

Advanced Information Processing

Series Editor

Lakhmi C. Jain

Advisory Board Members

Endre Boros

Clarence W. de Silva

Stephen Grossberg

Robert J. Howlett

Michael N. Huhns

Paul B. Kantor

Charles L. Karr

Nadia Magenat-Thalmann

Dinesh P. Mital

Toyoaki Nishida

Klaus Obermayer

Manfred Schmitt

Springer-Verlag Berlin Heidelberg GmbH

Chris Harris Xia Hong Qiang Gan

Adaptive Modelling, Estimation and Fusion from Data

A Neurofuzzy Approach

With 159 Figures



Springer

Chris Harris, Xia Hong
University of Southampton
Department of Electronics
and Computer Science
Southampton SO17 1BJ
United Kingdom

Qiang Gan
University of Essex
Department of
Computer Science
Colchester CO4 3SQ
United Kingdom

Library of Congress Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Harris, Chris: Adaptive modelling, estimation and fusion from data: a neurofuzzy approach/
Chris Harris; Xia Hong; Qiang Gang. - Berlin; Heidelberg; New York; Barcelona; Hong Kong;
London; Milan; Paris; Tokyo: Springer, 2002 (Advanced information processing)
ISBN 978-3-642-62119-2 ISBN 978-3-642-18242-6 (eBook)
DOI 10.1007/978-3-642-18242-6

ACM Subject Classification (1998): I.2, I.5, G.1, G.3, H.3.1

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 2002
Originally published by Springer-Verlag Berlin Heidelberg New York in 2002
Softcover reprint of the hardcover 1st edition 2002

The use of designations, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover Design: KünkelLopka, Heidelberg
Typesetting: Computer to film by author's data
Printed on acid-free paper SPIN 10853536 45/3142PS 5 4 3 2 1 0

This book is dedicated to
Connor Jordan, Tristan Jake,
Paige Jemma, James Zifan
and Lisa Ying.

We hope they will grow to appreciate
and be inspired by the dedication
behind this book.

Foreword

There is tremendous interest world wide in the implementation of intelligent machines which can mimic the behaviour of humans in a limited way. This process of creating intelligent machines is called artificial intelligence. In its relatively short history, the field of artificial intelligence has suffered much from swings between euphoria and despair. Over the last decade or so, the gloom of 1970s has been replaced by a growing confidence that the application of an ever-increasing array of artificial intelligence techniques can help provide solutions to a large number of real world problems in areas such as science, engineering, business and so on.

The main paradigms for generating intelligence in machines include artificial neural networks, evolutionary computing techniques, chaos theory, fuzzy systems, intelligent agents as well as the fusion of these paradigms.

We live in a century of intelligent machines. This novel book is a step towards realising intelligent machines. This book brings together for the first time the complete theory of data or empirical based neurofuzzy modelling exploiting the mathematical advantages of generalised linear neural networks and linguistic attributes of fuzzy logic in a single cohesive mathematical framework. A fundamental limitation of the practical application of these neurofuzzy systems has been the curse of dimensionality. This research book presents a number of innovative schemes using practical examples for resolving this limitation. A number of areas are covered by the authors including control, condition monitoring and real time nonlinear processes.

This research book will be of great benefit to the undergraduate, post-graduate students in many disciplines including electrical, electronic, mechatronic, computer systems, computer science, information technology, business and health sciences. The researchers, application engineers and practising engineers will appreciate the fusion of theory and applications in this book.

The authors must be congratulated for producing such a unique work in the area of neurofuzzy modelling and applications.

University of South Australia,
October 2001

L. C. Jain, Series Editor

Preface

“Data modelling is rather like a tin of sardines. We are all looking for the key. I wonder how many of us wasted years of looking behind the kitchen dresser for the key. Others think they have found the key. They roll back the sardine of modelling, they reveal the sardines, the riches of modelling therein, they get them out and enjoy them. You know there is always a little more in the corner you can’t quite get out” – after Alan Bennett (1958) “Beyond the Fringe”.

Over the past 35 years I have been concerned with the modelling and control of complex nonlinear system. Due to the enormous literature and associated powerful results generated for linear time invariant systems, my work has focused on a variety of algorithms that allow linear systems theoretic concepts to be used for nonlinear systems. These approaches have included the beguiling (but in practice not very useful) concept of kinematic similarity, whereby a time varying/nonlinear system is transformed by a Lyapunov transformation, which retains all dynamic properties such as stability, to a linear time invariant system – this earlier research resulted in the monograph Harris and Miles, “Stability of Linear Systems”, Academic Press, 1980. Interestingly much later developments in feedback linearisation (see also Chapter 8), are based on this old concept of system dynamic equivalence. Alternate forms of linearisation – inevitably locally based – have been investigated to exploit linear theoretic results in nonlinear systems, including loop transformations or local sector or gain limited based linearisation via the small gain theorems originated by George Zames. A variety of results have emerged each dependent upon the norms or metrics used for system input–output gains, and known smoothness properties of system elements, under certain restriction linear frequency domain stability criteria have been developed which allows direct integration into the wealth of linear multivariable frequency domain stability criteria originally developed by Howard Rosenbrock and Alistair MacFarlane in the early 1970s. The general theory of nonlinear frequency domain stability criteria for multivariable systems was covered in the research text of Harris and Valenca “The Stability of Input–Output Dynamical Systems”,

Academic Press, 1983. Each of these now classical approaches to control system design and dynamical system analysis require that process or plant models are known *a priori*, or at least known with predetermined structural or parameter uncertainty (usually in the form of additive uncertainty). This approach to partial process knowledge in control system design has led to the now famous H_∞ control system theory. This theory has been extended to a limited class of nonlinear systems – usually non-dynamic (static) nonlinear gains concatenated with known linear time invariant process models (i.e. similar to that developed for frequency domain nonlinear systems analysis). Even with the remarkable theoretical developments in H_∞ theory, few real dynamical processes satisfy the restrictive dynamics imposed by these theories. Compensation via robust control schemes certainly help, but they inevitably exchange system performance for inadequacy in process models as reflected in parametric/process uncertainty representation inherent in these approaches. An alternative approach has always been adaptive control, whereby process models are derived on-line whilst simultaneously deriving stable feedback control laws. For *a priori* unknown linear time invariant processes with additive Gaussian noise, the theory of adaptive control is now complete, but for more realistic and complex time varying and general nonlinear stochastic systems rigorous results are relatively rare and specialised to specific process models such as Brunovsky forms, affine models, etc. Yet such processes are frequently observable, often well understood (at least qualitatively) and often controlled by human operators with only experiential knowledge! This has provided the main impetus behind the rapid infusion of concepts from fuzzy logic and artificial neural networks into the now rapidly developing field of intelligent control where models based on learning from observational data (rather than phenomenological models) are central to the methodology. Whilst fuzzy logic has always been controversial, its linguistic readily interpreted rule base and ability to cope with process incompleteness and vagueness has been very attractive for practical implementation, but until recently has lacked a rigorous mathematical theory. Whereas artificial neural networks, as connectionist learning machines, generate desired input–output relationships and store knowledge in an opaque manner, yet some forms – such as associative memory or linear-in-the-parameters networks are highly analytic.

In 1986, given the highly influential publication by Rumelhart and McClelland on multilayer perceptrons and the then apparently unconnected research by Mamdani and his co-workers on adaptive fuzzy logic controllers, researchers at Southampton initiated a theory (now called neurofuzzy networks) that tied together the exciting and emerging field of neural networks with linguistic rule based algorithms such as fuzzy logic. In 1991, the first neurofuzzy network was published by Brown and Harris, in which a linear in the parameter neural network was derived based upon B-spline basis functions, which had a one-to-one identity with fuzzy logic if algebraic product

and sum operators were used for logical operations. An extensive theory for neurofuzzy adaptive data based modelling was first developed in the monograph by Brown and Harris “Neurofuzzy Adaptive Modelling and Control” (1994), in which it was shown that the B-spline based class of neurofuzzy network algorithms, integrated the then apparently disparate paradigms of artificial neural networks and fuzzy logic to provide data-based learning networks that ensured guaranteed learning (due to the inherent linear-in-the-parameters structure), convergence and stability, and network conditioning, yet incorporating the more usual connectionist machines attributes of generalisation, abstraction and the ability to include (and ultimately extract) knowledge about processes in the form of easily understood linguistic type rules. It was also demonstrated that whilst the ultimate objectives were to model and represent processes linguistically the training and modelling processes are best carried out by conventional neural network weight training algorithms, followed by a transformation from the ‘learning’ weight space parameterisation to a belief or confidence rule based parameterisation.

A fundamental weakness of all lattice based neural networks (such as B-splines and radial basis functions) and rule based algorithms such as fuzzy logic and expert systems, is that as the input space dimension increases the number of network parameters, rules and required training data all increase exponentially, limiting their application to low dimensional problems. Hence fuzzy logic has only been commercialised to two/three term controllers. In consequence the Southampton group embarked upon a series of research programmes to resolve (or at least partially resolve) the so-called curse of dimensionality for neurofuzzy modelling, whilst retaining their inherent linear optimisation and transparency attributes. This research book is a collection of those endeavours, and an attempt to integrate various neurofuzzy approaches into a coherent theory within conventional signal processing. Whilst the prime purpose of this book is to produce parsimonious models from data that have some interpretation in the form of rules, modelling is not usually the only purpose. Frequently models are generated for the design of controllers, or state estimators or for use in data fusion. In these applications simple models lead to simple controllers or low state dimensional estimators.

Acknowledgements

In developing this research, which has simultaneously generated over 200 publications over the past 15 years, a considerable credit for the work is due to the many PhD students, postdoctoral researchers and academic colleagues who have individually contributed to this research. I would like to take the opportunity of formally thanking the enormous contributions of: Edgar An, Alex Bailey, Kevin Bossley, Martin Brown, C-W Chan, Sheng Chen, Tony Dodd, Rory Doyle, Oliver Femminella, Mark French, Ming Feng, Steve Gunn,

Junbin Gao, Alan Lawrence, Chris Moore, Eric Rogers, Hong Wang, Richard Walker and Z-Q Wu.

The authors and publisher are grateful to various learned journals for permission to publish some of our previous research figures. In particular, Figures 6.11–6.14 have been reprinted from the *Proc. IEE Control Theory and Applications*, **146**, pp 234–40, 1999. Figures 7.10 and 7.11 have been reprinted from the *Proc. IEE Control Theory and Applications*, **147**, pp 337–343, 2000. Figures 6.11–6.14 are from the *Proc. IEE Control Theory and Applications*, (to appear). Figures 9.1, 9.5 and 9.18–25 have been reprinted from the *Information Fusion*, Vol 2(1), pp 17–29, 2001 with permission of Elsevier Science. Figures 10.3–10 have been reprinted from the *Engineering Applications of Artificial Intelligence*, Vol 14, pp 105–113, 2001 with permission from Elsevier Science. Figures 6.16–6.19 have been reprinted from the *Applied Intelligence* (to appear in 2002), with permission of Kluwer Academic Publishers. In addition to permission to publish the above figures, the IEEE have given permission to publish figures throughout the book, where their copyright is indicated by ©IEEE.

Southampton, January 2002

Chris Harris

Contents

1. An introduction to modelling and learning algorithms	1
1.1 Introduction to modelling	1
1.2 Modelling, control and learning algorithms	7
1.3 The learning problem	9
1.4 Book philosophy and contents overview	13
1.4.1 Book overview	13
1.4.2 A historical perspective of adaptive modelling and control	21
2. Basic concepts of data-based modelling	25
2.1 Introduction	25
2.2 State-space models versus input–output models	26
2.2.1 Conversion of state-space models to input–output models	26
2.2.2 Conversion of input–output models to state-space models	28
2.3 Nonlinear modelling by basis function expansion	29
2.4 Model parameter estimation	31
2.5 Model quality	33
2.5.1 The bias–variance dilemma	33
2.5.2 Bias–variance balance by model structure regularisation	34
2.6 Reproducing kernels and regularisation networks	39
2.7 Model selection methods	42
2.7.1 Model selection criteria	43
2.7.2 Model selection criteria sensitivity	44
2.7.3 Correlation tests	45
2.8 An example: time series modelling	49
3. Learning laws for linear-in-the-parameters networks	53
3.1 Introduction to learning	53
3.2 Error or performance surfaces	55
3.3 Batch learning laws	58
3.3.1 General learning laws	58
3.3.2 Gradient descent algorithms	59

3.4	Instantaneous learning laws	61
3.4.1	Least mean squares learning	61
3.4.2	Normalised least mean squares learning	62
3.4.3	NLMS weight convergence	63
3.4.4	Recursive least squares estimation	67
3.5	Gradient noise and normalised condition numbers	68
3.6	Adaptive learning rates	70
4.	Fuzzy and neurofuzzy modelling	71
4.1	Introduction to fuzzy and neurofuzzy systems	71
4.2	Fuzzy systems	74
4.2.1	Fuzzy sets	75
4.2.2	Fuzzy operators	83
4.2.3	Fuzzy relation surfaces	87
4.2.4	Inferencing	88
4.2.5	Fuzzification and defuzzification	89
4.3	Functional mapping and neurofuzzy models	91
4.4	Takagi–Sugeno local neurofuzzy model	95
4.5	Neurofuzzy modelling examples	97
4.5.1	Thermistor modelling	97
4.5.2	Time series modelling	99
5.	Parsimonious neurofuzzy modelling	103
5.1	Iterative construction modelling	103
5.2	Additive neurofuzzy modelling algorithms	106
5.3	Adaptive spline modelling algorithm (ASMOD)	107
5.3.1	ASMOD refinements	107
5.3.2	Illustrative examples of ASMOD	111
5.4	Extended additive neurofuzzy models	119
5.4.1	Weight identification	122
5.4.2	Extended additive model structure identification	124
5.5	Hierarchical neurofuzzy models	125
5.6	Regularised neurofuzzy models	129
5.6.1	Bayesian regularisation	129
5.6.2	Error bars	132
5.6.3	Priors for neurofuzzy models	133
5.6.4	Local regularised neurofuzzy models	136
5.7	Complexity reduction through orthogonal least squares	143
5.8	A-optimality neurofuzzy model construction (NeuDec)	144
6.	Local neurofuzzy modelling	153
6.1	Introduction	153
6.2	Local orthogonal partitioning algorithms	157
6.2.1	$k - d$ Trees	157
6.2.2	Quad-trees	161

6.3	Operating point dependent neurofuzzy models	164
6.4	State space representations of operating point dependent neurofuzzy models	168
6.5	Mixture of experts modelling	173
6.6	Multi-input–Multi-output (MIMO) modelling via input variable selection	187
6.6.1	MIMO NARX neurofuzzy model decomposition	187
6.6.2	Feedforward Gram–Schmidt OLS procedure for linear systems	191
6.6.3	Input variable selection via the modified Gram–Schmidt OLS for piecewise linear submodels	193
7.	Delaunay input space partitioning modelling	201
7.1	Introduction	201
7.2	Delaunay triangulation of the input space	202
7.3	Delaunay input space partitioning for locally linear models	204
7.4	The Bézier–Bernstein modelling network	209
7.4.1	Neurofuzzy modelling using Bézier–Bernstein function for univariate term $f_i(x_i)$ and bivariate term $f_{i_1,j_1}(x_{i_1}, x_{j_1})$	209
7.4.2	The complete Bézier–Bernstein model construction algorithm	219
7.4.3	Numerical examples	220
8.	Neurofuzzy linearisation modelling for nonlinear state estimation	225
8.1	Introduction to linearisation modelling	225
8.2	Neurofuzzy local linearisation and the MASMOD algorithm	228
8.3	A hybrid learning scheme combining MASMOD and EM algorithms for neurofuzzy local linearisation	236
8.4	Neurofuzzy feedback linearisation (NFFL)	239
8.5	Formulation of neurofuzzy state estimators	245
8.6	An example of nonlinear trajectory estimation	249
9.	Multisensor data fusion using Kalman filters based on neurofuzzy linearisation	255
9.1	Introduction	255
9.2	Measurement fusion	258
9.2.1	Outputs augmented fusion (OAF)	259
9.2.2	Optimal weighting measurement fusion (OWMF)	259
9.2.3	On functional equivalence of OAF and OWMF	260
9.2.4	On the decentralised architecture	262
9.3	State-vector fusion	263
9.3.1	State-vector assimilation fusion (SVAF)	263
9.3.2	Track-to-track fusion (TTF)	264

XVI Contents

9.3.3 On the decentralised architecture	265
9.4 Hierarchical multisensor data fusion – trade-off between centralised and decentralised Architectures .	266
9.5 Simulation examples	267
9.5.1 On functional equivalence of two measurement fusion methods	267
9.5.2 On hierarchical multisensor data fusion	271
10. Support vector neurofuzzy models	281
10.1 Introduction	281
10.2 Support vector machines	282
10.2.1 Loss functions	284
10.2.2 Feature space and kernel functions	284
10.3 Support vector regression	286
10.4 Support vector neurofuzzy networks	289
10.5 SUPANOVA	297
10.6 A comparison among neural network models	303
10.7 Conclusions	304
References	307
Index	319

1. An introduction to modelling and learning algorithms

“Modelling may be said to be a little like mathematics, it can never by fully learnt.” – after Issac Walton (1593 - 1683).

1.1 Introduction to modelling

Conventional science and engineering is based on a set of abstract physical or phenomenological principles based on an evolving scientific theory such as Newtonian mechanics, wave theory or thermodynamics, which are then validated by experimental or observational data. However for analytic tractability and ease of comprehension much of this theory is linear and time invariant, yet as the performance requirement and the range of operation of controlled processes increases, nonlinear, nonstationary and stochastic system behaviours need to be adequately represented. The power of current parallel computers allows simulation methods such as computational fluid dynamics, computational electromagnetics, or finite element methods to be integrated with observational data to resolve some of these nonlinear modelling problems. However such approaches are inappropriate to processes that are too complex to have ‘local’ phenomenological representations (such as Navier Stokes, or Maxwell equations), or the underlying causal processes or physical relationships are *a priori* unknown. If measurement data or information is available in sufficient quantities and richness, then models or relationships between a systems independent and dependent (input–output) variables can be estimated by empirical or data driven paradigms. This is not a new approach, since it is the basis of classical statistical estimation theory whereby the properties of some *a priori* unknown system output $y(t) = f(\mathbf{x}(t))$ is estimated from available sets of input–output data $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$, drawn from an unknown probability distribution, which is then utilised to predict future outputs to unseen input data $\{\mathbf{x}(t)\}$ (the principle of generalisation), where the observed data set is the collection of all observed input–output regressors of the unknown process $f(\cdot)$. All data-based learning processes have two fundamental steps: (i) modelling estimation based on training data sets and (ii) output predictions on unseen or future input data – given an appropriate model which can represent the underlying features of the unknown

process. Significant research progress has been made in recent years to integrate classical statistical estimation theory with modern learning paradigms such as neural networks and fuzzy logic – see for example [49, 200].

There are two essential classes of learning approaches *supervised* and *unsupervised*. In supervised learning, models are estimated by adjusting the model parameters or weights using input and output data so that the predicted output error is reduced. Classical classification and regression tasks are usually implemented in supervised learning algorithms; the majority of approaches developed in this book are supervised learning schemes. Unsupervised learning is used to organise a model’s structure based only on the training inputs presented to the model, this type of algorithm is usually used to derive a representative set of input features, for vector quantisation or for dimensionality reduction via say clustering prior to inputting into a supervised paradigm. (Only supervised learning is dealt with in this book.)

In classical statistical model estimation from observed data, there are a series of broader procedures that need to be addressed other than just model learning or parameterisation from data, they are (following [56]):

1. *Problem Specification*; whereby domain specific prior knowledge is exploited to clarify the fundamental underlying problem.
2. *Causality Hypothesis*; whereby functional dependencies between independent and dependent variables are hypothesised.
3. *Data Acquisition*; this is a critical stage in modelling since the data provision may be fixed by the environment or open to selection and control by the modeller. The appropriate sample rate or sampling distribution may not be known *a priori* and some adaptive feedback mechanism that finds an optimal balance between the training data set and the consequent derived model accuracy. Also it is essential that both the training and validation data sets come from the same sample distribution otherwise the generalisation capacity of the model is dubious. For meaningful analysis and modelling the data set should be free of nonsensical and inconsistent values; here prior knowledge of the underlying physical process is beneficial. The removal (de-duplication) of replica data is advisable in networks in which the model complexity is bound by the sample size.

An appropriate representation of the variables may enhance the interpretability of the data. In particular, data transformations may be necessary to include domain knowledge, process representation, data conditioning, or to reduce limitations of the model learning paradigm. Often nonlinear transformations can increase the smoothness of the functional relationship as well as to linearise relationships. Transformations induced by physical-based reasoning can augment or define some information features leading to improved model generalisation capability. Equally, standardisation or normalisation of

the data with respect to its mean and standard deviation tends to improve the numerical optimisation of the modelling, through improved learning convergence.

Collinearity between variables (one variable being nearly a linear combination of others) does not provide much new information for learning and as with unrepresentative and small data sets can lead to serious problems in empirical modelling – see [16] for further details.

4. Data Preprocessing; this is an essential, usually domain specific task carried out prior to model estimation. Its prime purpose is to produce the optimal set of encoded model input training data. Data preprocessing involves a variety of tasks such as input dimension reduction through feature abstraction, linear and nonlinear principal component analysis (PCA), independent component analysis (ICA), singular value decomposition (SVD); data normalisation to ensure that all training data is of equal significance in learning, and data filtering to remove aberrant outlier data and additive measurement noise.

There is a direct computational link between the dimension of the input training data and that of parameterisation of the consequent model through model or network identification, therefore any input data dimension reduction reduces training time and model or network parameterisation (as well reducing overfitting), with a consequent contribution to network transparency (see also Chapter 6).

5. Model Identification; the prime purpose here is to generate from a set of known dependencies between appropriately encoded (or preprocessed) input–output data pairs, parsimonious models that provide accurate predictions of output variables for unseen input variables. This book concentrates on a family of data-based models or networks, which are a generalisation of classical linear-in-the-parameters statistical models, and are applicable to a large number of *a priori* unknown dynamical processes with arbitrary nonlinear relationships.

6. Validation/Verification and Transparency of Models; predictive models or neural networks are validated against test data via a series of (usually statistical) tests based on performance criteria such as cross-correlation, Akaike information criterion (AIC), maximum description length (MDL), hypothesis tests, generalised cross validation (GCV), etc. (see Chapter 2). Each test is likely to lead to different conclusions for finite data sets, therefore the choice partly determines the quality of the resultant model and its ability to generalise well. Part of an aspect of model validation is the transparency or interpretability of the final model. Most classical statistical models lead to black box models that provide no parametric relationship to either real variables or process parameters providing only effective input–output mappings between observed input–output variables. A critical issue here is if the model

can provide sufficient insight and explanation of the underlying process that generated the training data. Such models are called grey box models, and are the main emphasis of this book into empirical modelling. Here the approach to model transparency is based on the authors [32] earlier development of neurofuzzy models which naturally merge linear-in-the-parameters artificial neural networks with fuzzy logic into a coherent grey box modelling paradigm.

This book is primarily about a neurofuzzy approach to empirical data modelling for nonlinear dynamical processes concentrating on steps 4 – 6 above as they are in part domain or application independent and as such some general learning theories can be developed. However it should be emphasised that there is no single approach which is optimal, the appropriate approach to data modelling depends heavily on the application domain (e.g. prior knowledge, data provision, modelling requirements).

In the remainder of this section we introduce the fundamental concepts of data modelling that form the basis of this book. Data modelling is a process of induction and inference, whose performance is dependent upon the quantity and quality of the measurements or observational data. If the underlying process is stochastic rather than deterministic then the modelling process is more exacting, in this book we focus primarily on deterministic processes, usually with a single output, as multiple output processes can always be reduced to a set of independent single output processes. Generally our modelling objective is to find an appropriate model from sets of learning data for a process with n inputs, whose model predicted output satisfies some performance criteria or objectives. Model performance is affected by several issues:

- Can the model adequately represent the complexity of the underlying process (i.e. is it sufficiently flexible to model data complexity)?
- Can all the inputs to the process be measured?
- Are the model training performance criteria appropriate?
- What measurement errors are there?
- What model validation criteria are appropriate?
- How can prior knowledge or assumptions be incorporated into the model and learning process?
- Is the model interpretable or comprehensible to the user?
- What is the optimal balance between prior knowledge and measurements?

There are two possible modelling scenarios, the modeller can influence the modelling process by changing some of the inputs to the system through for example changing the experiment that generates data, or the nature of the process restricts the modeller to simply observing the process input–output data. Furthermore the problem can be compounded in that some inputs may not be observable. Also by its nature real observed data is finite and sampled; typically sampling is nonuniform and also due to the high dimensionality of

the model input space the data forms a sparse distribution in the input space. Consequently empirical data modelling problems are nearly always ill-posed [163]. For a problem to be well posed it must have a unique solution which should vary continuously with the data. For finite data sets the model solution that minimises the empirical risk is nonunique since there are an infinite number of approximating functions that can interpolate the data points leading to various local solutions. Generally there are two approaches to the problem of ill-posed modelling, they are *regularisation* (see Section 2.5.2) and *hypothesis testing* (see Section 2.7), both of which can be interpreted in a Bayesian context.

The basic formulation of regularisation is to minimise some regularised risk functional usually composed of two parts. The first part corresponds to the empirical risk V_{D_N} , such as the ensemble mean squares error (MSE)

$$\begin{aligned} V_{D_N} &= E\{[y(\mathbf{x}(t)) - f(\mathbf{x}(t), \mathbf{w})]^2\} \\ &= \frac{1}{N} \sum_{t=1}^N [y(\mathbf{x}(t)) - f(\mathbf{x}(t), \mathbf{w})]^2, \end{aligned} \quad (1.1)$$

where \mathbf{w} is an unknown parameter vector for the model structure $f(\mathbf{x}(t), \mathbf{w})$. A second term in addition to (1.1) penalises some characteristic of the model that usually incorporates prior model knowledge about the model solution space, such as smooth outputs or limitations on the magnitude of model parameters (or equivalently weight vector \mathbf{w}). Prior knowledge reduces the effective complexity of the model at the expense of reduced capacity to model data. Significantly, historically model identification theory [155] has been based on linear models, leading to very simple and powerful models but with limited flexibility to model complexities such as nonlinearities. An obvious generalisation of linear models is to construct models which are linear combinations of nonlinear basis functions of the input vector $\mathbf{x}(t)$. This class includes polynomials, Fourier series, B-splines as well as certain classes of neural networks such as radial basis functions, cerebellar model articulation controller (CMAC), and wavelet networks. In this book we concentrate upon linear additive networks that utilise low order *local* polynomials (such as B-splines) or basis functions to approximate much higher order *global* polynomials or nonlinear functions. These models satisfy the condition for universal function approximation, such that it is always possible to find a model within the relevant class that will approximate the data with arbitrary approximation error. This attribute whilst of theoretic importance is of little value if the paradigm overfits the (noisy) training data leading to poor generalisation (prediction) results for unseen data. The power of these generalised linear models is their ability to incorporate prior knowledge structurally by choosing the type, number and position of basis functions, whilst the selection of the model parameter vector \mathbf{w} is achieved by linear optimisation, as these networks are all linear in \mathbf{w} . This is referred to as structural regularisation

and is the subject of the various approaches to model construction (rather than parameterisation) developed in Chapters 5 – 9. For these models prior knowledge can be incorporated by altering the model structure, for example if the prior is chosen to be Gaussian, then the regularisation functional has a simple interpretation (see Section 2.5), producing models with smooth outputs.

A further class of generalised linear models can be defined which do not depend upon a set of parameters, but for which the output is a linear combination of the observations weighted by kernel functions (see Section 2.6 and Chapter 10). The optimisation of the approximant is taken over a functional including some measure of fit to the data and a functional penalising certain characteristics of the approximant. However in contrast to parametric models the priors are taken over the possible functions which the model can approximate. Significant amongst kernel based models is the support vector machine (SVM – see Chapter 10) which readily applicable to sparse data sets in relatively high dimensional spaces. Common to nonparametric approaches to functional approximation is the idea of reproducing kernel Hilbert spaces (RKHS, see Section 2.6), which provides a rigorous mathematical framework for prior knowledge in terms of projecting data onto approximation spaces. The approximations then inherit properties derived directly from these subspaces. Interestingly these subspaces are uniquely defined by the so-called reproducing kernel which corresponds to a Gaussian covariance, support vector kernel or regularisation function in the models described above, in which RKHS provide a unifying framework to treat prior knowledge [49].

The second approach to resolving ill-posed models is *hypothesis* testing, which addresses the modelling process by considering what hypothesis (models) the data can endorse. In contrast to the classical regularisation approach it considers a range of models with varying degrees of flexibility and numbers of parameters. The prior may now be viewed as biasing the solution towards models with smaller number of parameters within a particular representational scheme. One advantage of this approach is that a suitably flexible and transparent representation can be chosen which can in turn enhance the interpretation of the constructed models. Furthermore since the inputs which affect the process may be unknown it is reasonable to include as many inputs in the model as are available, letting the modelling procedure determine which inputs are significant in predicting the outputs various attributes. A hypothesis testing approach has advantages from this perspective by distinguishing those inputs which contribute or otherwise to the model (or submodels) – see Chapters 5 and 6. A possible disadvantage of the hypothesis testing approach is that it may not guarantee that the problem is strictly well posed, i.e. there may be more than one solution, or hypothesis testing itself may be ill-posed. However in practice the user, utilising domain knowledge, can easily select the most appropriate model.

In summary, we recommend an *iterative* or cyclic approach to the generalised data problem (see also 4 – 6 above):

- *Design* – Choose a flexible model representation.
- *Train* – Estimate the model parameters.
- *Validate* – Test the model to determine its validity.
- *Interpret* – What insight and explanation does the model give the process?

In practice the designer will seldom get the cycle right first time and it will be necessary to iterate the procedure many times to obtain a satisfactory model, as illustrated in Figure 1.1. The transparency of the model is paramount in enabling the information gained in previous iterations to be synthesised within the next modelling cycle. No data set will be perfect and hence two elements are demanded of the modelling procedure: (i) establish a model of the underlying data and (ii) determine the significance of the data in supporting this model.

Ideally the chosen representation should be based on the particular modelling problem. However rarely is any detailed information available and therefore it is necessary to consider a universal representation. In this book we broadly restrict the class of models to generalised linear models with potentially transparent interoperations in the form of fuzzy IF–THEN rules, hence we use the neurofuzzy label for almost all empirical modelling algorithms introduced in this book.

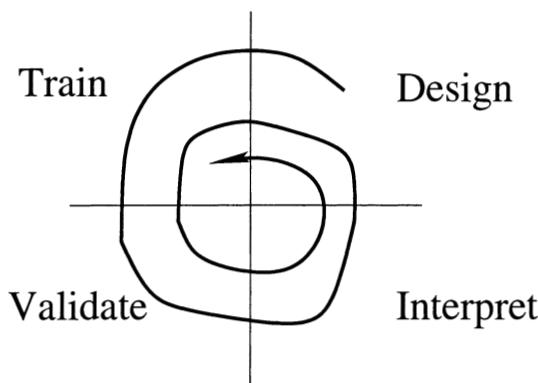


Fig. 1.1. The cyclic structure of modelling

1.2 Modelling, control and learning algorithms

Intelligent or adaptive modelling and control research has enjoyed considerable attention in recent years (see [104, 122, 175]. It is not a single cohesive

methodology or theory; rather it is a collection of complementary methodologies including fuzzy logic networks (FLN), artificial neural networks (ANN) or neurocomputing, evolutionary and emerging computing (EEC), within a unifying framework of machine learning. All are aimed at dealing with data driven processes subject to imprecision, uncertainty, nonlinearities and with little prior knowledge. Many attempts have been made to combine these apparent disparate methodologies to provide a better framework for say intelligent control; one of the most successful attempts has been that of neurofuzzy modelling by Brown and Harris [32] and Wang [198] with subsequent developments by Lin [135] and Jang [122]. The current interest in intelligent control is motivated by the need to control increasingly complex dynamical processes whilst simultaneously satisfying more stringent performance criteria, subject to process temporal and spatial variations subject to disturbances, faults and inherent nonlinearities. Fundamental in any controller design or theory is the formulation or identification of appropriate process models. In intelligent control, rather than utilise phenomenological models based upon physical–mathematical laws, models are derived from empirical data or observations/measurements of input–output data pairs or mappings of *a priori* unknown processes. The strength and weakness of this data-based approach is that it does not depend upon the restrictive but enabling assumption of linear time invariant plant. This is a weakness, since there is a significant number of highly analytical and powerful linear control theories, however by deriving local or operating point linear models from observational data (see Chapter 6) this linear control theory (see e.g. Lyapunov stability theory and linear matrix inequality controller design) can be directly applied [152].

Intelligent or learning controllers may be required for a variety of reasons including:

- The plant is time varying.
- The operating environment is only partial known or time varying.
- Prior knowledge about the plant and its structure is unknown or only partially available.
- Increased requirements for performance and robustness.
- A means for reducing design overheads, minimising human interaction.

Learning and adaptation are used interchangeably in many texts; here we mean to adapt “is to change a behaviour to conform to new circumstances”, whereas “learning generally implies the acquisition of new knowledge”. Adaptive technologies usually involve provable convergence (and stability) rules and theories for both linear and nonlinear processes, whereas learning algorithms are generally aimed at imprecisely defined processes utilising heuristics to formulate typically rule based systems. Certainly this book is focused on provable adaptive techniques, but many of them lead to knowledge discovery for ill-defined processes!

Almost all intelligent or learning controller incorporates some form of neural, fuzzy, neurofuzzy, or knowledge based algorithm. However as these algo-

rithms become more analytic and integrated into classical statistical learning theory or signal processing or into conventional control theory, it can be argued that they are no longer intelligent but rather adaptive learning algorithms with well understood stability and convergence behaviours – this book strongly subscribes to that view and attempts to develop this concept throughout.

Any adaptive or learning algorithm should have the ability or potential to [195]:

- *Sense* events within the environment;
- *Influence* its interaction within the environment;
- *Model* their cause and effective relationships;
- *Learn* in a provable and stable manner in real time.

Methods to implement such behaviours should be able to [32]:

- *Learn* from the interaction with the environment rather than by explicit programming;
- *Generalise* (that is interpolate and extrapolate) from the training data to similar but unseen situations;
- *Abstract* relevant concepts autonomously, which enable the networks to effectively generalise;
- *Incorporate* any prior knowledge about the process;
- *Extract* knowledge in a transparent manner about the underlying process from the trained model;
- *Learn* in a stable manner and in real time; and
- Provide *provable* learning convergence and stability properties.

