fuel reserves, IS92a seems exaggerated when looking 100 years into the future. Numerous new emission scenarios, higher and lower than IS92a, have been created recently."

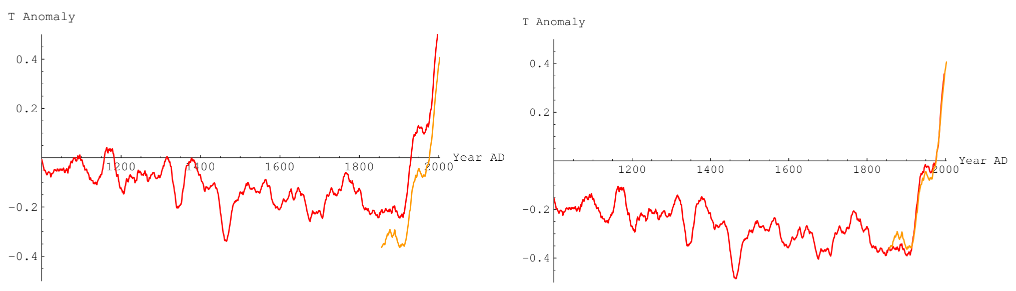
The paper gives a table for CO_2 ppm over the next century on the basis of different assumptions. We have constructed a similar (in places updated) Table 8.7.

Year	Linear extrap.	Expon. extrap.	IS92a
2000	367.89	367.89	367.89
2050	443	457	503
2100	517	567	705

Table 8.7: Predicted CO_2 levels on various hypotheses.

8.2.1 Amalgamating data sets for temperature

If we overlay the 25 yr smoothed data plot of the UEA Climate Research Unit (CRU) temperature data plotted in Figure B.3 with the 25 yr smoothed Mann *et al.* data of Figure B.1 we obtain Figure 8.8(a). From this it is clear that the two data sets share common features, but the baselines differ.



(a) Unadjusted temperature data overlay. (b) Adjusted temperature data overlay.

Figure 8.8: Combined temperature anomaly graphs 1000 AD to 2005 AD.

We decided to adjust the Mann *et al.* baseline by minimising the sum squared difference between the two data sets over their overlap. Using *Mathematica* `FindMinimum` we determined that the appropriate adjustment was to subtract 0.145911°C from the Mann *et al.* temperature anomalies. This produces the combined temperature anomaly graph of Figure 8.8(b).

We combined the two data sets by merging, sorting, averaging duplicates, and then, using linear interpolation, produced an equisampled one year interval data set.

8.2.2 Amalgamating data sets for CO_2

If we overlay the Law Ice Dome CO_2 data plot of Figure B.2 with the Mauna Lao instrumental CO_2 plot of Figure B.4(b), we obtain Figure 8.9.

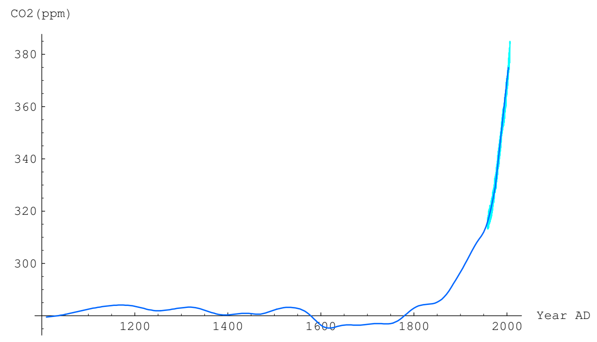


Figure 8.9: Combined CO_2 data plots 1010 AD to 2006 AD: Law Ice Dome (dark blue), Mauna Loa Instrumental CO_2 records (light blue).

We see that these two data sets, arrived at by totally different techniques, are extremely consistent.

We combined the two data sets by merging, taking direct CO_2 measurements where available, and then, using linear interpolation, produced an equisampled one year interval data set.

8.2.3 Input selection: millennium models

Δ-correlation for temperature

As before we begin the input selection procedure by examining the Δ-correlation plots in Figure 8.10 for temperature against CO_2 and itself. This leads to Table 8.8.

In fact the second and third highest Δ-correlations for CO_2 were lag 216 at 0.3747, and lag 215 at 0.3732. However, we decided that taking both of these was probably

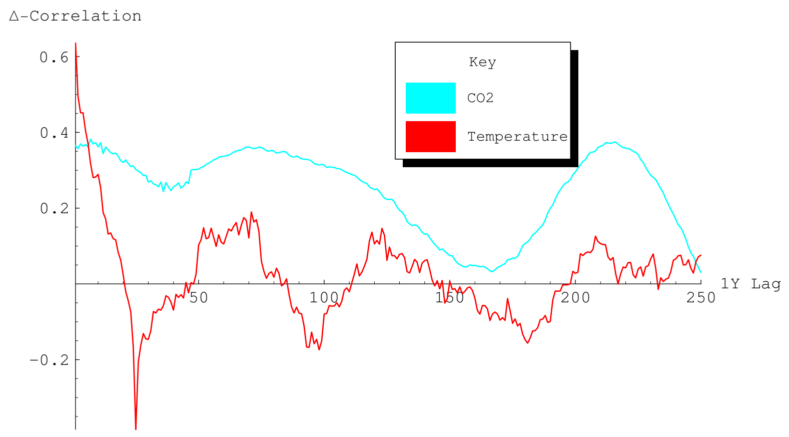


Figure 8.10: Δ-correlation plots for temperature against CO_2 and itself.

	Lag	Δ	Lag	Δ	Lag	Δ
CO_2	7	0.38224	216	0.37474	75	0.35990
Temperature	1	0.63639	2	0.49517	3	0.45177
Temp. Peaks	25	-0.38430	71	0.18977	98	-0.17397

Table 8.8: The three lags (in years) based on the highest absolute Δ-correlations, and the three best peaks, for temperature against CO_2 and itself, over the period 1011AD-2005AD.

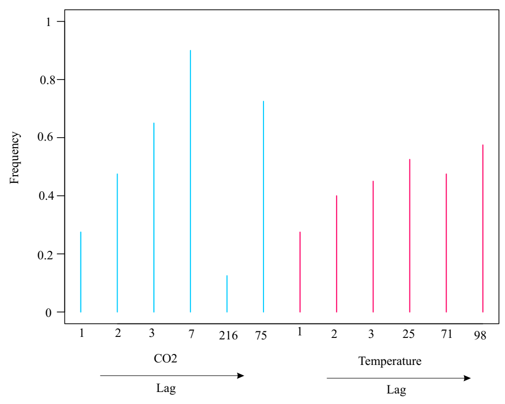
unnecessary and replaced lag 215 by lag 75, corresponding to the second peak in Figure 8.10. In addition we selected the three most recent CO_2 lags. For temperature the highest Δ-correlations were for the recent lags, so we included in addition the lags corresponding to the best three peaks. Thus six lags were selected for each of the two time series.

Embedding searches (1 yr sample rate)

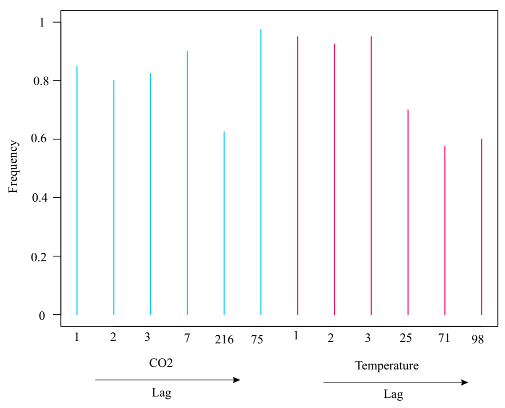
Having made a preliminary selection of 12 inputs using Δ-correlation we next proceed to use the tools described in Chapter 6 to refine the selection. In this case, since there are only 12 candidate inputs, we can do a Full Embedding search. The results are summarized in Table 8.9 and Figure 8.11.

Number of candidate inputs	12
Number of masks	4095
Durrant's method	$C0_2$ Lags 3, 7, 75
	Temp Lags 1, 2, 3
FCM Selected inputs	CO_2 Lags 3, 7, 75

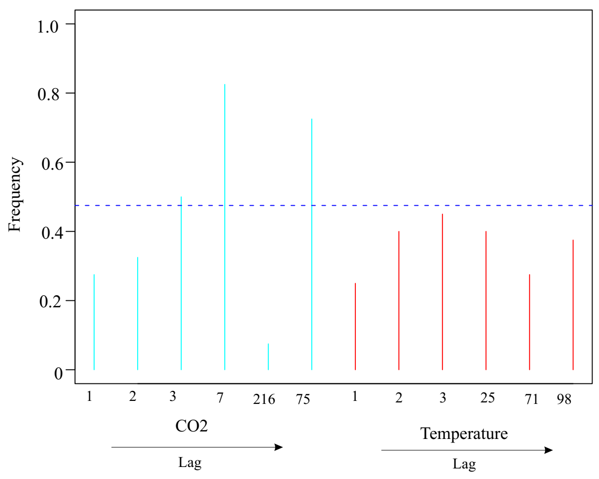
Table 8.9: The inputs selected from a full embedding search using Durrant's method to analyse the Γ-ordered input subsets and the Frequency Count Method. The candidate inputs were chosen using the Δ-correlation.



(a) Durrant: Low Gamma region.



(b) Durrant: High Gamma region



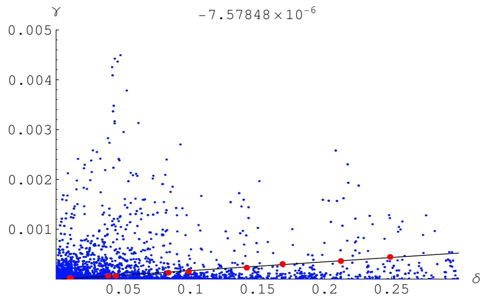
(c) FCM analysis.

Figure 8.11: Input selection methods: Durrant (examine frequency of each input in the high/low Gamma regions), FCM (combining frequencies of inputs from the high/low Gamma regions of the Gamma histogram from a random sample of masks).

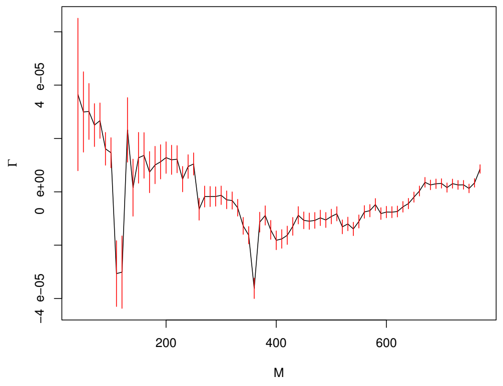
Both Durrant's method and FCM have selected CO_2 lags 3, 7, 75 as relevant inputs. Furthermore, Durrant's method (especially, the higher Gamma region) showed that the last three temperature values were also relevant. Running a Gamma test with both the lagged CO_2 and temperature inputs produced an absolute V_{ratio} of 0.0016484, which is smaller than that produced with just the CO_2 inputs alone ($|V_{ratio}| = 0.008305$). Hence, the final inputs selected are CO_2 at lags 3, 7 and 75 and temperature at lags 1, 2 and 3.