1.3 The learning problem

In this section we formalise the modelling learning problem in the context of statistical learning theory. Given a process with N input–output data pairs $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$, the fundamental goal of modelling is to choose a model $\hat{y} = f(\mathbf{x}, \mathbf{w})$, $\mathbf{w} \in \Omega$, from the *a priori* determined set of potential models (i.e. within the so-called hypothesis space \mathcal{H} , see Figure 1.2), which is nearest with respect to some error metric to the underlying function $f(\mathbf{x})$ in the target space (see (1.2)). Errors generated by this process occur from two sources:

(i) the approximation error – due to the hypothesis space being smaller than the target space, hence the underlying function may lie outside the hypothesis space. Note this problem is *theoretically* resolved by most of the modelling algorithms in this book having the potential to approximate with arbitrary accuracy. However for all finite data modelling problems, there is inevitably model mismatch due to either poor model choice or model structures with inflexible parameterisations.

- (ii) the estimation error – which is due to the learning algorithm selecting a non-optimal model within the hypothesis space.

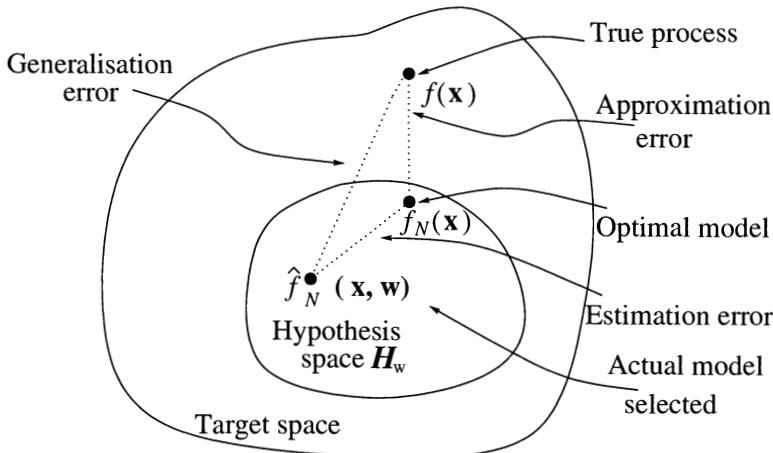


Fig. 1.2. Modelling errors

The approximation error and estimation error form the model generalisation error, see Figure 1.2. The quality of the model approximation generated by learning from observational data is that which minimises, say the L_2 loss or risk functional

$$V[f] = \int_{\mathbf{x} \times \mathbf{y}} (y - f(\mathbf{x}, \mathbf{w}))^2 p(\mathbf{x}, y) d\mathbf{x} dy, \quad (1.2)$$

where $p(\mathbf{x}, y)$ is probability density function (*pdf*). However $p(\mathbf{x}, y)$ is unknown, and $V[f]$ must be approximated, according to the empirical risk minimisation (ERM) principle [191], which provides the minimisation of the (known) empirical risk, a substitute for the (unknown) true expected risk

$$V_{emp}[f] = \sum_{t=1}^N (y(t) - f(\mathbf{x}(t), \mathbf{w}))^2. \quad (1.3)$$

Approximation produced by the ERM principle is for a fixed sample size N , biased estimates of the optimal function that minimises the true risk (as the true risk does not depend upon a particular sample); for any sample size N , $V_{emp}(f) < V[f]$, as the learning algorithm always chooses a function or model which minimises the empirical risk but not necessarily the true risk.

The ERM method is *consistent* if

$$\lim_{N \rightarrow \infty} \min_{f \in \mathcal{H}_w} V_{emp}[f] = \min_{f \in \mathcal{H}_w} V[f]. \quad (1.4)$$

Statistical learning theory [49, 190] does not rely on *a priori* knowledge about the underlying modelling problem that is to be solved and significantly

provides an upper bound, which depends on the empirical risk and the *capacity* of the model or function class, leads to the principle of *structural risk minimisation* (SRM).

The SRM inductive principle provides a formal method for selecting an optimal model structure for a finite sample data set. Unlike conventional statistical methods SRM provides accurate analytical estimates for model selection based upon generalisation bounds. According to the SRM principle solving a learning problem for a finite training data set requires *a priori* specification of the structure of a set of approximating (or loss) functions. Under SRM the set S of loss functions $V(\cdot)$, such as $V_{emp}[f] = \sum_{t=1}^N (y(t) - f(\mathbf{x}(t), \mathbf{w}))^2$, has a structure, i.e. it consists of the nested subsets (or elements) $S_k = \{V_{emp}[f(\mathbf{x}(t), \mathbf{w})], \mathbf{w} \in \Omega_k\}$, such that the hypothesis spaces,

$$S_1 \subset S_2 \subset \dots \subset S_k \subset \dots, \quad (1.5)$$

where each element of the structure S_k has finite Vapnik–Chervonenkis (VC) dimension h_k [190]. The VC dimension, h , is a scalar value that measures the *capacity* or expressive power of a set of functions or models realised by the learning algorithms. For linear methods the VC dimension is equivalent to the number of degrees of freedom of the model. By definition, modelling structure provides an ordering of its elements according to their complexity,

$$h_1 \leq h_2 \leq \dots \leq h_k \leq \dots \quad (1.6)$$

Statistical learning theory illustrates that it is essential to limit the class of models or functions f that is selected from those which have sufficient capacity, appropriate to the amount of available training data. For a given set of observations, the SRM method chooses the element S_k of the structure for which the smallest bound on the risk is achieved.

To control generalisation in the framework of this paradigm, two factors are significant; (i) the quality of the approximation of the data by the chosen model, and (ii) the capacity of the subset of models or functions from which the approximating function or model was selected.

The main difference between the SRM principle and classical statistical methods is in the control of the model general capacity factor (or VC dimension) instead of a specific model feature such as the number of model parameters. An important feature of the SRM principle is that model capacity control can be implemented in a variety of ways using different types of structures.

The theory of uniform convergence in probability [191] provides bounds on the derivation of the empirical risk from that of the expected risk. The upper bounds on the rate of the uniform convergence of the learning process evaluate the difference between the (unknown) true risk and the unknown empirical risk as a function of the sample size N , properties of the unknown pdf $p(\mathbf{x}, y)$, properties of the loss function $V_{emp}[f]$, and properties of the set of approximating functions [49]. These bounds not only provide the theoretical basis for the ERM inference principle but also motivate the SRM method

of inductive inference. Both the necessary and sufficient conditions for the consistency and rate of convergence of ERM principles depend on the capacity of the set of functions or model implemented by the learning algorithm. With probability $(1 - \delta)$ the bounds are given by [190]:

$$V[f] \leq V_{emp}[f] + \sqrt{\frac{h(\ln(2N/h) + 1) - \ln(\delta/4)}{N}}. \quad (1.7)$$

Remarkably this bound is independent of the probability density $p(\mathbf{x}, y)$. The right hand side of (1.7) is termed the risk bound or guaranteed risk. Minimisation of the expected risk $V[f]$ requires the VC dimension h to be the control variable. Hence for small expected risk (i.e. good generalisation performance), both the empirical risk and the (h/N) ratio have to be small. The first term in the inequality depends on a specific function of the set of models, whilst for a specific set of observations, D_N , the second term depends upon the VC dimension of the whole set of the models. Best generalisation performance is achieved by matching the learning algorithm to the amount of training data available.

From inequality (1.7), for a fixed number of training samples, the training error decreases as the capacity (or VC dimension) h is increased, whilst the confidence interval increases (the second term of inequality (1.7)). Accordingly both the guaranteed risk and the generalisation error converge to a minimum at an optimal VC dimension.

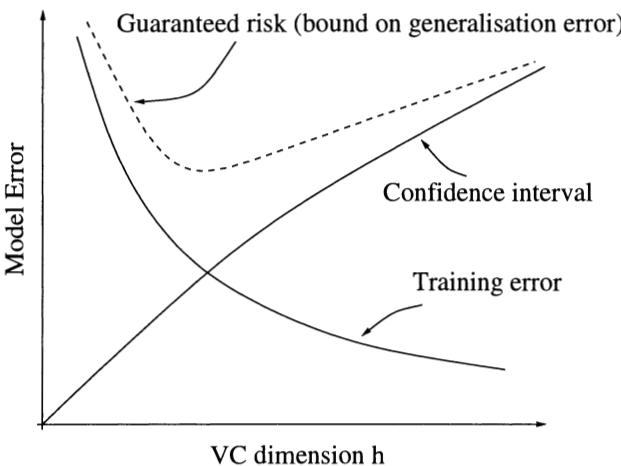


Fig. 1.3. The bound on the risk is the sum of empirical risk and the confidence interval

A trade-off between the approximation accuracy based on the training data and the learning machine's (or model's) capacity has to be made as formalised in the SRM principle before the minimisation is reached. The

learning problem is *over-determined* in that the learning capacity h is too small for the amount of training data, whereas beyond the minimum point, the learning problem is *under-determined* as the learning machine's capacity is too large for the amount of training data. For large sample sizes N , the confidence interval (or VC confidence) value becomes small, and the empirical risk is sufficient to use as a measure of the true risk. However for small sample sizes, a small value of empirical risk does not guarantee a small value of the expected risk.

The above leads to the following principle for controlling the generalisation ability of a learning machine:

“To achieve the smallest bound on the test error by minimising the training error, the set of the models or functions with the smallest VC dimension should be used.”

This is an extension of the Einstein principle of simplicity, in that the model with the simplest or least realisation or parameterisation, for a fixed training error, is always best. This general principle of parsimony is fundamental to the model construction algorithms developed in this book.

Unfortunately the basic theory of statistical learning theory is restricted to linear problems where the VC dimension can be evaluated analytically; in turn these problems are computationally intractable for high dimensional problems due to the curse of dimensionality. Much of this book is about generalised linear models which retain their nice linear analytic properties, yet the curse of dimensionality is reduced by various construction schemas. An alternative approach is to use universal constructive learning schema such as SVM that are founded in statistical learning theory [190], whose model parameterisation is independent of the problem dimensionality. We show in Chapter 10 the obvious links between neurofuzzy algorithms and the rapidly emerging field of SVMs; this is not perhaps surprising since SVMs are just linear additive models with a controller hyperparameter that controls network complexity.

1.4 Book philosophy and contents overview

1.4.1 Book overview

In this book almost all the various neurofuzzy modelling approaches satisfy the adaptive data modelling requirements listed in Section 1.2. The need for network or model provability conditions is essential for online learning, which in turn requires that the networks must be linear in the adjustable parameters. Throughout this book we will consider unknown but observable dynamical processes with N input–output observed data pairs $D_N = \{\mathbf{u}(t), \mathbf{y}(t)\}_{t=1}^N$, drawn from an *a priori* unknown probability distribution, the general modelling problem is then to find the functional mapping $\mathbf{f} : \mathbf{u} \rightarrow \mathbf{y}$ so as to:

- (i) provide a description that enumerates the dependence of \mathbf{y} on \mathbf{u} over the independent variable t ;
- (ii) allow inferences to be drawn on the relative contribution (influences/correlations) of the independent variables \mathbf{u} on \mathbf{y} ; and
- (iii) predict values of \mathbf{y} given set values of \mathbf{u} .

The training data set D_N should contain sufficient information to ensure that the various computational learning algorithms of this book can abstract an appropriate model, this is a basic assumption of any learning system and is the only prerequisite assumed throughout the book.

We consider throughout the book the general class of input–output dynamical nonlinear system represented by (see Chapter 2)

$$\mathbf{y}(t) = \mathbf{f}(\mathbf{y}(t-1), \dots, \mathbf{y}(t-n_y), \mathbf{u}(t-1), \dots, \mathbf{u}(t-n_u)) + \mathbf{e}(t), \quad (1.8)$$

where $\mathbf{y}(t) = [y_1(t), \dots, y_m(t)]^T$, $\mathbf{u}(t) = [u_1(t), \dots, u_r(t)]^T$, $\mathbf{e}(t) = [e_1(t), \dots, e_m(t)]^T$, are system output, input and measurement noise vectors respectively, and $\mathbf{f}(\cdot) : C(D) \in \Re^n \rightarrow \Re^m$ is an unknown vector valued nonlinear function defined on D , which is a compact subset of \Re^n . $C(D)$ denotes a set of continuous real valued function on D , and $n = mn_y + rn_u$ is the dimension of the model input information vector $\mathbf{x}(t)$ which is defined by

$$\mathbf{x}(t) = [\mathbf{y}^T(t-1), \dots, \mathbf{y}^T(t-n_y), \mathbf{u}^T(t-1), \dots, \mathbf{u}^T(t-n_u)]^T, \quad (1.9)$$

so that (1.8) can be represented as an input–output mapping

$$\mathbf{y}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{e}(t). \quad (1.10)$$

Frequently in model identification schemes the model input dimension n or model regressor orders (n_u, n_y) are assumed *a priori*, alternatively they have to be determined directly from D_N by some construction or structure determination algorithm – both are considered in this book. The general class of models or networks considered in this book, based on the decomposition of (1.10) into multi-input–single-output (MISO) subsystems, are of the form $\{f(\mathbf{x}) = \sum_{i=1}^p w_i \psi_i(\mathbf{x}, \mathbf{c}_i)\}$, where ψ_i are basis functions which usually have *compact support*. The support of a basis function ψ_i is the domain of the input space for which the output is nonzero $\{\mathbf{x} \in D : \psi_i(\mathbf{x}) \neq 0\}$, which is also known as its receptive field, and where w_i 's are the (linear) ‘output layer’ weights or parameters, and $\mathbf{c}_i([c_{i,1}, \dots, c_{i,n}])$ is the centre of the i th basis function ψ_i . Simply a nonlinear function $f(\mathbf{x})$ is being approximated by a series of weighted local basis functions, $\psi_i(\mathbf{x})$ (see Figure 1.4). The problem is now what order should these basis functions be, where should they be placed (widths and centres) and how should they be weighted in order to minimise the approximation error? Additionally we require the minimal number of basis functions (or weights/parameters), that provide the least approximation error and generalisation performance. The determination of the weights or parameters w_i is a linear optimisation problem

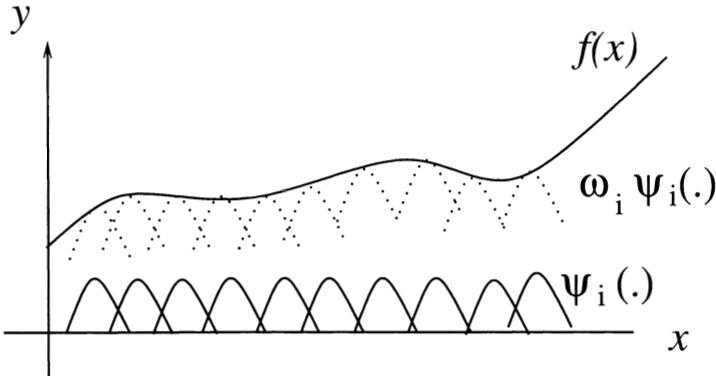


Fig. 1.4. An illustration of nonlinear functional approximation via basis functions

whereas determining basis order, centres, widths is a nonlinear optimisation problem.

A significant number of networks considered in this book (e.g. B-splines, Fuzzy logic, CMAC, radial basis functions (RBFs)) are so-called [32] lattice based associative networks (LAN) in that the network or model output is a linear combination of overlapping basis functions which are evenly distributed over an n -dimensional input subspace of \Re^n (see Figure 1.5(a)). Each of the basis functions is defined on receptive fields which are in this case hyper-rectangular subregions of \Re^n . Therefore the output of each basis function is nonzero only when the input lies in its receptive field. This feature allows LAN networks to generalise locally; i.e. similar inputs are mapped onto nearly hyper-rectangular receptive fields which produce similar outputs. Also LAN networks are ideal for online learning as learning is local, since when these networks are trained by instantaneous, rather than batch data, only those parameters or weights that contribute to the knowledge are stored in a transparent fashion, due to the fact that the network output depends only on a few basis functions, which can be interpreted as a set of linguistic rules.

LANs generally have a fast initial rate of convergence and long term parameter convergence can be established as the network's output is linearly dependent upon the parameter or weight vector w_i (see Sections 3.3 and 3.4). The rate of convergence of LANs depends on the condition number of the basis functions (see Section 3.2). LANs do have one serious disadvantage in that the number of basis functions $p(\gg n)$ is exponentially dependent on the input space dimension n , effectively limiting these networks to applications with low dimensional input spaces. Also since the number of adjustable parameters is exponentially dependent on the input space dimension, a similar amount of data is necessary to adequately train the network. An obvious

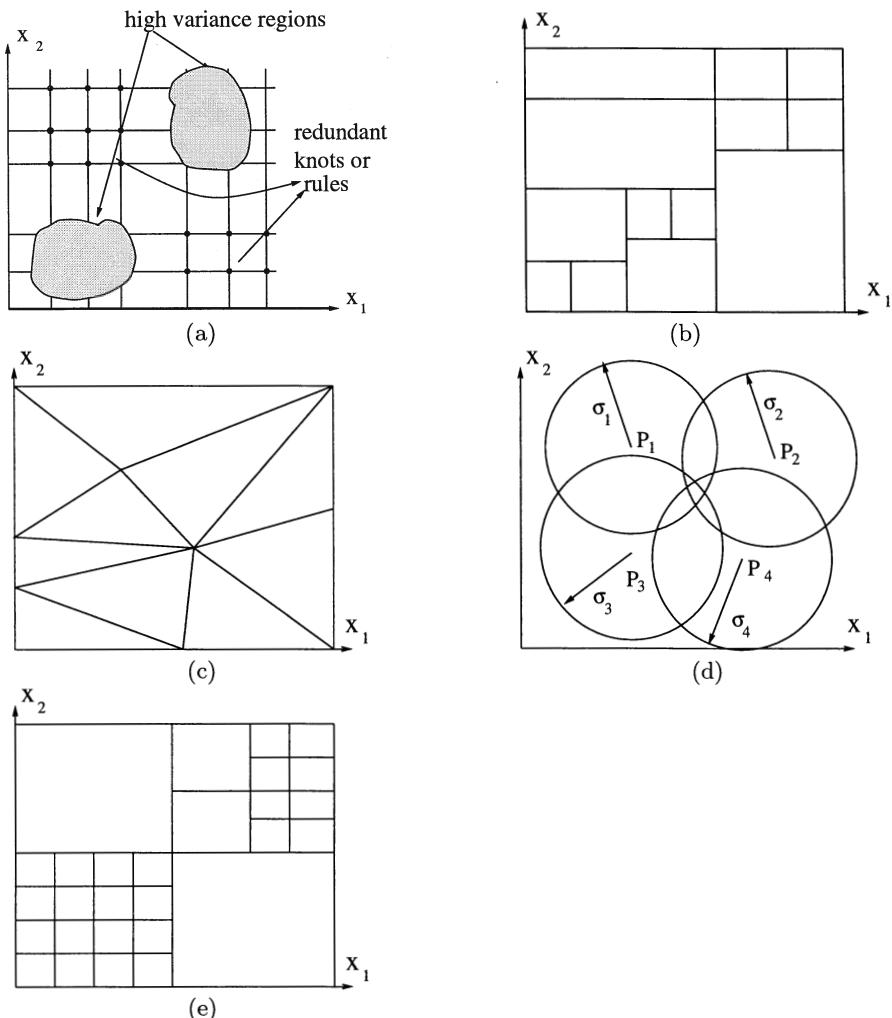


Fig. 1.5. Input space partitioning; (a) conventional orthogonal splits, (b) $k - d$ tree, (c) irregular simplexes, (d) centred Gaussians and (e) quad-trees

disadvantage of such networks is that of over-parameterisation leading to poor generalisation, any adaptive network should satisfy Occam's razor or Einstein's principle of simplicity, that is, the network should be as simple as possible, so when there are model structural choices the most parsimonious with just sufficient flexibility to store the required information is preferred.

For low dimensional modelling and control of nonlinear dynamical systems, associate memory networks such as LAN's ability to incorporate a unified approach to qualitative and quantitative knowledge representation within the same network coupled with a rigorous numerical modelling and learning theory suggest that there are compelling reasons to resolve the curse of dimensionality so that these networks can be applied to higher dimensional input space problems.

There are a variety of ways of designing networks defined on high dimensional input spaces. Clearly preprocessing algorithms such as linear and nonlinear principal component analysis (see Section 6.4), or independent component analysis or any other orthogonalisation or feature abstraction algorithm will significantly reduce the input space dimensionality, however such methods may lose transparency by operating in an eigenspace that does not relate to real or user comprehensible variables. One of the most successful approaches in reducing the effects of the curse of dimensionality is to exploit redundancy in the training data (whether this is based on *a priori* knowledge or acquired online). This knowledge may be structural so that the overall network can be modelled additively by several sub-networks of much lower dimensionality (see Section 5.2), or structured in an extended additive form that includes multiplicative terms (see Section 5.4), or modelled in an hierarchical form so that the output of one network is the input of the next (see Section 5.5). This latter approach is non-transparent since the input-output relationship of variables is highly nonlinear. Hierarchical network structuring is a well known technique for managing network complexity both for resolving complexity at each level as well as for introducing new input variables at a later stage – variants of the mixture of experts modelling schemas can be expressed in this form (see Section 6.5). Composite algorithms that automatically process data by reducing the required number of regressors followed by model identification are highly desirable in resolving the curse of dimensionality; this new approach is introduced in Section 5.8 through the A-optimality criteria.

The main cause of the curse of dimensionality is the orthogonal axis partitioning of the input space (a central requirement of all LANs). Clearly other decomposition of the input space are possible (see Figure 1.5). The conventional orthogonal split schemes produces redundant rules, knots, or intersections, other orthogonal schemes include $k - d$ trees (Figure 1.5(b)) which partition an n -dimensional space by a succession of axis orthogonal splits made across the entire domain of an existing partition. $k - d$ trees are unsuitable for neurofuzzy modelling since they are too flexible leading to poor learn-

ing convergence, are non-transparent and can lead to ill-conditioned weight optimisation problems. Quad-trees (Figure 1.5(e)) are less flexible (see Section 6.2) than $k - d$ trees resulting in a more structured local partitioning of the input space. This is achieved by successively partitioning cells of the n -dimensional space by n axis orthogonal splits, one for each axis producing 2^n cells. Berger [17] exploits this partitioning strategy to adaptively produce linear models whose output is piecewise linear across each of the hypercubes, also quad-trees have been extended to produce fuzzy box trees in fuzzy logic with some success [184].

Other *local* input space decompositions can be developed which do not depend upon orthogonal axis splits, including irregular simplexes (see Figure 1.5(c)) and data centred Gaussians (see Figure 1.5(d)) or other radial basis functions. This latter group have been extremely popular (and successful) when used in conjunction with fuzzy data clustering algorithms (see Section 6.1) and are usually based on *local linear* models of the Takagi–Sugeno (T–S) fuzzy type (see Section 4.4) being applicable to each region. Such approaches require significant *a priori* knowledge and work well for low dimensional processes subject to low measurement noise and adequate amounts of training data. The triangular decomposition of the input space (Figure 1.5(c)) via Delaunay triangulation (see Chapter 7) is a new approach, which apart from utilising the analysis of variance (ANOVA) expansion for a multivariate function $f(\mathbf{x})$ into additive lower dimensional sub-models, requires very little prior knowledge and is an automatic model construction algorithm that offers transparency.

Divide and conquer is a principle widely used to attack complex problems by dividing a complex problem into simpler subproblems whose solution can be combined to yield a solution to the overall complex problem. There are three critical issues in divide and conquer methods:

(i) The decomposition of the complex problem, preferably by utilising prior knowledge or by some data driven automatic procedure. In the case of nonlinear modelling this essentially reduces to the partitioning of the input or operating space as in the $k - d$ or quad-trees, or Takagi–Sugeno approaches (see Sections 6.2 and 6.3).

(ii) The design of the corresponding local models to generate global models that generalise well. These local models maybe linear as in the case of the Takagi–Sugeno (see Section 8.1), or neurofuzzy local linearisation (NFLL) (see Chapter 8), or nonlinear in the case of the Bézier–Bernstein polynomial models that utilise Delaunay decomposition of the input space (see Chapter 7).

(iii) Combination of the local linear models. Here fuzzy and probabilistic methods usually provide smooth/local model interpolation. Jordan’s mixture of experts algorithm is a probabilistic approach which finds a weighted sum of local probabilistic models. This is extended in Section 8.3 via a hybrid learning algorithm, that optimally combines the analysis of variance functional

expansion of a multivariate function and the expectation maximisation algorithm of Jordan and Jacobs [126] to automatically generate local neurofuzzy linear models which retain full model transparency yet remove the curse of dimensionality and the manual decomposition of the input space.

Much of modern estimation and control theory is based on state space models rather than input–output models. Of particular importance are linear canonical state space representations, which raised a significant issue in data-based modelling. Can canonical state space process models be derived for nonlinear dynamical processes directly from data? Certainly when the input space is decomposed so that locally linear models are appropriate local canonical state space models can be derived (see Sections 2.2 and 6.3), equally when a globally linear model with operating point dependent parameters is realised a neurofuzzy state space model follows directly (see Sections 2.2 and 6.3). A more generic and theory based approach is to consider exploiting output-feedback linearisation using adaptive algorithms or linear-in-the-parameters neural networks. In output-feedback linearisation, the state space system

$$\begin{aligned}\dot{\mathbf{x}} &= f(\mathbf{x}) + \mathbf{q}_0(\mathbf{x}, u) + \sum_{i=1}^p w_i \mathbf{q}_i(\mathbf{x}, u) \\ y &= h(\mathbf{x})\end{aligned}\tag{1.11}$$

are often considered, where all functions assumed smooth and *a priori* known, with only uncertainties in the parameters w_i . By using a state coordinate transformation $\mathbf{z} = T(\mathbf{x})$, and a feedback control law, (1.11) can be converted into the following canonical form

$$\begin{aligned}\dot{\mathbf{z}} &= \mathbf{A}_c \mathbf{z} + \Psi_0(y, u) + \sum_{i=1}^p w_i \Psi_i(y, u) \\ y &= \mathbf{C}_c \mathbf{z},\end{aligned}\tag{1.12}$$

for $\mathbf{A}_c, \mathbf{C}_c$ canonical matrices and Ψ_i smooth vector functions in the observables $\{u, y\}$.

Currently the application of neural networks to output feedback linearisation has been limited to the identification of uncertainty parameters or weights with fixed basis functions, Ψ_i , and assume that the state vector \mathbf{z} is observable or that the transformation $\mathbf{z} = T(\mathbf{x})$ can be constructed correctly. In Section 8.4, a recurrent neurofuzzy network based on the neurofuzzy linearisation schema of Section 8.3, is developed for state space model realisation from empirical data-based only on the assumptions:

- (i) The original output measurement process is linear – this is reasonable since major efforts are made by sensor manufacturers to ensure that sensors are linear over applicable ranges.
- (ii) The state coordinate transformation $\mathbf{z} = T(\mathbf{x})$ is linear, but unknown.
- (iii) The nonlinear functions $f(\cdot), h(\cdot)$ are *a priori* unknown.

A further problem remains, even if canonical state space models can be derived from empirical data, since in practice the observations are noisy some form of state space estimation is required to reconstruct the system's state from noisy observations. Fortunately the neurofuzzy operating point dependent models or neurofuzzy local linearisation models or recurrent feedback linearisation models are structured so as to be directly applicable to the famous Kalman filter for state vector estimation in Section 8.5. Whilst the underlying processes are inherently nonlinear, problems with convergence and stability associated with the extended Kalman filter are not prevalent here since no state or measurement functional linearisation has been performed.

One of the most important roles for state estimation is for target tracking where the target dynamics are unknown and highly nonlinear. Here the task is from a range of differing modality sensors to reconstruct and predict the future target trajectory, which is superior to those generated from the individual sensors – this is called multisensor data fusion. Given that we have derived in Chapter 8 a series of Kalman filter estimators based upon the neurofuzzy empirical modelling techniques, Chapter 9 is concerned about optimal informational integration techniques that ensure that the original target tracks can be recovered utilising a range of sensors and observational data only. The approach to data fusion is based upon a distributed-decentralised architecture to ensure maximal robustness to the loss of sensors or data sources and considers two fundamentally different forms of data fusion – state-vector fusion (see Section 9.3) and output measurement fusion (see Section 9.2). Whilst distributed decentralised data fusion architecture offers maximal robustness and flexibility it does have a high communication bandwidth overhead, since all sensors of data processing centres have to be connected. Therefore a hybrid hierarchical multisensor data fusion architecture introduced in Section 9.5 allows a trade-off between centralised and decentralised data fusion through a controllable feedback element. A ship tracking example is given throughout Chapter 9 to evaluate the efficacy of the various approaches to state estimation and data fusion from observational data.

An alternative approach to dealing with the curse of dimensionality is to use nonparametric models whose output is a linear combination of the observable \mathbf{x} , where the linear weighting functions are determined by the characteristics of kernel functions [49] which form the basis of the rapidly emerging theory of support vector machines [191] in empirical data modelling. In nonparametric models, the model parameters are not predetermined but depend upon the data itself and are used to reconcile the resultant modelling capacity with the associate model data set. In Chapter 10 we briefly introduce support vector machines and their links to input space based modelling algorithms such as neurofuzzy algorithms introduced earlier. The support vector neurofuzzy modelling algorithm and the support vector analysis of variance (SUPANOVA) algorithms are introduced in Section 10.4 and 10.5

respectively, illustrated by benchmark examples that have been evaluated throughout the book for comparative purposes.

All the neurofuzzy algorithms introduced in this book have been evaluated by the authors on real word data modelling problems including:

- Submersible vehicle modelling and control [24]
- Gas turbine modelling and fault detection [29]
- Car driver and driving modelling [3]
- Obstacle detection and collision avoidance for cars [3]
- Missile tracking and guidance laws [150]
- Car engine torque modelling [34]
- Ship tracking, guidance and control [102]
- Ship collision avoidance systems [189]
- Helicopter collision avoidance system [57]
- Processing property relationships of aerospace (Al alloys) [64]

1.4.2 A historical perspective of adaptive modelling and control

Adaptive learning theory is certainly not new with the earliest neural networks being derived in the 1950s by Widrow and the associated least mean squares parameter learning law being derived much earlier by Kaczmarz in 1937. Widrow's simple adaptive linear element (ADALINE) was taught to stabilise and control an inverted pendulum, and is still a benchmark problem for emerging adaptive controllers, e.g. [15, 147, 148]. The ADALINE was soon followed by Rosenblatt's Perceptron in 1961, whose simple architecture has been the basis of many more sophisticated versions of neural networks. One of the most significant developments in neuro-controllers in the 1970s was Albus' cerebellum model articulation controller (CMAC); this was the first of a family of associative memory or lattice based neural networks which are linear-in-the-parameters. Albus proposed CMAC as a tabular or look up associative memory device that reflected the functioning of the cerebellum. It has been extensively used in robotic control, chemical process control and car driving where provable control stabilisation is required. Miller extended the theory of CMAC in the 1980s followed by Tolle and Ersu [187], and Brown and Harris [32] who have shown its links to the ADALINE, as well as to fuzzy logic (see [32] for review article). In his seminal and controversial paper of 1963 Zadeh introduced the concept of fuzzy logic, as fuzzy logic suffers from the curse of dimensionality it has been limited in application to typically two input systems with extensive application to domestic products such as cameras, washing machines, toilet seats as well as more complex systems such as chemical plants and city transport (see [150] for review article). The basis of fuzzy logic control was first introduced by Brae and Rutherford in 1979 and Tong [188] in 1977 and later by Pedrycz [161] in 1993. Significantly Mamdani and his students introduced the direct adaptive fuzzy controller (i.e. model free) in 1979, followed by the indirect (i.e. model based) adaptive

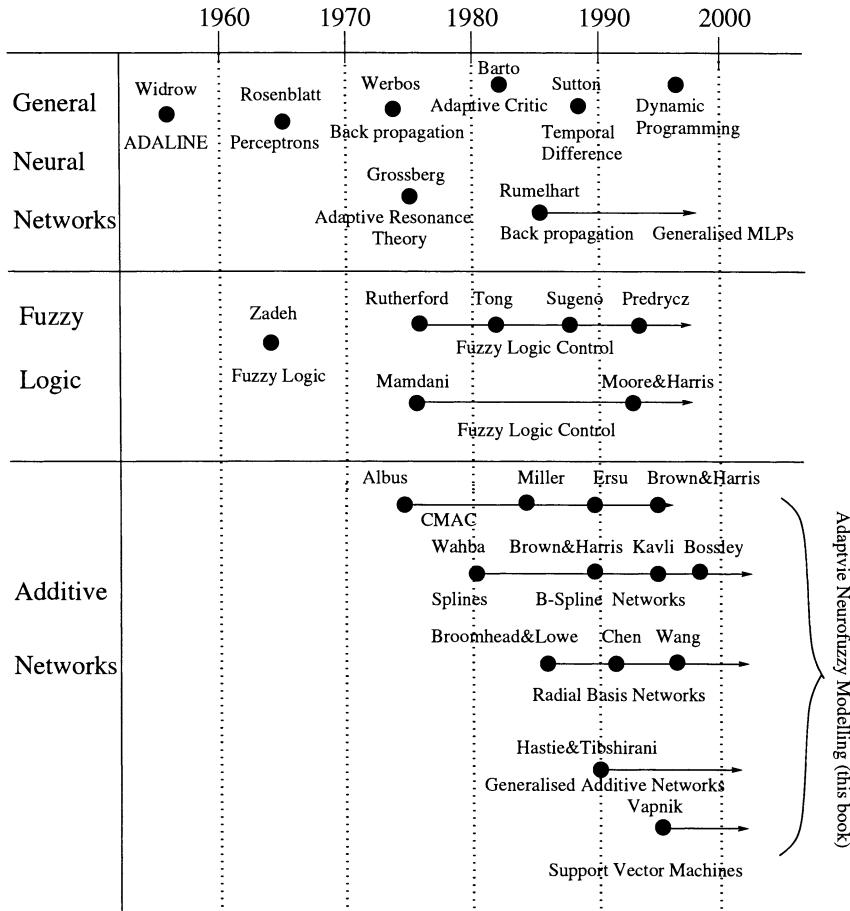


Fig. 1.6. A historical perspective on adaptive learning modelling and control

fuzzy controller in 1994 by Moore and Harris [150]. The obvious analytical capabilities of associative memory neural networks were recognised and first exploited in the late 1980s when various basis functions were exploited to generate generalised linear adaptive networks such as the radial basis function networks of Broomhead and Lowe [31], the B-spline neural network of Brown and Harris [32], the generalised additive networks of Hastie and Tibshirani [103]. The work of Brown and Harris [32] and Wang [198] for B-splines and Gaussian neural networks respectively established the first links between neural networks and fuzzy logic, producing the first neurofuzzy adaptive networks which have the linguistic transparency of fuzzy logic coupled with the analytical tractability of neural networks. The structure of these empirically generated models make them ideal for neuro-control due to their linear-in-the-adjustable-parameters structure. Unfortunately associative memory networks

suffer from the curse of dimensionality therefore considerable research has been devoted to resolving this problem whilst retaining the transparency and linear in the parameter attributes of neurofuzzy algorithms. Notable amongst this research has been the adaptive spline modelling algorithms (ASMOD) of Kavli [129] and its derivatives by Bossley, Brown, Gan and Harris (1997 – 2000) by the Southampton group. Alternative recent research based on orthogonal least squares, A-optimality design criteria and Bézier–Bernstein polynomials are discussed in this book indicating that the theory of generalised additive networks is far from complete.

In 1995, Vapnik [190] exploited the very early work of Aronszajin [7] in 1970 on reproducing kernel Hilbert spaces to provide a rigorous mathematical treatment of nonparametric additive models in terms of projections of data or observations onto approximating subspaces. Here the models are instead linear weightings of the observations where the weights are determined by kernel functions. This new theory of support vector machines is founded in statistical learning theory and provides a useful framework for considering other Bayesian networks such as Gaussian processes due to Neal [153] and Mackay [139]. Significantly, support vector machines can be interpreted as neurofuzzy networks and applicable to sparse data sets in relatively high dimensional spaces. Figure 1.6 illustrates the interlinking of the adaptive/learning algorithms discussed in this section, which form the basis of the remainder of the book.

2. Basic concepts of data-based modelling

“Never let your data get in the way of analysis.” – T. J. Lowi

2.1 Introduction

In this chapter we provide the basic model mathematical representations that are used throughout the book. Both parametric and nonparametric models require structural representations which reflect the modeller’s prior knowledge of the underlying process, or is selected so as to provide a form that ensures process identification easily from observed data, or is selected with another end process in mind such as control or condition monitoring. Generally in control and fault diagnosis problems, the model representation is in state space form (see Section 2.2) as knowledge of the unknown systems states are required for detection of process faults, or for state feedback control, or for state vector data fusion (see Section 9.3). Fundamental to this book is the representation of nonlinear observable processes by additive basis function expansions for which the basis functions are locally defined (i.e. have compact local support) rather than global basis functions such as general polynomials. In the sequel this representation coupled with linearly adjustable parameters is shown to have many knowledge and computational based advantages such as parameterisation via linear optimisation, easy incorporation of prior knowledge, and model transparency with direct links to fuzzy rule base representations.

Two fundamental problems of all data-based modelling approaches with finite data sets is resolving the bias–variance dilemma and dealing with ill-posedness (see Section 2.5). Both problems can be resolved by imposing various forms of structural regularisation that controls the model predicted output’s smoothness or the number and magnitude of model parameters. For specific models such as generalised linear models and neurofuzzy models, regularisation – both global and local, is dealt with in detail in Chapter 5, however the general principle of model quality is discussed in Section 2.5.

For a fixed data set D_N , just as there are a wide variety of models that can be selected to fit that data, there are also various cost functions $V_N(\mathbf{w})$ that can be used to evaluate the model. Generally, a quadratic cost functional

is used in this text (see Section 2.4), since the linear-in-the-parameter (\mathbf{w}) networks developed in this book lead to a linear optimisation solution with this cost function, if the basis functions are *a priori* fixed. However a quadratic cost functional leads to a nonlinear optimisation problem if the model is non linear-in-the-parameters, such as basis width, centres or knots. In Section 2.7 a variety of model selection criteria are introduced that are appropriate to generalised linear-in-the-parameters networks. A simple time series modelling example is used to compare the efficacy of each approach.

2.2 State–space models versus input–output models

2.2.1 Conversion of state–space models to input–output models

Whilst nonlinear dynamical systems have only observable input–output data pairs, they have unobservable internal state, $\mathbf{z}(t)$, which represent the minimum number of variables that incorporate the complete dynamics of the process, which together with the state transition matrix completely defines the underlying dynamic process. The generation of state space models from input–output data pairs is essential to the formulation of nonlinear state estimators – a prerequisite of any data fusion or tracking algorithm (see Chapters 8 and 9). In this book we consider discrete-time time-invariant nonlinear dynamic state representations Σ :

$$\begin{aligned} \mathbf{z}(t+1) &= \mathbf{g}[\mathbf{z}(t), \mathbf{u}(t)] \\ \mathbf{y}(t) &= \mathbf{h}[\mathbf{z}(t), \mathbf{u}(t)], \end{aligned} \quad (2.1)$$

where $t \in \mathcal{T}$ is a set of integers, $\mathbf{z}(t) \in \mathcal{Z}$, the system state of dimension l , $\mathbf{u}(t) \in \mathcal{U}$, the input of dimension r , $\mathbf{y}(t) \in \mathcal{Y}$, the output of dimension m ; $\mathbf{g} : \mathcal{Z} \times \mathcal{U} \rightarrow \mathcal{Z}$, the one-step-ahead transition function, and $\mathbf{h} : \mathcal{Z} \times \mathcal{U} \rightarrow \mathcal{Y}$, the output or measurement function. The state transition matrix Φ can be formed by repeated application of the state equation $\mathbf{g}[\cdot]$ in (2.1). Define the set $\mathbf{u}^* = \{\mathbf{u}(t+k-1), \dots, \mathbf{u}(0)\} \in \mathcal{U}^*$, for $k > 0$, then $\mathbf{z}(t+k) = \Phi[\mathbf{z}(0), \mathbf{u}(t+k-1), \dots, \mathbf{u}(0)]$, where $\Phi : \mathcal{Z} \times \mathcal{U}^* \rightarrow \mathcal{Z}$. From (2.1) we have

$$\mathbf{y}(t+k) = \mathbf{h}[\Phi[\mathbf{z}(0), \mathbf{u}(t+k-1), \dots, \mathbf{u}(0)], \mathbf{u}(t+k)]. \quad (2.2)$$

The state $\mathbf{z}(0)$ is an equilibrium state if there exists an input $\mathbf{u}(0)$ such that $\mathbf{z}(0) = \mathbf{g}[\mathbf{z}(0), \mathbf{u}(0)]$ and $\mathbf{y}(0) = \mathbf{h}[\mathbf{z}(0), \mathbf{u}(0)]$.

Equation (2.2) can be rewritten as an input–output mapping or system response:

$$\mathbf{y}(t+k) = \mathbf{f}[\mathbf{z}(0), \mathbf{u}(t+k), \mathbf{u}(t+k-1), \dots, \mathbf{u}(0)]. \quad (2.3)$$

It has been shown by Leontaritis and Billings [133] that if the system (2.3) can be described by a state–space equation in a finite dimensional space and when the system is close to its equilibrium point it can be approximated by

a linearised system. For a single-input-single-output (SISO) system, it can be represented in a recursive input-output form as

$$y(t) = f[y(t-1), \dots, y(t-n_y), u(t-1), \dots, u(t-n_u), \mathbf{w}] + e(t), \quad (2.4)$$

for $f[\cdot]$ some nonlinear mapping; n_y and n_u are positive integers representing lags in the system observable inputs/outputs. In practice, $y(t)$ is subject to noisy observations or model mismatch through the noise term $e(t)$ (usually assumed as an uncorrelated Gaussian sequence with variance σ^2).

Approximate (2.4) by a linearised system:

$$\begin{aligned} \hat{y}(t|\mathbf{w}) &= -a_1 y(t-1) - \dots - a_{n_y} y(t-n_y) + b_1 u(t-1) + \\ &\quad \dots + b_{n_u} u(t-n_u) \\ &= \mathbf{x}^T(t)\mathbf{w}, \end{aligned} \quad (2.5)$$

where $\mathbf{w} = [-a_1, \dots, -a_{n_y}, b_1, \dots, b_{n_u}]^T \in \Re^n$, ($n = n_y + n_u$), is an unknown parameter vector and $\mathbf{x}(t) = [y(t-1), \dots, y(t-n_y), u(t-1), \dots, u(t-n_u)]^T$ is a known input/output observation vector or a set of system regressors. Suppose that there are N data points $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$ available to estimate the model parameter \mathbf{w} and that a mean squared error (MSE) criterion is used for measuring the approximation of $y(t)$ by $\hat{y}(t|\mathbf{w})$:

$$\begin{aligned} V_N(\mathbf{w}, D_N) &= \frac{1}{N} \sum_{t=1}^N [y(t) - \hat{y}(t|\mathbf{w})]^2 \\ &= \frac{1}{N} \sum_{t=1}^N [y(t) - \mathbf{x}^T(t)\mathbf{w}]^2. \end{aligned} \quad (2.6)$$

Then the optimal parameter vector $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \{V_N(\mathbf{w}, D_N)\}$ is given by differentiating (2.6) with respect to \mathbf{w} and setting to zero,

$$\hat{\mathbf{w}} = [\mathbf{X}^T \mathbf{X}]^{-1} \mathbf{X}^T \mathbf{y}, \quad (2.7)$$

where $\mathbf{X} = [\mathbf{x}(1), \dots, \mathbf{x}(N)]^T \in \Re^{N \times n}$ is a regression matrix, and $\mathbf{y} = [y(1), \dots, y(N)]^T \in \Re^N$, the output vector. Clearly the type of model optimised depends on the choice of regressors used in $\mathbf{x}(t)$. They can for example include past inputs $u(t-k)$, past outputs $y(t-k)$, recurrent model outputs from past u , i.e. $\hat{y}(t-k|\hat{\mathbf{w}})$, and past model residuals $e(t-k|\hat{\mathbf{w}}) = y(t-k) - \hat{y}(t-k|\hat{\mathbf{w}})$. These various regressors have been extensively used for linear modelling. For example, finite impulse response (FIR) models use past inputs $u(t-k)$ only, autoregressive models use $\{u(t-k), y(t-k)\}$ as regressors, and autoregressive moving average models use $\{u(t-k), y(t-k), e(t-k|\mathbf{w})\}$ as regressors.

For the nonlinear case with noisy observations, (2.4) becomes

$$\begin{aligned} y(t) &= f[y(t-1), \dots, y(t-n_y), u(t-1), \dots, u(t-n_u), e(t-1), \\ &\quad \dots, e(t-n_e), \mathbf{w}] + e(t), \end{aligned} \quad (2.8)$$

where n_e denotes the lag of the noise term. Clearly ignoring the regressors in $\{e(t)\}$ in the model (2.8) will lead to a biased solution to the parameter estimation.

Assuming that noise is only additive in (2.8), it is natural to use the same set of regressors for nonlinear systems as used in linear modelling, so that a general “black-box” model for (2.8) could be

$$\hat{y}(t|\mathbf{w}) = f(\mathbf{x}(t); \mathbf{w}). \quad (2.9)$$

Special cases of this have been developed, for example, Chen and Billings [45] have developed:

- (i) Nonlinear finite impulse response models which use only $u(t - k)$ as regressors;
- (ii) Nonlinear autoregressive (NARX) models which use $u(t - k)$ and $y(t - k)$ as regressors;
- (iii) Nonlinear autoregressive moving average (NARMAX) models which use $u(t - k)$, $y(t - k)$, and $e(t - k|\mathbf{w})$ as regressors.

The NARX model is non-recurrent (with no $\hat{y}(t|\mathbf{w})$ in the regressor vector) and quite general, allowing easy parameter estimation, but requires a high dimensional regressor vector $\mathbf{x}(t)$ for good approximation and includes any noise dynamics within the model of the process dynamics. However, replacing y by \hat{y} in the regressor changes a NARX model into a nonlinear output error (NOE) model which only models the system dynamics and not the noise dynamics. To provide a separate parameterisation of the additive noise terms, the regressor vector needs to be amended to include prediction errors $e(t - k|\mathbf{w})$, to produce the most general model NARMAX [176, 177]. In practice the noise model is restricted to linear in the residuals to reduce overall model flexibility. In this case, the NARMAX models are of the form

$$\hat{y}(t) = f(\mathbf{x}(t); \mathbf{w}) + C(q^{-1})e(t), \quad (2.10)$$

for $C(q^{-1})$ some linear filter, and q^{-1} the unit delay operator. Note that this NARMAX model is simply a NARX model with an additive linear noise term.

Other nonlinear models include Wiener and Hammerstein models [133], which cascade a static nonlinearity $z = g(u)$ followed by a linear system. For example the overall input–output Hammerstein model is

$$\begin{aligned} y(t) &= -a_1y(t-1) - \dots - a_{n_y}y(t-n_y) \\ &\quad + b_1g(u(t-1)) + \dots + b_{n_u}g(u(t-n_u)) + e(t). \end{aligned} \quad (2.11)$$

2.2.2 Conversion of input–output models to state–space models

Input–output models are special cases of state–space models. Conversely, state–space models may be constructed from input–output models based on generalised regressors via the Witney–Takens theorem [38]. The formation of

the state space models are not unique, depending on the specific definition of state vector. For example, if we take the generalised linear ARMA model with time-varying coefficients:

$$\begin{aligned} y(t) = & -a_1(t)y(t-1) - \dots - a_{n_y}(t)y(t-n_y) \\ & + b_1(t)u(t-1) + \dots + b_{n_u}(t)u(t-n_u) + e(t), \end{aligned} \quad (2.12)$$

and denote $\tilde{u}(t) = b_1(t)u(t-1) + \dots + b_{n_u}(t)u(t-n_u)$, then model (2.12) becomes

$$y(t) = -\sum_{i=1}^{n_y} a_i(t)y(t-i) + \tilde{u}(t) + e(t) \quad (2.13)$$

Define $z_i(t) = y(t-n_y+i-1)$. It follows that $z_i(t+1) = y(t-n_y+i) = z_{i+1}(t)$, $i = 1, 2, \dots, n_y-1$, and $z_{n_y}(t+1) = y(t) = -\sum_{i=1}^{n_y} a_i(t)z_{n_y-i+1}(t) + \tilde{u}(t) + e(t)$, leading to the following canonical controllable state-space realisation:

$$\begin{aligned} \mathbf{z}(t+1) = & \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_{n_y}(t) & -a_{n_y-1}(t) & -a_{n_y-2}(t) & \dots & -a_1(t) \end{bmatrix} \mathbf{z}(t) \\ & + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tilde{u}(t) + \mathbf{v}(t), \end{aligned}$$

$$y(t) = [-a_{n_y}(t) \dots -a_2(t) -a_1(t)]\mathbf{z}(t) + \tilde{u}(t) + \xi(t), \quad (2.14)$$

where $\mathbf{z}(t) = [z_1(t), \dots, z_{n_y}(t)]^T \in \Re^{n_y}$, $\mathbf{v}(t)$ is an additional state or process noise, and $\xi(t) = e(t)$. Clearly (2.14) can be written in a state-space form:

$$\begin{aligned} \mathbf{z}(t+1) = & A(t)\mathbf{z}(t) + B\tilde{u}(t) + \mathbf{v}(t) \\ y(t) = & C(t)\mathbf{z}(t) + D\tilde{u}(t) + \xi(t). \end{aligned} \quad (2.15)$$

This input-output model to control canonical state space representation is fully exploited in the sequel in generating the neurofuzzy Kalman filter (see Section 6.4), the A-optimality NeuDec algorithm of Section 5.8, and the various local linearised estimators of Chapter 8.

2.3 Nonlinear modelling by basis function expansion

Consider again the general nonlinear mapping (2.9), a natural means of expressing this is in the form of function expansion in terms of basis functions, i.e.

$$\hat{y}(t|\mathbf{w}) = f(\mathbf{x}(t), \mathbf{w}) = \sum_{i=1}^p w_i \psi_i(\mathbf{x}(t), \mathbf{c}_i) = \psi^T(\mathbf{x}(t))\mathbf{w}, \quad (2.16)$$

where $\psi_i(\cdot)$ are basis functions, since they may form a functional space basis for fixed regressor vector. Basis functions are usually of two generic types defined by their coverage of the input space:

- *Local basis functions*, which have gradient with bounded support, and are usually defined over a (local) interval. Examples include wavelets [176], RBFs [85], kernel functions [194], B-splines and fuzzy systems [32], and Bézier–Bernstein polynomials ([105], see also Chapter 7).
- *Global basis functions*, which are functions with infinitely spreading (bounded or unbounded) gradient covering all of the input space. Examples include Fourier series, sigmoidal neural networks [104], hinging hyperplanes [28], and projection pursuit regressors [113].

For the normal approximation problem, multivariable basis functions have to be constructed from univariate basis functions. Three of the most useful methods are:

- *Tensor product*. Given n univariate functions $\{\psi_{ij}(x_j)\}$ where one basis function is defined on each input axis, the multivariable basis function is the tensor product of the univariates, i.e. $\psi_i(\mathbf{x}) = \prod_{j=1}^n \psi_{ij}(x_j)$. Good examples here are B-splines (see Figure 2.1) and fuzzy sets, which are computationally simple and whose shape may incorporate prior knowledge about the underlying process (see Chapter 4). For example a relay type surface representing discrete changes in output could be modelled using order 1 basis functions, whereas a locally linear nonlinearity could be used via order 2 basis functions. In this latter case these ‘triangular’ basis functions or fuzzy sets are very popular in the literature due to their inherent simple representation and consequent analytical properties. For continuous smooth nonlinearities quadratic or cubic basis functions may be utilised. Mixed multivariable basis functions may be used to reflect local quadratic behaviour on one axis and local linear on another (a form of *a priori* knowledge influencing model selection).
- *Radial construction*. Generally the radial construction of multivariable basis functions is of the form: $\psi_i(\mathbf{x}) = \psi_i(\|\mathbf{c}_i - \mathbf{x}\|_2)$, where \mathbf{c}_i is the i th centre defined in the input space. One important choice is the localised Gaussian function given by $\psi_i(\|\mathbf{c}_i - \mathbf{x}\|_2) = \prod_{j=1}^n \exp(-0.5(\frac{c_{ij} - x_j}{\sigma_i})^2)$, where σ_i indicates the influence or width of the local Gaussian around its centre c_{ij} . This form of construction has led to the radial basis function network [45, 46, 47, 48], which like the B-spline network has strong analytical properties and direct connections to fuzzy representations.
- *Ridge construction*. Let $\varphi(\cdot)$ be any single variable function, then for all $\mathbf{c}_i \in \mathbb{R}^n$, $a_i \in R$, a ridge function is defined as $\psi_i(\mathbf{x}) = \psi_i(\mathbf{x}, a_i, \mathbf{c}_i) = \varphi(\mathbf{c}_i^T \mathbf{x} + a_i)$. This function is semi-global since it is constant for all \mathbf{x} in

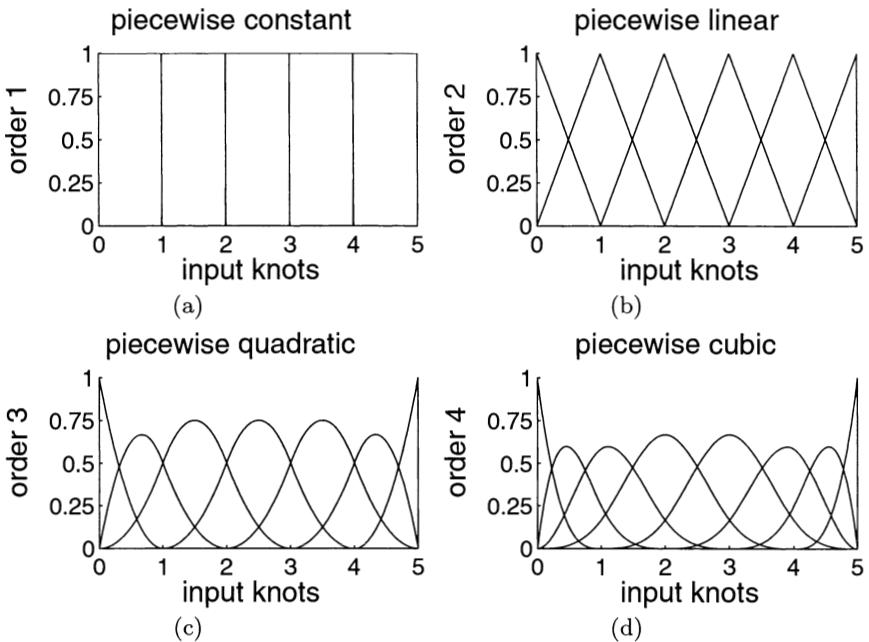


Fig. 2.1. B-splines for orders 1 – 4

the subspace $\{\mathbf{x} \in R^n : \mathbf{c}_i^T \mathbf{x} = \text{constant}\}$, hence the name ridge. The most famous example of ridge basis function is when $\varphi(\cdot)$ is a sigmoid (Figure 2.2), leading to the well known multilayer perceptron neural network [23]. The hinging hyperplane model can be expressed as a ridge construction plus an additional linear term.

2.4 Model parameter estimation

The selected model must inevitably reflect its intended use in say condition monitoring, knowledge discovery, control or tracking. Any prior knowledge, relevant regressors, assumed structure, and smoothness properties should be encoded into the initial model structure. The model identification problem requires several interconnected subproblems to be resolved: (i) Generation of the appropriate data set D_N which is sufficiently rich and persistent; (ii) Selection of the appropriate set of regressors $\mathbf{x}(t)$; (iii) Selection of the appropriate basis functions, number and orders; (iv) Estimation of the model parameters \mathbf{w} ; and (v) Validation of the model against unseen data to evaluate the model's ability to generalise.

Given a data set $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$, and some model structure $f(\mathbf{x}(t), \mathbf{w})$, the objective of parameter estimation is to draw inferences about

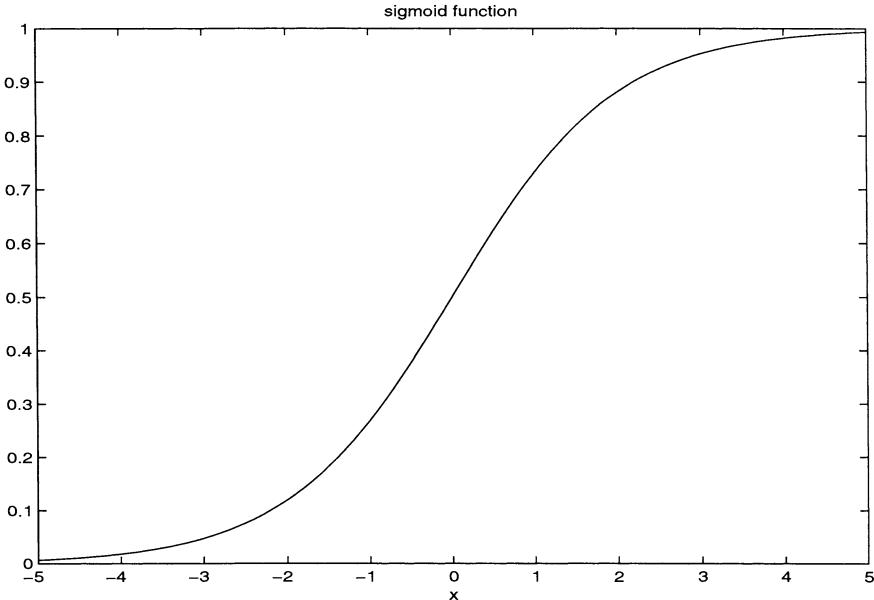


Fig. 2.2. Sigmoidal function

the parameter vector \mathbf{w} , so that the model produces a good approximation to the true system. Here the objective is to find the most probable \mathbf{w} given the data set D_N and the model structure $f(\cdot)$. The probability density function (pdf) for \mathbf{w} given D_N is given by Bayes theorem:

$$p(\mathbf{w}|D_N) = \frac{p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \mathbf{w}) p(\mathbf{w})}{p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N)}. \quad (2.17)$$

This *a posteriori* pdf is composed of three pdf's: the likelihood function, $p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \mathbf{w}) = p(D_N|\mathbf{w})$, which is the conditional pdf of the data given the parameter vector \mathbf{w} ; the prior pdf $p(\mathbf{w})$, which contains prior knowledge on \mathbf{w} before any data has been utilised; and the evidence pdf $p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N)$, which is a normalising term in (2.17). The most common approach to parameter estimation is to find the parameter vector \mathbf{w} that maximises this *a posteriori* pdf, i.e. the maximum *a posteriori* (MAP) estimate. Returning now to the classical regression problem of estimating the parameter vector \mathbf{w} of $f(\mathbf{x}(t), \mathbf{w})$ by making measurements of the function for any $\mathbf{x}(t)$:

$$y(t) = f(\mathbf{x}(t), \mathbf{w}) + e(t), \quad (2.18)$$

where the error $e(t)$ is independent of $\mathbf{x}(t)$, and distributed with known pdf $p_e(e)$. Based on the observation data $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$, the log-likelihood is given by

$$\log p(D_N | \mathbf{w}) = \sum_{t=1}^N \ln \log p_e[y(t) - f(\mathbf{x}(t), \mathbf{w})]. \quad (2.19)$$

Assuming that the error is normally distributed with zero mean and variance σ^2 , then (2.19) becomes

$$p(D_N | \mathbf{w}) = -\frac{1}{2\sigma^2} \sum_{t=1}^N [y(t) - f(\mathbf{x}(t), \mathbf{w})]^2 - N \ln(\sqrt{2\pi}\sigma). \quad (2.20)$$

Maximising this likelihood is equivalent to minimising the empirical loss function described in (2.6) with respect to \mathbf{w} :

$$V_N(\mathbf{w}, D_N) = \frac{1}{N} \sum_{t=1}^N [y(t) - f(\mathbf{x}(t), \mathbf{w})]^2, \quad (2.21)$$

leading to the celebrated maximum likelihood (ML) estimate. This cost function is clearly a MSE criterion, leading to an estimate with the variance of the assumed additive model noise $e(t)$. If the model output is a linear function of the parameters \mathbf{w} (as in the networks of B-splines, RBF's, etc.), then $V_N(\cdot)$ is a quadratic cost function which can be minimised by standard linear optimisation techniques. Various learning laws for parameter estimation will be discussed in detail in Chapter 3. The main weaknesses of the ML estimation are: (i) Priors are ignored, and when there is mismatch between the model and the data this can lead to poor generalisation; (ii) Minimisation is carried out over just one training data set D_N , whilst the aim is to capture the behaviour of all training data sets $\{D_N\}_{N=1}^\infty$. However the ML procedure is a very successful approach to model parameter estimation. The problems encountered in data-based modelling are usually due to incorrect model structure, finite data sets, failure to incorporate priors, etc. Some of these issues can be better understood by considering the bias-variance dilemma [82] discussed together with its solution in the next section.

2.5 Model quality

2.5.1 The bias–variance dilemma

Assume that an optimal least squares estimate, $f_o(\mathbf{x}(t))$, of the model $f(\mathbf{x}(t); \mathbf{w}, p)$ exists, where p is the number of degrees of freedom of the model (frequently the dimension of \mathbf{w}). For a finite data set D_N , the objective of identification is to find a model $\hat{y}(\mathbf{x}(t), D_N)$ such that the expected error (or ensemble MSE)

$$E_{D_N}[(\hat{y}(\mathbf{x}(t), D_N) - f_o(\mathbf{x}(t)))^2] \quad (2.22)$$

across all data sets of size N is minimised.

Unfortunately this cannot be directly evaluated, but can be expanded to give error terms which help to direct model parameterisation:

$$\begin{aligned}
& E_{D_N}[(\hat{y}(\mathbf{x}) - f_o(\mathbf{x}))^2] \\
&= E_{D_N}[((\hat{y}(\mathbf{x}) - E_{D_N}[\hat{y}(\mathbf{x})]) + (E_{D_N}[\hat{y}(\mathbf{x})] - f_o(\mathbf{x})))^2] \\
&= E_{D_N}[(\hat{y}(\mathbf{x}) - E_{D_N}[\hat{y}(\mathbf{x})])^2 + (E_{D_N}[\hat{y}(\mathbf{x})] - f_o(\mathbf{x}))^2 \\
&\quad + 2(\hat{y}(\mathbf{x}) - E_{D_N}[\hat{y}(\mathbf{x})]) \cdot (E_{D_N}[\hat{y}(\mathbf{x})] - f_o(\mathbf{x}))] \\
&= E_{D_N}[(\hat{y}(\mathbf{x}) - E_{D_N}[\hat{y}(\mathbf{x})])^2] + E_{D_N}[(E_{D_N}[\hat{y}(\mathbf{x})] - f_o(\mathbf{x}))^2] \\
&\quad + 2E_{D_N}[(\hat{y}(\mathbf{x}) - E_{D_N}[\hat{y}(\mathbf{x})]) \cdot (E_{D_N}[\hat{y}(\mathbf{x})] - f_o(\mathbf{x}))]. \tag{2.23}
\end{aligned}$$

The last term is zero as $E_{D_N}[\hat{y}(\mathbf{x}) - E_{D_N}[\hat{y}(\mathbf{x})]]$ is zero, hence we have

$$\begin{aligned}
& E_{D_N}[(\hat{y}(\mathbf{x}) - f_o(\mathbf{x}))^2] \\
&= E_{D_N}[(\hat{y}(\mathbf{x}) - E_{D_N}[\hat{y}(\mathbf{x})])^2] + (E_{D_N}[\hat{y}(\mathbf{x})] - f_o(\mathbf{x}))^2, \tag{2.24}
\end{aligned}$$

where the first term represents the model variance, and the second term the model bias, which is the expected error between the model and the true system. The expected error between the model and the data is the above plus the variance of the additive noise σ^2 . In (2.24) there are two terms:

- *Bias*, which represents how the average model within a given structure differs from the true system $f_o(\cdot)$. As $N \rightarrow \infty$, (2.24) only involves the bias term, and the consequent estimate $\hat{\mathbf{w}}^*$ represents the best approximation to the true system for given model structure and size. If it converges to the true system as $N \rightarrow \infty$, then the model is unbiased or well matched to the underlying system (a highly desirable attribute of any identification algorithm).
- *Variance*, which represents how sensitive the derived model is to different data sets, or simply the covariance between the parameter estimate $\hat{\mathbf{w}}$ and the optimal estimate $\hat{\mathbf{w}}^*$, i.e. $E_{D_N}[(f(\mathbf{x}, \hat{\mathbf{w}}) - f(\mathbf{x}, \hat{\mathbf{w}}^*))^2]$, which approximately equals $\sigma^2(\frac{p}{N})$ [82].

The above model variance approximation illustrates the bias–variance trade-off. Whilst increasing the number of parameters, p , may give better model flexibility which reduces the model bias, the model variance is increased by σ^2/N (when N is finite) for every additional parameter, leading to overfitting through redundant parameters. By reducing the number of these redundant parameters such that the variance is reduced with a small bias, such that $E_{D_N}[(\hat{y}(\mathbf{x}) - f_o(\mathbf{x}))^2]$ is minimised. This is the so-called principle of model parsimony: the model containing the least degrees of freedom, which matches the true system or data, is the best model. We adopt this philosophy through the algorithms developed in this book.

2.5.2 Bias–variance balance by model structure regularisation

There are essentially two methods of controlling the bias–variance contributions to model performance: model structural regularisation and adaptation.

Model structure, i.e. the position, number, and order (shape) of basis functions, can be adapted off-line via a series of so-called model construction algorithms, such as MARS, CART, ASMOD (see Chapter 5) to reduce the number of degrees of freedom whilst simultaneously minimising say the MSE, or to find an appropriate balance between model bias and variance.

A natural way to approach the problem of minimising (2.23) with respect to model ‘order’ p is to try a sequence of models for increasing p , and to select a suitable model by testing the models on a validation data set. An alternative is to add a penalty term to the MSE of (2.21), which relates to for example the amplitude and number of parameters \mathbf{w} . This technique of regularisation [85] overcomes the prime weakness of the ML estimation in that it encodes prior knowledge about the model/network parameters (weights) before the data is utilised.

By defining the prior distribution as a Gaussian $p(\mathbf{w}) = \exp[-\alpha V_{\mathbf{w}}(\mathbf{w})]/Z_{\mathbf{w}}$, where $Z_{\mathbf{w}}$ is a normalisation constant, then the MAP estimate becomes the parameter vector that minimises the amended or regularised cost:

$$\begin{aligned} V_R(\mathbf{w}) &= \frac{\beta N}{2} V_N(\mathbf{w}, D_N) + \alpha V_{\mathbf{w}}(\mathbf{w}) \\ &= \frac{\beta}{2} \sum_{t=1}^N [y(t) - \hat{y}(\mathbf{x}(t))]^2 + \alpha V_{\mathbf{w}}(\mathbf{w}). \end{aligned} \quad (2.25)$$

The ratio α/β is known as the regulariser coefficient, controlling the bias-variance trade-off. Large α leads to a small (p) model structure, with small variance and large bias (underfit), whereas small α leads to a large model structure with large variance (overfit) and small bias.