M-test and scatter plot analysis (1 yr sample rate)

As can be seen from the scatterplot of Figure 8.12 we have an excellent(δ, γ)-regression line. Indeed from Table 8.10 we see that the MSE of the regression line is 1.337236×10^{-10}, although the 'empty wedge' is not particularly noticeable in the left upper quadrant.



(a) Scatter Plot: $\Gamma = -7.5784 \times 10^{-6}$.



(b) M-test.

Figure 8.12: Scatterplot and M-test with confidence intervals, for the 6-input/1-output data set (1 yr sampling rate), $M = 600$.

Examining the M-test graph of Figure 8.12(b) suggests that we should take $M = 600$ for the training set. The statistics are summarised in Table 8.10. Initially we will build a model on $M = 600$ data points (i.e. up to 1826AD) so that we can test from $M = 610$ to $M = 779$. Subsequently, for the purposes of attempting forward projection of temperature, we shall extend the training set for the model to include *all* available data to 2005(6).

		Training set	
Inputs used		CO_2 Lags: 3, 7 and 75	
		Temp Lags: 1, 2 and 3	
Start point (yr)		1226 AD	
End point (yr)		1826 AD	
M		600	
Γ	V_{ratio}	-7.578492×10^{-6}	-0.001648475
Heuristic confidence intervals		$(-9.7307 \times 10^{-6}, -5.4262 \times 10^{-6})$ at 95% level	
MSE$_{\delta\gamma}$ of Γ regression		1.337236×10^{-10}	

Table 8.10: Key statistics for the temperature data (1 year sampling rate) over the (initial) training set.

Millennium temperature models

For the temperature anomaly of the last few decades, we are faced with attempting predictions in a situation where there is no comparable historical data. The current value of the variable CO_2 is climbing rapidly and is considerably outside the range of the historical data we might use for model training (see Figure B.6). Any data-driven modelling technique is going to experience difficulties when faced with this situation. In particular, as the predictions of the target variable temperature can also be expected to fall outside the range of the training data, we conclude that any straightforward application of neural network modelling is likely to be unsuitable. This problem becomes particularly acute when we are attempting to make genuine future predictions: attempting to predict 'into the cliff face of the singularity' as it were.

We elected to construct our models using Local Linear Regression (LLR). Here we construct a *kd*-tree on the input data and, given a query point, use a specified number of near neighbours to construct a local linear model. We then use this model to predict the output value for the query. One way to look at constructing the local linear model is that in essence we are constructing a pseudo-inverse matrix as described in Chapter 2.

Two points of caution. First, if the query point for the model is sufficiently far out of sample (in a specific fixed direction) then the same near neighbour set will be returned every time, so the predictions would reduce to repeatedly applying the same linear model (and would therefore be rather trivial). We shall discuss this in the next section. Second, in order to check our results, we used both $winGamma^{TM}$ and $Mathematica^{TM}$ and discovered in the process that for this data, particularly approaching present time, the specific algorithm used for linear regression in LLR is

Section of test set (in years)	MSE	RMSE	Mean change	Turning points (%)
1826-2005	0.0002	0.0151	0.0050	70

Table 8.11: The performance of the one-year ahead predictions produced by the Millennium local linear regression model (using 100 nearest neighbours) on *unseen* temperature data.

critical. Initially we were using the standard `PseudoInverse` in *Mathematica*, and the results were poor compared with *winGamma*, which uses Singular Value Decomposition (SVD). Only when we used SVD in *Mathematica* were we able to reproduce the *winGamma* results. It would appear that SVD is a superior algorithm, particularly for ill conditioned matrices.

Millennium model for temperature trained to 1826

We build the *kd*-tree using data points constructed from the lags identified in Table 8.10. The training data covers the period 1011 AD to 1825 AD. The results are shown in Figure 8.13 and summarised in Table 8.11.

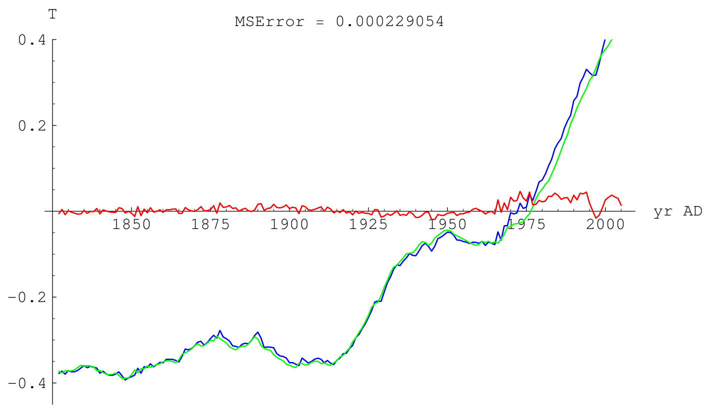


Figure 8.13: Millennium model: test results for one step 6-input/1-output temperature LLR prediction model on unseen test data from 1826 to 2005 (sampling rate is 1 yr). Green: actual temperature. Blue: predicted temperature. Red: error.

Scenario projections using the Millennium model trained to 2005

We now use the *entire* data set to train the model (i.e. to build the *kd*-tree). It is interesting to then allow this model to iterate into the future, using the idea sketched in Figure 8.14(a). If we do so we obtain the result in Figure 8.15.

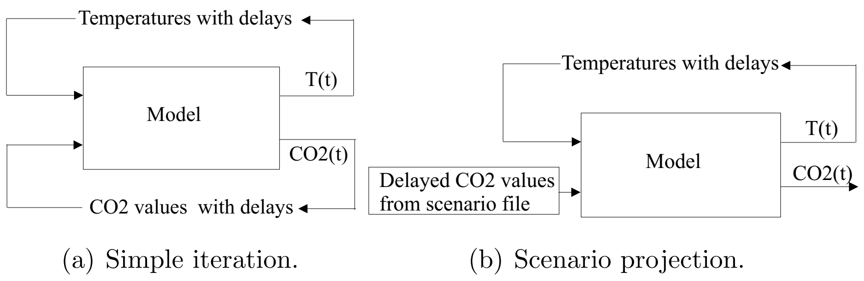


Figure 8.14: Two ways of future projection using the Millennium LLR model

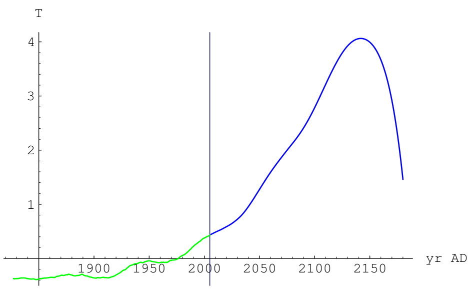


Figure 8.15: Millennium model: iterating into the future beyond 2005.

This roughly[7] amounts to asking: what does the model predict if no further human CO_2 emissions occurred? Figure 8.15 shows the result of this experiment. Whilst we should not take this at all seriously[8], it is interesting because the non-linearities show that the model does not fixate on the same locally linear projection every time.

Of somewhat more interest are the projections of future temperature based on the three scenarios postulated for CO_2 emissions in Table 8.7. We created files of equi-sampled CO_2 levels for each of the three scenarios and used these as inputs to the model as in Figure 8.14(b). The results are shown in Figure 8.16 and Table 8.12.

	Linear		Exponential		IS92a	
Year	CO_2	Temp	CO_2	Temp	CO_2	Temp
2006	380.06	0.447	380.06	0.447	380.06	0.447
2050	443	1.254	457	1.380	503	1.798
2100	517	1.705	567	2.231	705	3.723

Table 8.12: The local linear regression predictions for global surface temperature at 2006, 2050, and 2100AD under the three different hypotheses for future CO_2 emissions.

What is interesting about these projections is that the model, constructed directly

[7]Of course, trained to 2005 the model will certainly have captured some of the trends in human CO_2 emissions.

[8]The model uses CO_2 as inputs, but the lags are optimised for temperature - to do this properly needs two models - one for optimised for CO_2 and one optimised for temperature.

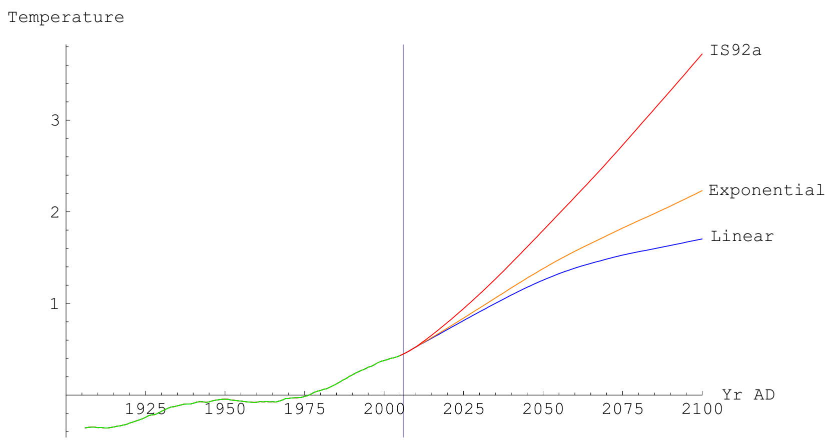


Figure 8.16: Millennium model: predicting temperature on the various CO_2 hypotheses of Table 8.7.

from the data and having inherited no in-built knowledge concerning the relationship between CO_2 and temperature, nevertheless predicts higher temperatures for the higher CO_2 scenarios.

8.2.4 Reflections on the millennium models

Hitherto we have been interested in merely developing techniques, and analysing existing data sets. By building the data-driven millennium models, and actually making forward predictions of global surface temperature we have taken a somewhat risky step into the unknown. Whilst we have every confidence in the methods described herein, they are applicable only in a clearly defined set of circumstances. The principal requirement for our modelling methods to produce reliable results is that the dynamic interplay of the variables we are measuring is fixed, i.e. does not change between the training and test data[9]. But already, in our examination of the Vostok paleodata, we had cause to observe that over the last 11.6 kyr the environment has become more unpredictable. Moreover, a glance at Figure B.6 shows clearly that something unusual is happening to recent atmospheric greenhouse gases compared with the last half-million years.

The dynamics are changing as a result of human consumption of fossil fuels etc. The system is no longer autonomous, so our data-driven models, based on data predominantly prior to the industrial revolution, are already suspect. In addition the more reliable temperature models require inputs from the CO_2 time series, which is hardly surprising, but means, as we project temperature into the future, we are dependent

[9]That the noise distribution is assumed fixed is less critical.

on political judgements regarding future fossil fuel consumption in making our own projections. Thus we regard these efforts as an interesting technical exercise, but make no claims as to the detailed accuracy of our predictions.