Common priors in regularisation include *pdf*’s which lead to small weights through

$$V_{\mathbf{w}}(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T K \mathbf{w}, \quad (2.26)$$

and to small model output covariance or model smoothness by the second order condition

$$V_{\mathbf{w}}(\mathbf{w}) = E\left[\left|\frac{d^2 \hat{y}(t)}{d\mathbf{x}^2}\right|\right], \quad (2.27)$$

where K is a prior matrix with associate dimension. Substituting (2.26) into (2.25) gives

$$V_R(\mathbf{w}) = \frac{N\beta}{2} V_N(\mathbf{w}, D_N) + \frac{\alpha}{2} \mathbf{w}^T K \mathbf{w}. \quad (2.28)$$

Selecting a nonlinear model which is linear-in-the-parameters \mathbf{w} (see (2.16)): $\hat{y} = \psi^T(\mathbf{x}(t))\mathbf{w}$. Substituting this model and (2.21) into (2.28) gives after simplification:

$$V_R(\mathbf{w}) = \frac{N\beta}{2} [\mathbf{w}^T R \mathbf{w} - 2\mathbf{w}^T \mathbf{m} + \frac{\mathbf{y}^T \mathbf{y}}{N}] + \frac{\alpha}{2} \mathbf{w}^T K \mathbf{w}, \quad (2.29)$$

where $R = \frac{1}{N} \mathbf{A}^T \mathbf{A}$ is an autocorrelation matrix of the transformed input vector $\psi(\cdot)$, $\mathbf{m} = \frac{1}{N} \mathbf{A}^T \mathbf{y}$ is a cross-correlation vector, $\mathbf{y} = [y(1) \dots y(N)]^T$ is the output vector, and $\mathbf{A} = [\psi(\mathbf{x}(1)) \dots \psi(\mathbf{x}(N))]^T$. As (2.29) is quadratic in \mathbf{w} , differentiating with respect to the parameter vector \mathbf{w} and setting to zero gives the optimal regularised parameter vector

$$\mathbf{w}_R^* = \beta[\beta\mathbf{A}^T \mathbf{A} + \alpha K]^{-1} \mathbf{A}^T \mathbf{y}. \quad (2.30)$$

Regularisation offers several advantages:

- (i) Zero order regularisation ($K = I$ in (2.30)) leads to parameter estimation in which smaller weights/parameters are more likely than larger ones.
- (ii) Second order regularisation (via (2.27)) leads to smoother outputs, leading to improved generalisation for noisy or sparse data sets.
- (iii) For a quadratic regulariser, the resulting prior distribution is a multivariate normal distribution with covariance matrix $[\alpha K]^{-1}$, which may be selected to produce the desired prior.
- (iv) The condition number of the unregularised network is $C[R]$, whereas for the quadratic regularised network it is $C[\beta NR + \alpha K]$, leading to improved network conditioning, where the condition number is defined by $C[R] \equiv \max\{\lambda_i(R)\}/\min\{\lambda_i(R)\}$, for $\lambda_i(R) \neq 0$, the eigenvalues of R .
- (v) It has been shown [35] that all linear-in-the-parameters models can be expressed as a linear smoother, defined by

$$\hat{\mathbf{y}} = S\mathbf{y}, \quad (2.31)$$

where S is a smoother matrix given by

$$S = \mathbf{A} N^{-1} R^{-1} \mathbf{A}^T = \mathbf{A} [\mathbf{A}^T \mathbf{A}]^{-1} \mathbf{A}^T = U \Lambda [\Lambda^T \Lambda]^{-1} \Lambda^T U^T, \quad (2.32)$$

for \mathbf{A} whose singular decomposition is UAV^T , for U and V orthonormal matrices whose columns, respectively, represent the eigenvectors of $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$, and Λ is $N \times p$ matrix, whose diagonal is the vector of the eigenvalues of \mathbf{A} . The number of degrees of freedom of a model is a measure of the number of independent variables (\mathbf{w}) required to fit the data. For a linear smoother this is defined as the trace of S . If r is the rank of \mathbf{A} , then from (2.32), S has r non-zero eigenvalues, which are all unity. Hence the number of degrees of freedom of a linear-in-the-parameters model (see (2.16)) is given by the rank of \mathbf{A} . Regularisation changes the form of the smoother matrix, and consequently the model's number of degrees of freedom. For the quadratic regularised model (2.29), the resultant linear amended smoother matrix is

$$S_{\alpha, \beta} = \beta \mathbf{A} [\beta NR + \alpha K]^{-1} \mathbf{A}^T. \quad (2.33)$$

Defining $H = [\beta NR + \alpha K]$, which is the Hessian of the cost function, the number of degrees of freedom (*dof*) of the regularised network is

$$dof = Ntr(RH^{-1}) = Ntr(R[\beta NR + \alpha K]^{-1}). \quad (2.34)$$

Clearly the hyperparameters α and β control the number of degrees of freedom for fixed data length N , and autocorrelation matrix R , which in turn

depends on the basis functions $\psi(\cdot)$ and network structure. For zero-order regularisation, $K = \alpha I$, then (2.34) simplifies to [24]

$$dof = \sum_{i=1}^p \frac{\lambda_i}{\lambda_i + \alpha}, \quad (2.35)$$

where $\lambda_i = \lambda_i[\beta \mathbf{A}^T \mathbf{A}]$. When λ_i is small in comparison to α , the likelihood function gives little confidence to this direction in the parameter space, preferring the prior, whereas when λ_i is larger than α , the data can identify the associated direction in the parameter space with high confidence.

Note that the degrees of freedom of a model also affect the unbiased estimate of the noise variance in the data. This unbiased estimate of variance is given by

$$\hat{\sigma}^2 = \sum_{t=1}^N \frac{[y(t) - \hat{y}(\mathbf{x}(t), \mathbf{w}_R^*)]^2}{N - p} = \frac{N \times MSE}{N - p}. \quad (2.36)$$

A fundamental question arises in regularisation, how are the hyperparameters (α, β) to be evaluated? If we are only interested in obtaining the weight or parameter vector that minimises the cost function, then the single smoothing regularisation coefficient $\lambda = \frac{\alpha}{\beta N}$ suffices. However, (α, β) can give insight to an estimate of the noise variance as well as weight control via α .

An illustrative example. Consider the problem of fitting a curve through noisy data produced from the following function:

$$y = \exp(2x - 1) \sin(20(x - 0.6)^2) + \mathcal{N}(0, 0.09).$$

where $\mathcal{N}(\mu, \sigma^2)$ represents noise drawn from a normal distribution with mean μ and variance σ^2 . A data set of 50 input–output pairs was produced, and a cubic B-spline (see Chapter 4) was used to fit the data. The ML estimates produced for three cubic B-spline approximations with 4, 11 and 43 basis functions evenly distributed across $x \in [0, 1]$ are shown in Figures 2.3(a)–(c), respectively. The solid lines show the fits produced from the various models using maximum likelihood estimation, while the broken line shows the noise-free function which produced the data, i.e. $E[y|\mathbf{x}]$.

It is clear that the simple model with only four basis functions has a large bias but small variance; the model is not flexible enough to reproduce the function. Conversely, the model consisting of 43 basis functions is associated with high variance of this model, as demonstrated by the fit to the noise. The model is too flexible and hence too sensitive to the individual data sets. A model that generalises well, producing a good compromise between bias and variance, is one that consists of eleven basis functions, Figure 2.3(b). As a demonstration of regularisation, the fit shown in Figure 2.3(d) has been produced by using a penalty function to a model consisting of 43 basis functions. The effect of the regularisation term that penalises the high output curvature on the model generalisation is obvious.

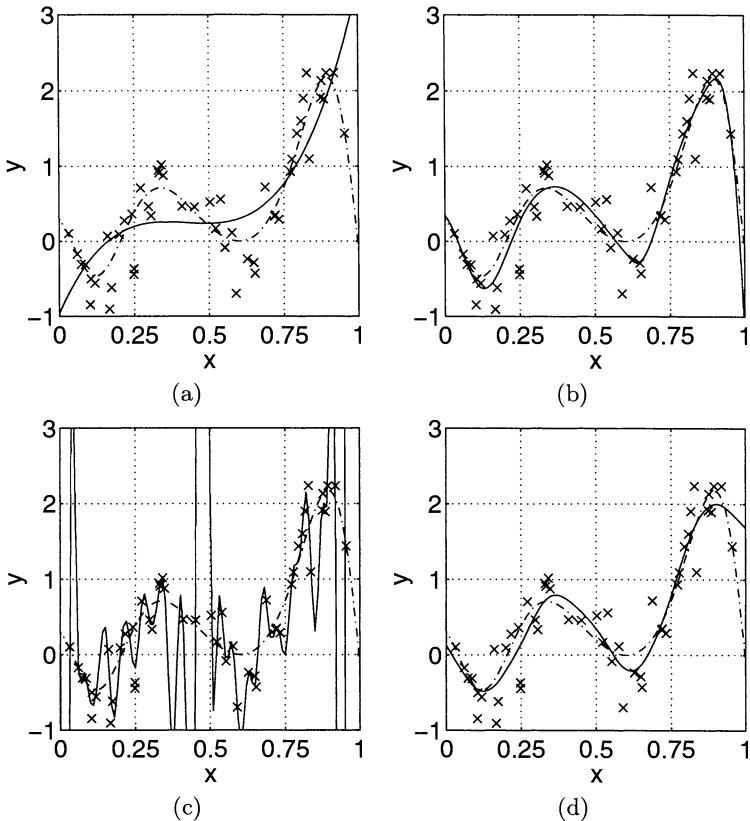


Fig. 2.3. An example demonstrating the bias–variance dilemma. The figures (a), (b) and (c) are from B-spline models with four, 11 and 43 basis functions, respectively. Maximum likelihood estimation is used to identify these models. (d) is the result of applying regularisation to the B-spline model with 43 basis functions. In all these figures, the *solid line* shows the fit produced by the model and the *dashed line* represents the true function

2.6 Reproducing kernels and regularisation networks

In Section 2.5 we considered the problem of model ill-posedness (for finite data sets D_N , i.e. data sparsity), network or model smoothness, and unconstrained weights/parameters by introducing a constrained quadratic cost functional

$$V_R(\mathbf{w}) = \frac{1}{N} \sum_{t=1}^N [y(t) - f(\mathbf{x}(t), \mathbf{w})]^2 + \lambda \mathbf{w}^T K \mathbf{w}. \quad (2.37)$$

By slightly generalising this formulation to

$$V_R(f) = \frac{1}{N} \sum_{t=1}^N [y(t) - f(\mathbf{x}(t))]^2 + \lambda \|f\|_k^2, \quad f \in \mathbf{H}, \quad (2.38)$$

where \mathbf{H} is a reproducing kernel Hilbert space (RKHS), then all networks in this book can be interpreted as special cases of regularisation networks with natural links to recent research into support vector neural networks [62]. A RKHS is a Hilbert space [194] \mathbf{H} defined on some bounded domain $\mathcal{X} \in \Re^n$ with the property that for each $\mathbf{x} \in \mathcal{X}$, the evaluation functionals $f(\mathbf{x})$ are linear and bounded. To every RKHS \mathbf{H} there corresponds a positive definite reproducing kernel function $k(\mathbf{x}(t_2), \mathbf{x}(t_1))$ of two variables in \mathcal{X} , with the reproducing property

$$f(\mathbf{x}) = \langle f(\mathbf{x}(t_2)), k(\mathbf{x}(t_2), \mathbf{x}) \rangle_{\mathbf{H}}, \quad \forall f \in \mathbf{H}, \quad (2.39)$$

where $\langle \cdot, \cdot \rangle_{\mathbf{H}}$ is a scalar product in Hilbert space. Suppose that we have a set of linearly independent (not necessarily orthonormal) functions $\psi_i(\mathbf{x})$ and a sequence of positive numbers α_i , such that they define a bounded function

$$k(\mathbf{x}(t_2), \mathbf{x}(t_1)) = \sum_{i=0}^{\infty} \alpha_i \psi_i(\mathbf{x}(t_1)) \psi_i(\mathbf{x}(t_2)). \quad (2.40)$$

Take the Hilbert space \mathbf{H} to be the set of additive functions (not necessarily infinite sums) of the form:

$$f(\mathbf{x}) = \sum_i w_i \psi_i(\mathbf{x}), \quad (2.41)$$

for any $w_i \in \Re$, then the scalar product is defined by

$$\langle \sum_i w_i \psi_i(\mathbf{x}(t_1)), \sum_j w'_j \psi_j(\mathbf{x}(t_2)) \rangle_{\mathbf{H}} \equiv \sum_i \frac{w_i w'_i}{\alpha_i}. \quad (2.42)$$

Clearly if $f(\cdot)$ is bounded, then such a Hilbert space \mathbf{H} is a RKHS since

$$\begin{aligned} \langle f(\mathbf{x}(t_2)), k(\mathbf{x}(t_2), \mathbf{x}) \rangle_{\mathbf{H}} &= \sum_i \frac{w_i \alpha_i}{\alpha_i} \psi_i(\mathbf{x}) \\ &= \sum_i w_i \psi_i(\mathbf{x}) = f(\mathbf{x}). \end{aligned} \quad (2.43)$$

For the more usual case when there are a finite number of basis functions $\psi_i(\mathbf{x})$, α_i can be arbitrary finite numbers and may be set equal to unity.

Since convergence of the linear-in-the-parameters expansion (2.41) is guaranteed [194], $\psi_i(\mathbf{x})$ defines a space of functions of dimension p , spanned by $\psi_i(\mathbf{x})$ which are in turn basis functions. (2.42) shows that the norm of the RKHS is of the form

$$\|f\|_k^2 = \sum_{i=1}^{\infty} \frac{w_i^2}{\alpha_i}, \quad (2.44)$$

which is of course the same as the regularisation constraint of (2.26). It can be easily seen that the functional that minimises (2.38) has the form

$$f(\mathbf{x}) = \sum_{i=1}^N w_i k(\mathbf{x}, \mathbf{x}(i)) = \mathbf{w}^T \psi(\mathbf{x}), \quad (2.45)$$

where the coefficients w_i 's depend on the data $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$, and are given by

$$\mathbf{w} = (\mathbf{A} + \lambda I)^{-1} \mathbf{y} \quad \text{or} \quad f(\mathbf{x}) = [(\mathbf{A} + \lambda I)^{-1} \mathbf{y}]^T \psi(\mathbf{x}), \quad (2.46)$$

where $\mathbf{y} \in \Re^N$ is the output vector, $\mathbf{w} = [w_1, \dots, w_N]^T \in \Re^N$, the parameter vector, $\mathbf{A} = \{k(\mathbf{x}(i), \mathbf{x}(j))\} \in \Re^{N \times N}$, and λ is a very small positive regularisation coefficient.

The additive linear-in-the-parameters model (2.45) represents a network with a single hidden layer, changing the kernel leads to various regularisation networks, some of which addressed in this book are given in Table 2.1.

Of particular importance in this book is the inherent ability of RKHS to produce tensor products and additive sums of several RKHS in terms of a reproducing kernel [7]. For example in the sequel we use

- *Tensor product splines* for generating multivariate basis functions. In this case the kernel is of the form

$$k(\mathbf{x}(t_1), \mathbf{x}(t_2)) = \prod_{j=1}^n k(x_j(t_1), x_j(t_2)), \quad (2.47)$$

where $k(x_j(t_1), x_j(t_2))$ are positive definite functions, so that

$$f(\mathbf{x}) = \sum_{i=1}^N w_i \left(\prod_{j=1}^n k(x_j, x_j(i)) \right), \quad (2.48)$$

- *Additive splines*, in which the multivariate kernel is formed by

$$k(\mathbf{x}(t_1), \mathbf{x}(t_2)) = \sum_{j=1}^n k(x_j(t_1), x_j(t_2)), \quad (2.49)$$

so that

Table 2.1. Regularisation networks

Regularisation networks	Network characteristics	Kernel functions
Polynomial	Global	$(1 + \langle \mathbf{x}(t_1), \mathbf{x}(t_2) \rangle)^p$
Multilayer perceptron (MLP)	Semi-global	$\tanh(\langle \mathbf{x}(t_1), \mathbf{x}(t_2) \rangle - \alpha)$
Gaussian radial basis function (RBF)	Radial	$\exp(-\frac{\ \mathbf{x}(t_1) - \mathbf{x}(t_2)\ ^2}{2\sigma^2})$
B-splines ^a	Local	$N_j(\mathbf{x}) = \prod_i^n B_{i,k_i}(\mathbf{x})$ $\sum_j N_j(\cdot) = 1$
Bézier–Bernstein ^b polynomials	Local	$B_{i,j,k}^{(d)}(\mathbf{x}) = \frac{d!}{i!j!k!} u^i v^j (1-u-v)^k$ $\sum_{i+j+k=d} B_{i,j,k}^{(d)}(\mathbf{x})(\mathbf{x}) = 1, \mathbf{x} \in \Re^2$ $0 < u, v, (1-u-v) < 1$ $\mathbf{x} \rightarrow \{u, v\}$

^a For formal definitions, see (4.8) and (4.40).

^b For formal definitions, see (7.20).

$$\begin{aligned}
 f(\mathbf{x}) &= \sum_{i=1}^N w_i \left(\sum_{j=1}^n k(x_j, x_j(i)) \right) \\
 &= \sum_{j=1}^n \left(\sum_{i=1}^N w_i k(x_j, x_j(i)) \right) \\
 &= \sum_{j=1}^n f_j(x_j).
 \end{aligned} \tag{2.50}$$

That is an additive decomposition into univariate functions.

– *Hybrid tensor/additive splines*, in which the kernel is of the form:

$$k(\mathbf{x}(t_1), \mathbf{x}(t_2)) = \sum_{l=1}^p \prod_{j=1}^n k_l(x_j(t_1), x_j(t_2)), \tag{2.51}$$

$$\begin{aligned}
 f(\mathbf{x}) &= \sum_{i=1}^N w_i \left(\sum_{l=1}^p \prod_{j=1}^n k_l(x_j, x_j(i)) \right) \\
 &= \sum_{l=1}^p \left(\sum_{i=1}^N w_i \prod_{j=1}^n k_l(x_j, x_j(i)) \right).
 \end{aligned} \tag{2.52}$$

Note: In Section 2.5 we considered the maximum *a posteriori* (MAP) estimate via Bayes rule. If the noise is normally distributed with variance σ^2 ,

then the conditional probability of the data $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$, given f , is $p(D_N|f) \propto \exp(-\frac{1}{2\sigma^2} \sum_{t=1}^N [y(t) - f(\mathbf{x}(t))]^2)$. Assuming that the prior probability $p(f) \propto \exp(-\lambda \|f\|_k^2)$, then by Bayes theorem the *a posteriori* probability of f is $p(f|D_N) \propto \exp(-[\frac{1}{2\sigma^2} \sum_{t=1}^N [y(t) - f(\mathbf{x}(t))]^2 + \lambda \|f\|_k^2])$. Therefore, the MAP estimate of f is the minimiser of the functional $\sum_{t=1}^N [y(t) - f(\mathbf{x}(t))]^2 + \lambda \|f\|_k^2$, so we can clearly see that in the context of regulariser networks, the stabilizer is a prior on the regressor function f [55].

2.7 Model selection methods

Clearly there are a wide variety of different possible models that can be used to fit the data D_N , as there are various cost functions $V_N(\mathbf{w})$ that can be used to evaluate the model and to estimate the unknown model parameters \mathbf{w} . Even when the models are limited to the class of additive, linear-in-the-parameters, and the cost function is the MSE, the expected value of $V_N(\mathbf{w})$, $E[V_N(\mathbf{w})]$, depends upon the actual structure of the model chosen (e.g. centres and width parameters of RBFs, or knot position and order of B-splines), and the quantity and statistical properties of the data points. The choice of model is naturally application dependent, however the model selection process must resolve the trade-off between the model bias and variance. Unfortunately the resulting error contributions cannot be directly measured. In empirical modelling, the resulting model is highly dependent on the quality and quantity of the training and validation data. Therefore model selection criteria have been derived based on limited empirical data, most being based on hypothesis testing concepts [137]. As all the models in this book are linear-in-the-parameters, we need only consider those model (order) selection techniques. Before reviewing the various methods an overview of hypothesis testing is given.

Consider a model with parameter vector \mathbf{w} , composed of two disjoint subvectors such that $\mathbf{w} = \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \end{bmatrix}$, where $\mathbf{w}_0 \in R^s$, $\mathbf{w}_1 \in R^{p-s}$, for $\mathbf{w} \in R^p$. Two hypotheses are: $H_0 : \mathbf{w}_0 = \mathbf{0}$ (null hypothesis) and $H_1 : \mathbf{w}_0 \neq \mathbf{0}$ (alternative hypothesis), where $\mathbf{0}$ is a zero vector, as the null hypothesis represents a reduced model with s parameters equal to zero. The aim of hypothesis testing is to determine if the data supports the larger model. Associated with any statistical test there are two possible errors: rejecting H_0 when $\mathbf{w}_0 = \mathbf{0}$ and rejecting H_1 when $\mathbf{w}_0 \neq \mathbf{0}$, with probabilities α and β , respectively. The probability α is known as the test significance level (usually set by the modeller). There is a trade-off between α and β : as α decreases, β increases. A common statistic for restricted model comparison is the log determinate ratio (LDR) test

$$N \log \left\{ \frac{V_N(\hat{\mathbf{w}}_0)}{V_N(\hat{\mathbf{w}}_1)} \right\}, \quad (2.53)$$

which is asymptotically equivalent to the classical statistical F-test, and converges to a $\chi^2(s)$ distribution, where s is the number of degrees of freedom of this distribution, as $N \rightarrow \infty$ [134]. From this, given α , the acceptance of the null hypothesis may be determined. Above we have only considered two models, one of which is a subset of the other, this can be generalised to criteria which adhere to the principle of model parsimony, i.e. finding the balance between model complexity (bias) and model accuracy (variance).

2.7.1 Model selection criteria

(i) Hypothesis Testing

The above criteria can be extended to multiple competing models by considering a cost function that measures the ability of the model to fit the data set, e.g. $(N \log(V_N(\hat{\mathbf{w}})))$, penalised by a complexity term dependent on p . A natural criteria would be

$$V_m = N \log(V_N(\hat{\mathbf{w}})) + pK(1), \quad (2.54)$$

where $K(1)$ defines a parameter dependent on the significance level α , which is defined by the probability of rejecting a smaller correct model, when comparing models whose degree of freedom differs by 1. The above concept has been around some while in the context of model order detection utilising information measures [171].

(ii) Information Measures

Akaike [1] used the Kullback–Leibler distance measure to evaluate the distance between the estimated *pdf* of D_N and the true one. The derived Akaike information criterion (AIC) is

$$AIC = N \log(V_N(\hat{\mathbf{w}})) + 2p, \quad (2.55)$$

which is the same as (2.54) with the significance level set by $K(1) = 2$. The AIC criterion tends to overestimate the model's size, due to this small significance level.

$V_N(\hat{\mathbf{w}})$ can be interpreted as an estimate of the noise in the data, and hence may be used for either biased or unbiased estimates. For small data sets, this choice becomes significant, and for an unbiased estimate the AIC criterion needs an additional compensation term $N \log(\frac{N}{N-p})$. Clearly for $N \gg p$, this term disappears. In a related form Akaike also derived the final prediction error (FPE) term given by $FPE = V_N(\hat{\mathbf{w}})(\frac{N+p}{N-p})$ so that

$$N \log(FPE) = N \log(V_N(\hat{\mathbf{w}})) + N \log(\frac{N+p}{N-p}), \quad (2.56)$$

which is asymptotically equivalent to the AIC as $N \rightarrow \infty$.

The amount of data, N , may be limited, as an alternative, the minimum description length (MDL) criterion

$$MDL = \log(V_N(\hat{\mathbf{w}})) + p \frac{\log(N)}{N} \quad (2.57)$$

was introduced to find the model that accurately represents the data with the minimum amount of information. Other similar criteria can be readily derived based on hypothesis testing. For example, a significance level determined by data set size, i.e. $\log(N)$ (see [23] on Bayesian estimation).

(iii) Cross Validation (CV)

Cross validation is a very natural approach to model selection by splitting the available data into two parts: the identification or estimation data set D_{N_1} to find the model parameters, $\hat{\mathbf{w}}_{N_1} = \arg \min\{V_{N_1}(\mathbf{w}, D_{N_1})\}$, and the validation data set, D_{N_2} , for which the criterion $EV_{N_2}(\hat{\mathbf{w}}) = V_{N_2}(\hat{\mathbf{w}}_{N_1}, D_{N_2})$ is evaluated. The procedure tries out a number of model structures and selects the one that minimises $EV_{N_2}(\hat{\mathbf{w}})$.

There are several problems with CV: (i) the data is frequently limited in size and insufficient to populate the input space; (ii) the validation set can guide model selection in such a manner as to fit its own idiosyncrasies; so ideally a new and fully independent validation data set is required for each test; and (iii) it is computationally expensive. A computationally efficient scheme is the generalised cross validation (GCV) criterion [86]

$$GCV = \frac{V_N(\hat{\mathbf{w}})}{(1-p/N)^2}, \\ N \log(GCV) = N \log(V_N(\hat{\mathbf{w}})) - N \log((1-p/N)^2). \quad (2.58)$$

The GCV is asymptotically equivalent to the FPE (see (2.56)).

2.7.2 Model selection criteria sensitivity

The above model selection criteria are asymptotically similar (as $N \rightarrow \infty$), however an important measure as to their efficiency is their sensitivity, defined as the factor improvement in the data fit (or reduction in $V_N(\hat{\mathbf{w}})$) required to justify an extra degree of freedom in the model. The sensitivity of each of the above criteria are shown in Table 2.2, all based on unbiased noise estimation.

To illustrate this, consider a model based on a fixed size data set ($N = 100$). As shown in Figure 2.4, when the model size p increases, the sensitivity of all measures decrease, preventing the model size approaching the data set, which in turn prevents overfitting. However, when biased estimates are used, the metrics AIC, MDL, and hypothesis test remain constant for a given data size (N), irrespective of the model size p . When the model size p is fixed, all the model selection criteria asymptotically approach unity for increasing data

Table 2.2. Model selection criteria sensitivity

Model selection criteria	Model sensitivity
Hypothesis test	$(\frac{N-p-1}{N-p}) \exp(-\frac{4}{N})$
AIC	$(\frac{N-p-1}{N-p}) \exp(-\frac{2}{N})$
FPE	$\frac{(N-p-1)^2(N+p)}{(N+p+1)(N-p)^2}$
MDL	$\frac{(N-p-1)}{(N-p)} \exp(-\frac{\log(N)}{N})$
GCV	$(\frac{N-(p+1)/N}{N-p/N})^3$

size N , as shown in Figure 2.5 with $p = 50$. Generally, as N increases, FPE and GCV approach AIC, with MDL being the most conservative.

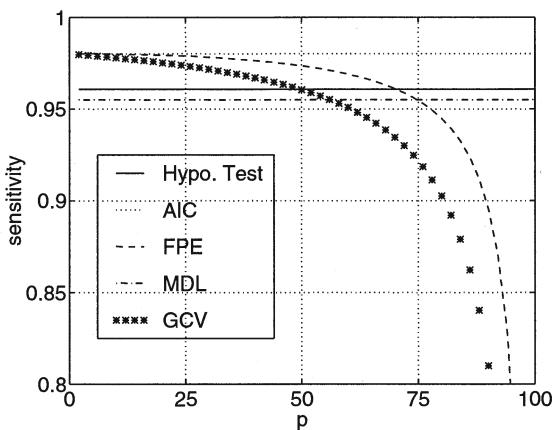


Fig. 2.4. The sensitivity of the various model selection criteria, with a fixed data size set and an increasing model size

2.7.3 Correlation tests

In practice the above hypothesis based methods may lead to inappropriate models, therefore, further validation tests are used to confirm model design. For example, if the models are transparent, then expert qualitative interrogation is useful. However, more traditional statistical tests based on the lack of correlation between model residuals and the data are to be recommended as well. Correlation based validation involves comput-

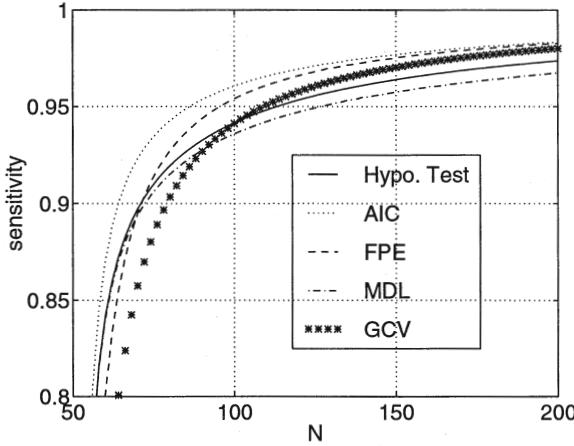


Fig. 2.5. The sensitivity of the various model selection criteria, with a fixed model size ($p = 50$) and an increasing data size

ing correlation functions which are composed of model residuals and system inputs and testing if these satisfy certain conditions given in the form of confidence intervals. The classical approach to validate identified linear models consists of computing the autocorrelation function of the residuals and the cross-correlation function between the residuals and the input [181]. Define a matrix $M(\mathbf{x}^T(t)) = \{\mathbf{m}(t), \mathbf{m}(t-1), \dots, \mathbf{m}(t-t_d)\}$, where $\mathbf{x}^T(t) = \{\mathbf{y}^{t-1}, \mathbf{u}^{t-1}, \mathbf{e}^{t-1}\}^T$ is the observational vector of model inputs, outputs and errors seen up to iteration at time t , and $\mathbf{m}(t)$ is a monomial of elements of the vector $\mathbf{x}^T(t)$, e.g. $\mathbf{m}(t) = \mathbf{y}(t-1)\mathbf{u}^2(t-2)$.

Two hypotheses can be defined: H_0 : $\mathbf{e}(t)$ is uncorrelated with $M(\mathbf{x}^T(t))$; $E[\mathbf{e}(t)M(\cdot)] = 0$; and H_1 : $\mathbf{e}(t)$ is correlated with $M(\mathbf{x}^T(t))$; $E[\mathbf{e}(t)M(\cdot)] \neq 0$. Two different test statistics can be used to ascertain if H_0 holds.

(i) χ^2 Test

In χ^2 test, a reliable statistic is given by [134]:

$$\mathbf{d} = N[\sum_{t=1}^{N-t_d} \mathbf{e}^2(t)]^{-1} [\sum_{t=1}^{N-t_d} M(t)\mathbf{e}(t)] \\ \times [\sum_{t=1}^{N-t_d} \mathbf{m}^T(t)\mathbf{m}(t)]^{-1} [\sum_{t=1}^{N-t_d} \mathbf{m}(t)\mathbf{e}(t)],$$

which is asymptotically a $\chi^2(s)$ distribution if H_0 holds, with s being the number of delays, t_d . For a given acceptance level (typically 95%), H_0 is rejected if elements of \mathbf{d} are outside this acceptance level.

(ii) Model Validity Tests

A series of model validity tests for nonlinear models have been introduced [18, 19, 21]. Model validity tests are general procedures to detect the inadequacy of a fitted model. If the model structure and the estimated parameters

are correct then the residual sequence $e(t)$ should be unpredictable from all linear and nonlinear combinations of past inputs $u(t)$ and outputs $y(t)$. This condition will hold only if the following conditions hold [21]

$$\begin{aligned}
 \rho_{ee}(\tau) &= \frac{\sum_{t=1}^N e(t)e(t-\tau)}{\sum_{t=1}^N e^2(t)} \\
 &= \delta(\tau) \\
 \rho_{ue}(\tau) &= \frac{\sum_{t=1}^N u(t-\tau)e(t)}{\sqrt{\sum_{t=1}^N u^2(t)}\sqrt{\sum_{t=1}^N e^2(t)}} \\
 &= 0, \quad \forall \tau \\
 \rho_{[uu]e}(\tau) &= \frac{\sum_{t=1}^N [u^2(t-\tau) - \overline{u^2(t)}]e(t)}{\sqrt{\sum_{t=1}^N [u^2(t-\tau) - \overline{u^2(t)}]^2}\sqrt{\sum_{t=1}^N e^2(t)}} \\
 &= 0, \quad \forall \tau \\
 \rho_{[uu][ee]}(\tau) &= \frac{\sum_{t=1}^N [u^2(t-\tau) - \overline{u^2(t)}][e^2(t) - \overline{e^2(t)}]}{\sqrt{\sum_{t=1}^N [u^2(t-\tau) - \overline{u^2(t)}]^2}\sqrt{\sum_{t=1}^N [e^2(t) - \overline{e^2(t)}]^2}} \\
 &= 0, \quad \forall \tau \\
 \rho_{e[eu]}(\tau) &= \frac{\sum_{t=1}^N e(t)e(t-1-\tau)u(t-1-\tau)}{\sqrt{\sum_{t=1}^N e^2(t)}\sqrt{\sum_{t=1}^N [e(t)u(t) - \overline{e(t)u(t)}]^2}} \\
 &= 0, \quad \tau \geq 0,
 \end{aligned} \tag{2.59}$$

where δ denotes *Kronecker delta*, and the overbar signifies mean average, $\bullet(t) = \frac{1}{N} \sum_{t=1}^N \bullet(t)$. The system output can be included to form the new higher order correlation tests [19].

$$\begin{aligned}
 \rho_{ee}(\tau) &= \frac{\sum_{t=1}^N e(t)e(t-\tau)}{\sum_{t=1}^N e^2(t)} \\
 &= \delta(\tau) \\
 \rho_{[ye][e^2]}(\tau) &= \frac{\sum_{t=1}^N [y(t)e(t) - \overline{y(t)e(t)}][e^2(t-\tau) - \overline{e^2(t)}]}{\sqrt{\sum_{t=1}^N [y(t)e(t) - \overline{y(t)e(t)}]^2}\sqrt{\sum_{t=1}^N [e^2(t) - \overline{e^2(t)}]^2}} \\
 &= k_1 \delta(\tau) \\
 \rho_{[y\xi][u^2]}(\tau) &= \frac{\sum_{t=1}^N [y(t)e(t) - \overline{y(t)e(t)}][u^2(t-\tau) - \overline{u^2(t)}]}{\sqrt{\sum_{t=1}^N [y(t)u(t) - \overline{y(t)u(t)}]^2}\sqrt{\sum_{t=1}^N [u^2(t) - \overline{u^2(t)}]^2}} \\
 &= 0, \quad \forall \tau,
 \end{aligned} \tag{2.60}$$

where $k_1 > 0$ is a constant. When the above new higher order correlation tests are used for time series, the tests involving system input $u(t)$ are not used.

Considering that in practice only a finite data length will be used in estimation, confidence bands should be used to indicate if the correlation between variables is significant or not. The model will be regarded as adequate if all the tests fall within the 95% confidence limits at approximately $\frac{1.96}{\sqrt{N}}$ of a normal distribution, where N is the number of the samples.

Correlation tests provide an indication of the validity of the fitted model. If validation indicates a poor model the identification should be repeated. However, in iterative modelling, a statistically validated model should be or have been subject to other model selection metrics such as model generalisation, etc. Although it is a general practice that a model which is not invalidated is assumed to be adequate, in nonlinear modelling, a match between the dynamics of the identified model with underlying system is often viewed as the ultimate goal of the modelling process. The qualitative validation of the model dynamics such as comparing model parameters, bifurcation parameters, etc. are generally problem domain specific and especially important for the modelling of chaotic systems [37, 211].