8.3 Chapter summary

In this chapter we have applied the full range of techniques developed earlier in the thesis to two areas of interest.

In the first case we looked at the Vostok paleodata for the last half million years, and were able to get very good 50 yr predictive models for temperature. For most of the last half million years the data indicates that the earth's climate was locked into a limit cycle, with a period of around 102.4 kyr, and exhibits every indication of stable dynamics. However, in the last 11.6 kyr there is a strong hint in our results that the dynamics was becoming more unpredictable.

In the second case we looked at the data for the last millennium, sampled at yearly intervals, and attempted to build short term models for temperature prediction. Here the problem of the rapidly changing atmospheric dynamics became particularly acute. We were pushing the envelope of our techniques[10] and one should not take the numerical conclusions too seriously. Still, one thing is clear from our models: whether temperature leads CO_2 or vice versa, the two are intimately related. Taken together the three data-driven models convey the clear implication that higher levels of CO_2 lead inexorably to higher temperatures.

[10]Even to the point where the precise choice of algorithm to compute the pseudo-inverse made a significant difference.

Chapter 9

Conclusions

We briefly survey the work covered, and discuss possible future extensions.

9.1 New data analysis tools

We have developed a number of new algorithms for confidence intervals and feature selection. In this section we briefly review the highlights.

9.1.1 Heuristic confidence intervals for the Gamma test

In Chapter 3 we gave an account of the Gamma test. In the first instance the Gamma test estimates the sample noise variance modulo any smooth function. Since the sample noise variance converges in probability to the noise distribution variance, the Gamma statistic also converges to the distribution noise variance. Still, it seemed that it would be helpful to have some estimate of confidence for the Gamma statistic estimate of the sample noise variance. A formal computation of confidence intervals would require a knowledge of the distribution of the Gamma statistic (knowledge that we do not have). However, using the observation that the Gamma distribution is approximately normal for large M, based on the M-test we have been able to construct an efficient heuristic algorithm to estimate Gamma confidence intervals. This joint work was published in (Jones and Kemp, 2006).

9.1.2 New embedding search routines

One of the key areas of application of the Gamma test is to provide an objective criterion for the selection of useful predictive inputs. From early in the development of Gamma test many embedding search techniques have been proposed: Hill-climbing, genetic algorithm, increasing embedding search and full embedding search, some of which are described in Chapter 3. In Chapter 6 we communicated two new *faster* embedding search routines for high input dimensions called approximate nearest neighbour full embedding search, and random embedding search.

It somewhat surprisingly emerged that a simple random embedding search is the fastest routine and robust even with a relatively small sample of embeddings. We therefore recommended that the random embedding search should be adopted as the preferred routine for high input dimensions.

9.1.3 Frequency Combination Method for input selection

Durrant provided a method for selecting input variables based on analysing the results of an embedding search routine. In Chapter 6 we describe what we believe to be a more robust input variable selection tool: The *Frequency Combination Method* (FCM) of section 6.2.3. This enhanced algorithm was applied to a wide range of popular time series models and also real-world datasets. In each case we compare and contrast the results of our algorithms with established techniques such as correlation.

This work was presented at *The 2005 International Symposium on Forecasting*, San Antonio, Texas, USA.

9.2 Noisy time series

In statistics the study of stochastic time series is well developed. Here the noise, or perturbation, of the signal at one moment feeds through to affect the value at the next moment. However, there is another factor in time series analysis which significantly affects our ability to construct predictive models. This is the noise associated with each sample value, typically as a result of measurement error. In Chapter 4 we treat the little studied case of *noisy time series*, where the noise *does not* feed through to affect the system. We illustrate that the optimal predictive model in such a situation depends on both the underlying smooth process *and* the noise. In the case of small noise variance we also determine in Proposition 4.1 an upper bound for the effective model noise variance (that estimated by the Gamma test) in terms of the noise variance on the signal and the complexity of the embedding surface (as determined by the underlying smooth process).

In one of our examples the process is quadratic, but the noise on the inputs is such that the optimal predictive model is virtually linear. Thus one cannot even reliably infer a *process law* from noisy data. Since almost all measurements have associated errors this fact has some unsettling implications.

This was joint work with A. J. Jones and D. Evans and is expected to appear in *Physica D*.

9.3 R-tools

The new data analysis tools described in this thesis were implemented into an R package with accompanying help files. Both R and the Gamma test package are freely available to download from the *www.r-project.org* web-site.

The key component of the algorithms is a fast implementation of the *kd*-tree. To achieve this we re-engineered the approximate nearest neighbour C++ library (ANNLIB) so that it could be used on all platforms and not just UNIX. Having an approximate nearest neighbour algorithm allowed us to ask the question of whether the trade-off between speed and precision of nearest neighbours was beneficial enough to be used in a full embedding search. Our experiments demonstrated that full embedding computing times could be halved without having a detrimental effect on Durrant's method or on FCM as techniques for selecting inputs.

Given the popularity of R among the statistical and time series community we entertain the hope that our techniques maybe become more widely adopted by the community.

9.4 Applications: crime data

Our first application of the techniques described and developed in this thesis was to the Cardiff crime data. Although, it was reasonable to assume that these methods could be of use in predicting daily criminal activity, it turned out that the final models were not predictively useful. This may have been a symptom of incomplete available data - more causal inputs need to be captured (e.g. unemployment rates, hourly alcohol sales). Alternatively, it might be the case that whatever data is provided, the stochastic component of criminal activity dominates any deterministic dynamics.

9.5 Applications: climate data

Based on the Vostok ice core data we constructed predictive models for the paleo-climate over the last half-million years. The results for the 50 yr prediction model were remarkably good, and interesting in that we were able to conclude that the climate started to become less predictable around 11.6 kyr ago. Although the precise reasons for this remain obscure.

Finally, we addressed the issue of forward prediction of temperature for the next century. Here we were faced with a difficult data-driven modelling problem in that the dynamic system under consideration was clearly undergoing major perturbations due to human emissions of CO_2. Particular care had to be taken in the choice of modelling methods and algorithms. The input data and target variable were clearly developing values which were outside the historical ranges. We were asking the models to predict in circumstances they had not seen before. Nevertheless, whilst the precise numerical projections should not be taken too literally, our final results showed a reasonable degree of intuitive acceptability. In particular they suggest that higher CO_2 emissions will lead to higher temperatures.

This work is the subject of a current journal submission.

9.6 Future work

The analysis of noisy non-linear time series and the construction of data-driven models is an area which has many important and varied applications. We have seen, for example, the importance of large scale automated data collection in the context of climate modelling.

The subject is now sufficiently developed that one can envisage a more comprehensive suite of time series modelling tools, in which many of the analysis and modelling steps we have developed and used in this work could be combined into a single data manipulation, analysis and modelling environment for multiple time series.

Another area of development is in the data-driven modular construction of complex dynamical systems. For example consider Figure 9.1.

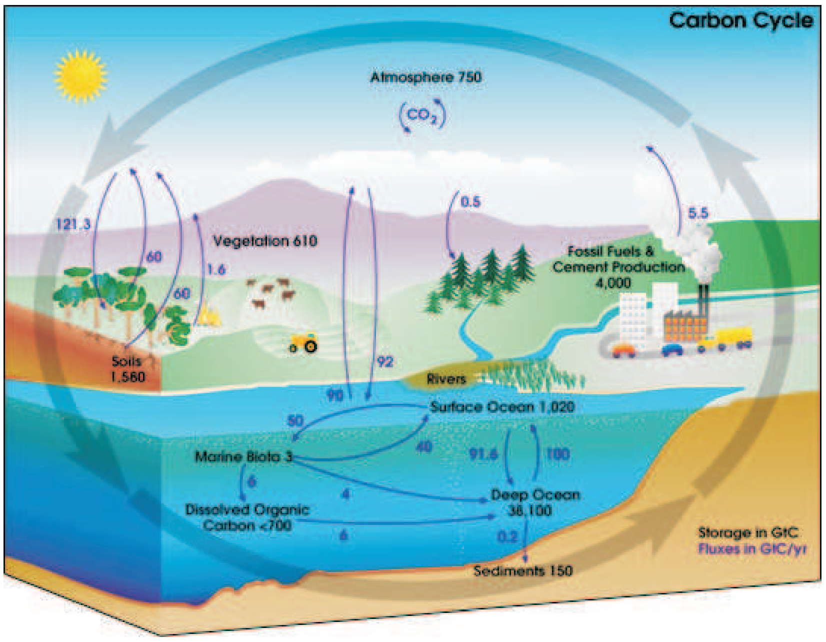


Figure 9.1: The Carbon Cycle. An example of a complex dynamic system. (From: Wikipedia.)

Here we have an example of a complex dynamical system which is of special interest. Measuring individual data flows can help us to understand the dynamical properties of such systems, but only if suitable modelling software is available. In particular we are interested in the long term *stability* of such dynamical systems. (Jones, 2004) outlines a proposal for such a system:

> "Building on the earlier ideas of *Simula* we could generate a system rather similar to *STELLA*, in that flow process charts could be used via a graphical user interface to specify inputs and outputs for specific nodes and link the nodes together into a model of a complex dynamic system. The difference would be that nodes would be non-linear input-output models automatically generated from the data. Each node would have an associated Gamma statistic, which would quantify the expected error relative to the actual data. Using this information the propagation of errors through the system could be studied and quantified. In this way we could identify critical nodes which limit overall accuracy, and assign measures of confidence to the outcomes of simulations."

Whilst many of the R-tools generated during this study could be used to prototype modules for such an environment, the vision is sufficiently ambitious as to warrant a fresh start in some platform independent language.

Such tools, together with automated monitoring programs, will prove invaluable in building accurate data-driven models of vital complex non-linear systems. A good predictive model, even if it is only short term, is a first pre-requisite to *control*. Following (Ott et al., 1990) we now know that even if the system is chaotic, provided we know enough about the dynamics, it can often be controlled (stabilised) by a tiny, easily computed, signal.

Thus the modelling is merely a stepping stone to effective control and management. Using such models we can examine the potential outcomes of different policy options, so that choosing policy becomes more a question of deciding goals and less a preoccupation with means.

- In order to form effective policies the need for accurate models is paramount.

Appendix A

Simulated Datasets

A summary of the different models used to generate the time series processes used in Chapter 6 is shown below in Table A.1.

Dataset Name	**Model Type**	**Nature of f**
A	Autoregressive	Linear
B	SETAR	Non-linear/Jump discontinuities
C	Transfer Functions	Linear
D	Vector Autoregressive	Linear
E	TARSO	Non-linear/Jump discontinuities

Table A.1: The simulated datasets.

A.1 Dataset A: Linear AR time series

The 20 time series models used to generate this dataset are shown in Table A.2. For all of the processes $\sigma^2 = 1$.

A.2 Dataset B: SETAR models

The self-exciting threshold autoregressive (SETAR) model parameters used to generate the nonlinear univariate time series are shown in Table A.3. For each process $M = 500$ and $\sigma^2 = 1$.

Series Name	Model	M
A.1	$y_t = 0.9y_{t-1} + e_t$	274
A.2	$y_t = -0.9y_{t-1} + e_t$	274
A.3	$y_t = 0.4y_{t-1} + e_t$	274
A.4	$y_t = -0.4y_{t-1} + e_t$	274
A.5	$y_t = 0.2y_{t-1} + e_t$	274
A.6	$y_t = -0.2y_{t-1} + e_t$	274
A.7	white noise series	274
A.8	$y_t = 0.4y_{t-1} + 0.5y_{t-2} + e_t$	274
A.9	$y_t = -0.4y_{t-1} + 0.5y_{t-2} + e_t$	274
A.10	$y_t = 0.4y_{t-1} + -0.5y_{t-2} + e_t$	274
A.11	$y_t = -0.4y_{t-1} + -0.5y_{t-2} + e_t$	274
A.12	$y_t = 0.7y_{t-1} + -0.9y_{t-2} + e_t$	274
A.13	$y_t = 0.9y_{t-1} + -0.7y_{t-2} + e_t$	274
A.14	$y_t = 0.3y_{t-1} + 0.2y_{t-2} + 0.3y_{t-3} + e_t$	274
A.15	$y_t = -0.3y_{t-1} + 0.2y_{t-2} + 0.3y_{t-3} + e_t$	274
A.16	$y_t = 0.3y_{t-1} + -0.2y_{t-2} + 0.3y_{t-3} + e_t$	274
A.17	$y_t = 0.3y_{t-1} + 0.2y_{t-2} + -0.3y_{t-3} + e_t$	274
A.18	$y_t = -0.3y_{t-1} + 0.2y_{t-2} + -0.3y_{t-3} + e_t$	274
A.19	$y_t = -0.3y_{t-1} + -0.2y_{t-2} + -0.3y_{t-3} + e_t$	274
A.20	$y_t = 0.9y_{t-1} + -0.6y_{t-2} + -0.3y_{t-3} + e_t$	274

Table A.2: The linear autoregressive models used.

A.3 Dataset C: Transfer functions

Recall from section 2.3.1 that a transfer function model is specified by the equation

$$\Phi(B)\Lambda(B)y_t = \Phi(B)\Omega(B)x_{t-b} + \Lambda(B)\Theta(B)e_t \tag{A.1}$$

where $\Omega(B)$, $\Lambda(B)$, $\Phi(B)$ and $\Theta(B)$ are all operators, b is the delay, x_t is the input time series, and e_t is an i.i.d. Gaussian random variable representing the noise.

For the 18 time series generated for this dataset, the input time series $\{x_t\}$ used was the differenced advertising series in the "sales-advertising data" provided in (Bowerman and O'Connell, 1993), which follows the following ARMA(1,1) model

$$(1 + 0.30027B)x_t = (1 + 0.78714B)e_t \tag{A.2}$$

The parameters of the transfer function models used to generate the open-looped time series are shown in Table A.4. In all cases $C = 1$, $\sigma^2 = 1$, $M = 97$, and $b = 3$

A.4 Dataset D: VAR time series

Table A.5 displays the $\boldsymbol{\phi}(B)$ parameters used to generate the vector autoregressive (VAR) time series, in all cases $M = 500$ and $\sigma^2 = 1$ for both e_t^1 and e_t^2.

A.5 Dataset E: TARSO time series

The input time series x_t was specified by the following ARMA(1,1) model

$$(1 - 0.5B)x_t = (1 - 0.5B)e_t \tag{A.3}$$

where $M = 500$ and $\mathrm{Var}(e_t) = 1$. The TARSO model parameters used to generate the output are specified in Table A.6 where $M = 500$ and $\sigma^2 = 1$.

Series Name	Model Parameters $\{\phi_0, (\phi_1, \ldots, \phi_k)\}$
B.1	$\{0.5, (0.8)\}$ if $y_{t-1} \geq 0$ $\{-2.3, (0.4)\}$ if $y_{t-1} < 0$
B.2	$\{1, (0.5)\}$ if $y_{t-1} \geq 0$ $\{4.5, (0.3)\}$ if $y_{t-1} < 0$
B.3	$\{1, (-0.22)\}$ if $y_{t-1} \geq 0$ $\{-1, (-0.721)\}$ if $y_{t-1} < 0$
B.4	$\{0.3, (0.6)\}$ if $y_{t-1} \geq 0$ $\{2.1, (-0.8)\}$ if $y_{t-1} < 0$
B.5	$\{0.5, (0.5)\}$ if $y_{t-1} \geq 0$ $\{2, (-0.3)\}$ if $y_{t-1} < 0$
B.6	$\{3.3, (0.8)\}$ if $y_{t-1} \geq 0$ $\{-1.2, (0.7)\}$ if $y_{t-1} < 0$
B.7	$\{0.3, (0.33)\}$ if $y_{t-1} \geq 0$ $\{0.1, (-0.27)\}$ if $y_{t-1} < 0$
B.8	$\{1.3, (0.8, -0.33)\}$ if $y_{t-1} \geq 0$ $\{2.1, (-0.27, 0.44)\}$ if $y_{t-1} < 0$
B.9	$\{2, (-0.66, -0.54)\}$ if $y_{t-1} \geq 0$ $\{-1, (0.6, 0.3)\}$ if $y_{t-1} < 0$
B.10	$\{0.3, (0.9, -0.44)\}$ if $y_{t-1} \geq 0$ $\{2.1, (-0.6, 0.9)\}$ if $y_{t-1} < 0$
B.11	$\{-0.2, (-0.9, -0.32)\}$ if $y_{t-1} \geq 0$ $\{-1.2, (0.22, 0.33\}$ if $y_{t-1} < 0$
B.12	$\{2, (-0.12, 0.32)\}$ if $y_{t-1} \geq 0$ $\{-1.5, (0.55, 0.33)\}$ if $y_{t-1} < 0$
B.13	$\{1.5, (-0.66, -0.32)\}$ if $y_{t-1} \geq 0$ $\{-1.5, (0.15, 0.13)\}$ if $y_{t-1} < 0$
B.14	$\{-3, (-0.7, -0.5)\}$ if $y_{t-1} \geq 0$ $\{4, (-1.3, 0.6)\}$ if $y_{t-1} < 0$
B.15	$\{2, (0.26, -0.32)\}$ if $y_{t-1} \geq 0$ $\{-2, (0.9, -1.2)\}$ if $y_{t-1} < 0$
B.16	$\{2, (0.26, -0.32, 0.5)\}$ if $y_{t-1} \geq 0$ $\{1.5, (-0.9, 0.5, -0.3)\}$ if $y_{t-1} < 0$
B.17	$\{4, (0.39, -0.32, -0.52)\}$ if $y_{t-1} \geq 0$ $\{-1.5, (0.36, -0.5, 0.6)\}$ if $y_{t-1} < 0$
B.18	$\{1.2, (0.44, 0.2, -0.2)\}$ if $y_{t-1} \geq 0$ $\{-1.5, (0.36, -0.5, 0.6)\}$ if $y_{t-1} < 0$
B.19	$\{1.2, (-0.99, 0.33, 0.33)\}$ if $y_{t-1} \geq 0$ $\{1.5, (0.36, -0.5, -0.2)\}$ if $y_{t-1} < 0$
B.20	$\{0.5, (0.22, 0.33, 0.33)\}$ if $y_{t-1} \geq 0$ $\{0.3, (0.44, -0.5, 0.2)\}$ if $y_{t-1} < 0$

Table A.3: The SETAR models used.

Series Name	Model Parameters $\{(\omega_1,\ldots,\omega_{k_1}),(\delta_1,\ldots,\delta_{k_2}),(\phi_1,\ldots,\phi_{k_3}),(\theta_1,\ldots,\theta_{k_4})\}$
C.1	$\{(0.88),(),(),()\}$
C.2	$\{(-0.88),(),(),()\}$
C.3	$\{(0.66),(),(),()\}$
C.4	$\{(-0.33),(),(),()\}$
C.5	$\{(0.88),(0.22),(),()\}$
C.6	$\{(0.22),(0.88),(),()\}$
C.7	$\{(-0.66),(0.44),(),()\}$
C.8	$\{(0.33),(0.55),(),()\}$
C.9	$\{(0.88),(0.22),(0.99),()\}$
C.10	$\{(0.22),(0.88),(0.66),()\}$
C.11	$\{(-0.66),(0.44),(0.74),()\}$
C.12	$\{(0.33),(0.55),(0.22),()\}$
C.13	$\{(1.5),(0.55,0.44),(0.44,-0.55),(0.22,0.33)\}$
C.14	$\{(2),(0.55,0.44),(0.44,-0.55),(0.22,0.33)\}$
C.15	$\{(1.33),(0.35,-0.14),(0.41,-0.15),(0.44,0.55)\}$
C.16	$\{(1.11),(0.35,-0.14),(0.41,-0.15),(0.22,0.11)\}$
C.17	$\{(0.44),(0.35,-0.14),(0.41,-0.15),(0.22,0.11)\}$
C.18	$\{(1),(0.35,-0.14),(0.41,-0.15),(0.22,0.11)\}$

Table A.4: The parameters used to generate the transfer function processes.