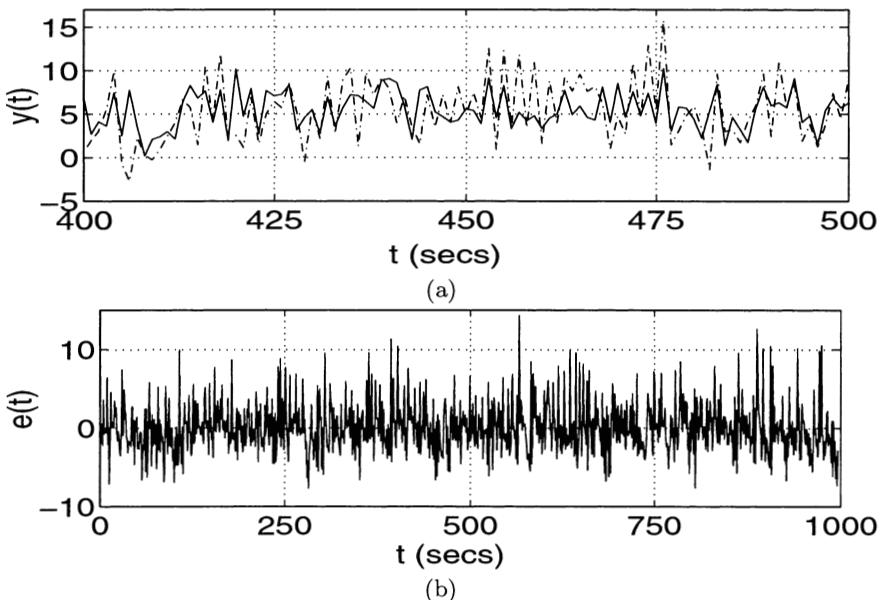


Fig. 2.6. Result of identifying the time series with a linear model; (a) is subsection of the response, *solid line*, across the data set superimposed onto the correct output, *dashed line*, $y(t)$, and (b) is the error between the true response and the model, $e(t)$

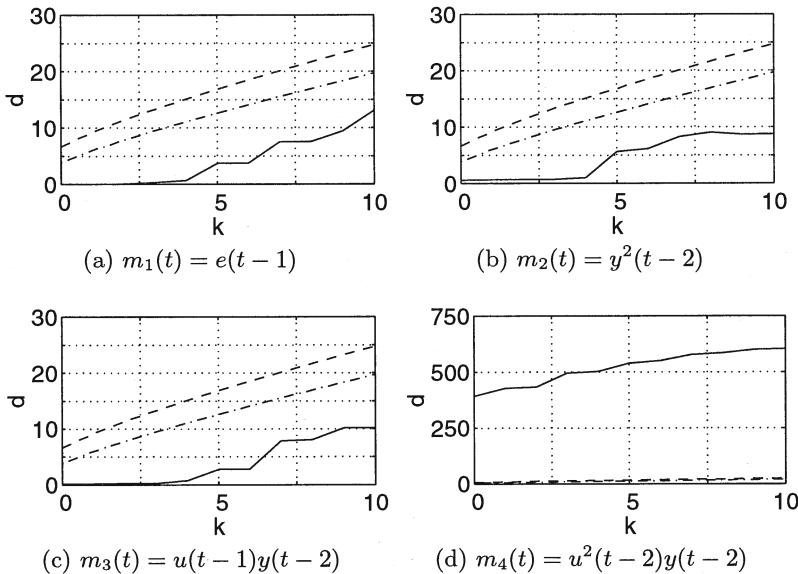


Fig. 2.7. χ^2 correlation tests for the nonlinear model

2.8 An example: time series modelling

To illustrate the modelling issues introduced in this chapter, consider the following simple input/output time series:

$$\begin{aligned} y(t) = & 0.5 + 0.8y(t-2) - 1.5u^2(t-2)y(t-2) \\ & + 0.8(u(t-1)-2)^2 + N(0, 1). \end{aligned} \quad (2.61)$$

The input $u(t)$ is drawn from a uniform distribution over $[-1, 1]$. A set of 1000 data pairs were generated together with another 1000 noise free data pairs for validation.

We consider initially a linear model $\hat{y}(t) = \mathbf{x}^T(t)\mathbf{w}$, where the regressor vector $\mathbf{x}(t) = [1 \ u(t-1), \dots, \ u(t-4), \ y(t-1), \dots, \ y(t-4)]^T$ is selected on the basis of no prior knowledge about the underlying process. The maximum likelihood estimate of \mathbf{w} was found by minimising $V_N(\cdot)$, the model's response is shown in Figure 2.6, which clearly is inadequate.

A polynomial nonlinear model is examined, whose regressor represents all the unique terms $u^p(t - n_u)y^q(t - n_y)$, where $\{n_u, n_y\} \in [0, 4]$, and $\{p, q\} \in [0, 3]$. Figure 2.8 shows the result of the nonlinear model, which is significantly better than the linear model. The MSE error is 0.85, which is below the variance of the additive noise, suggesting that the model is overfitting, using its spare degrees of freedom to model the noise (see Table 2.3). This is overfitted as shown by the high MSE in the validation set. Clearly

regularisation would have controlled the model's redundant degrees of freedom.

For the nonlinear model validation using χ^2 correlation tests, consider four $M(\cdot)$ matrices constructed from the following monomials: $m_1(t) = e(t - 1)$, $m_2(t) = y^2(t - 2)$, $m_3(t) = u(t - 1)y(t - 2)$, $m_4 = u^2(t - 2)y(t - 2)$. For example, $M_3(t) = [m_3(t), \dots, m_3(t - t_d)]^T = [u(t - 1)y(t - 2), \dots, u(t - 1 - t_d)y(t - 2 - t_d)]^T$. Taking $t_d = 10$, and the $\chi^2(s)$ test correlation between these matrices and model residuals, the \mathbf{d} statistic is shown in Figure 2.7. The dotted lines represent the 95% and 99% confidence levels. If \mathbf{d} is above these lines, the model is inadequate ($m_4(t)$ is correlated with the residuals).

Table 2.3. Model data summary

Model	Data set MSE	Test set MSE	p	Model variance	Model bias ²
Linear	10.23	8.99	9	$9.52e - 2$	10.4
Overpara- meterised	0.85	0.26	241	$1.67e - 1$	$9.95e - 1$
MDL	1.01	$5.8e - 3$	6	$5.56e - 3$	$9.95e - 1$
Hypothesis test	1.00	$9.3e - 3$	7	$6.43e - 3$	$9.95e - 1$
AIC, GCV, FPE	0.97	$3.7e - 2$	10	$1.48e - 2$	$9.95e - 1$
Actual time series	1.01	0.00	5	n/a	n/a

This example is now used to evaluate the various model selection criteria of Section 2.7.1. A polynomial model using forward selection was iteratively constructed to minimise each criterion. At each iteration, a single monomial was added to the current model. These monomials were drawn from the above overparameterised model. As the data set is large, MDL performs best with only one extra term than in the actual time series (the MDL model time series prediction is shown in Figure 2.9).

The hypothesis test produced the same model as MDL with an additional term that models the noise in the data (see Table 2.4). As AIC, GCV and FPE are equivalent for N large, all produced the same model with 16 monomial terms. This overfitting is due to the low significance level assigned to these tests (see also Table 2.4).

The MSE of various models should be compared in Table 2.3 with the sample noise variance from the data. Note that the linear model has high bias and low variance, whereas the overparameterised model has large variance and low bias, with the MDL model producing the best balance between model bias and variance.

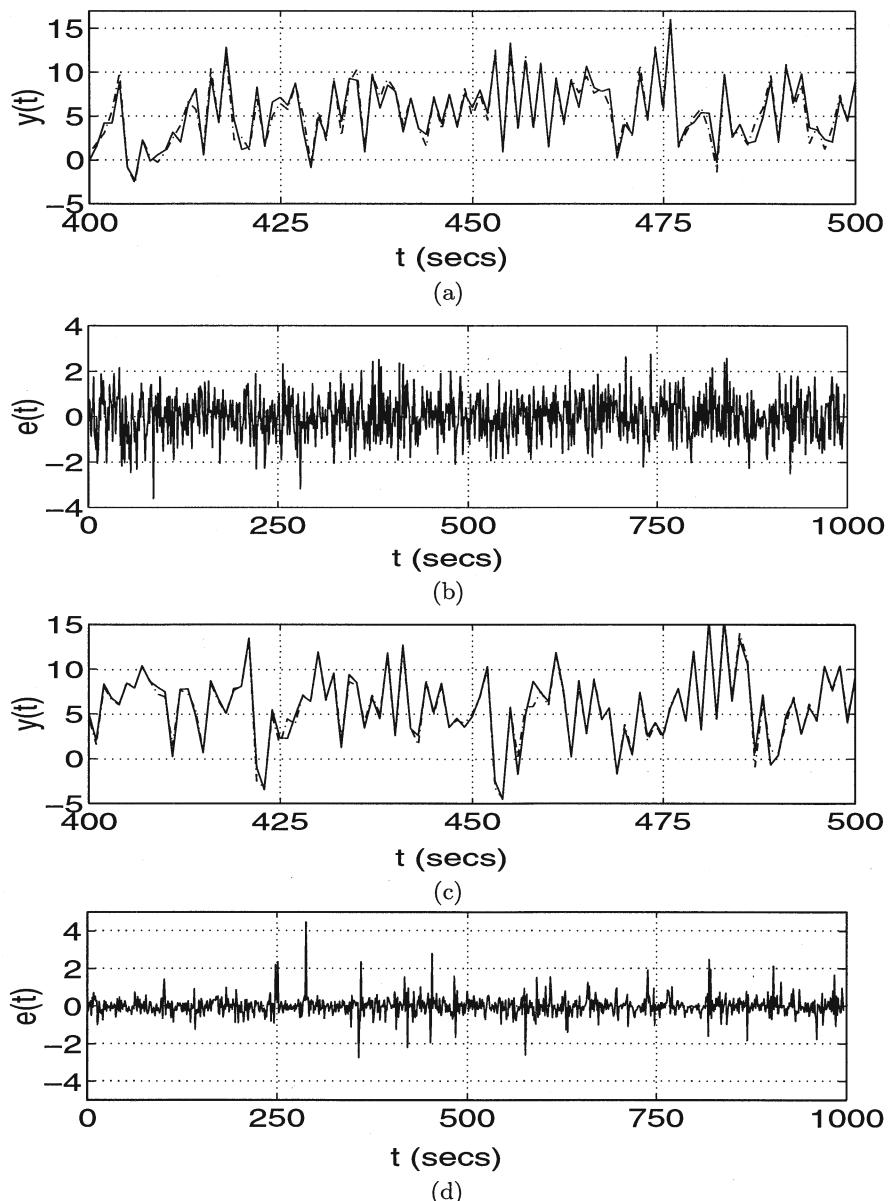


Fig. 2.8. Result of identifying the time series with a nonlinear model; (a) shows the response across a subsection of the data set, (b) is error across this set, while (c) and (d) represent the same for the noise-free validation set

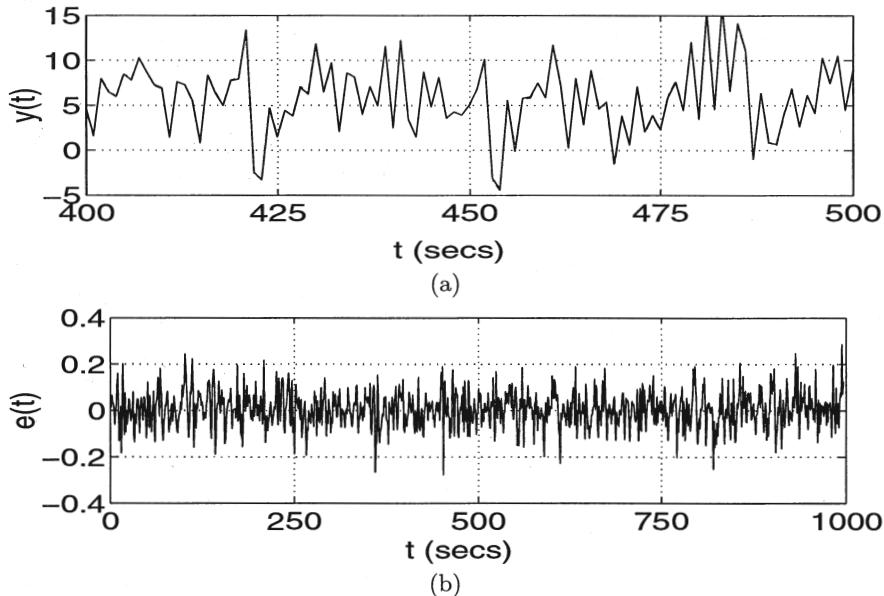


Fig. 2.9. The response of the resulting model constructed by a stepwise regression based on the MDL selection criteria, (a) shows the response across a subsection of the noise-free validation set and (b) shows the error across the complete set. Both the true system response and model response are shown in (a) but due to low errors (as shown in (b)) the discrepancies cannot be seen

Table 2.4. Regressors for various models generated by stepwise regression

Criteria	Regressors $\mathbf{x}(t)$
MDL	$[1, u(t-1), u^2(t-2), u^2(t-1)y(t-2), y(t-2)u^2(t-2)]^T$
Hypothesis Test	$[1, u(t-1), u^2(t-2), u^2(t-1)y(t-2), y(t-2)u^2(t-2), y(t-2)u^3(t-1)]^T$
AIC, GCV, FPE	$[1, u(t-1), u^2(t-1), u^3(t-1)y^2(t-1), u^3(t-1)y^3(t-1), u^3(t-1)y(t-2), u^2(t-2), u^2(t-2)y(t-1), u^2(t-2)y(t-2), u(t-2)y^2(t-4), u^2(t-2)y^2(t-2), u^3(t-3)y(t-2), u^3(t-3)y^2(t-2), u^2(t-4)y(t-1), y(t-2), y^3(t-4)]^T$

3. Learning laws for linear-in-the-parameters networks

“Discovery consists of seeing what everybody has seen, – and thinking what nobody has thought.” – Albert Szent-Gyorgyi

3.1 Introduction to learning

Despite the fact that artificial neural networks (ANNs) have been proposed primarily as nonlinear learning systems, considerable insight into the behaviour of these networks can be gained from linear modelling techniques, for which the literature is vast (see for example [8, 137, 181]), and significantly the resultant theory is directly applicable to a powerful and special class of ANNs, i.e. linear-in-the-parameters networks, which forms the basis of this book. An obvious reason to use ANNs is their ability to approximate arbitrarily well any continuous nonlinear function, with the specific network architecture and parameter/weight adjustment algorithm determining how well learning is achieved. Of particular interest in adaptive control and estimation is the ability of algorithms to model, track and control on-line. However, it is infeasible to assume that the input signals excite the whole of the state space (a prerequisite of identification theory), and so it is necessary to consider the effects of a reduced input signal on overall functional approximation for various network architectures and associated learning laws. Here learning must be local, in that adjustable network weights or parameters should only affect the network’s output locally. In ANNs, the vast majority of supervised learning rules are based on the assumption that the nonlinear network can be locally linearised, so that it is natural to develop on-line learning algorithms with provable learning network stability and convergence conditions, and to choose ANNs which have linear-in-the-parameters with local behavioural characteristics. In particular, nonlinear networks, such as the cerebellum model articulation controller (CMAC), radial basis function (RBF), B-splines (see [32]), Bézier–Bernstein polynomial (see Section 7.4) networks, all have an output layer of linear parameters, with basis functions having local compact support. These so-called linear-in-the-parameters networks generally have large memory requirements for generating the optimal

network parameter or weight vector from batch data, where direct optimisation methods, generally based on matrix inversion with a computational cost of $O(p^3)$ (p represents network size), can be used. However, as the system equations for these networks are generally sparse and singular, then numerically stable matrix inversion techniques that exploit these characteristics should be utilised. Alternatively, iterative techniques that avoid matrix inversion to calculate the optimal weight vector can be derived (see Section 3.3, and for a detailed discussion of learning for linear-in-the-parameters networks see Brown and Harris [32]).

In order that an adaptive network can track an unknown nonlinear time-varying system, it must be able to reorganise itself in real-time via on-line instantaneous learning or training that uses an instantaneous estimate of the current network performance prior to weight updating. This is a significantly different problem to recursive or iterative weight updating from batch data, where iteration is used for computational ease (e.g. to avoid matrix inversion), since in instantaneous learning only the estimate of the instantaneous performance value, gradient, etc. are available, which may well be significantly different from the true value. In this case, a variety of instantaneous least mean squares (LMS) learning rules have been developed for use with on-line modelling and control (see Section 3.4), in which the instantaneous error is used to approximate the total cost function, such as MSE, leading to convergence problem. The problem of using instantaneous gradient estimates introduces an additional noise term into the learning process, causing weight convergence to a domain or region around the optimal weight vector (see Section 3.4.3).

As linear-in-the-parameters networks are used throughout this book, then gradient descent rules are highly appropriate. Also since all linear-in-the-parameters networks map the input regressor vectors to a higher dimensional sparse space, prior to the output weight layer, then instantaneous learning laws can exploit this sparse structure with its inherent localised response, as only weights that contribute locally to the output are updated. Learning is local as dissimilar inputs are mapped to different weight sets, ensuring that instantaneous learning is highly appropriate for weight training for linear-in-the-parameters networks.

Throughout this chapter, various learning laws are developed, which have a common structure, such that the new weight value is the past value plus an output error multiplied by the transformed input regressor vector. Generally performance is based on the network's MSE, as it leads to simple, analytically tractable laws, which give acceptable models, and when subject to additive Gaussian noise it is also the maximum likelihood estimate.

The instantaneous learning laws developed in this chapter can also be considered as line search optimisation techniques [70], since they produce a search direction or path, along which the weights or parameters are updated, and then compute a step size which ensures stable learning, albeit

with limited training data. Throughout this book we utilise instantaneous normalised least squares learning (Section 3.4.2) due to independence of the learning rate, δ , or the basis functions and its inherent stable learning properties compared with other learning laws. The following sections develop the interrelationships between error performance surfaces and various parametric learning laws.

3.2 Error or performance surfaces

A network's output error $e_y(t) = y(t) - \hat{y}(t)$ is available in supervised learning as an instantaneous measure of the current model's performance, and as such is useful for feedback via some learning law derived from a performance function for weight or parameter updating. Learning laws are derived so as to modify the estimate of \mathbf{w} , such that as the amount of data increases, an optimal parameter vector (weight) is derived by globally minimising some prespecified cost function $V_N(\mathbf{w})$ over all available training data D_N , as $\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} [V_N(\mathbf{w}, D_N)]$, where $V_N(\mathbf{w}, D_N)$ can be defined as various cost functions, such as

$$V_N(\mathbf{w}, D_N) = \begin{cases} E[|e_y(t)|] \\ E[e_y^2(t)] \\ \max_t |e_y(t)|. \end{cases} \quad (3.1)$$

Only when a weight vector exactly models a desired function, are the solutions to the various cost functions in (3.1) the same, since each places a different emphasis upon the instantaneous error. The choice of the cost function influences the type of learning law, its computational complexity, its convergence properties, as well as the resultant final model. For general classes of computational models with multiple adaptive layers of adjustable weights, such as multilayer perceptrons (MLPs), the error function and the consequent cost function will be a highly nonlinear function of the weights, for which many local minima exist, satisfying $\nabla V_N(\mathbf{w}, D_N) = 0$, where $\nabla V_N(\mathbf{w}, D_N)$ is the gradient of $V_N(\mathbf{w}, D_N)$.

Here it is not in general possible to find closed form analytical solutions for \mathbf{w} . This problem also applies to linear-in-the-parameters networks, such as RBFs, various neurofuzzy networks, CMAC, etc. in which the basis functions positions, orders, dilation parameters, etc. are simultaneously adjusted with the parameters \mathbf{w} . In this chapter we assume that these model structural properties are fixed or determined off-line by some model construction algorithms (see Chapter 5). In this case, the network's MSE (see (2.6)) produces a (hyper)quadratic surface in p -dimensional weight space with a unique global minimum $\hat{\mathbf{w}}$ (see Figure 3.1 for the 2-D weight space case).

Substituting the following model (2.16)

$$\hat{y}(t) = \psi^T(\mathbf{x}(t))\mathbf{w} \quad (3.2)$$

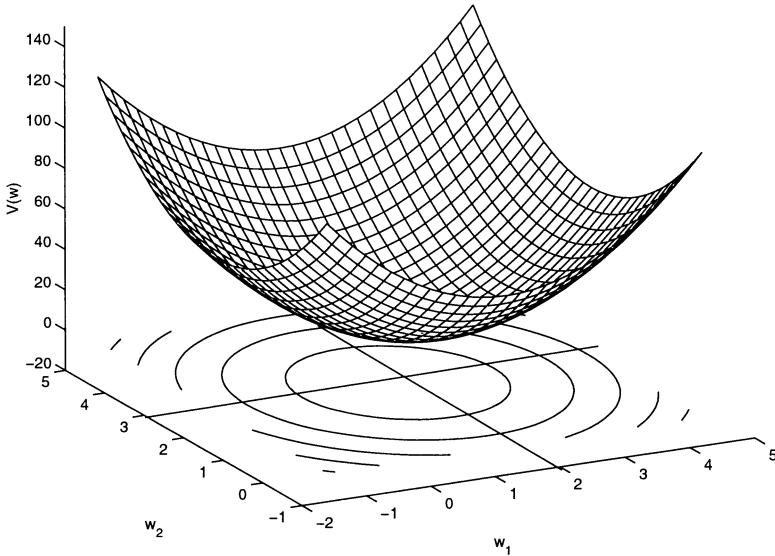


Fig. 3.1. Error surface in a 2-D weight space. The optimal weight occurs at $\mathbf{w} = [2, 3]^T$

into the quadratic cost function

$$V_N(\mathbf{w}, D_N) = \frac{1}{N} \sum_{t=1}^N [y(t) - \hat{y}(t)]^2, \quad (3.3)$$

gives

$$V_N(\mathbf{w}, D_N) = \frac{\mathbf{y}^T \mathbf{y}}{N} + \mathbf{w}^T R \mathbf{w} - 2\mathbf{m}^T \mathbf{w}, \quad (3.4)$$

where R is a $p \times p$ semi-positive definite, symmetric autocorrelation matrix of the basis functions defined by

$$\begin{aligned} R &= E[\psi(t)\psi^T(t)] = \frac{1}{N} \mathbf{A}^T \mathbf{A} \\ &= \frac{1}{N} [\psi(\mathbf{x}(1)) \dots \psi(\mathbf{x}(N))] [\psi(\mathbf{x}(1)) \dots \psi(\mathbf{x}(N))]^T \end{aligned} \quad (3.5)$$

$$= \begin{bmatrix} \frac{1}{N} \sum_{t=1}^N [\psi_1^2(t)] & \frac{1}{N} \sum_{t=1}^N [\psi_1(t)\psi_2(t)] & \dots & \frac{1}{N} \sum_{t=1}^N [\psi_1(t)\psi_p(t)] \\ \frac{1}{N} \sum_{t=1}^N [\psi_2(t)\psi_1(t)] & \frac{1}{N} \sum_{t=1}^N [\psi_2^2(t)] & \dots & \frac{1}{N} \sum_{t=1}^N [\psi_2(t)\psi_p(t)] \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{N} \sum_{t=1}^N [\psi_p(t)\psi_1(t)] & \frac{1}{N} \sum_{t=1}^N [\psi_p(t)\psi_2(t)] & \dots & \frac{1}{N} \sum_{t=1}^N [\psi_p^2(t)] \end{bmatrix},$$

and $\mathbf{m} = \frac{1}{N} \mathbf{A}^T \mathbf{y}$ is a cross-correlation vector.

When R is nonsingular, the optimal weight vector $\hat{\mathbf{w}}$ is given by differentiating (3.4) with respect to \mathbf{w} and setting it to zero, resulting in the set of (so-called normal) linear equations:

$$R\hat{\mathbf{w}} = \mathbf{m}, \quad (3.6)$$

whose solution is given by the pseudo-inverse as

$$\hat{\mathbf{w}} = R^{-1}\mathbf{A}^T\mathbf{y} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}. \quad (3.7)$$

For this least squares (also maximum likelihood if the additive noise process $e(t)$ in (2.4) is a Gaussian process) solution is unique if $\text{rank}(\mathbf{A}) = p$, otherwise it is under-determined with an infinity of solutions. In this situation it is desirable to find the minimum norm solution ($\min \|\mathbf{w}\|_2$), since such weight values mean that the network should generalise sensibly across its input space (see Section 2.4). For many problems the solution matrix \mathbf{A} is sparse, whilst R is not, and hence algorithms that do not explicitly form R can often prove computationally efficient [32].

Fortunately the networks discussed in this book, e.g. B-spline neurofuzzy systems, are naturally sparse due to their inherent structure and this sparsity is preserved in R . Irrespective of what method is used in solving (3.7) the ability to find a solution is dependent upon the condition number, $C(R)$, of the matrix R :

$$C(R) = \frac{\max \text{ nonzero } \lambda_i}{\min \text{ nonzero } \lambda_i}, \quad (3.8)$$

where λ_i are the eigenvalues of R . If the condition number is large (e.g. 10^6), then the problem is ill-conditioned, whilst if it is close to unity, it is well conditioned (see [32] for detailed discussion on conditioning of neural networks).

The minimum MSE, $V_N(\hat{\mathbf{w}}) = V_{\min}$, occurs when $\mathbf{w} = \hat{\mathbf{w}}$ in (3.4). Noting that R is symmetric, we have

$$V_{\min} = \frac{\mathbf{y}^T\mathbf{y}}{N} - \mathbf{m}^T\hat{\mathbf{w}}. \quad (3.9)$$

So defining the current weight error as $\mathbf{e}_w = \hat{\mathbf{w}} - \mathbf{w}$, then the cost (3.4) becomes on simplification via (3.9):

$$V_N(\mathbf{w}) = V_{\min} + \mathbf{e}_w^T R \mathbf{e}_w, \quad (3.10)$$

which defines a quadratic function around $\mathbf{w} = \hat{\mathbf{w}}$ (as R is positive definite). When \mathbf{e}_w lies in the null space of R or is zero, $V_N = V_{\min}$, and as with the network condition number, $C(R)$, the autocorrelation matrix R is critical in determining network performance.

Note: Consider the above model is used to determine the nonlinear process of (2.4), via (2.16), in which the additive noise is uncorrelated with zero mean and variance σ^2 , the least squares solution is identical to (3.7) and the estimate $\hat{\mathbf{w}} = E(\mathbf{w})$ (i.e. $\hat{\mathbf{w}}$ is unbiased) and $\text{cov}(\hat{\mathbf{w}}) = \sigma^2(\mathbf{A}^T\mathbf{A})^{-1}$ (see

[104]), which tends to zero as $N \rightarrow \infty$ if $(\mathbf{A}^T \mathbf{A})^{-1} \rightarrow 0$ (i.e. $\hat{\mathbf{w}}$ is a consistent estimator).

Now the autocorrelation matrix R can be decomposed into its normal form

$$R = U \Lambda U^T,$$

where U is the unitary matrix composed of the orthonormal eigenvectors of R and $\Lambda = \text{diag}\{\lambda_i\}$, is the $p \times p$ diagonal matrix of the non-negative eigenvalues of R . Defining the vector

$$\mathbf{v} = U^T \mathbf{e}_w,$$

then (3.10) can be expressed in terms of the eigenvalues of R as

$$V_N(\mathbf{w}) = V_{min} + \mathbf{v}^T \Lambda \mathbf{v} = V_{min} + \sum_{i=1}^p v_i^2 \lambda_i. \quad (3.11)$$

So differentiating (3.11) twice with respect to v_i gives

$$\lambda_i = \frac{1}{2} \frac{\partial^2 V_N(\mathbf{w})}{\partial v_i^2}, \quad \forall i.$$

That is, the performance function curvature is equivalent to the eigenvalues of R . The cost function, V_N , defines in the \mathbf{v} -space a hyperellipsoid, whose principal axes are the eigenvectors of R , but since U is a unitary matrix, then \mathbf{v} is a rotated version of \mathbf{e}_w in the original weight (\mathbf{w}) space. For any $\lambda_i = 0$, the performance function produces a valley in the rotated \mathbf{v} space (with an infinite number of solutions for $\hat{\mathbf{w}}$) along its principal axis in the rotated \mathbf{v} performance space (singular).

3.3 Batch learning laws

3.3.1 General learning laws

For some modelling problems a batch of data $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$ is available, and single shot solution for the optimal weight vector $\hat{\mathbf{w}}$, such as (3.7), can be utilised. However since the cost or performance function is quadratic with a global minimum, iterative and recursive based methods can be used to avoid computational problems associated with large data storage. Typically they use weight updates of the form:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \delta(k) \mathbf{d}(k), \quad (3.12)$$

where k represents the iteration index, $\delta(k)$ is the learning rate or step size, and $\mathbf{d}(k)$ a direction or search vector in the p -dimensional weight space. The various learning laws which have to generate those $\{\delta(k), \mathbf{d}(k)\}$ ensure that

$$E[V(\mathbf{w}(k+1))] = E[V(\mathbf{w}(k) + \delta(k) \mathbf{d}(k))] < E[V(\mathbf{w}(k))].$$

Clearly, irrespective of the shape of the performance function $V(\mathbf{w})$, it can be locally expanded in a Taylor series as a quadratic function for small steps $\delta(k)$:

$$\begin{aligned} V(\mathbf{w}(k+1)) \simeq V(\mathbf{w}(k)) + \nabla^T V(\mathbf{w}(k))\Delta\mathbf{w}(k) + \\ \frac{1}{2}\Delta\mathbf{w}^T(k)\nabla^2 V(\mathbf{w}(k))\Delta\mathbf{w}(k), \end{aligned} \quad (3.13)$$

where $\Delta\mathbf{w}(k) = \mathbf{w}(k+1) - \mathbf{w}(k)$, and $\nabla^2 V(\mathbf{w}) = \mathbf{H}$ the Hessian matrix of $V(\mathbf{w})$, and $\nabla^T V(\mathbf{w})$ is the gradient of $V(\mathbf{w})$. Assuming that the inverse of \mathbf{H} exists, then the optimal iteration to this approximated quadratic is the Newton–Raphson method ($\delta(k) = 1$)

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mathbf{H}^{-1}\nabla V(\mathbf{w}(k)). \quad (3.14)$$

If \mathbf{H} is positive definite and $V(\mathbf{w})$ is quadratic, then Newton's method goes to the optimal weight vector $\hat{\mathbf{w}}$ in one step! (Note: If \mathbf{H} is not positive definite, the Newton direction may point towards a local maximum or saddle point. \mathbf{H} can be made positive definite by adding a positive matrix, say $\lambda\mathbf{I}$, so that replacing \mathbf{H} by $\mathbf{H}' = \mathbf{H} + \lambda\mathbf{I}$ in (3.14), resulting in the Levenberg–Marquardt algorithm; this is a form of regularisation and ridge regression.)

3.3.2 Gradient descent algorithms

Computing the Hessian matrix is computationally expensive and inappropriate for batch processes, so many practical algorithms just utilise the gradient $\nabla V = \partial V / \partial \mathbf{w}$, which for the quadratic cost (3.3) is

$$\nabla V = \partial V / \partial \mathbf{w} = 2R\mathbf{w} - 2\mathbf{m} = -2R\mathbf{e}_w. \quad (3.15)$$

Again we note that it depends on the autocorrelation matrix R . Transforming (3.15) to the eigenspace \mathbf{v} via $\mathbf{v} = U^T \mathbf{e}_w$, we have alternatively

$$\nabla V_{\mathbf{v}} = \partial V / \partial \mathbf{v} = 2\Lambda\mathbf{v}.$$

That is, the gradient, with respect to v_i , depends only on the i th eigenvalue of R .

Substituting (3.15) into (3.12) with $\mathbf{d}(k) = -\frac{\nabla V}{2}$ gives the gradient descent weight updating rule:

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) - \delta(k)(R\mathbf{w}(k) - \mathbf{m}), \\ \text{or } \Delta\mathbf{w}(k) &= \delta(k)R\mathbf{e}_w(k), \end{aligned} \quad (3.16)$$

where $\Delta\mathbf{w}(k) = \mathbf{w}(k+1) - \mathbf{w}(k)$, and $\mathbf{w}(k+1) - \mathbf{w}(k) = \mathbf{e}_w(k+1) - \mathbf{e}_w(k)$. Substituting $\hat{\mathbf{w}}$ from (3.6) into (3.16) gives

$$\mathbf{e}_w(k+1) = (1 - \delta(k)R)\mathbf{e}_w(k),$$

or since $\mathbf{v} = U^T \mathbf{e}_w$, in decoupled components,

$$v_i(k+1) = (1 - \delta(k)\lambda_i)v_i(k) \quad \forall i, \quad (3.17)$$

whose solutions are

$$v_i(k+1) = (1 - \delta(k)\lambda_i)^k v_i(1). \quad (3.18)$$

Clearly for stable learning, $|1 - \delta(k)\lambda_i| < 1, \forall i$, or the learning rate $\delta(k)$ must satisfy

$$0 < \delta(k) < \frac{2}{\lambda_{max}}, \quad (3.19)$$

and the model weight vector \mathbf{w} tends to $\hat{\mathbf{w}}$ as $k \rightarrow \infty$ (see (3.18)). For zero eigenvalues of R , the error along the i th principal axis does not decay, as there are an infinite number of global minima in the weight space. In this case the error along the principal axis constrains parameter adjustment to a unique value which is dependent upon the initial weight values. Note that replacing δ by $\delta' = \delta/\lambda_{max}$ enables various gradient based learning algorithms to be compared as stability is dictated by the condition $0 < \delta < 2$. The rate of convergence in weight adjustment can be estimated by placing an exponential envelope on the iteration (3.18) as

$$v_i(1) \exp(-\frac{k}{\tau_i}) = v_i(1)(1 - \delta'\lambda_i)^k.$$

From which the effective time constant is

$$\tau_i = -\frac{1}{\ln(1 - \delta\lambda_i/\lambda_{max})}.$$

As the largest time constant, τ_{max} , corresponds to the smallest eigenvalue, λ_{min} , of R , then

$$\tau_{max} = -\frac{1}{\ln(1 - \delta\lambda_{min}/\lambda_{max})} = -\frac{1}{\ln(1 - \delta/C(R))}. \quad (3.20)$$

For a well conditioned network, $C(R)$ is close to unity and weight convergence is fast; for an ill-conditioned network ($C(R)$ is large), the performance function is valley shaped rather than bowl shaped, leading to slow convergence.