Series Name	$\boldsymbol{\phi}(B)$	M
D.1	$\begin{pmatrix} 0.7B & -0.5B \\ 0.5B & 0.3B \end{pmatrix}$	500
D.2	$\begin{pmatrix} -0.7B & 0.23B \\ -0.6B & 0.48B \end{pmatrix}$	500
D.3	$\begin{pmatrix} 0.87B & 0.21B \\ -0.33B & -0.48B \end{pmatrix}$	500
D.4	$\begin{pmatrix} 0.27B & -0.21B \\ -0.33B & 0.25B \end{pmatrix}$	500
D.5	$\begin{pmatrix} -0.77B & -0.55B \\ 0.66B & 0.12B \end{pmatrix}$	500
D.6	$\begin{pmatrix} 0.22B & -0.22B \\ 0.22B & -0.22B \end{pmatrix}$	500
D.7	$\begin{pmatrix} 0.22B + 0.33B^2 & 0.22B + 0.33B^2 \\ 0.22B + 0.33B^2 & 0.22B + 0.33B^2 \end{pmatrix}$	500
D.8	$\begin{pmatrix} 0.22B - 0.33B^2 & 0.22B - 0.33B^2 \\ -0.22B + 0.33B^2 & 0.22B - 0.33B^2 \end{pmatrix}$	500
D.9	$\begin{pmatrix} 0.47B - 0.38B^2 & -0.82B + 0.93B^2 \\ 0.62B + 0.33B^2 & -0.42B - 0.33B^2 \end{pmatrix}$	500
D.10	$\begin{pmatrix} 0.54B + 0.38B^2 & 0.42B + 0.43B^2 \\ -0.62B + 0.33B^2 & -0.42B - 0.33B^2 \end{pmatrix}$	500
D.11	$\begin{pmatrix} -0.54B + 0.28B^2 & 0.22B + 0.43B^2 \\ -0.62B + 0.44B^2 & 0.42B + 0.33B^2 \end{pmatrix}$	500
D.12	$\begin{pmatrix} -0.14B + 0.2B^2 & 0.12B + 0.13B^2 \\ 0.12B + 0.14B^2 & 0.12B + 0.33B^2 \end{pmatrix}$	500
D.13	$\begin{pmatrix} 0.24B - 0.66B^2 & -0.72B - 0.63B^2 \\ 0.42B + 0.24B^2 & -0.42B - 0.33B^2 \end{pmatrix}$	500
D.14	$\begin{pmatrix} 0.38B - 0.99B^2 & -0.72B + 0.63B^2 \\ 0.22B - 0.23B^2 & 0.12B + 0.63B^2 \end{pmatrix}$	500
D.15	$\begin{pmatrix} -0.38B + 0.49B^2 + 0.22B^3 & 0.42B - 0.35B^2 + 0.56B^3 \\ 0.32B - 0.23B^2 + 0.22B^3 & -0.52B + 0.37B^2 + 0.32B^3 \end{pmatrix}$	500
D.16	$\begin{pmatrix} 0.28B + 0.29B^2 + 0.22B^3 & 0.22B - 0.35B^2 + 0.16B^3 \\ 0.22B - 0.23B^2 + 0.22B^3 & -0.32B + 0.21B^2 + 0.38B^3 \end{pmatrix}$	500
D.17	$\begin{pmatrix} 0.48B + 0.2B^2 + 0.2B^3 & -0.3B + 0.35B^2 + 0.2B^3 \\ 0.62B - 0.18B^2 + 0.23B^3 & -0.52B - 0.34B^2 + 0.38B^3 \end{pmatrix}$	500
D.18	$\begin{pmatrix} -0.18B - 0.2B^2 + 0.3B^3 & -0.3B + 0.35B^2 + 0.2B^3 \\ -0.42B - 0.38B^2 + 0.53B^3 & -0.32B + 0.24B^2 + 0.28B^3 \end{pmatrix}$	500
D.19	$\begin{pmatrix} -0.8B + 0.32B^2 + 0.36B^3 & 0.2B - 0.5B^2 + 0.2B^3 \\ -0.42B + 0.38B^2 + 0.23B^3 & 0.32B - 0.24B^2 + 0.28B^3 \end{pmatrix}$	500
D.20	$\begin{pmatrix} -0.4B - 0.42B^2 + 0.56B^3 & 0.32B - 0.4B^2 + 0.42B^3 \\ -0.6B + 0.58B^2 + 0.42B^3 & -0.42B - 0.24B^2 + 0.58B^3 \end{pmatrix}$	500

Table A.5: The linear multivariate time series models.

Series Name	Model Parameters $\{\phi_0, (\phi_1, \ldots, \phi_p), (\theta_1, \ldots, \theta_q)\}$
E.1	$\{0.5, (0.9), (-1.5)\}$ if $y_{t-1} \geq 0$ $\{-2.3, (0.1), (1.7)\}$ if $y_{t-1} < 0$
E.2	$\{1, (0.5), (0.5)\}$ if $y_{t-1} \geq 0$ $\{4.5, (0.3), (0.7)\}$ if $y_{t-1} < 0$
E.3	$\{1, (0.2), (0.2)\}$ if $y_{t-1} \geq 0$ $\{-1, (0.2), (0.2)\}$ if $y_{t-1} < 0$
E.4	$\{1, (-0.9), (3.2)\}$ if $y_{t-1} \geq 0$ $\{2.1, (0.6), (4.2)\}$ if $y_{t-1} < 0$
E.5	$\{0.3, (0.6), (1.2)\}$ if $y_{t-1} \geq 0$ $\{2.1, (0.8), (2)\}$ if $y_{t-1} < 0$
E.6	$\{3.3, (0.4), (1)\}$ if $y_{t-1} \geq 0$ $\{2.1, (-0.7), (0.6)\}$ if $y_{t-1} < 0$
E.7	$\{0.3, (0.8), (0.7)\}$ if $y_{t-1} \geq 0$ $\{0.1, (-0.6), (0.5)\}$ if $y_{t-1} < 0$
E.8	$\{1.3, (0.8), (2.7, 0.8)\}$ if $y_{t-1} \geq 0$ $\{2.1, (0.6), (3.5, 0.9)\}$ if $y_{t-1} < 0$
E.9	$\{1, (0.8), (0.8, 2.7)\}$ if $y_{t-1} \geq 0$ $\{-1, (0.6), (0.5, 1.9)\}$ if $y_{t-1} < 0$
E.10	$\{1, (0.8), (0.8, -2.7)\}$ if $y_{t-1} \geq 0$ $\{-1, (0.6), (0.5, 1.9)\}$ if $y_{t-1} < 0$
E.11	$\{8, (-0.9), (2.1, -0.7)\}$ if $y_{t-1} \geq 0$ $\{4, (0.5), (-2.5, 0.9)\}$ if $y_{t-1} < 0$
E.12	$\{8, (0.9), (0.2, 0.3)\}$ if $y_{t-1} \geq 0$ $\{-4, (0.5), (0.3, 0.2)\}$ if $y_{t-1} < 0$
E.13	$\{2, (0.4), (0.2, 0.2)\}$ if $y_{t-1} \geq 0$ $\{-2, (0.4), (0.2, 0.2)\}$ if $y_{t-1} < 0$
E.14	$\{2, (0.4), (-0.2, -0.2)\}$ if $y_{t-1} \geq 0$ $\{-2, (0.4), (0.2, -0.2)\}$ if $y_{t-1} < 0$
E.15	$\{2, (0.7), (-0.2, 1.6, 1.4)\}$ if $y_{t-1} \geq 0$ $\{-2, (0.7), (0.2, -2.6, 1.7)\}$ if $y_{t-1} < 0$
E.16	$\{4, (0.4), (-0.2, 1.2, -1.6)\}$ if $y_{t-1} \geq 0$ $\{8, (0.5), (0.2, -1.2, 2.9)\}$ if $y_{t-1} < 0$
E.17	$\{1, (0.3), (1.7, 1.2, 1.6)\}$ if $y_{t-1} \geq 0$ $\{-1, (0.2), (1.8, 1.2, 1.9)\}$ if $y_{t-1} < 0$
E.18	$\{1, (0.7, -0.7), (1.2, 1.6)\}$ if $y_{t-1} \geq 0$ $\{-1, (0.7, -0.8), (1.2, 1.9)\}$ if $y_{t-1} < 0$
E.19	$\{-3, (0.5, -0.7), (2.2, 1.89)\}$ if $y_{t-1} \geq 0$ $\{4, (0.8, -0.5), (1.7, 1.66)\}$ if $y_{t-1} < 0$
E.20	$\{0.5, (0.4, 0.35), (0.91, -0.91)\}$ if $y_{t-1} \geq 0$ $\{0.3, (0.3, 0.45), (-0.91, 0.91)\}$ if $y_{t-1} < 0$

Table A.6: The TARSO models used.

Appendix B

Data sets for climate modelling

In seeking to build data-driven models of global climate the major part of the problem from our perspective lies in the available data sets. For example, time scales on ice-core data sets are often not clearly tied to real-time, are not equispaced in time (which is helpful for many non-linear analysis packages), and each data time series often covers a different period.

We list in

http://users.cs.cf.ac.uk/Antonia.J.Jones/ClimateModelling

the original data sets we have used, and also give the *Mathematica* file which consolidates the data (i.e. lines them up according to a consistent time scale), and uses interpolation functions of order 1, 2, or 3 to construct and output equi-time-spaced series covering the maximal period possible. In massaging the data in this way we recognise that the time measurements will have significant noise, but without such data consolidation we cannot apply the analysis tools appropriate for data-driven modelling (such as those discussed in the Gamma Archive).

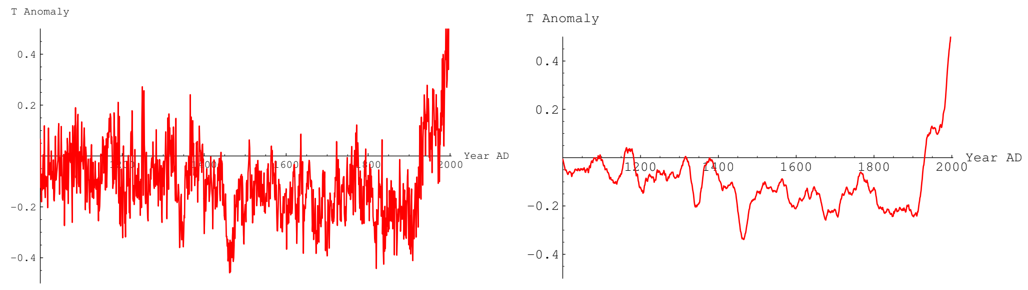
B.1 The last millennium

B.1.1 Historical times: Mann proxy temperature data

This data set consists of 998 data pairs giving annual mean surface temperature anomaly from 1000 AD to 1997 AD and is arguably too long to reproduce here. It is taken from:

http://www.co2science.org/scripts/CO2ScienceB2C/subject/other/data/mannetal_nhtemp.jsp

and represents the deviation of N.hemisphere surface temperature from the 1961 − 1990 average. We have archived a csv file at the site referenced above. Plots of this data are given in Figure B.1.



(a) Mann et al raw temperature data.

(b) Mann et al 25 yr smoothed temperature data.

Figure B.1: Proxy annual N.hemisphere surface temperature anomaly 1000 AD to 1997 AD, after (Mann et al., 1998, 1999).

B.1.2 Historical times: Law dome ice core data

Historical CO_2 record from the Law dome DE08, DE08-2, and DSS ice cores June 1998. This covers the period 1010 AD to 1975 AD and is 75 yr smoothed (Etheridge et al., 2002).

Source. D.M. Etheridge, L.P. Steele, R.L. Langenfelds, R.J. Francey: Division of Atmospheric Research, CSIRO, Aspendale, Victoria, Australia. J.-M. Barnola: Laboratoire of Glaciologie et Geophysique de l'Environnement, Saint Martin d'Heres-Cedex, France. V.I. Morgan: Antarctic CRC and Australian Antarctic Division, Hobart, Tasmania, Australia.

The data is plotted in Figure B.2.

B.1.3 Recent times: CRU Temperature data

Annual N. Hemisphere Temperature anomaly from 1856 to 2005. The data is shown in Figure B.3

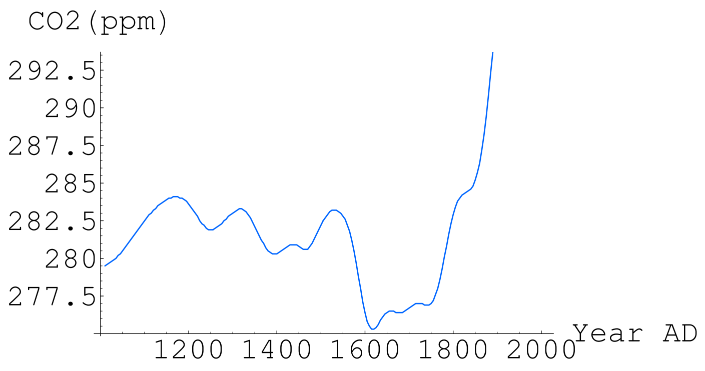
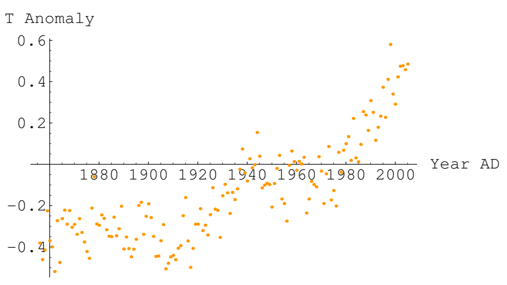
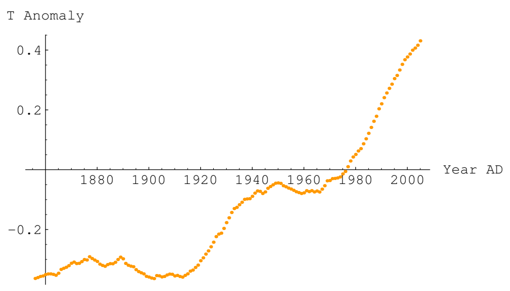


Figure B.2: Law Dome Ice core data: annual CO_2 from 1010 AD to 1975 AD.



(a) CRU Raw temperature data.



(b) CRU 25 yr smoothed temperature data.

Figure B.3: The UEA Climate Research Unit Temperature data 1856 to 2005 AD.

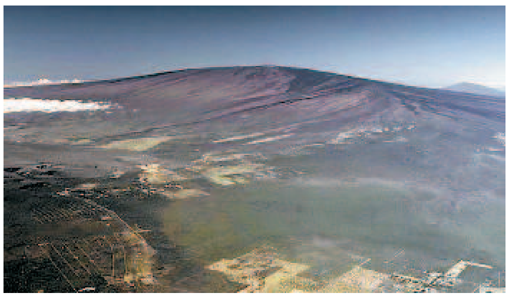
B.1.4 Recent times: Mauna Loa CO_2 data

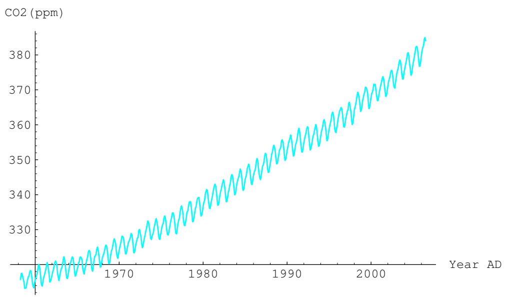
Rising gradually to more than 4 km above sea level, Mauna Loa[1], see Figure B.4(a), is the largest volcano on our planet. Its long submarine flanks descend to the sea floor an additional 5 km, and the sea floor in turn is depressed by Mauna Loa's great mass another 8 km. This makes the volcano's summit about 17 km (56, 000 ft) above its base! The enormous volcano covers half of the Island of Hawaìi and by itself amounts to about 85 percent of all the other Hawaiian Islands combined.

The 40th anniversary of the atmospheric CO_2 work being done at the Mauna Loa Observatory, Hawaii, by Dr. Charles D. Keeling, of the Scripps Institution of Oceanography (SIO) at the University of California, San Diego, was celebrated in 1997.

Widely recognized as the "Keeling curve", the Mauna Loa atmospheric CO_2 concentration measurements, taken since 1958 and now extending through 1997, constitute the

[1]Material in this subsection is taken from the Mauna Loa Observatory website.


(a) View of Mauna Loa (Courtesy: J. D. Griggs)



(b) Mauna Loa Instrumental CO_2 records. The annual variation can be clearly seen.

Figure B.4: Mauna Loa and the monthly CO_2 data 1958 to 2006 AD.

longest, continuous record of atmospheric CO_2 concentrations available in the world and are considered to be a reliable indicator of the regional trend in the concentration of atmospheric CO_2 in the middle layers of the troposphere. The methods and equipment used to obtain these measurements have remained essentially unchanged during the 40-year monitoring program. The CO_2 concentrations taken at Mauna Loa Observatory are obtained using a nondispersive, dual detector, infrared gas analyzer.

The Mauna Loa record shows a 15.2% increase in the mean annual concentration, from 315.83 parts per million by volume (ppmv) of dry air in 1959 to 363.82 ppmv in 1997. Each year, the atmospheric CO_2 concentration varies between a high value in winter (because of biospheric respiration) and a low value in summer (because of drawdown by photosynthesis); thus, a wave-like pattern is superimposed on the year-to-year increasing trend.

The data is taken monthly from March 1958 to June 2006 and is shown in Figure B.4(b).

B.2 Paleodata: The last 422 kilo years

B.2.1 Vostok ice core data

NAME OF DATA SET: Vostok Ice Core CO_2 Data, 1105-2856m.

CONTRIBUTORS: M. Wahlen, H. Fischer, J. Smith, D. Mastroianni, B. Deck, Scripps Institution of Oceanography.

The data was downloaded from

http : //www.ncdc.noaa.gov/paleo/icecore/antarctica/vostok/vostok_data.html

Located at 78°27′S, 106°52′E, central ice plateau, East Antarctica the Vostok Ice station is one of the coldest, driest places on Earth. In July 2004 the mean monthly temperature was −70.3°C (i.e. −94.5°F).

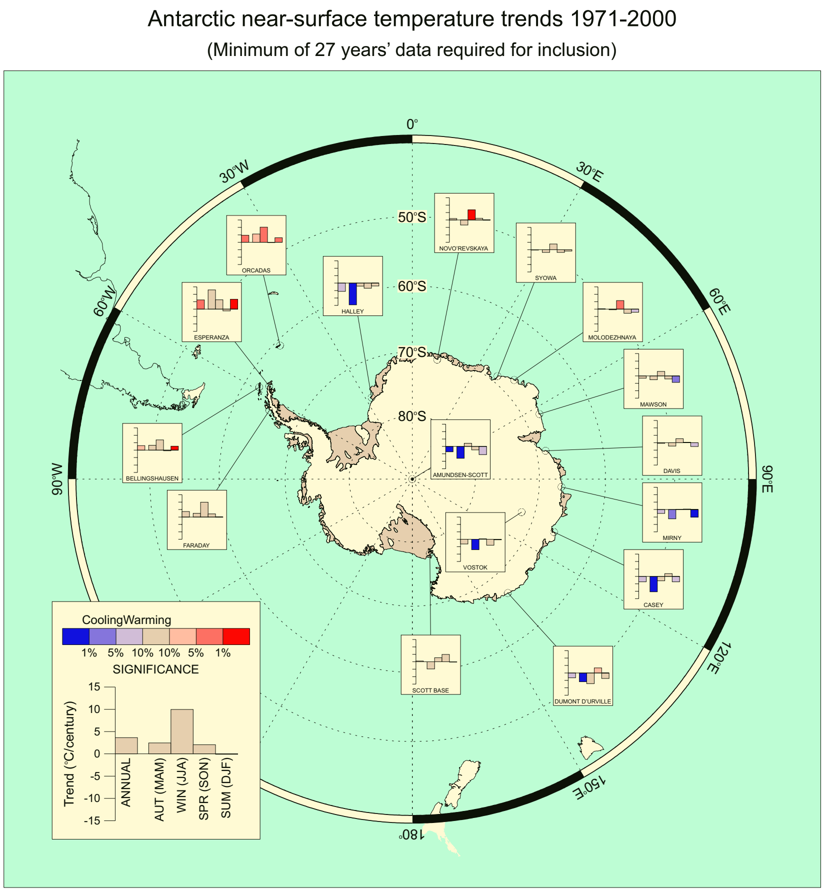


Figure B.5: Antarctic near surface temperature-trends 1971 − 2000. Courtesy: READER (**RE**ference **A**ntarctic **D**ata for **E**nvironmental **R**esearch).

We have used the data files co2nat.txt (CO_2 ppm), ch4.txt (CH_4 ppbv), duetnat.txt (Delta Temperature in degrees C from present), and dustnat.txt (dust conc. ppm), and archived the corresponding csv files at the site mentioned at the beginning of this Appendix. Graphs of this data are given in Figure B.6.

B.2.2 Insolation data

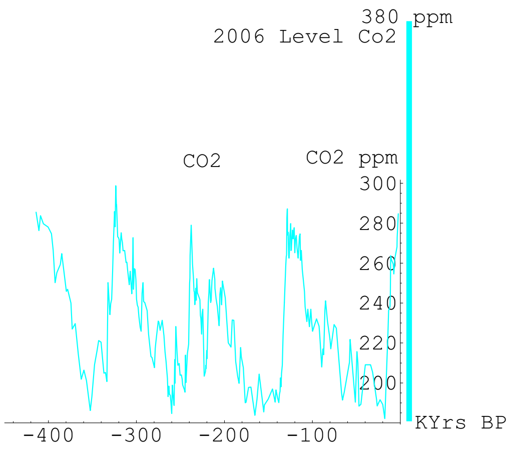
NAME OF DATA SET: Orbital Variations and Insolation Database LAST UPDATE: 5/92 (Addition of 1991 Solution)

CONTRIBUTOR: A. Berger, Catholic University of Louvain IGBP PAGES/WDCA Data Contribution Series #: 92-007

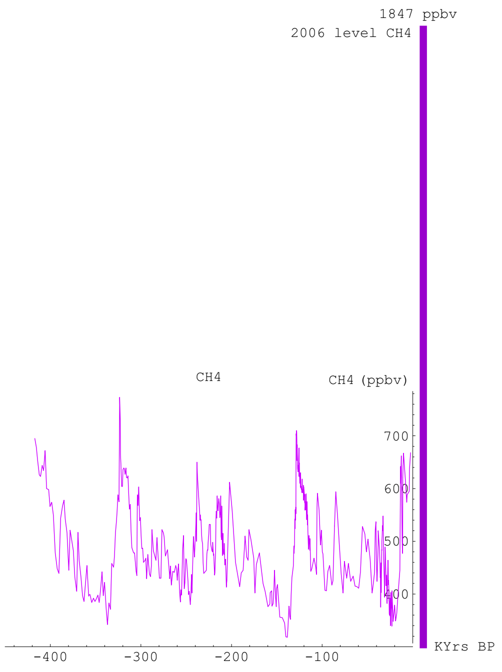
SUGGESTED DATA CITATION: Berger, A., 1992, Orbital Variations and Insolation Database. IGBP PAGES/World Data Center-A for Paleoclimatology Data Contribution Series # 92-007. NOAA/NGDC Paleoclimatology Program, Boulder CO, USA.