Steepest descent is one of the most popular gradient algorithms for parametric optimisation; it computes the one-step-ahead optimal step size and moves parallel to the negative gradient by this amount. It is more complex, but faster in convergence than the standard gradient algorithm (3.16). The weight update rule is

$$\Delta\mathbf{w}(k) = \delta(k)\mathbf{d}(k),$$

where $\mathbf{d}(k) = -\frac{1}{2}\partial V/\partial \mathbf{w} = -(R\mathbf{w}(k) - \mathbf{m}) = -\mathbf{r}(k)$, for $\mathbf{r}(k)$ the residual errors in the estimated cross-correlation vector \mathbf{m} . The ideal step size is problem dependent. A good practical choice is to use a $\delta(k)$ such that $V_N(\mathbf{w}(k))$ is minimised, which results in choosing δ such that the current search direction is orthogonal to the previous ones, giving [178]

$$\delta(k) = \frac{\mathbf{r}^T(k)\mathbf{r}(k)}{\mathbf{r}^T(k)\mathbf{A}\mathbf{r}(k)}. \quad (3.21)$$

Note that the steepest descent is the same as the Levenberg–Marquardt (Newton’s amended algorithm) for $\lambda \rightarrow \infty$.

An important aspect of network modelling is the relationship between the weight error vector \mathbf{e}_w , on which most learning laws operate, and the consequent network output prediction error \mathbf{e}_y . This functional relationship is given by [32] as

$$\frac{\|\mathbf{e}_w\|}{\|\mathbf{w}\|} \leq \sqrt{C(R)} \frac{\|\mathbf{e}_y\|}{\|\mathbf{y}\|}. \quad (3.22)$$

Hence for a poorly conditioned network, a small measured output error does not imply a small weight or parametric error. The network’s ability to generalise locally depends on the weight error, and terminating the learning prematurely (as e_y is small) may mean that some weights are not close to their optimal values. Condition (3.22) together with (3.20) and (3.7) shows how important proper selection of the basis functions, their order and positions, and training data is in determining effective network generalisation behaviour.

Notes:

(i) Conditioning of linear-in-the-parameters models depends on the shape of the basis functions and the distribution of the training data. The degree of activation and overlap of the basis functions determines the form of the autocorrelation, e.g. semi-global basis functions such as sigmoidal may lead to ill-conditioned networks. Generally linear-in-the-parameters networks are well conditioned as the basis functions are locally unimodal, mutually orthogonal (R is sparse), and the basis function power $E(\psi_i^2)$ is generally greater than its interaction with other basis functions as R tends to be diagonally dominant (see Gersgorin’s theorem). For examples of various neural network model conditioning see [32].

(ii) Frequently R has at least one zero eigenvalue due to basis functions having no training data lying in its support. Whilst this may cause problems in inverting R , it does not affect overall learning behaviour, such as its learning rate. It only influences which optimal solution (amongst the infinite number possible) will be found, as gradient based algorithms always find the minimum norm solution for \mathbf{w} if appropriately initiated.

3.4 Instantaneous learning laws

3.4.1 Least mean squares learning

For on-line control, conditioning monitoring and tracking problems, and for the modelling of time varying dynamical processes data is generated in real time, therefore estimation must be carried out as data is generated. Also in adaptive control, instantaneous learning algorithms are used where

the unknown parameters or weights are recursively estimated using currently available input–output data. Only the instantaneous estimate of the network performance is available for evaluating the network prior to parametric update. The instantaneous MSE, $\frac{1}{2}e_y^2(t) = \frac{1}{2}[y(t) - \hat{y}(t|t-1)]^2$, where $\hat{y}(t|t-1) = \psi^T(t)\mathbf{w}(t-1)$, is frequently used as it is both simple and utilises only the latest available observable data for training. The unbiased instantaneous estimate of the true gradient at t is

$$\hat{\nabla}V = -[y(t) - \hat{y}(t)]\psi(t) = -e_y(t)\psi(t). \quad (3.23)$$

Updating the weight vector in proportion to this gradient estimate gives the least mean squares (LMS) algorithm:

$$\Delta\mathbf{w}(t) = \delta e_y(t)\psi(t), \quad (3.24)$$

where δ is the learning rate. Here the weights are updated in proportion to the current error multiplied by the size of its contribution to the output via the transformed input $\psi(t)$. As $\psi(t)$ is sparse, only those weights that influence the output are updated, leading to a simple but highly efficient algorithm. After updating, the *a posteriori* network output is

$$\hat{y}(t) = \psi^T(t)\mathbf{w}(t) = \delta\|\psi(t)\|^2y(t) + (1 - \delta\|\psi(t)\|^2)\hat{y}(t|t-1),$$

where $\|\psi(t)\|^2 = \psi^T(t) \cdot \psi(t)$ and the *a posteriori* output error is

$$e_y(t) = y(t) - \hat{y}(t) = (1 - \delta\|\psi(t)\|^2)e_y(t|t-1), \quad (3.25)$$

where $e_y(t|t-1) = y(t) - \hat{y}(t|t-1)$ is a *a priori* error. For stability in learning we require that the *a posteriori* error $e_y(t)$ is less than the *a priori* error $e_y(t|t-1)$. This depends on both the learning rate δ and the Euclidean norm $\|\psi(t)\|^2$ of the transformed input vector (i.e. on the basis function ψ and data), since from (3.25) we have

$$\|e_y(t+1)\| < \|e_y(t)\|, \quad \text{if } \delta \in (0, \frac{2}{\|\psi(t)\|^2}). \quad (3.26)$$

Note: If the variance of the input signal is large, then $\|\psi(t)\|^2$ is large, leading to slow learning, as δ is small. Small δ also provides insensitivity to model mismatch and measurement noise.

3.4.2 Normalised least mean squares learning

The dependence of the learning rate δ on the norm of the transformed input $\psi(t)$, can be removed by setting

$$\delta(t) = \frac{\delta'}{\|\psi(t)\|^2}, \quad (3.27)$$

giving the normalised least mean squares (NLMS) weight update law:

$$\Delta\mathbf{w}(t) = \frac{\delta' e_y(t)\psi(t)}{\|\psi(t)\|^2}, \quad \delta' \in [0, 2]. \quad (3.28)$$

The normalisation term means that information is always stored when $\delta' = 2$ (*a posteriori* error=0). However, the NLMS law (3.28) no longer minimises the MSE, but rather a normalised version of it, $E[\frac{e_y^2(t)}{\|\psi(t)\|^2}]$. But if there exists a unique weight vector such that $e_y(t) = 0$, $\forall t$ (i.e. no modelling error), or if $\|\psi(t)\|^2$ is constant, then the LMS and NLMS lead to the same optimal weight vector, otherwise their minima occur at different points in the weight space. The rate of convergence of the NLMS algorithm and its condition depend on the normalised autocorrelation matrix \bar{R} .

The NLMS learning law (3.28) is a special case ($r = 2$) of the generalised NLMS rule [10] that satisfies the following conditions: find a \mathbf{w} such that

$$\hat{y}(t) = \psi^T(t)\mathbf{w}(t), \quad \|\Delta\mathbf{w}(t)\|_r \text{ is minimised} \quad (3.29)$$

for different values of r , $1 \leq r < \infty$. The general solution to (3.29) is

$$\Delta\mathbf{w}(t) = \delta \frac{e_y(t)\mathbf{s}[\psi(t)]}{\psi^T(t)\mathbf{s}[\psi(t)]}, \quad (3.30)$$

where $\mathbf{s}[\cdot]$ is a modified search direction such that

$$s_i(\psi) = \begin{cases} |\psi_i|^{q-1} sgn(\psi_i) & \text{if } 1 \leq q < \infty, \\ \delta_{i,k} & \text{if } q = \infty, \end{cases}$$

where $\delta_{i,k}$ is the Kronecker delta function, $k = \arg \max_j |\psi_j|$, and q is the value that satisfies $(1/r + 1/q) = 1$.

Special cases of the generalised NLMS law (3.30) include $r = 2$ (Euclidean norm, see (3.28)); $r = 1$ (the max-NLMS rule):

$$\Delta w_i = \delta \frac{e_y(t)}{\psi_k(t)} \delta_{i,k},$$

and $r = \infty$ (the sgn-NLMS rule):

$$\Delta w_i = \delta \frac{e_y(t) sgn[\psi(t)]}{\|\psi(t)\|_1}.$$

It is easily seen that the *a posteriori* error is always zero for these learning laws (with $\delta = 1$), so that the only difference between them is how they search the weight space. The weight updates in the weight space for the three learning rules are shown in Figure 3.2. The max-NLMS always updates the weight vector parallel to an axis (with largest error), the sgn-NLMS rule causes the update rule to be set at 45 degrees to an axis, whereas the standard ($r = 2$) NLMS rule always projects the weight vector perpendicularly onto the solution hyperplane (for $\delta = 1$).

3.4.3 NLMS weight convergence

Consider now the weight error $\mathbf{e}_w(t) = \hat{\mathbf{w}} - \mathbf{w}(t)$. From (3.30), an iterative equation in $\mathbf{e}_w(t)$ follows as:

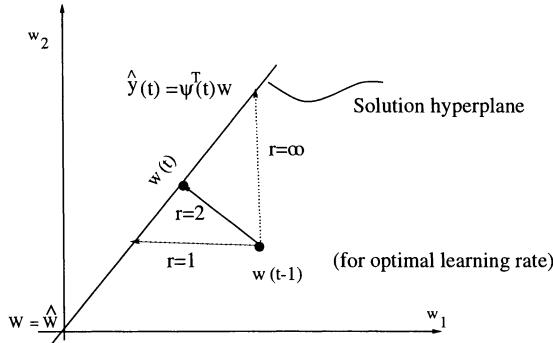


Fig. 3.2. Weight updates in the weight space

$$\mathbf{e}_w(t) = [I - \delta \frac{\mathbf{s}^T(t)\psi(t)}{\psi^T(t)\mathbf{s}(t)}] \mathbf{e}_w(t-1). \quad (3.31)$$

Denoting the matrix on the right hand side by $P(t)$, then the *a posteriori* weight error is

$$\|\mathbf{e}_w(t)\|_r = \|P(t)\mathbf{e}_w(t-1)\|_r \leq \|P(t)\|_r \|\mathbf{e}_w(t-1)\|_r \quad (3.32)$$

when $\delta = 1$, $P(t)$ is a projection matrix or operator [61], as:

$$P(P(\mathbf{e}_w(t-1))) = P(\mathbf{e}_w(t-1)),$$

whose structure determines the stability and convergence properties of different learning laws.

For the standard NLMS training algorithm, the projection operator is given by

$$P(t) = I - \delta \frac{\psi(t)\psi^T(t)}{\|\psi(t)\|_2^2},$$

from which it is easy to show that it has $(p-1)$ unity eigenvalues whose corresponding eigenvectors are orthogonal to $\psi(t)$, and one eigenvalue of $(1-\delta)$ whose corresponding eigenvector is $\psi(t)$. The magnitudes of the weight errors from a strictly non-increasing sequence as $\|P(t)\|_2 = 1$, and when the weight error vector has a non-zero component that is parallel to $\psi(t)$, $\|\mathbf{e}_w\|_2$ will decrease.

For the max-NLMS learning algorithm, the corresponding projection matrix is equivalent to an identity matrix except for the k th row which corresponds to the weight that is being updated. In this case the projection matrix k th row is

$$[-\delta \frac{\psi_1(t)}{\psi_k(t)}, \dots, -\delta \frac{\psi_{k-1}(t)}{\psi_k(t)}, 1 - \delta, -\delta \frac{\psi_{k+1}(t)}{\psi_k(t)}, \dots, -\delta \frac{\psi_p(t)}{\psi_k(t)}].$$

Therefore in order that $\|P(t)\|_\infty \leq 1$, and that the weight error is a non-increasing function of t , then the absolute sum of elements of the k th row of

P must sum to ≤ 1 [112]. Therefore the input training pattern must be input or diagonally dominant, i.e.

$$|\psi_k(t)| \geq \sum_{i=1, i \neq k}^p |\psi_i(t)|,$$

which is independent of the learning rate δ which must also be such that $\delta \in [0, 1]^2$. Whilst the input dominance condition is sufficient to guarantee convergence, but this is not always necessary, as the matrix norm bound is always a worst case analysis.

For the sgn-NLMS training law, the projection matrix k th row is given by

$$\left[-\delta \frac{\psi_1(t) \operatorname{sgn}(\psi_k(t))}{\|\psi(t)\|_1}, \dots, 1 - \delta \frac{\psi_k(t) \operatorname{sgn}(\psi_k(t))}{\|\psi(t)\|_1}, \dots, -\delta \frac{\psi_p(t) \operatorname{sgn}(\psi_k(t))}{\|\psi(t)\|_1} \right],$$

whose norm is given by [112]

$$\|P(t)\|_1 = 1 + (p-2)\delta \frac{\|\psi(t)\|_\infty}{\|\psi(t)\|_1} \leq 1 + (p-2)\delta. \quad (3.33)$$

Therefore for any three (or higher) input dimensional network, the sgn-NLMS algorithm may be unstable for any non-zero value of the learning rate and any type of (non-zero) input vector. Whether or not the algorithm is unstable depends on the signs and magnitudes of the components of the current weight error vector. For input signals that are input dominant, then the sgn-NLMS algorithm is potentially more unstable as $\|\psi(t)\|_\infty/\|\psi(t)\|_1$ is closer to unity.

In practice convergence depends on the properties of the transformed input process $\psi(t)$. For example, if $\psi(t)$ is not persistently exciting, but say settles to a constant value or varies slowly, then the standard NLMS algorithm will not converge to its optimal solution. The max-NLMS and sgn-NLMS perform even worse, for example, the sgn-NLMS does not converge in any sense when all the components of $\psi(t)$ are positive (or negative), equally the max-NLMS cannot converge when an independent element of $\psi(t)$ never attains the maximum value. For statistical signals we can consider convergence in the mean squared sense, i.e. if $\lim_{t \rightarrow \infty} E[\mathbf{e}_w^T(t)\mathbf{e}_w(t)] \rightarrow 0$.

From (3.31) for all three learning laws,

$$\begin{aligned} \mathbf{e}_w^T(t)\mathbf{e}_w(t) &= \mathbf{e}_w^T(t-1) \left[I - \delta \frac{\psi(t)\mathbf{s}^T(t)}{\psi^T(t)\mathbf{s}(t)} - \delta \frac{\mathbf{s}(t)\psi^T(t)}{\psi^T(t)\mathbf{s}(t)} \right. \\ &\quad \left. + \delta^2 \frac{\psi(t)\mathbf{s}^T(t)\mathbf{s}(t)\psi^T(t)}{(\psi^T(t)\mathbf{s}(t))^2} \right] \mathbf{e}_w(t-1). \end{aligned}$$

Denote $v^2(t) = E[\mathbf{e}_w^T(t)\mathbf{e}_w(t)]$ as the weight error variance, then from the above

$$v^2(t) = (1 - 2p^{-1}\delta + \beta\delta^2)v^2(t-1), \quad (3.34)$$

where

$$\beta = \begin{cases} \beta_{\text{standard NLMS}} = p^{-1} \\ \beta_{\text{sgn-NLMS}} = E\left[\frac{\psi^T(t)\psi(t)}{\psi^T(t)\text{sgn}(\psi(t))^2}\right] \\ \beta_{\text{max-NLMS}} = p^{-1}E\left[\frac{\psi^T(t)\psi(t)}{\psi_k^2(t)}\right] \end{cases}. \quad (3.35)$$

Clearly for convergence in the weight error variance we require that

$$0 < r = (1 - 2p^{-1}\delta + \beta\delta^2) < 1,$$

from which the optimal learning rate is

$$\delta^* = (p\beta)^{-1} \text{ with } r^* = (1 - p^{-1}\delta^*). \quad (3.36)$$

Generally

$$\delta_{\text{standard NLMS}}^* = 1 \geq \delta_{\text{sgn-NLMS}}^*, \delta_{\text{max-NLMS}}^*,$$

equally

$$r_{\text{sgn-NLMS}}^*, r_{\text{max-NLMS}}^* \geq r_{\text{standard NLMS}}^* = (1 - p^{-1}).$$

To evaluate the optimal learning rate and region of convergence for the modified NLMS algorithms, the probability distribution $p(\psi_i)$ is necessary, so for example take $p(\psi_i) = \alpha^{-1} \exp(-|\psi_i|/\alpha)$ (a Laplace distribution), it can be shown [10] that

$$\beta_{\text{sgn-NLMS}} = \frac{2}{(p+1)}, \quad \delta_{\text{sgn-NLMS}}^* = \frac{(p+1)}{2p}.$$

So that the convergence rate for p large for the sgn-NLMS is approximately twice that of the standard NLMS algorithm.

If the model (3.2), $\hat{y}(t) = \psi^T(t)\mathbf{w}$, is also subject to additive white noise $e(t)$ with variance σ_ξ^2 , then the above variance analysis can be repeated to give [10]

$$v^2(t) = (1 - 2p^{-1}\delta + \beta\delta^2)v^2(t-1) + \delta^2\gamma\sigma_\xi^2, \quad (3.37)$$

where

$$\gamma = \begin{cases} \gamma_{\text{standard NLMS}} = E[(\psi^T(t)\psi(t))^{-1}] \\ \gamma_{\text{sgn-NLMS}} = pE[(\psi^T(t)\text{sgn}(\psi(t)))^{-1}] \\ \gamma_{\text{max-NLMS}} = E[\psi_k^{-1}(t)] \end{cases}, \quad (3.38)$$

so that as $t \rightarrow \infty$, we have the weight error variance

$$v^2 = p\sigma_\xi^2\gamma \frac{\delta}{(2 - \beta\delta p)}. \quad (3.39)$$

So that after an initial transient convergence all three NLMS learning algorithms will converge to a mean weight value of $\hat{\mathbf{w}}$, but will then ‘jitter’ around the mean value with a variance v^2 . This region is called a minimal capture zone [32]. For a specific input distribution, the size of the minimal capture zone can be minimised by selecting a small value of δ , but this in turn decreases the rate of convergence.

The above modified NLMS learning algorithms are computationally efficient as they search the weight space in orthogonal directions, however they introduce serious problems of learning stability and associated rate of convergence. Only the standard NLMS is unconditionally stable, updating weight vectors parallel to the input vectors.

3.4.4 Recursive least squares estimation

Equation (3.2) can be rewritten for all the data D_N as

$$\mathbf{y} = \mathbf{Aw} + \mathbf{e},$$

where $\mathbf{e} = \mathbf{y} - \mathbf{Aw}$, whose least mean squared solution is given by (3.7) at the t th data point as

$$\mathbf{w}(t) = (\mathbf{A}_t^T \mathbf{A}_t)^{-1} \mathbf{A}_t^T \mathbf{y}_t$$

for $\mathbf{A}_t = [\psi(\mathbf{x}(1)), \dots, \psi(\mathbf{x}(t))]^T$, $\mathbf{y}_t = [y(1), \dots, y(t)]^T$. Rather than solve this equation as a single shot or batch solution it can be more efficiently solved iteratively by progressing through the data set D_N sequentially.

Suppose that at iteration $t+1$, the new data is $y(t+1)$ and $\psi(\mathbf{x}(t+1)) \equiv \psi(t+1)$ (where $\mathbf{x}(t+1)$ is the observed regressor vector at $t+1$). Hence

$$\begin{aligned} \mathbf{w}(t+1) &= \left(\begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix}^T \begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix} \right)^{-1} \\ &\quad \times \begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix}^T \begin{bmatrix} \mathbf{y}_t \\ y(t+1) \end{bmatrix}. \end{aligned} \quad (3.40)$$

Define $P(t) = (\mathbf{A}_t^T \mathbf{A}_t)^{-1}$, then

$$\begin{aligned} P(t+1) &= \left(\begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix}^T \begin{bmatrix} \mathbf{A}_t \\ \psi^T(t+1) \end{bmatrix} \right)^{-1} \\ &= [\mathbf{A}_t^T \mathbf{A}_t + \psi(t+1)\psi^T(t+1)]^{-1} \\ &= [P^{-1}(t) + \psi(t+1)\psi^T(t+1)]^{-1} \\ &= P(t) - P(t)\psi(t+1)[1 + \psi^T(t+1)P(t)\psi(t+1)]^{-1}\psi^T(t+1)P(t) \\ &= P(t) - \frac{P(t)\psi(t+1)\psi^T(t+1)P(t)}{1 + \psi^T(t+1)P(t)\psi(t+1)}. \end{aligned} \quad (3.41)$$

This iterative equation for $P(t)$ has been derived by the matrix inversion lemma [87]. Hence the weight update can be rewritten as

$$\begin{aligned} \mathbf{w}(t+1) &= P(t+1)[\mathbf{A}_t^T \mathbf{y}_t + \psi(t+1)y(t+1)] \\ &= P(t+1)[P^{-1}(t)\mathbf{w}(t) + \psi(t+1)y(t+1)] \\ &= \mathbf{w}(t) + P(t+1)\psi(t+1)[y(t+1) - \psi^T(t+1)\mathbf{w}(t)]. \end{aligned} \quad (3.42)$$

That is, the new estimate is the old estimate plus a correction term.

Note: Assuming that $\mathbf{e} \sim N(0, \sigma^2 I)$, then it can be easily shown that:

- $P(t)$ is proportional to the estimator covariance matrix $\text{cov}(\mathbf{w})$.
- Equations (3.41) and (3.42) form a Kalman filter type estimator [50] (see also Chapter 8)
- $\hat{\mathbf{w}}$ is an unbiased estimate.
- $\hat{\mathbf{w}}$ is a consistent estimate.
- The above least mean squares estimator is also a maximum likelihood estimator.
- This recursive algorithm requires no matrix inversion, utilises only the last iterative values of $\mathbf{w}(t)$, $P(t)$, and therefore is computationally efficient.

3.5 Gradient noise and normalised condition numbers

The condition of the autocorrelation matrix R not only determines the rate of convergence of batch gradient based weight training algorithms, it also influences the rate of convergence of the instantaneous LMS and NLMS learning rules as well. As we have seen above a full theoretical comparison of even simple learning laws such as LMS, NLMS is difficult since they depend strongly on the particular data input distribution used to train the network, so these results usually assume that statistically independent inputs are drawn randomly from a Gaussian input distribution with zero-mean.

The instantaneous LMS (or NLMS) rule typically uses the instantaneous gradient $\hat{\nabla}$, rather than the batch performance gradient ∇ , introducing an error

$$\mathbf{n}(t) = \nabla(t) - \hat{\nabla}(t).$$

This difference in gradients is called gradient noise, so that the instantaneous gradient (see (3.23)) can be written as

$$e_y(t)\psi(t) = E[e_y(t)\psi(t)] + \mathbf{n}(t).$$

The gradient noise covariance matrix $\mathbf{N}(t) = E[\mathbf{n}(t)\mathbf{n}^T(t)]$ is therefore

$$\mathbf{N}(t) = E[\psi(t)e_y(t)e_y(t)\psi^T(t)] - E[e_y(t)\psi(t)]E[e_y(t)\psi^T(t)].$$

This noise covariance expression describes the relative spread of the gradient noise components and is important for analysing the relative convergence rates for stochastic inputs for LMS (NLMS) learning laws. However this expression contains a forth order moment term which is difficult to analyse, but assuming that ψ_i 's are stationary, Gaussian distributed, and their zero mean components are statistically independent over t iterations, then it can be simplified [4] by using the real Gaussian factoring theorem to

$$\mathbf{N} = R\mathbf{e}_w\mathbf{e}_w^T R^T + R(v_{min} + \mathbf{e}_w^T R\mathbf{e}_w) + R\bar{e}_y^2 + R\mathbf{e}_w(\overline{e_y\psi^T}), \quad (3.43)$$

where $v_{min} = \min_{\mathbf{w}} E(y - \hat{y})^2$. Assuming that ψ and e_y are unbiased then (3.43) simplifies to

$$\mathbf{N} = R\mathbf{e}_w\mathbf{e}_w^T R^T + R(v_{min} + \mathbf{e}_w^T R\mathbf{e}_w). \quad (3.44)$$

Clearly the orientation and magnitude of the gradient noise covariance matrix is a function of the current weight error vector \mathbf{e}_w , the autocorrelation matrix R , and the minimum MSE v_{min} . Roughly speaking, the gradient noise is correlated with the steepest principal axis in the weight space accounting for the jittery learning behaviour of the weight vector when updated using LMS (NLMS) as illustrated in Figure 3.3 [24].

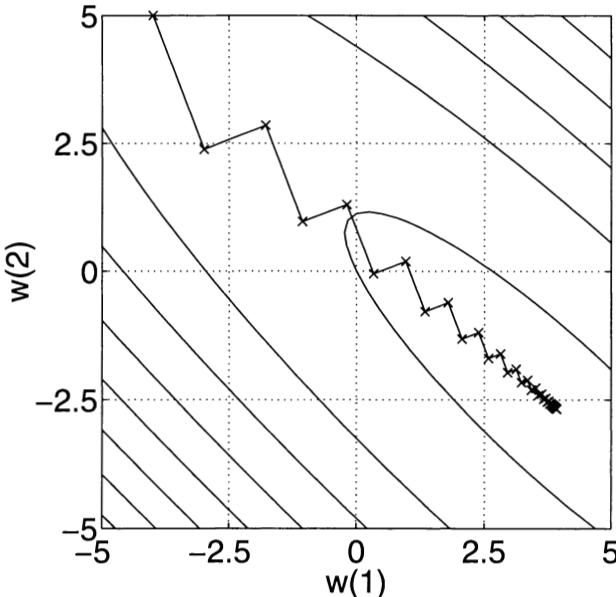


Fig. 3.3. A illustration of the jittery learning behaviour (here the parameter estimate still converges)

Near the optimal weight value $\hat{\mathbf{w}}$, the weight error vector is $\mathbf{e}_w \simeq 0$, this occurs for a well conditioned network, or for very large number of iterations (i.e. $t \rightarrow \infty$), in this case the gradient noise covariance is

$$\mathbf{N} = Rv_{min}. \quad (3.45)$$

Therefore whenever there exists a modelling error or there is measurement noise ($\sigma^2 \neq 0$), LMS (NLMS) rules do not converge to a point but rather to a domain called the minimal capture zone (see Section 3.4.3 and (3.39)), whose size is dependent upon the autocorrelation matrix R .

Finally (3.22) relates the observable network output error to the network's weight error through the network's condition number $C(R)$. In general the

NLMS algorithm provides faster convergence and better network conditioning than the conventional LMS learning algorithms as $C(R_{NLMS}) \simeq \sqrt{C(R_{LMS})} < C(R_{LMS})$ [5].

3.6 Adaptive learning rates

In the above instantaneous LMS, NLMS learning laws, it has been assumed that the learning rate δ is a positive constant number which satisfies inequalities (3.26) to ensure learning stability. Whenever (in all real situations) there exist modelling mismatch errors, observation noise and gradient noise, then these algorithms will only converge if the learning rate is scheduled against iterations to filter out these noise influences as well as minimising minimal capture domains. There are various methods for achieving this, e.g. assign an individual learning rate, δ_i , to each basis function and reduce δ_i over time as the belief in a particular weight value increases. This approach was originated as the Robbins–Munro stochastic approximation algorithm for which the necessary conditions for convergence on the learning rates, δ_i , associated with the i th basis function are:

$$\begin{aligned}\delta_i(t) &> 0 \\ \lim_{t \rightarrow \infty} \delta_i(t) &= 0 \\ \sum_{t=1}^{\infty} \delta_i(t) &= \infty \\ \sum_{t=1}^{\infty} \delta_i^2(t) &< \infty.\end{aligned}$$

The first condition is to ensure that the weight change direction is towards the hyperplane solution, the second condition ensures that the algorithm terminates asymptotically, the third condition implies that $\delta_i(t) \rightarrow 0$ and the final condition ensures that finite energy is used in achieving learning. One such condition that satisfies all these conditions is

$$\delta_i(t) = \frac{\delta_1}{(1 + t_i/\delta_2)},$$

where $\delta_1, \delta_2 > 0$ and t_i is the number of times that the i th basis function has been updated. The stochastic NLMS learning algorithm is of the form

$$\Delta w_i(t) = \delta_i \frac{\psi_i(t)}{\|\psi(t)\|_2^2} e_y(t).$$

In this chapter we have introduced the basic theory of least squares parametric learning laws for both batch and on-line training. Whilst high order learning algorithms are possible they are unnecessary for linear-in-the-parameters networks. Throughout the remainder of this book we utilise either the normalised least mean squares algorithm of Section 3.4.2 or recursive least squares estimators of Section 3.4.4 (or its Kalman filter derivatives).

4. Fuzzy and neurofuzzy modelling

4.1 Introduction to fuzzy and neurofuzzy systems

Fuzzy logic and fuzzy systems have received considerable attention both in scientific and popular media, yet the basic techniques of vagueness go back to at least the 1920s. Zadeh's seminal paper in 1965 on fuzzy logic introduced much of the terminology that is used conventionally in fuzzy logic today. The considerable success of fuzzy logic products, as in automobiles, cameras, washing machines, rice cookers, etc. has done much to temper much of the scorn poured out by the academic community on the ideas first postulated by Zadeh. The existing fuzzy logic literature, number of international conferences and academic journals, together with a rapidly increasing number and diversity of applications is a testament to the vitality and importance of this subject, despite the continuing debate over its intellectual viability.

The main reason for the criticism of fuzzy logic is that the resulting models are mathematically opaque, or there is no formal mathematical representation of the systems behaviour, which prevents application of conventional empirical modelling techniques to fuzzy systems, making model validation and benchmarking difficult to perform. Equally carrying out stability analysis on fuzzy logic controllers (the main area of applications) had been until recently impossible [197]. Also fuzzy logic is claimed to represent a degree of uncertainty or imprecision associated with variables, yet there is no distribution theory associated with fuzzy logic that allows the propagation of uncertainty to be tracked through the various stages of information processing in a fuzzy reasoning system. Whilst this is true (despite efforts to link fuzzy logic to possibility theory and probability theory) [206], fuzzy logic is primarily about linguistic *vagueness* through its ability to allow an element to be a *partial member* of a set, so that its membership value can lie between 0 and 1 and be interpreted as: the degree to which an event may be classified as something. It critically allows elements to be members of different sets with varying degrees of membership at the same time, and also allows ordering information to be retained in the class of membership values. Uncertainty is not an inherent property of the event, but rather occurs through the classification systems. Natural language terms such as *close* and *cold* are imprecise or vague yet humans reason with them and use them to convey useful information, expecting any such reasoning system to convey useful information

and to be able to *generalise* between neighbouring concepts without the need for rules to define every possible situation. For example, *rules* such as:

$$\begin{aligned} \text{IF (the room is hot)} & \text{ THEN (turn the heating down)} \\ \text{IF (the room is warm)} & \text{ THEN (keep the heating constant)} \\ \text{IF (the room is cold)} & \text{ THEN (turn up the heating)} \end{aligned} \quad (4.1)$$

involve natural language statements, and through graded fuzzy sets allow the system to use a finite number of rules to represent an infinite number of situations (temperatures) through interpolation or generalisation, as shown in Figure 4.1.

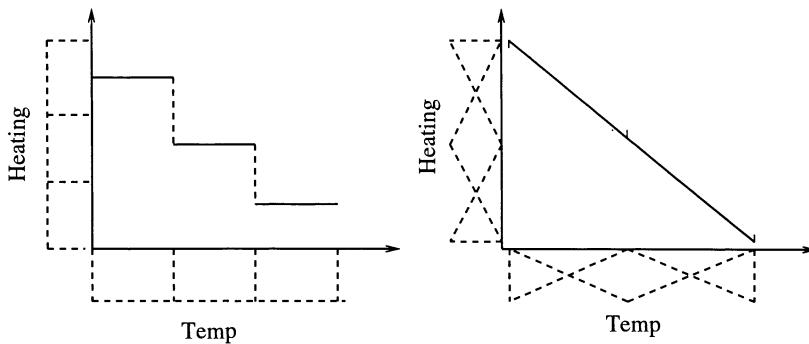


Fig. 4.1. The output of a binary (left) and a fuzzy (right) expert system

Generalisation or interpolation between concepts/rules is natural to both natural language and fuzzy logic (through its graded or multivalued set membership). Fuzzy logic can cope with ordinary set theory, partial set membership, ordinal variables (through grade set membership), interval variables and ratio variables, therefore as an information representation, fuzzy systems are a richer representation of knowledge than conventional sets.

The mathematical opaqueness of fuzzy systems has been partially removed through the development of *neurofuzzy systems* [32, 122, 197, 198], which attempt to integrate the analytical properties of a class of neural networks (usually linear-in-the-weight networks) with a class of fuzzy systems (usually using algebraic operators and centre of gravity defuzzification). Such techniques allow prior knowledge in the form of linguistic rules to be incorporated into empirical data-based modelling as well as allowing the final process model to be interpreted as a set of linguistic rules (so-called network transparency) or alternatively as a neural network (usually in the form of a readily interpreted surface for low dimensional systems – see Chapters 5 and 10). Neurofuzzy models are essentially *adaptive* fuzzy systems, where the adaptation is based on the fundamentals of feedback learning and weight adjustment found in conventional parametric optimisation. Within the class of

approximation models they are a particular form of generalised linear models that form an ideal framework for performing nonlinear system identification.

In system identification there are a number of modelling attributes that need to be considered when evaluating a particular algorithm, such as

- *Model accuracy:* The model should accurately approximate the derived system across the training data.
- *Generalisation:* The model should be able to generalise, this is the model's ability to model the desired system accurately for unseen inputs.
- *Interpretability or transparency:* Often the prior understanding of the underlying system is limited and it is beneficial to the modeller if the model process provides knowledge about the underlying physical process, i.e. qualitative physics (or grey box modelling as against black box modelling), such knowledge is useful in validating the behaviour of the model (in say diagnostic problems, and in problems of knowledge discovering).
- *Ability to encode a priori knowledge:* *A priori* knowledge is often available describing certain aspects of the system's operation. Any such knowledge should be exploited in modelling via say rule/weight initialisation.
- *Efficient implementation:* The modelling technique must use computational resources (i.e. data, speed, memory, ...) efficiently as in many applications these are inherently limited.
- *Fast adaptation:* This property is critical when the model is employed on line, where process adaptation is carried out as new system knowledge is acquired.