This data set contains data on changes in the earth's orbital parameters, and the resulting variations in insolation (Berger and Loutre, 1991). The directory contains both the 1978 calculations and the latest (1991) solution by A. Berger. The 1978 solution is preferred for time 0 (1950 A.D.). The two solutions are equivalent through 800 kyr BP, and the 1991 solution is preferred for times greater than 800 kyr BP.

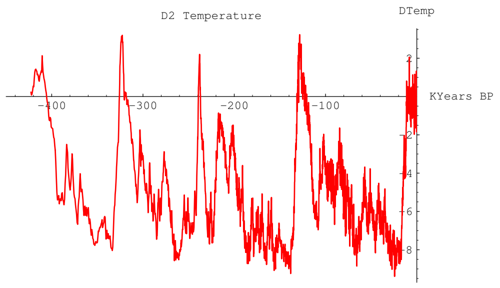
Here we use only the first 422 kyr and have averaged across months of the year and latitude to obtain a mean insolation for Earth at the given time. The corresponding graph is shown in Figure 8.3. This is theoretically computed from orbital data and cannot take account of variations in atmospheric dust or of solar luminosity.



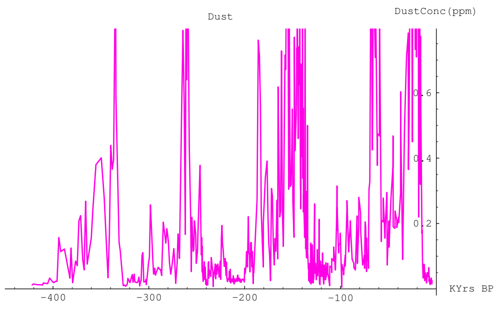
(a) Vostok Ice core: CO_2.



(b) Vostok Ice core: CH_4.



(c) Vostok Ice core: change in Temperature.



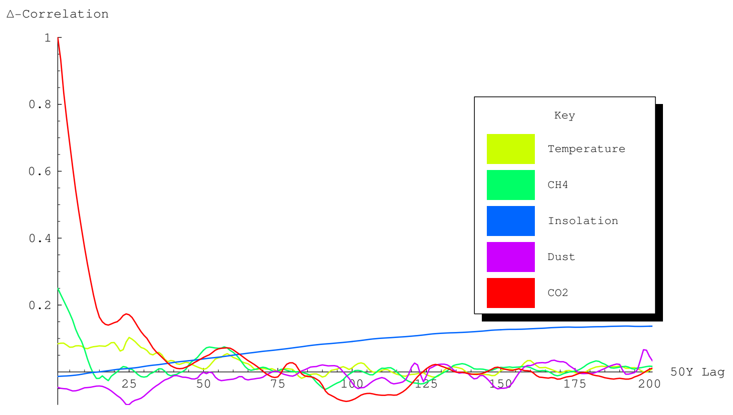
(d) Vostok Ice core: dust

Figure B.6: The Vostok 422K year ice core data.

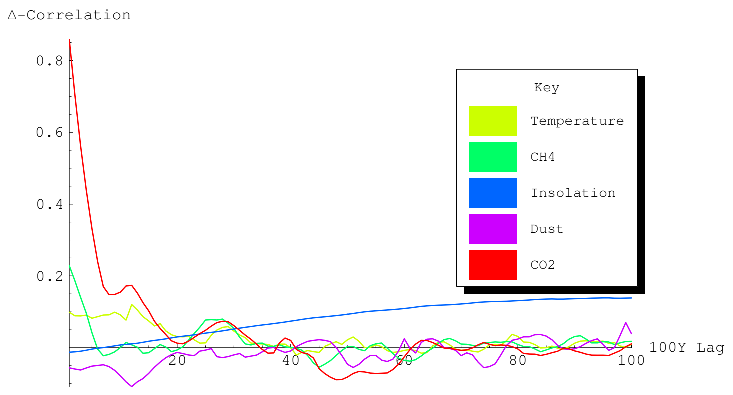
Appendix C

Analysis of CO_2 Δ-correlation lags

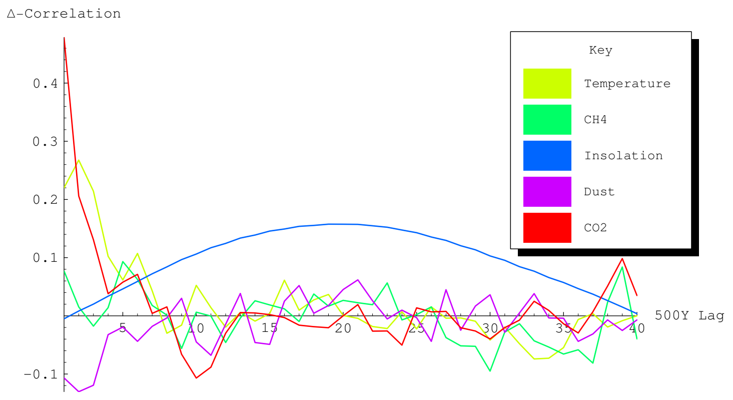
For completeness and to inform the discussion we include here the results of the CO_2 Δ-correlation analysis. From the Δ-correlation results illustrated in Figure C.1(a) we construct Tables C.1 to C.3



(a) 50 year lags over 10,000 years



(b) 100 year lags over 20,000 years



(c) 500 year lags over 20,000 years

Figure C.1: Δ-correlation plots for CO_2 against itself and the other variables for different sampling intervals (lags).

	Lag (kyr)	Δ	Lag (kyr)	Δ	Lag (kyr)	Δ
Temperature	1.20	0.103	1.25	0.098	1.30	0.091
CH_4	0.05	0.231	0.10	0.213	0.15	0.194
Insolation	9.55	0.137	9.60	0.137	9.50	0.137
Dust	1.20	-0.098	1.15	-0.098	1.25	-0.089
CO_2	0.05	0.933	0.10	0.835	0.15	0.756

Table C.1: The three best lags (in kyr) based on the strongest Δ-correlations for CO_2 (sampling rate is 50 yr).

	Lag (kyr)	Δ	Lag (kyr)	Δ	Lag (kyr)	Δ
Temperature	1.30	0.120	1.2	0.105	0.90	0.099
CH_4	0.10	0.229	0.20	0.186	0.30	0.140
Insolation	9.60	0.139	9.50	0.139	10.00	0.139
Dust	1.20	-0.108	1.10	-0.095	1.30	-0.095
CO_2	0.10	0.859	0.20	0.703	0.30	0.562

Table C.2: The three best lags (in kyr) based on the strongest Δ-correlations for CO_2 (sampling rate is 100 yr).

	Lag (kyr)	Δ	Lag (kyr)	Δ	Lag (kyr)	Δ
Temperature	1.00	0.268	0.50	0.220	1.50	0.214
CH_4	15.00	-0.095	2.50	0.093	19.50	0.084
Insolation	9.50	0.158	10.00	0.157	10.50	0.157
Dust	1.00	-0.130	1.50	-0.119	0.50	-0.107
CO_2	0.50	0.859	1.00	0.703	1.50	0.562

Table C.3: The three best lags (in kyr) based on the strongest Δ-correlations for CO_2 (sampling rate is 500 yr).

References

Arya, S., D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu (1998). An optimal algorithm for approximate nearest neighbour searching. *Journal of the ACM 45*, 891–923.

Balkin, S. D. and J. K. Ord (2000). Automatic neural network modeling for univariate time series. *International Journal of Forecasting 16*(4), 509–515.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative search. *Commun ACM 18*, 309–517.

Berger, A. and M. Loutre (1991). Insolation values for the climate of the last 10 million years. *Quaternary Sciences Review 10*, 297–317.

Bishop, C. M. (1996). *Neural Networks for Pattern Recognition.* Oxford University Press. ISBN 0-19-853864-2.

Bowerman, B. L. and R. T. O'Connell (1993). *Forecasting and Time series: An Applied Approach* (3rd ed.). Wadsworth, Inc. ISBN 0-534-93251-7.

Box, G. E. and G. M. Jenkins (1970). *Time Series Analysis forecasting and control.* Holden-Day.

Brent, R. (2002). *Algorithms for minimization without derivatives* (2nd ed.). Dover Publications, Mineola, New York. ISBN: 0-486-41998-3.

Caillon, N., J. P. Severinghaus, J. Jouzel, J.-M. Barnola, J. Kang, and V. Y. Lipenkov (2002). Timing of atmospheric co2 and antarctic temperature changes across termination iii. *SCIENCE 299*, 1728–1731.

Callen, J. L., C. C. Y. Kwan, P. C. Y. Yip, and Y. Yuan (1996). Neural network forecasting of quarterly accounting earnings. *International Journal of Forecasting* (12), 475–482.

C.Fröhlich and J. Lean (1998a). The sun's total irradiance: Cycles, trends and related climate change uncertainties since 1978. *Geophys.Res.Let. 25*, 4377–4380.

C.Fröhlich and J. Lean (1998b). Total solar irradiance variations. In F. et al. (Ed.), *New Eyes to see inside the Sun and Stars*, pp. 89–102. London: Chapman and Hall.

Chatfield, C. (1993). Neural networks: Forecasting breakthrough or passing fad? *International Journal of Forecasting* (9), 1–3.

Cherkassky, V. and F. Mulier (1998). *Learning From Data: Concepts, Theory, and Methods*. John Wiley and Sons. ISBN 0-471-15493-8.

Cherkauer, K. J. and J. W. Shavlik (1995). Rapidly estimating the quality of input representations for neural networks. In *IJCAI-95 Workshop on Data Engineering for Inductive Learning*.

Cohn, E. G. (1993). The prediction of police calls for service: The influence of weather and temporal variables on rape and domestic violence. *Journal of Environmental Psychology 136*, 71–83.

Cohn, E. G. (1996). The effect of weather and temporal variations on calls for police service. *American Journal of Police 15*, 23–43.

Cohn, E. G. and J. Rotton (1997). Assault as a function of time and temperature: A moderator-variable time series analysis. *Journal of Personality and Social Psychology 72*(6), 1322–1334.

Cohn, E. G. and J. Rotton (2000). Weather, seasonal trends and property crimes in minneapolis 1987-1988: A moderator-variable time series analysis of routine activities. *Journal of Environmental Psychology 20*(3), 257–272.

Cohn, E. G. and J. Rotton (2003). Even criminals take a holiday: Instrumental and expressive crimes on major and minor holidays. *Journal of Criminal Justice 31*(4), 351–360.

Corcoran, J. J., I. D. Wilson, and J. A. Ware (2003). Predicting the geo-temporal variations of crime and disorder. *International Journal of Forecasting* (19), 623–634.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signal and Systems 2*, 303–314.

Durrant, P. J. (2002). *WinGamma^{TM}: A Non-Linear Data Analysis and Modelling Tool with Applications to Flood Prediction.* Ph. D. thesis, Department of Computer science, Cardiff University, Wales, UK.