Most neurofuzzy algorithms satisfy all these attributes in some degree, however, as with fuzzy/expert/associative memory systems, neurofuzzy systems become computationally inefficient as the input space dimension increases due to the curse of dimensionality. The curse of dimensionality was a phrase introduced by Bellman in 1961 referring to the exponential increase in resources (data, memory, ...) required by a system as the input dimension increases. For example, for a complete base fuzzy system with n inputs, with p_i fuzzy sets on each input, the total number of rules or combinations of linguistic input terms is $p = \prod_{i=1}^n p_i$. Figure 4.2 illustrates the typical growth in fuzzy rules as n increases, equally the amount of training time and training data increases at the same rate. This fundamental weakness of fuzzy/neurofuzzy systems has limited their practical application to systems for which $n < 4$. In this book we develop model construction algorithms (such as ASMOD, MASMOD in Chapters 5 and 8 respectively) that avoid the curse of dimensionality.

Research into fuzzy systems and neurofuzzy systems is undergoing a renaissance, as researchers are recognising and exploiting abilities of neural networks in conjunction with rule based algorithms. It is *not* the purpose of this book to develop the theory of fuzzy systems. There are excellent books such as [51, 59, 100, 212] which cover the fundamental theory of fuzzy systems. Equally there are complementary research books on adaptive and neurofuzzy

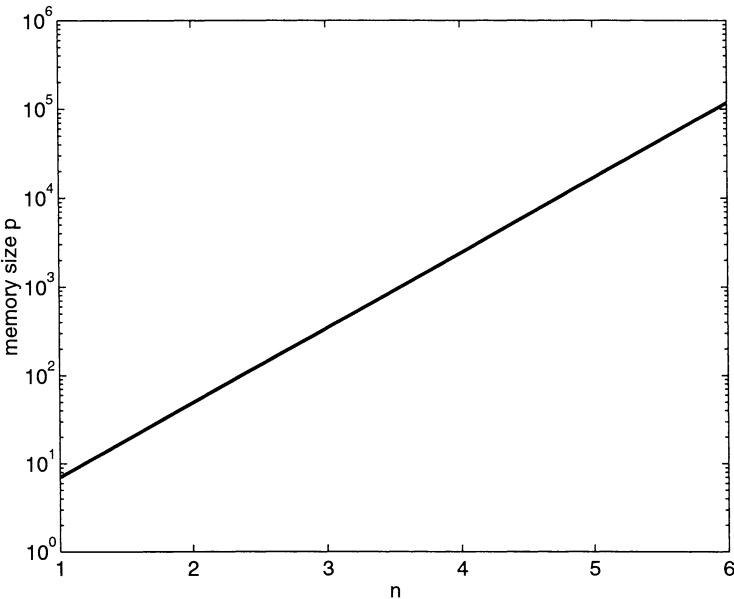


Fig. 4.2. Number of rules for increasing n , with $p_i = 7, \forall i$

modelling and control approaches, i.e. [32, 122, 175, 197, 198], which are well worth consulting by readers as reflecting the major techniques.

In this chapter the basic concepts of fuzzy systems are introduced, in particular the restrictive (but of wide applicability) class which demonstrates the similarities and links between fuzzy systems and generalised linear models from which neurofuzzy models are derived together with their inherent limitations and some mechanisms (such as construction algorithms) to resolve them.

4.2 Fuzzy systems

A fuzzy system is (usually) a nonlinear system whose behaviour is described by a set of linguistic rules in a rule base such as (4.1). The rule base or fuzzy algorithm represents expert knowledge in the form of IF–THEN production rules that relate vague input statements to vague output decisions/actions. The input or rule antecedent (IF) defines imprecisely the system states, and the consequent part (THEN) represents actions which could be taken to remedy the state condition. A fuzzy system contains all the components necessary to implement a fuzzy algorithm, it is composed of four elements:

- *A knowledge base* which contains definitions of fuzzy sets and the fuzzy operators;

- An *inference engine* which performs all the output calculations;
- A *fuzzifier* which represents the real valued (crisp) inputs as fuzzy sets;
- A *defuzzifier* which transforms the fuzzy output set into a real valued output (crisp).

The basic structure of a fuzzy system is shown in Figure 4.3, in which the inputs/outputs are usually crisp or deterministic variables. Central to fuzzy logic is the concept of fuzzy sets, since they give the designer a method for providing a precise representation of vague natural language terms such as *hot*, *cold*, *warm*, etc. Fuzzy logic provides the necessary mechanisms for manipulating and inferring conclusions based upon this vague information.

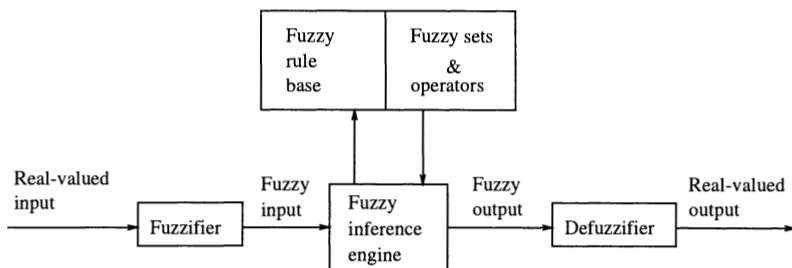


Fig. 4.3. Basic components of a fuzzy system

4.2.1 Fuzzy sets

Fuzzy sets were introduced to overcome some of the limitations of classical (boolean or binary) sets. Just as a classical set is defined by its characteristic function, a fuzzy set is represented by its *membership function* $\mu(\cdot) \in [0, 1]$, which represents the degree to which an event may be classified as something. It does *not* measure the frequency of occurrence of an event (that would be related to probability) or necessarily an individual's uncertainty about an event (subjective probability), rather it allows real world human orientated events to be classified using a finite number of linguistic classes. Generally, fuzzy membership functions are defined as one variable (univariate), but need not be the case as some logical connectives can be used to generate fuzzy membership functions defined on several variables from univariate ones.

Definition 4.1 (Fuzzy membership function $\mu_A(\cdot)$) The fuzzy membership function of a fuzzy set A is defined on its universe of discourse X , and is characterised by the function $\mu_A(\cdot) : X \rightarrow [0, 1]$, which maps each element of X to a real number that lies in the unit interval $[0, 1]$. For a particular input, the value of the fuzzy membership function represents the degree of membership of that set.

Membership functions provide an interface between the real valued (input) space and an expert's vague, linguistic sets. As such they provide precise representations of vague concepts, but this precision is useful because it contains an expert's domain specific knowledge about a particular situation. For the vast majority of applications, the power of fuzzy systems is through its ability to generalise, and as such the definition of one fuzzy set is related to the definition of its neighbouring ones, i.e. the definition of a fuzzy set (or basis function) is relative to how its neighbours are defined.

If fuzzy sets are to be general they must also include conventional sets: in this case a fuzzy membership function $\mu_A(\cdot)$ is crisp if $\mu_A(\cdot) \in \{0, 1\}, \forall x \in X$, and a *singleton* if

$$\mu_{\bar{x}}(x) = \begin{cases} 1 & \text{if } x = \bar{x}, \\ 0 & \text{otherwise,} \end{cases} \quad (4.2)$$

for actual measurements, as shown in Figure 4.4 (a) and (b).

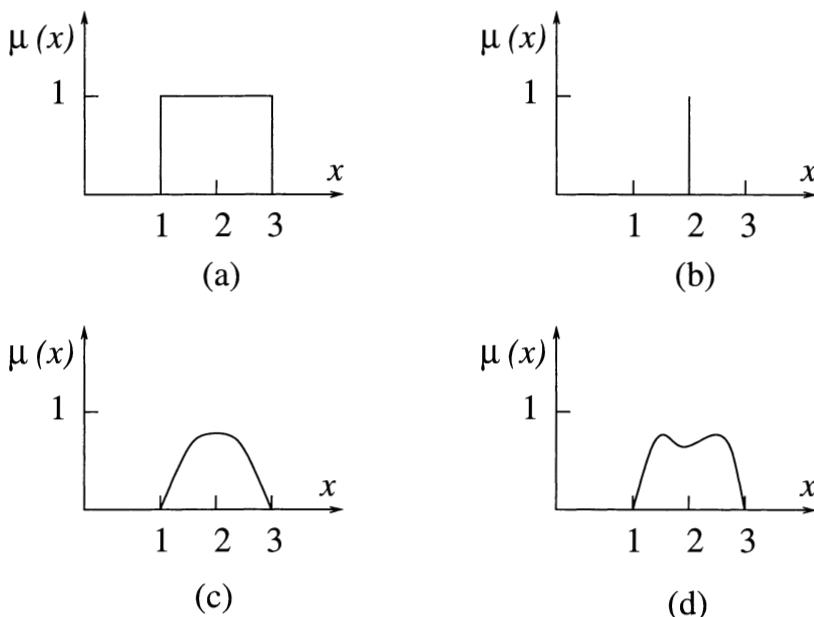


Fig. 4.4. Some membership functions; (a) crisp set; (b) singleton set; (c) unimodal set and (d) multimodal set

Definition 4.2 (Unimodal fuzzy sets) The fuzzy membership function $\mu_A(\cdot)$ is unimodal if

$$\mu_A(\lambda x + (1 - \lambda)y) \geq \min\{\mu_A(x), \mu_A(y)\}, \quad \forall x, y \in X, \forall \lambda \in [0, 1].$$

A unimodal fuzzy membership function is also known as *convex* and can be interpreted as an “inverse” distance measure. The vast majority of fuzzy membership functions are unimodal since they imply that the linguistic term has only a local influence on the overall rule base (see Figure 4.4 (c) and (d)).

Definition 4.3 (Fuzzy support) The support of a fuzzy set S_A , is given by the following set

$$S_A = \{x \in X : \mu_A(x) > 0\}.$$

The support of a fuzzy set A is that part of the input space for which its membership function is activated to a degree greater than zero. A related concept is *compact support*, which refers to the size of its support being strictly less than its original universe of discourse (see Figure 4.5).

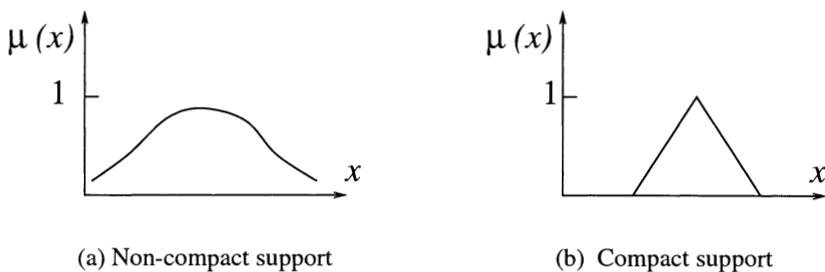


Fig. 4.5. Fuzzy support

The support of a fuzzy membership function therefore determines which inputs will activate fuzzy rules that have the corresponding linguistic term as part of their antecedent. If the fuzzy membership functions have non-compact support then every rule will be activated by each input, and the important concept of local knowledge and retrieval may be lost. Any unimodal membership function with non-compact support can be transformed into a (discontinuous) membership function with compact support by taking an α -cut, defined by

$$\mu_{A^\alpha}(x) = \begin{cases} \mu_A(x) & \text{when } \mu_A(x) \geq \alpha, \\ 0 & \text{otherwise.} \end{cases} \quad \text{for } \alpha \in [0, 1].$$

Definition 4.4 A fuzzy set is *normal* when $\max_{x \in X} \{\mu_A(x)\} = 1$ (subnormal otherwise) and the fuzzy set A is known as a fuzzy number when it is normal and defined on a real line.

It is important to note that all these concepts are *local* to a particular fuzzy set, whereas the power of a fuzzy system comes from its ability to generalise locally between neighbouring sets and rules. The main use of fuzzy (and neurofuzzy) algorithms is to provide local interpolation and extrapolation,

therefore it is difficult to determine if a particular fuzzy set or basis function is well designed or appropriate without reference to the remaining ones in the representation.

To describe or represent a real world measurement as a symbolic or fuzzy label, it is necessary to define two or more fuzzy sets on that particular variable. For example, the temperature of the room may be described as *cold*, *warm*, *hot*, each definition is relative to its neighbouring sets. Once the number of linguistic terms to model a variable is fixed, their form specified, then the complete set of fuzzy membership functions is known as a *fuzzy variable* V_x . A fuzzy set, A , is uniquely associated with its universe of discourse X , so a fuzzy variable can be considered as the group of all the fuzzy sets associated with this particular variable. More precisely, a fuzzy variable is defined as follows.

Definition 4.5 (Fuzzy variable) A fuzzy variable V_x is defined as a 4-tuple: $\{X, L, \chi, M_x\}$, where

- X is the symbolic name of a linguistic variable, such as *room temperature*.
- L is the set of linguistic labels associated with X , such as *cold*, *warm*, *hot*.
- χ is the domain over which L is defined on the universe of discourse of X . For real valued variables this could be a continuous interval such as $[-15, 45]$ (degrees centigrade) or a discrete, sampled set.
- M_x is a semantic function that returns the meaning of a given linguistic label in terms of the elements of X and the corresponding values of the fuzzy membership function.

A fuzzy variable is a collection of all the information used to represent a particular measurement as a fuzzy linguistic set. Fuzzy variables are very important in the overall system as a single fuzzy rule with a single fuzzy membership function is no richer in information than a crisp rule. Just as a simple fuzzy membership function has certain properties such as compact support, there are two important properties of fuzzy variables that need to be defined.

Definition 4.6 (Completeness) A fuzzy variable V_x is said to be complete if for each $x \in X$ there exists a fuzzy set A such that $\mu_A(x) > 0$.

When a fuzzy variable is incomplete, then there exists inputs which have no linguistic interpretation in terms of the current term set and hence the output of any system that is based on these linguistic sets will be undefined or zero.

Definition 4.7 (Partition of unity) The fuzzy variable forms a partition of unity (sometimes known as a fuzzy partition) if for every input x , we have

$$\sum_{i=1}^p \mu_{A^i}(x) = 1. \quad (4.3)$$

A partition of unity is a sufficient condition for completeness, and as such is a restrictive condition similar to the fundamental axiom of probability where all probabilities sum to unity. However in many engineering applications it can be shown that the partition of unity leads to improved generalisation as well as interpretation (i.e. transparency). For many fuzzy systems the final defuzzification operation involves a type of normalisation and any fuzzy variables that do not have this property have it implicitly imposed on them.

Any complete fuzzy system can be transformed into a system with a partition of unity by normalising the membership function accordingly

$$\mu_{\hat{A}^i}(x) = \frac{\mu_{A^i}(x)}{\sum_{j=1}^p \mu_{A^j}(x)}, \quad i = 1, 2, \dots, p, \quad (4.4)$$

since the modified fuzzy membership functions now satisfy $\sum_{i=1}^p \mu_{\hat{A}^i}(x) = 1$.

The actual shape of fuzzy membership functions that are used to represent linguistic terms are relative, subjective, and context dependent, as well as ill-defined. In much of fuzzy systems research piecewise linear (i.e. triangular) fuzzy sets are used, not because they are relative to the problem, but because they are easy to analyse. The basic form of fuzzy membership functions should satisfy the following two points:

- It must broadly possess the properties that are representative of the fuzzy linguistic terms. For example, it may be required that the membership functions are unimodal, have compact support and form a partition of unity.
- The membership functions must have a simple representation so that their form can be easily stored and so that the membership of a particular input can be quickly and accurately evaluated.

Unlike probability theory, where there is a unique probability density function determined by the statistics of the input–output signals, a fuzzy membership function is more difficult to determine and frequently selected based upon analytical tractability (e.g. triangular sets) than on *a priori* knowledge or distribution theory. If the input–output relationship is piecewise linear, then piecewise linear sets are appropriate (see Figure 4.6 (b)), however if continuous and smooth, then quadratic membership functions are appropriate (see Figure 4.6 (c) and (d)). Therefore in practice simple shapes such as Gaussians, or piecewise polynomial B-splines are used to represent fuzzy membership functions, or else they are learnt directly from some training data.

B-spline Membership Functions

B-spline basis functions are piecewise polynomial of order k , which have been widely used in surface fitting applications, but they also can be used as a

technique for designing fuzzy variables. B-spline basis functions are defined on a (univariate) real-valued measurement and are parameterised by the *order* of a piecewise polynomial k and also by the *knot vector* which is simply a set of values defined on the real-line that break it up into a number of intervals. This information is sufficient to specify a set of basis (membership) functions defined on the real-line whose shape is determined by the order k and where each membership function has a compact support k unit wide. In addition, the set of membership functions form a partition of unity (see Definition 4.7). The different shapes of membership functions, for different values of k , are shown in Figure 4.6 and it can be seen that they can be used to implement binary, crisp fuzzy sets ($k = 1$) or triangular fuzzy membership functions ($k = 2$) as well as smoother representations [32]. The B-spline basis functions therefore provide the designer with a flexible set of fuzzy set shapes, all of which can be evaluated efficiently.

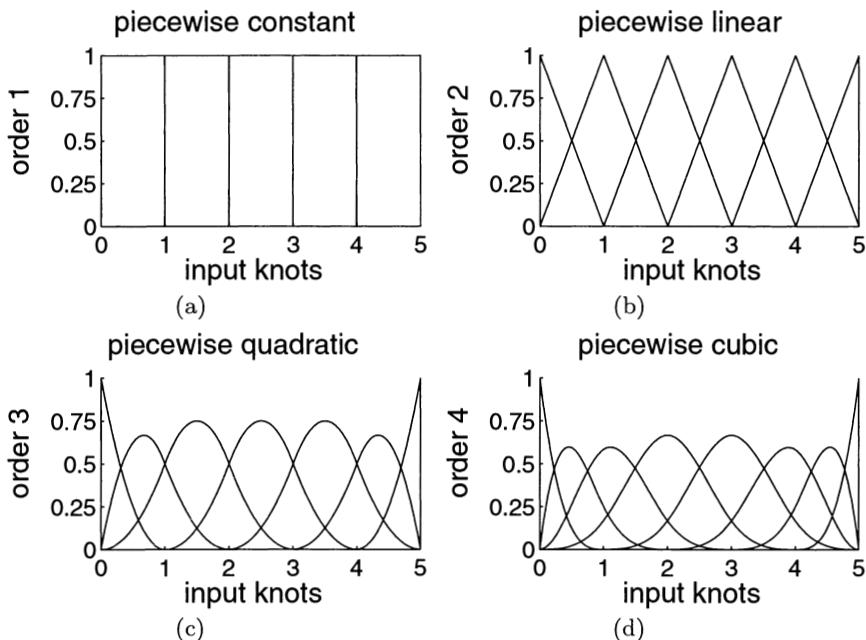


Fig. 4.6. B-spline fuzzy membership functions of orders 1 – 4

As well as choosing the shape (or order) of the membership functions, the designer must also supply a knot vector which determines how the membership functions are defined on its universe of discourse. Suppose there exist m linguistic terms (and hence fuzzy membership functions) in the fuzzy variable, then the membership functions are defined on an interior space $m - k + 1$

intervals wide and the designer must specify $(m + k)$ knot values, λ_i , which satisfy the following relationship:

$$x_{min} < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_{m-k} < x_{max}. \quad (4.5)$$

These knots roughly correspond to the centres of the individual basis functions and as such can be used to distribute them such that there is a fine resolution (large number of basis functions) in areas of interest and a coarse resolution (small number of basis functions) otherwise. A set of the extra knot values which define the basis functions at each end must also be specified and these should satisfy:

$$\lambda_{-k+1} \leq \dots \leq \lambda_0 = x_{min}, \quad (4.6)$$

$$x_{max} = \lambda_{m-k+1} \leq \dots \leq \lambda_m, \quad (4.7)$$

and this is illustrated in Figure 4.7 for order 2 (triangular) fuzzy membership functions.

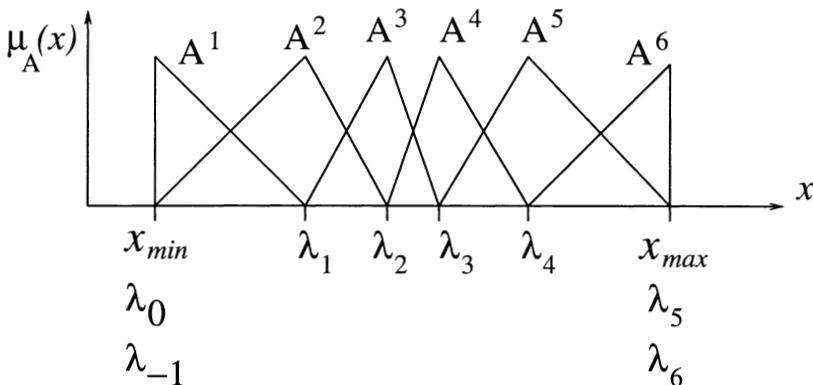


Fig. 4.7. Six B-spline fuzzy membership functions of order $k = 2$ where a non-uniform knot placement strategy is used

The output of B-spline membership functions can also be calculated using the following simple and stable recursive relationship:

$$\begin{aligned} \mu_{A_j^k}(x) &= \left(\frac{x - \lambda_{j-k}}{\lambda_{j-1} - \lambda_{j-k}} \right) \mu_{A_{j-1}^{k-1}}(x) + \left(\frac{\lambda_j - x}{\lambda_j - \lambda_{j-k+1}} \right) \mu_{A_j^{k-1}}(x) \\ \mu_{A_j^1}(x) &= \begin{cases} 1 & \text{if } x \in [\lambda_{j-1}, \lambda_j], \\ 0 & \text{otherwise} \end{cases}, \end{aligned} \quad (4.8)$$

where $\mu_{A_j^k}(x)$ is the j th membership function of order k . Therefore using B-spline basis functions as a framework for fuzzy membership functions has several important properties:

- A simple and stable recursive relationship can be used to evaluate the degree of membership.
- The basis functions have a compact support which means that knowledge is stored locally across only a small number of basis functions.
- The basis functions form a partition of unity which also implies that the corresponding fuzzy variables are complete.

Many of the piecewise polynomial fuzzy membership functions that have been used in the literature are simply particular types of standard, additive or dilated B-splines [114, 132].

Gaussian Membership Functions

Another fuzzy membership function that is often used to represent vague, linguistic terms is the Gaussian which is given by:

$$\mu_{A^i}(x) = \exp\left(-\frac{(c_i - x)^2}{2\sigma_i^2}\right), \quad (4.9)$$

where c_i and σ_i are the centre and width of the i th fuzzy set A^i , respectively. This is illustrated in Figure 4.8. Gaussian fuzzy sets have some very desirable properties in that both their spatial and frequency content (a Fourier transform of an exponential is another exponential) is local, although not strictly compact, and the output is very smooth in that it can be differentiated as many times as you like. However, it is important to note that although these functions can also represent (scaled) probability density functions, their meaning in this context is generally different. Probability density functions can be used to calculate the probability (or relative frequency) and a measurement will lie in an interval, whereas fuzzy set theory provides a measurement of the degree of membership that an exact measurement satisfies a vague concept. Gaussian functions can be used to model both situations but the underlying meaning is very different.

Multivariate Gaussian functions can be formed from the product of the univariate sets and this is one of the reasons why Gaussian fuzzy sets are popular; the multivariate radial basis functions can be expressed as a product of univariate ones. This emphasises the relationship between fuzzy sets and distance measures. An interesting compact support Gaussian-type function has also been proposed [202], and this membership function has a strictly compact support property and is also infinitely differentiable:

$$\mu_{A^i}(x) = \begin{cases} \exp^{-1}(-1) * \exp\left(-\frac{(\lambda_{i,2} - \lambda_{i,1})^2}{4(\lambda_{i,2} - x)(x - \lambda_{i,1})}\right) & \text{if } x \in (\lambda_{i,1}, \lambda_{i,2}), \\ 0 & \text{otherwise.} \end{cases} \quad (4.10)$$

Gaussian fuzzy membership functions are quite popular in the fuzzy logic literature, as they are the basis for the connection between fuzzy systems and radial basis function (RBF) neural networks.

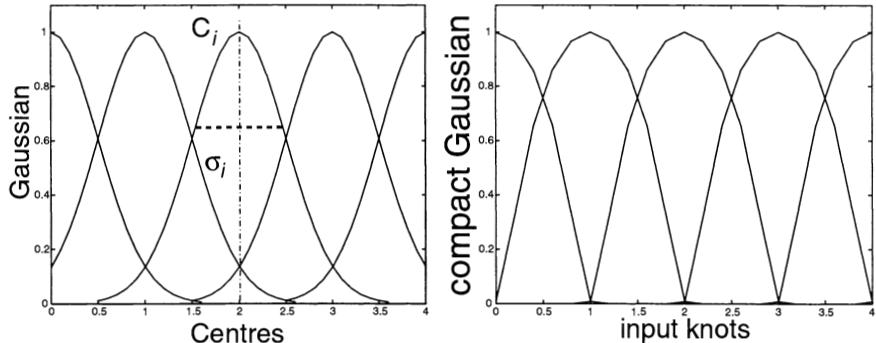


Fig. 4.8. Gaussian fuzzy sets

4.2.2 Fuzzy operators

Fuzzy sets and membership functions are the reason fuzzy logic was first introduced, since they capture the concept of vague membership of a set. However this ability to map data to fuzzy set memberships is not useful without a set of operators for combining this information and making inferences about its state. Fuzzy operators, such as AND, OR, NOT, etc. provide this flexibility.

Fuzzy Intersection: AND

The fuzzy intersection of two sets A and B refers to linguistic statement

$$(x \text{ is } A) \text{ AND } (y \text{ is } B),$$

where x and y could potentially refer to the same variable. A new fuzzy membership function is generated by this operator defined on the $X \times Y$ space, and is denoted by $\mu_{A \cap B}(x, y)$, where \cap refers to fuzzy set intersection. For binary (crisp) sets these operators are well defined and tabulated in truth tables, but there exist many possible generalisations for fuzzy logic operators (see [100]). For fuzzy intersection, the family of operators is known as the set of triangular or T-norms and the new membership function is generated by:

$$\mu_{A \cap B}(x, y) = \mu_A(x) \hat{*} \mu_B(y), \quad (4.11)$$

where $\hat{*}$ is the T-operator. For ease of notation, let $a, b, c, d \in [0, 1]$ denote the value of the fuzzy membership functions. The T-norm is defined as follows.

Definition 4.8 (T-norm) The set of triangular norms or T-norms is the class of functions which obey the following relationships:

- $a \hat{*} b = b \hat{*} a$
- $(a \hat{*} b) \hat{*} c = a \hat{*} (b \hat{*} c)$
- If $a \leq c$ and $b \leq d$ then $a \hat{*} b \leq c \hat{*} d$
- $a \hat{*} 1 = a$
- If a T-norm is Archimedean then $a \hat{*} a < a$, $\forall a \in (0, 1)$.

There are many possible operators which satisfy these conditions, but the two most popular are the algebraic product and min functions:

$$\mu_{A \cap B}(x, y) = \mu_A(x) * \mu_B(y), \quad (4.12)$$

$$\mu_{A \cap B}(x, y) = \min\{\mu_A(x), \mu_B(y)\}, \quad (4.13)$$

for which the former is an Archimedean T-norm. Historically the min operator has been used in fuzzy logic, especially for its simple implementation. However, more recently the algebraic product operator has been shown [32] to perform better, since the ‘min’ operator acts as a truncation operator losing information contained in the membership functions $\{\mu_A(x), \mu_B(y)\}$. Also using the product operator allows error information to be back propagated through the network as the first derivative is well defined. It also generally gives a smoother output surface when univariate B-spline and Gaussian fuzzy membership functions are used to represent each linguistic statement. Here a multivariate membership function is simply a multivariate B-spline or Gaussian basis function.

It can be shown [100] that for any T-norm, we have

$$\mu_A(x) * \mu_B(x) \leq \min\{\mu_A(x), \mu_B(x)\}, \quad (4.14)$$

therefore the ‘min’ operator forms an upper bound on the space of fuzzy intersection operators.

In a fuzzy algorithm, the antecedent of a fuzzy production rule is formed from the fuzzy intersection of n -univariate fuzzy sets:

$$(x_1 \text{ is } A_1^i) \text{ AND } \dots \text{ AND } (x_n \text{ is } A_n^i),$$

which produces a new multivariate membership function

$$\mu_{A_1^i \cap \dots \cap A_n^i}(x_1, \dots, x_n) \text{ or } \mu_{A^i}(\mathbf{x}),$$

defined on the original n -dimensional input space whose output is given by

$$\mu_{A^i}(\mathbf{x}) = \hat{\prod}(\mu_{A_1^i}(x_1), \dots, \mu_{A_n^i}(x_n)), \quad (4.15)$$

where $\hat{\prod}$ is the multivariate T-norm operator.

Fuzzy Union: OR

The fuzzy union of two sets A and B refers to a linguistic statement of the form:

$$(x \text{ is } A) \text{ OR } (y \text{ is } B),$$

where x and y could potentially refer to the same variable. A new fuzzy membership function is generated by this operation defined on the $X \times Y$ space, and is denoted by $\mu_{A \cup B}(x, y)$ where the union operator \cup mirrors the binary \vee operator. There are many ways of generalising union in fuzzy logic.

This family of operators is known as triangular w -norms or S -norms [100] and new membership functions are obtained from:

$$\mu_A \hat{\cup}_B (x, y) = \mu_A(x) \hat{+} \mu_B(y), \quad (4.16)$$

where $\hat{+}$ is the binary S -norm operator. Again, letting $a, b, c, d \in [0, 1]$ denote the value of the fuzzy membership functions, we have:

Definition 4.9 (S-norm) The set of triangular w -norms or S -norms is the class of functions which satisfy the following:

- $a \hat{+} b = b \hat{+} a$,
- $(a \hat{+} b) \hat{+} c = a \hat{+} (b \hat{+} c)$,
- If $a \leq c$, $b \leq d$, then $a \hat{+} b \leq c \hat{+} d$,
- $a \hat{+} 0 = a$.

Two of the most commonly used fuzzy OR operators are the algebraic sum and max functions:

$$\mu_A \hat{\cup}_B (x, y) = \mu_A(x) + \mu_B(y), \quad (4.17)$$

$$\mu_A \hat{\cup}_B (x, y) = \max\{\mu_A(x), \mu_B(y)\}. \quad (4.18)$$

If the fuzzy membership functions do not form a partition of unity, the sum operator is sometimes replaced by the *bounded* sum:

$$\mu_A \hat{\cup}_B (x, y) = \mu_A(x) + \mu_B(y) - \mu_A(x) * \mu_B(y), \quad (4.19)$$

which always has a membership value that lies in the unit interval. The max operator can be shown to be the most pessimistic S -norm as

$$\max\{\mu_A(x), \mu_B(y)\} \leq \mu_A(x) \hat{+} \mu_B(y). \quad (4.20)$$

Unlike T -norms, when the arguments of an S -norm are unimodal, the resulting membership function is unlikely to retain the unimodel property, as shown in Figure 4.9, where max and sum operators are compared, and it is clearly seen that non-unimodal membership functions may be produced by the union operator.

Fuzzy Implication: IF (.) THEN (.)

Fuzzy implication relationships are an expert's knowledge about how a vague linguistic input set is related to an output set. This can be represented as: $A \rightarrow B$ or B is true whenever A is true.

Its usage in fuzzy systems differs from its common definition in standard logic, since for binary logic implication is represented by

$$A \rightarrow B = A^c \vee B = (A \wedge B) \vee A^c,$$

where A^c denotes set complement, so even when A is false, B will be activated as the implication is true.

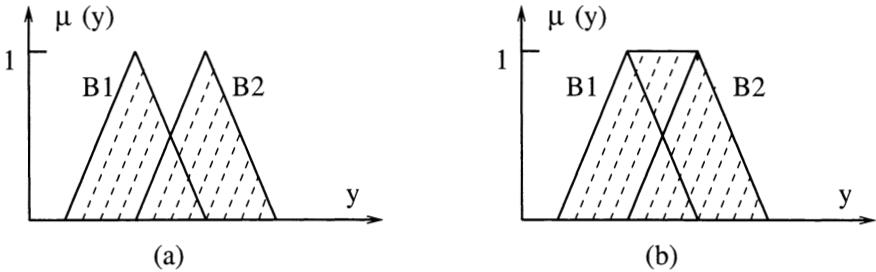


Fig. 4.9. Comparison of max (a) and sum (b) operators where the shaded area represents the membership function $\mu_{B^1 \cup B^2}(\cdot)$

In fuzzy systems, implication represents a causal relationship between input and output sets, where the ideas of local knowledge representation are particularly important. A general fuzzy algorithm is composed of a set of production rules of the form:

$$\begin{aligned} r_{ij} : & \text{ IF } (x_1 \text{ is } A_1^i \text{ AND } \dots \text{ AND } x_n \text{ is } A_n^i) \\ & \text{THEN } (y \text{ is } B^j) \quad c_{ij}, \end{aligned} \quad (4.21)$$

where r_{ij} rule is the ij th fuzzy production rule which relates the i th input fuzzy set A^i to the j th output fuzzy set B^j . The degree of confidence, with which the input fuzzy set A^i (which is composed of fuzzy intersection (AND) of several univariate fuzzy sets) is related to the output fuzzy set B^j , is given by a *rule confidence* $c_{ij} \in [0, 1]$. When c_{ij} is zero, the rule is inactive and does not contribute to the output. Otherwise, it partially fires whenever its antecedent is activated to a degree greater than zero. The rule base is characterised by a set of rule confidence $\{c_{ij}\}$ ($i = 1, 2, \dots, p$; $j = 1, 2, \dots, q$) which can be stored in a rule confidence matrix $C = \{c_{ij}\}$. Once fuzzy membership functions have been defined, the rule confidences encapsulate the experts knowledge about a particular process and also form a convenient set of parameters to train the fuzzy system from training data (see [32] for confidence rule learning algorithms).

The rule r_{ij} (4.21) can be also written as

$$r_{ij} : \text{IF } (\mathbf{x} \text{ is } A^i) \text{ THEN } (y \text{ is } B^j), \quad c_{ij},$$

then the degree to which \mathbf{x} is related to element y is represented by the $(n+1)$ -dimensional membership function $\mu_{r_{ij}}(\mathbf{x}, y)$ defined on the product space $A_1 \times \dots \times A_n \times B$ by

$$\mu_{r_{ij}}(\mathbf{x}, y) = \mu_{A^i}(\mathbf{x}) \hat{*} c_{ij} \hat{*} \mu_{B^j}(y), \quad (4.22)$$

where $\hat{*}$ represents T-norm operation. The fuzzy membership function $\mu_{r_{ij}}(\mathbf{x}, y)$ represents the confidence in the output being y given that the input is \mathbf{x} for the ij th fuzzy rule.