Durrant, P. J. and A. J. Jones (2003). Non-linear modelling of river levels using the gamma test. *Technical report: winGamma Archive*.

Etheridge, D. M., L. Steele, R. Langenfelds, R. Francey, J.-M. Barnola, and V. Morgan (2002). Natural and anthropogenic changes in atmospheric co2 over the last 1000 years from air in antarctic ice and firn. *Journal of Geophysical Research 101*, 4115–4128.

Evans, D. (2002). *Data-derived estimates of noise for unknown smooth models using near-neighbour asymptotics.* Ph. D. thesis, Department of Computer science, Cardiff University, Wales, UK.

Evans, D. and A. J. Jones (2002). A proof of the gamma test. *Proc. Roy. Soc. Lond. Series A 458(2027)*, 2759–2799.

Evans, D., A. J. Jones, and W. M. Schimidt (2002). Asymptotic moments of near neighbour distance distributions. *Proc. Roy. Soc. Lond. Series A 458(2028)*, 2839–2849.

Faraway, J. and C. Chatfield (1998). Time series forecasting with neural networks: a comparative study using the airline data. *Applied Statistics* (47, Part 2), 231–250.

Fletcher, R. (1987). *Practical Methods of Optimization* (2nd ed.). John Wiley and Sons.

Friedman, J. H. and J. L. Bentley (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software 3*, 209–226.

Gasser, T., L. Sroka, and C. Jennen-Steinmetz (1986). Residual variance and residual pattern in nonlinear regression. *Biometrika* (73), 625–633.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* MIT Press.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 356–366.

Ivaha, C. (2006). *A geo-temporal analysis of crime and disorder*. Ph. D. thesis, Faculty of Advanced Technology, University of Glamorgan, Wales, UK.

Jones, A. J. (2004). New tools in non-linear modelling and prediction. *Computational Management Science 1*(2), 109–149. ISSN 1619-697.

Jones, A. J., P. J. Durrant, and S. Margetts (1998). The gamma test and how to use it: A practioners guide.

Jones, A. J., D. Evans, S. Margetts, and P. J. Durrant (2002). *Heuristic and Optimisation for Knowledge Discovery*, Chapter The Gamma Test. Hershey, PA: Idea Group Publishing.

Jones, A. J. and S. E. Kemp (2006). Heuristic confidence intervals for the gamma test. *The 2006 International Conference on Artificial Intelligence (ICAI'06): June 26-29, 2006, Las Vegas, USA 2*, 54–60.

Judd, K. and A. Mees (1998). Embedding as a modelling problem. *Physica D* (120), 273–286.

Kantz, H. and T. Schreiber (2004). *Nonlinear Time Series Analysis* (2nd ed.). Cambridge University Press.

Kemp, S. E., I. D. Wilson, and J. A. Ware (2004). A tutorial on the gamma test. *International Journal of Simulation: Systems, Science and Technology 6*(1-2), 67–75.

Kennel, M. B., R. Brown, and H. D. I. Abarbanel (1992). Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical Review A 45*(6), 3403–3411.

Kerr, T. H. (1985). The proper computation of the matrix pseudoinverse and its impact in MVRO filtering. *IEEE Transactions on Aerospace and Electronic Systems 21*(5), 711–724.

Le Cun, Y. (1986). Learning processes in an asymmetric threshold network. In E. Bienenstock, F. F. Souli, and G. Weisbuch (Eds.), *Computer and Systems Sciences*. Springer.

Mann, M., Bradley, R.S., and M. Hughes (1998). Global-scale temperature patterns and climate forcing over the past six centuries. *Nature 392*, 779–787.

Mann, M., Bradley, R.S., and M. Hughes (1999). Northern hemisphere temperatures during the past millennium: Inferences, uncertainties, and limitations. *Geophysical Research Letters 26*, 759–762.

Margetts, S. (2001). *Adaptive Genotype to Phenotype Mappings for Evolutionary Algorithms.* Ph. D. thesis, Department of Computer science, Cardiff University, Wales, UK.

McCulloch, W. and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5*, 115–133.

Minsky, M. and S. Papert (1969). *Perceptrons.* MIT Press.

Olson, D. and C. Mossman (2003). Neural network forecasts of canadian stock returns using accounting ratios. *International Journal of Forecasting 19*(3), 453–465.

Ott, E., C. Grebogi, and J. Yorke (1990). Controlling chaos. *Physical Review Letters 64*(11), 1196–1199.

Park, Y. R., T. J. Murray, and C. Chen (1996). Predicting sun spots using a layered perceptron neural network. *IEEE Transactions Neural Networks* (7), 501–505.

Parker, D. (1985). Learning logic. *Technical report TR-47, MIT Center for Research in Computational Economics and Management Science, Cambridge, MA.*

Penrose, R. (1955). A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society 51*, 406–413.

Penrose, R. (1956). On best approximate solution of linear matrix equations. *Proceedings of the Cambridge Philosophical Society 52*, 17–19.

Petit, J., J. Jouzel, D. Raynaud, N. Barkov, J. Barnola, I. Basile, M. Bender, J. Chappellaz, J. Davis, G. Delaygue, M. Delmotte, V. Kotlyakov, M. Legrand, V. Lipenkov, C. Lorius, L. Ppin, C. Ritz, E. Saltzman, and M. Stievenard (1999). Climate and atmospheric history of the past 420,000 years from the vostok ice core antarctica. *Nature 399*, 429–436.

Pfleger, K., G. John, and R. Kohavi (1994). Irrelevant features and the subset selection problem. In W. W. Cohne and H. Hirsh (Eds.), *Machine Learning: Proceedings of the Eleventh International Conference*, pp. 121–129.

Pi, H. and C. Peterson (1994). Finding the embedding dimension and variable dependencies in time series. *Neural Computation 5*, 509–520.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992). *Numerical Recipes in C* (Second ed.). Cambridge University Press. ISBN 0-521-43108-5.

Rice, J. A. (1984). Bandwidth choice of nonparametric regression. *Annals of Statistics* (12), 1215–1230.

Rosenblatt, F. (1962). *Priciples of neurodynamics: perceptrons and the theory of brain mechanics*. Spartan.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986, 9 October). Learning representations by back-propagating errors. *Nature 323*, 533–536.

Scherf, M. and W. Brauer (1997). Feature selection by means of a feature weighting approach. Technical Report FKI-221-97, Forschungsberichte Künstliche Intelligenz, Institut für Informatik, Technische Universität München.

Shewchuk, J. R. (1994, August). An introduction to the conjugate gradient method without the agonizing pain. ed. $1\frac{1}{4}$. Technical report, School of Computer Science Carnegie Mellon University. http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.pdf.

Stefánsson, A., N. Končar, and A. J. Jones (1997). A note on the gamma test. *Neural Computing Applications 5*, 131–133.

Takens, F. (1981). Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence*, Volume 898 of *Lecture notes in Mathematics*, pp. 366–381. Springer-Verlag.

Tiao, G. C. and G. E. P. Box (1981). Modeling multiple time series with applications. *Journal of the American Statistical Association 76*(376), 802–816.

Tong, H. (1990). *Non-Linear Time Series*. Oxford: Clarendon.

Tong, H. and K. Lim (1980). Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statisical Society Series B 42*, 245–292.

Tsui, A. (1999). *Smooth Data Modelling and Stimulus-Response via Stabilisation of Neural Chaos*. Ph. D. thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London.

Tsui, A. P. M., A. J. Jones, and A. G. de Oliveira (2002). The construction of smooth models using irregular embeddings determined by a gamma test analysis. *Neural Computing & Applications 10(4)*, 318–329.

Werbos, P. (1974). *Beyond regression: new tools for prediction and analysis in the behavioural sciences.* Ph. D. thesis, Harvard University, Boston, MA.

Wilson, I. D., A. J. Jones, D. H. Jenkins, and J. A. Ware (2004). Predicting housing value: Attribute selection and dependence modelling utilising the gamma test. *Advances in Econometrics* (19), 243–275.

Wilson, I. D., S. D. Paris, J. A. Ware, and D. H. Jenkins (2002). Residential property price time series forecasting with neural networks. *Journal of Knowledge Based Systems 15/5-6*, 73–79.

Index

albedo, 151
ANNFES, 126, **126–127**
ANNLIB, 95
ARMA model, 28, 31
autocorrelations, 28

BFGS algorithm, 47–49
Box-Jenkins approach, 28, 49

Cardiff, 137
chaos, 76
    Lyapunov exponents, 76
closed system, 30, 32
crime prediction, 150
criminality, 136
cross-correlation, 32, 33

delay vector, 25, 115
delta correlation, 124, 156, 167, 196
deterministic time series, 26
difference-based estimators, 55
    The δ-test, 57
    GSJ estimator, 56
    Rice estimator, 55

effective noise, 84
embedding dimension, 21, 25, 76, 115
Euclidean distance, 59

false nearest neighbour algorithm, 76
frequency combination method, 126, **128–131**

The Gamma test, 22, 57, **59–80**, 82
    class of models, 60
    confidence intervals, 67
    full embedding search, 72, 118
        frequency count analysis, 72
        Gamma histogram, 72
    Gamma histogram, 120, 131
    increasing embedding search, 78, 117
    M-test, 66, 117
    noise distribution, 61
    sampling distribution, 61
    setting p, **61–64**
global warming, 164

Hénon map, 90

input variable selection, 24, **123–135**
input vector, 21
insolation, 154, 155

kd-tree, 21, 59, **95–99**, 109
    bounds overlap ball algorithm, 99
    buckets, 96
    construction, 96
    partitioning, 96
    searching, 97

leading indicator, 30
linearity, 22
Local linear regression, **50–53**

mask, 72
mean squared error, 84
measurement error, 24, 27, 82
memory management, 110
multivariate, 21, 30, 32

nearest neighbours, 94
neural networks, **37–53**
    activation function, 40
    architecture, 39
    backpropagation, 38, 43, 44
    conjugate gradient descent, 42, 45
    hidden layer, 39
    input layer, 39
    multi-layer perceptrons, 38
    perceptrons, 37
    quasi-Newton methods, 42
    steepest gradient descent, 42
    weights, 40
noisy time series, 23, 27, 82
non-autonomous time series, 55
non-parametric modelling, 21

Occam's razor, 89
one-step prediction, 26
open system, 31
open-source, 94

parametric modelling, **34–37**
partial autocorrelation, 28
Polak-Ribiere form, 46

R, 24, 94

CRAN, 108
environment, 107
packages, 108
RES, 126, **127–128**
residual error, 28

singular value decomposition, 52
smooth function, 22, 82
solar constant, 154
solar insolation, 151
solar irradiance, 154
stationary, 27
stochastic, 20
stochastic time series, 23, 26
SVD, *see* singular value decomposition

Takens theorem, 21, 25
temperature anomaly, 153
threshold autoregressive models, 35
time series, 20
transfer function, 31

univariate, 21

VARMA model, 32
Vostok limit cycle, 164

white noise, 28
winGamma, 94