There are several other methods for implementing the implication operator [59], however the above is particularly amenable to forming the mathematical link between fuzzy and neural systems.

4.2.3 Fuzzy relation surfaces

For simple fuzzy algorithms, the union operator is generally used to connect the outputs of the different rules, although it is also informative to consider the union of all the rules defined on both the input and output universes. When p univariate fuzzy input sets A^i map to q univariate fuzzy output sets B^j , there are pq overlapping $(n+1)$ -dimensional membership functions formed using the intersection and implication operators, one for each relation. The individual pq fuzzy membership functions are $\mu_{r_{ij}}(\mathbf{x}, y)$ (defined on the space $\{A \times B\} \subseteq R^{n+1}$) which can be connected to form a fuzzy rule base \mathbf{R} by taking the union of these membership functions through

$$\begin{aligned}\mu_{\mathbf{R}}(\mathbf{x}, y) &= \bigcup_{i,j} (IF \ \mu_{A^i}(\mathbf{x}) \ THEN \ \mu_{B^j}(y), \ c_{ij}) \\ &= \widehat{\sum}_{i,j} \mu_{r_{ij}}(\mathbf{x}, y) \\ &= s(t(\mu_{A^1}(\mathbf{x}), \mu_{B^1}(y), c_{11}), \dots, t(\mu_{A^p}(\mathbf{x}), \mu_{B^q}(y), c_{pq})),\end{aligned}\quad (4.23)$$

where $\widehat{\sum}$ is the multivariable S -norm operator, and $s(\cdot)$, $t(\cdot)$ are respectively the univariate $\{S, T\}$ operators. (In distribution terms, $\mu_{\mathbf{R}}(\mathbf{x}, y)$ is a joint ‘probability’ density function.)

The union of all the individual relational membership functions forms a *ridge* or *fuzzy relational surface* in the input–output space which represents how individual input–output pairs are related and can be used to infer a fuzzy output membership function given a particular input measurement, a process known as fuzzy inferencing. A typical relational surface is shown in Figure 4.10, where four triangular fuzzy sets (B-splines of order 2) are defined on each variable, the algebraic functions are used to implement the logic operators, and the fuzzy algorithm is given by

$$\begin{array}{lll} r_{1,1} & IF \ (x \text{ is } AZ) \ THEN \ (y \text{ is } AZ) & 1.0 \\ r_{2,1} & OR \ IF \ (x \text{ is } PS) \ THEN \ (y \text{ is } AZ) & 0.4 \\ r_{2,2} & OR \ IF \ (x \text{ is } PS) \ THEN \ (y \text{ is } PS) & 0.6 \\ r_{3,2} & OR \ IF \ (x \text{ is } PM) \ THEN \ (y \text{ is } PS) & 0.2 \\ r_{3,3} & OR \ IF \ (x \text{ is } PM) \ THEN \ (y \text{ is } PM) & 0.8 \\ r_{4,4} & OR \ IF \ (x \text{ is } PL) \ THEN \ (y \text{ is } PL) & 1.0. \end{array} \quad (4.24)$$

This produces a fuzzy relational surface which is piecewise linear between rule centres and the general trend of the input, output relationship, almost a linear mapping, is obvious from the contour plot.

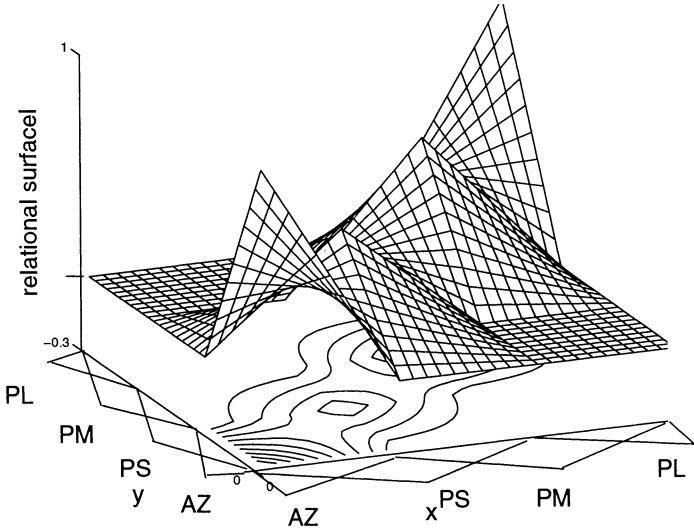


Fig. 4.10. A fuzzy relational surface, $\mu_R(x, y)$ and associated contour plot for a single input, single output fuzzy system with normalised rule confidence vectors and algebraic fuzzy operators. Each peak corresponds to a fuzzy rule

When the input is known, the fuzzy inferencing algorithms produce a single fuzzy output set for each rule and the output of the fuzzy rule base is the union of all these membership functions defined on the output universe. It should therefore be clear that when a fuzzy algorithm is implemented, fuzzy membership function intersection generally occurs across different universes whereas fuzzy union usually takes place on the same universe.

4.2.4 Inferencing

Inferencing is the process of reasoning about a particular state, using all available knowledge to produce a best estimate of the output. In a fuzzy system, the inference engine is used to pattern match the current fuzzy input set $\mu_A(x)$ with the antecedents of all the fuzzy rules and to combine their responses, producing a single fuzzy output set $\mu_B(y)$. This is defined by

$$\mu_B(y) = \widehat{\Sigma}_x(\mu_A(x) * \mu_R(x, y)), \quad (4.25)$$

where the triangular w -norm $\widehat{\Sigma}_x$ is taken over *all* possible values of x , and the triangular-norm computes a match between two membership functions for a particular value of x . When $\widehat{\Sigma}$ and $\widehat{\Pi}$ are chosen to be the integration (sum) and the product operators, respectively, then:

$$\mu_B(y) = \int_D \mu_A(x) \mu_R(x, y) dx, \quad (4.26)$$

which for an arbitrary fuzzy input set requires an n -dimensional integral to be evaluated over the input domain D . The calculated fuzzy output set depends on the fuzzy input set $\mu_A(\cdot)$, the relational surface $\mu_R(\cdot)$, as well as the actual inferencing operators. Using singleton fuzzification this reduces to $\mu_A(\mathbf{x}^s, y)$ which is a slice through the relational surface. The result is a fuzzy output representing the degree to which the input is related to each of the fuzzy output sets. (In distribution terms, $\mu_B(y)$ is the “conditional” probability density function of y conditioned on \mathbf{x} .)

As long as there exists an overlap between the fuzzy input set and the antecedents of the rule base, then the fuzzy system is able to *generalise* in some sense. The ability to generalise information about neighbouring states is one of the strengths of fuzzy logic, but their actual interpolation properties are poorly understood. The neurofuzzy systems studied in this book are particularly important as their approximation abilities can be both determined and analysed theoretically, which has many important consequences for practical systems.

4.2.5 Fuzzification and defuzzification

The fuzzy membership functions are the interface between the real-valued world outside the fuzzy system and its own internal rule based representation. Hence, a real-valued input must be represented as a fuzzy set in order to perform the inferencing calculations and the information contained in the fuzzy output set must be compressed to a single number which is the real-valued output of the fuzzy system. This section discusses different methods for performing these operations.

Fuzzification

The process of representing a real-valued signal as a fuzzy set is known as *fuzzification* and is necessary when a fuzzy system deals with real-valued inputs. There are many different methods for implementing a fuzzifier, but the most commonly used is the *singleton* that maps the input \mathbf{x} to a crisp fuzzy set with membership

$$\mu_{\tilde{\mathbf{x}}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} = \tilde{\mathbf{x}}, \\ 0 & \text{otherwise.} \end{cases} \quad (4.27)$$

For inputs that are corrupted by noise, the shape of the fuzzy set can reflect the uncertainty associated with the measurement process. For example, a triangular fuzzy set may be used where the vertex corresponds to the mean of some measurement data and the base width is a function of the standard deviation. If the model input is a linguistic statement, a fuzzy set must be found that adequately represents this statement. Unless the input is a linguistic statement, there is no justification for fuzzifying the input using the same

membership functions used to represent the linguistic statements such as *x is small*. The latter membership functions are chosen to represent *vague* linguistic statements whereas the input fuzzy sets reflect the uncertainty associated with the *imprecise* measurement process, and these two quantities are generally distinct. A fuzzy input distribution effectively low pass filters or averages neighbouring outputs and as the width of the input set grows (increasingly imprecise measurements), a greater emphasis is placed on neighbouring output values and the system becomes conservative in its recommendations.

Defuzzification

When a fuzzy output set $\mu_B(y)$ is formed as the output of the inferencing process, it is necessary to compress this distribution to produce a single value, representing the output of the fuzzy system. This process is known as defuzzification and currently there are several commonly used methods. Perhaps the two most widely used are the mean of maxima (MOM) and the centre of gravity (COG) algorithms which are illustrated in Figure 4.11. These can be classed as truncation and algebraic defuzzification methods, respectively, as the former bases the output estimate on only one piece of information (or at most an average of several) because the output is the value which has the largest membership in $\mu_B(y)$, whereas the latter uses the normalised weighted contribution from every point in the output distribution. The COG defuzzification algorithm tends to give a smoother output surface as there is a more gradual transition between the rules as the input is varied.

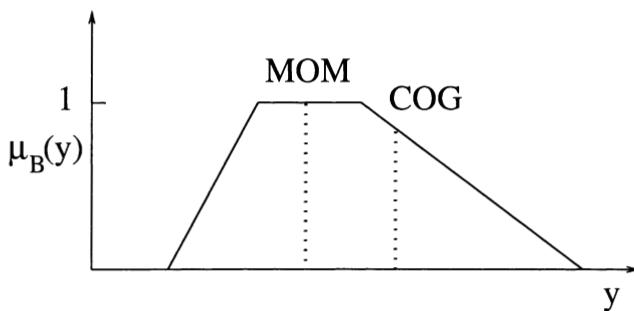


Fig. 4.11. The mean of maxima and centre of gravity defuzzification algorithms

The COG defuzzification process is defined by

$$y(\mathbf{x}) = \frac{\int_Y \mu_B(y)ydy}{\int_Y \mu_B(y)dy}, \quad (4.28)$$

and the whole of the output distribution contributes to determining the network's output. This is in direct contrast with the MOM procedure where only

the elements with maximal membership are considered and the rest of the distribution is taken as being unimportant. This can be expressed as

$$y(\mathbf{x}) = \frac{\int_Y \mu_{B^H}(y) y dy}{\int_Y \mu_{B^H}(y) dy}, \quad (4.29)$$

where $\mu_{B^H}(y)$ is the fuzzy set obtained by taking the α -cut at the height, H_B , of set B.

Just as there exists a whole family of T -norms and S -norms, there is a large number of defuzzification algorithms [59]. In practice, the COG defuzzification procedure is most widely used.

4.3 Functional mapping and neurofuzzy models

Many engineering applications require a fuzzy system that simply operates as a functional mapping, mapping real-valued inputs to real-valued outputs where the task is to approximate a function $y = f(\mathbf{x})$ on a bounded (*compact*) area of the input space. In contrast to the data-driven methods used to train artificial neural networks, fuzzy systems are designed using human-centred engineering techniques where the system is used to encode the heuristic knowledge articulated by a domain-specific expert. A finite number of vague or fuzzy rules forms the basis for the fuzzy system's knowledge base and to *generalise* or interpolate between these rules, the inference engine weights each rule according to its firing strength, which in turn is determined by both the shape of the fuzzy membership functions and the logical operators used by the inference engine. This section shows that when a centre of gravity defuzzification algorithm is used in conjunction with algebraic operators, then the type of functional mapping performed by the system is directly dependent on the shape of the fuzzy input sets. The rule confidence matrix is a set of parameters that determines the magnitude (height) of the fuzzy mapping, but it is the fuzzy input sets that determine its form.

Consider a fuzzy system that uses a centre of gravity defuzzification algorithm, then the network's output is given by

$$y(\mathbf{x}) = \frac{\int_Y \mu_B(y) y dy}{\int_Y \mu_B(y) dy}. \quad (4.30)$$

When the T -norm and S -norm operators are implemented using product and sum functions, respectively, then the centre of gravity defuzzification algorithm becomes

$$y(\mathbf{x}) = \frac{\int_Y \int_X \mu_A(\mathbf{x}) \sum_{ij} \mu_{A^i}(\mathbf{x}) \mu_{B^j}(y) c_{ij} y d\mathbf{x} dy}{\int_Y \int_X \mu_A(\mathbf{x}) \sum_{ij} \mu_{A^i}(\mathbf{x}) \mu_{B^j}(y) c_{ij} d\mathbf{x} dy}. \quad (4.31)$$

But for bounded and symmetric fuzzy output sets (such as B splines) the integrals $\int_Y \mu_{B^j}(y) dy$, for all j , are equal and so the following relationship holds:

$$\frac{\int_Y \mu_{B^j}(y) y dy}{\int_Y \mu_{B^j}(y) dy} = y_j^c,$$

where y_j^c is the centre of the j th output set, and (4.31) therefore reduces to

$$y(\mathbf{x}) = \frac{\int_X \mu_A(\mathbf{x}) \sum_i \mu_{A^i}(\mathbf{x}) \sum_j c_{ij} y_j^c d\mathbf{x}}{\int_X \mu_A(\mathbf{x}) \sum_i \mu_{A^i}(\mathbf{x}) \sum_j c_{ij} d\mathbf{x}}.$$

Suppose that the multivariate fuzzy input sets form a partition of unity, i.e. $\sum_i \mu_{A^i}(x) = 1$ and that the i th rule confidence vector $\mathbf{c}_i = (c_{i1}, \dots, c_{iq})^T$ is normalised, i.e. $\sum_j c_{ij} = 1$, then the defuzzified output becomes

$$y(\mathbf{x}) = \frac{\int_X \mu_A(\mathbf{x}) \sum_i \mu_{A^i}(\mathbf{x}) w_i d\mathbf{x}}{\int_X \mu_A(\mathbf{x}) d\mathbf{x}}, \quad (4.32)$$

where $w_i = \sum_j c_{ij} y_j^c$ is the *weight* associated with the i th fuzzy membership function. The transformation from the weight w_i to the vector of rule confidence \mathbf{c}_i is a one-to-many mapping, although for fuzzy sets defined by symmetric B-splines of order $k \geq 2$, it can be inverted in the sense that for a given w_i there exists a *unique* \mathbf{c}_i that will generate the desired output (see [32], for confidence learning rules). That is $c_{ij} = \mu_{B^j}(w_i)$, or the rule confidence represents the different grade of membership of the weight to the various fuzzy output sets. Clearly we can alternate between the weight or rule confidence space whilst preserving the encoded information. This is the only method which is consistent within the fuzzy methodology for mapping between weights and rule confidences.

It should also be emphasised that when using weights in place of rule confidence vectors provides a considerable reduction in both the storage requirements and the computational cost; this is also relevant to the discussion on adaptive network training given in Chapter 5.

When the fuzzy input set $\mu_A(\mathbf{x})$ is a singleton, the numerator and denominator integrals in (4.32) cancel to give

$$y^s(\mathbf{x}) = \sum_i \mu_{A^i}(\mathbf{x}) w_i, \quad (4.33)$$

where $y^s(\mathbf{x})$ is called the fuzzy singleton output. This is an important observation since $y^s(\mathbf{x})$ is a *linear* combination of the fuzzy input sets and does not depend on the choice of fuzzy output sets. It also provides a useful link between fuzzy and neural networks and allows both approaches to be treated within a unified framework. The reduction in the computational cost of implementing a fuzzy system in this manner and the overall algorithmic simplification is illustrated in Figure 4.12.

The analysis also illustrates how the centre of gravity defuzzification procedure *implicitly* imposes a partition of unity on the fuzzy input membership functions. Consider the above system when the fuzzy input sets do not sum to unity, which could be due to their univariate shape or the operator used to represent fuzzy intersection. The output is then given by

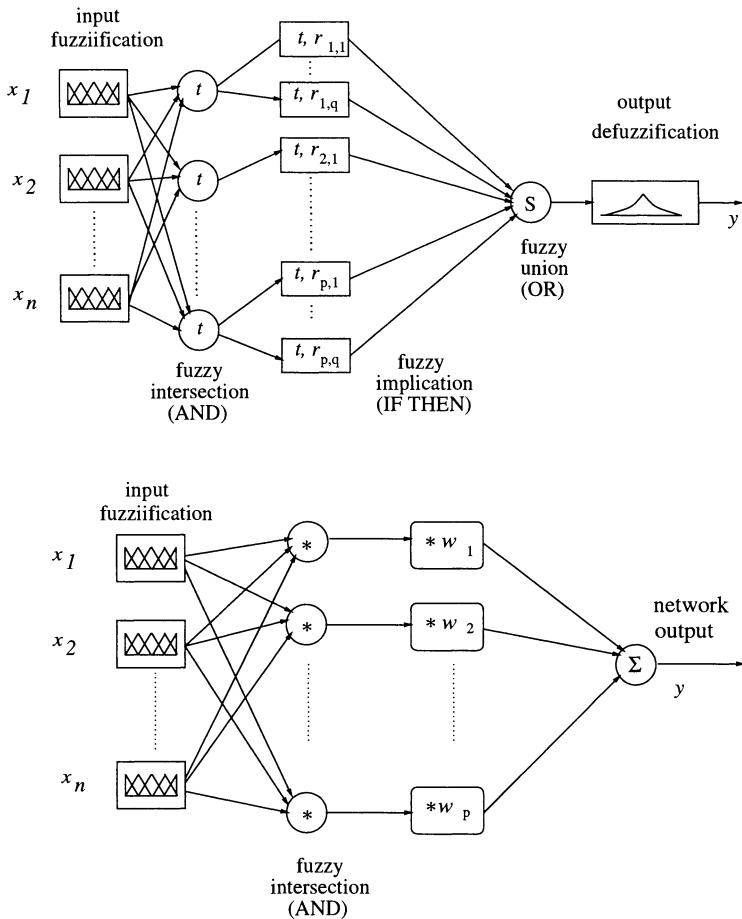


Fig. 4.12. An illustration of the information flow through a fuzzy system (*top*) and the resulting simplification (*bottom*) when algebraic operators are used in conjunction with a centre of gravity defuzzification algorithm, and the singleton input is represented by a crisp fuzzy set

$$y^s(\mathbf{x}) = \frac{\sum_i \mu_{A^i}(\mathbf{x}) w_i}{\sum_j \mu_{A^j}(\mathbf{x})} = \sum_i \mu_{\hat{A}^i}(\mathbf{x}) w_i, \quad (4.34)$$

where the normalised fuzzy input membership functions $\mu_{\hat{A}^i}(\mathbf{x}) = \mu_{A^i}(\mathbf{x}) / \sum_j \mu_{A^j}(\mathbf{x})$ form a partition of unity. This normalisation step is very important because it determines the *actual* influence of the fuzzy set on the system's output and can make previously convex sets non-convex.

Equation (4.34) leads to the following fundamental neurofuzzy modelling theorem that links fuzzy logic with neural network representations:

Theorem 4.1 [32] When algebraic operators are used to implement the fuzzy logic function, crisp inputs are fuzzified using singleton fuzzification and centre of gravity defuzzification is employed, the input–output functional mapping can be represented by a linear combination of the input fuzzy membership functions:

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^p \left(\frac{\mu_{A^i}(\mathbf{x})}{\sum_j \mu_{A^j}(\mathbf{x})} \right) w_i \equiv \sum_{i=1}^p \psi_i(\mathbf{x}) w_i.$$

This *neurofuzzy network* is simply a weighted sum of nonlinear normalised basis functions $\psi_i(\mathbf{x})$, for which the weights can be trained by a linear optimisation training algorithm (see Chapter 3). This neurofuzzy model satisfies the Stone–Weirstrass theorem and as such is a universal approximator and can hence approximate any continuous nonlinear function $f(\mathbf{x})$ defined on a compact domain with arbitrary accuracy.

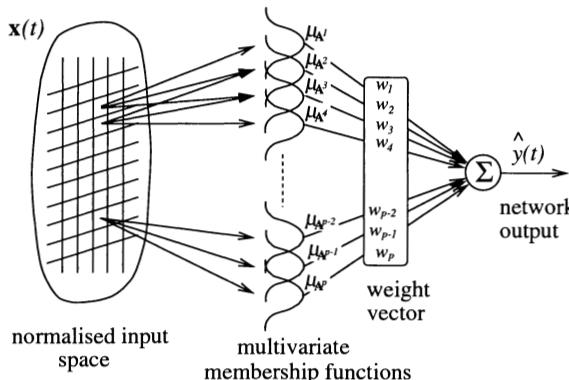


Fig. 4.13. The basic structure of a neurofuzzy network, resulting from Theorem 4.1

Figure 4.13 illustrates the basic structure of a generalised neurofuzzy network. Note that when the input to the fuzzy system is a fuzzy distribution rather than a singleton, it is possible to substitute (4.34) into (4.32) giving

$$y(\mathbf{x}) = \frac{\int_X \mu_A(\mathbf{x}) y^s(\mathbf{x}) d\mathbf{x}}{\int_X \mu_A(\mathbf{x}) d\mathbf{x}}. \quad (4.35)$$

The defuzzified output is a weighted average of the fuzzy singleton outputs over the support of the fuzzy input set $\mu_A(\mathbf{x})$ and the effect is to *smooth* or *low pass filter* the system's output y . This is illustrated in Figure 4.14. It can be seen that as the width of the fuzzy input set increases, the overall output of the system becomes *less sensitive* to the shape of either the input set or the sets used to represent the linguistic terms. However this is not always desirable as the output also becomes less sensitive to individual rules and the input variable, and in the limit as the input set shape has an arbitrarily large width (representing complete uncertainty about the measurement) the system's output will be constant everywhere.

4.4 Takagi–Sugeno local neurofuzzy model

A very popular approach to neurofuzzy modelling and control is to use local fuzzy modelling approaches [185], in which fuzzy rules are in the form:

$$IF (x_1 \text{ is } A_1^{j_1} \text{ AND } \dots \text{ AND } x_n \text{ is } A_n^{j_n}) \text{ THEN } (y \text{ is } \hat{y}_j(\mathbf{x})), \quad (4.36)$$

where $j = 1, 2, \dots, p$ is an index corresponding to the sequence $\{j_1, \dots, j_n\}$, with $j_i = 1, 2, \dots, m_i$, $p = \prod_{i=1}^n m_i$ and the output $\hat{y}_j(\mathbf{x})$ is a deterministic function. In practice $\hat{y}_j(\mathbf{x})$ is chosen as a linear combination of components of the input vector, i.e.

$$\hat{y}_j(\mathbf{x}) = w_{1j}x_1 + w_{2j}x_2 + \dots + w_{nj}x_n, \quad (4.37)$$

allowing linear stability control theory to be used. Using algebraic operators (product/sum), the truth value of the antecedent part of the fuzzy rule (4.36) is

$$\mu_j(\mathbf{x}) = \prod_{i=1}^n A_i^{j_i}(x_i).$$

Using centre of gravity defuzzification of the rule set (4.36) leads to the real output

$$\hat{y}(\mathbf{x}) = \frac{\sum_{j=1}^p \mu_j(\mathbf{x}) \hat{y}_j(\mathbf{x})}{\sum_{l=1}^p \mu_l(\mathbf{x})} = \frac{\sum_{j=1}^p \prod_{i=1}^n A_i^{j_i}(x_i) \hat{y}_j(\mathbf{x})}{\prod_{i=1}^n (\sum_{j_i=1}^{m_i} A_i^{j_i}(x_i))}. \quad (4.38)$$

If B-splines are used as fuzzy membership functions, i.e. $A_i^{j_i}(x_i) = B_{i,j_i}(x_i)$, which are defined by (4.8), then (4.38) becomes

$$\hat{y}(\mathbf{x}) = \frac{\sum_{j=1}^p \prod_{i=1}^n B_{i,j_i}(x_i) \hat{y}_j(\mathbf{x})}{\prod_{i=1}^n (\sum_{j_i=1}^{m_i} B_{i,j_i}(x_i))}. \quad (4.39)$$

Since B-splines have a partition of unity, i.e. $\sum_{j_i=1}^{m_i} B_{i,j_i}(x_i) = 1$, (4.39) is simplified to

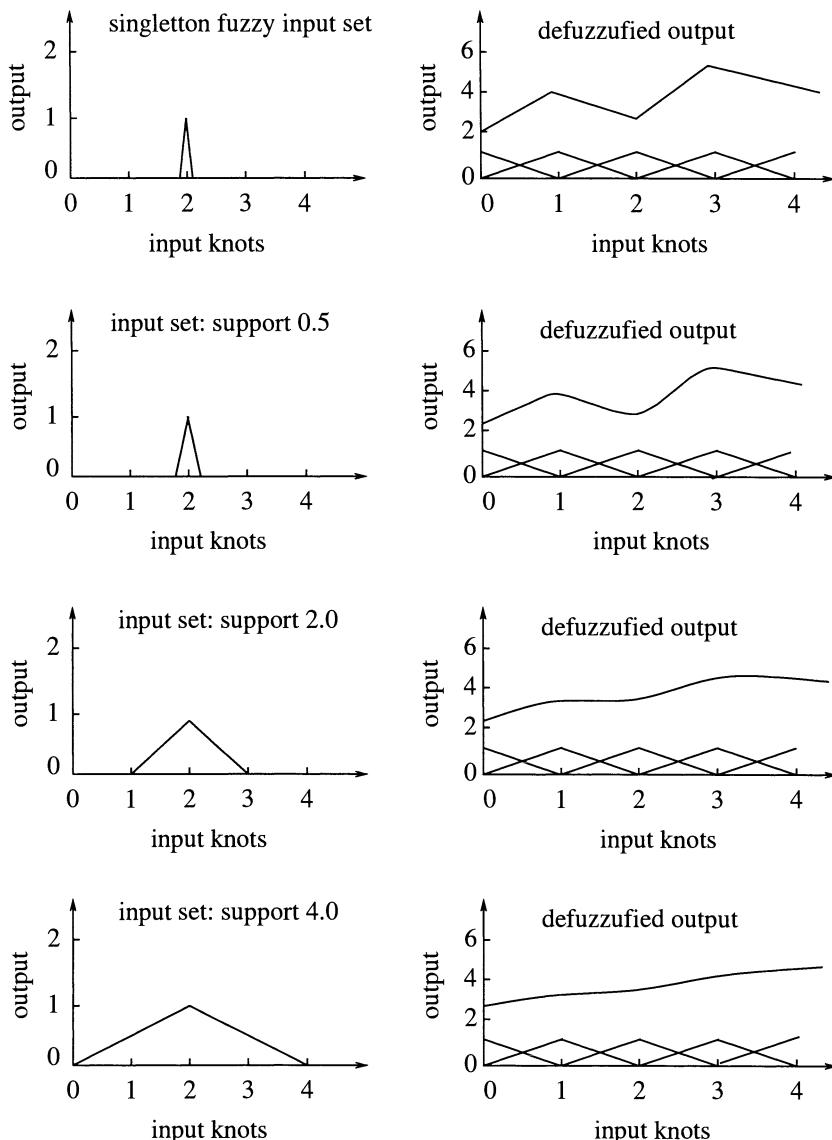


Fig. 4.14. Four fuzzy input sets and their corresponding defuzzified outputs, when the fuzzy rule base consists of triangular membership functions. The original triangular membership functions used to represent the linguistic terms are shown on the bottom of the graphs on the right and it can clearly be seen that as the width of the input set increases the system becomes less sensitive to the input variable and the set shapes

$$\hat{y}(\mathbf{x}) = \sum_{j=1}^p \prod_{i=1}^n B_{i,j_i}(x_i) \hat{y}_j(\mathbf{x}) = \sum_{j=1}^p N_j(\mathbf{x}) \hat{y}_j(\mathbf{x}). \quad (4.40)$$

Substituting (4.37) into (4.40) gives the following weighted linear model

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^n a_i(\mathbf{x}) x_i, \quad (4.41)$$

where the coefficients or weights $a_i(\mathbf{x}) = \sum_{j=1}^p N_j(\mathbf{x}) w_{ij}$ are effectively sub-neurofuzzy networks.

Takagi–Sugeno modelling has been proven popular due to the use of local linear models and the almost universal use of triangular (order 2) basis functions owing to their analytical tractability. Whilst local linear modelling is justified for many dynamic processes, restricting basis functions to order 2 is not justified unless the fundamental relationship is piecewise linear. One of the fundamental problems of Takagi–Sugeno models is how to decompose an *a priori* unknown global nonlinear system into a series of local models ((4.36) and (4.36)), which are minimum in number as well as with prescribed model approximation error. We consider this problem further in Chapter 6 within the context of a generalisation of Takagi–Sugeno model to operating point models (Section 6.3) and mixture of experts models (Section 6.5).

An obvious application of the local neurofuzzy model (4.41) is to time series modelling where $\mathbf{x}(t)$ is the set of model regressors $[y(t-1), \dots, y(t-n_y), u(t-1), \dots, u(t-n_u)]^T$. In this case, (4.41) becomes an ARMA type model. For further theory and applications, see Section 6.3.

4.5 Neurofuzzy modelling examples

4.5.1 Thermistor modelling

This is a simple static empirical data set (see Figure 4.15), from which we wish to find a set of fuzzy rules that relate temperature (*temp*) to resistance (*res*), for an electrician who finds exponential curves difficult to comprehend. Figure 4.16 are the *a priori* definition of fuzzy sets for the batch of unknown resistors. From the 100 data points of a sample of 10 thermistors, at 10 random temperatures in the range of 5 – 45°C, the neurofuzzy algorithm of Theorem 4.1 has produced the piecewise linear model of Figure 4.17.

This model can be represented as a set of (transparent) fuzzy rules, by transforming the weights into the associated rule confidences. Consider the rule antecedent (*temperature* is *cold*), the associated weight, w_2 , is fuzzified to produce the rule confidence $\mathbf{c}_2 = [0, 0.3, 0.7, 0, 0]^T$ as illustrated in Figure 4.17. These rule confidences represent the grade membership of the weight to the fuzzy output sets. Each weight produces two non-zero rule confidences, giving the following rule base:

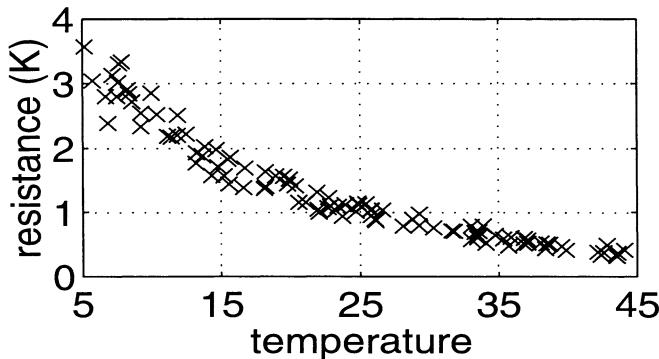


Fig. 4.15. Data produced from a sample of 10 thermistors

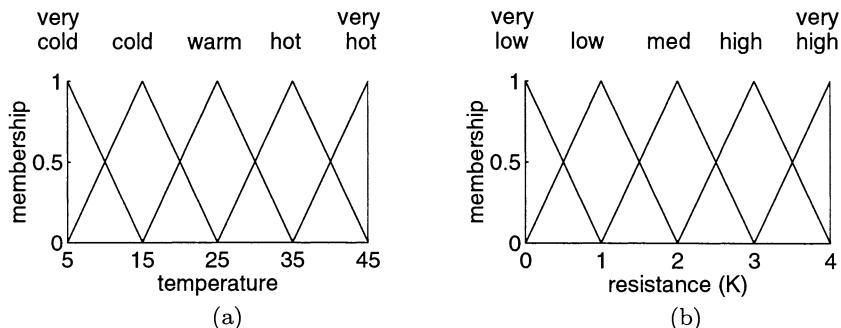


Fig. 4.16. Fuzzy sets used to characterise the behaviour of a batch of unknown thermistors, (a) temperature and (b) resistance of the thermistor

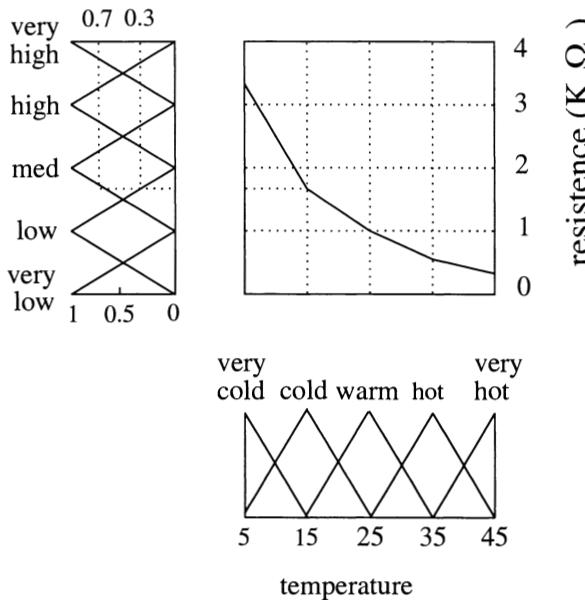


Fig. 4.17. The neurofuzzy models response across its input space. The fuzzy input and output sets are also shown, and the fuzzification of the weight, w_2 , associated with the fuzzy input set *cold* is demonstrated

IF (*temp* is *very cold*) THEN(*res* is *high*) 0.66
 OR IF (*temp* is *very cold*) THEN(*res* is *very high*) 0.34
 OR IF (*temp* is *cold*) THEN(*res* is *low*) 0.30
 OR IF (*temp* is *cold*) THEN(*res* is *med*) 0.70
 OR IF (*temp* is *warm*) THEN(*res* is *low*) 1.00
 OR IF (*temp* is *warm*) THEN(*res* is *med*) 0.00
 OR IF (*temp* is *hot*) THEN(*res* is *very low*) 0.41
 OR IF (*temp* is *hot*) THEN(*res* is *low*) 0.59
 OR IF (*temp* is *very hot*) THEN(*res* is *very low*) 0.68
 OR IF (*temp* is *very hot*) THEN(*res* is *low*) 0.32

For this example we have generated a small set of linguistic rules which make common sense. For each antecedent two rules can be excited with a total belief of unity. Also since order 2 (triangular) basis functions have been used the resultant ‘surface’ approximation (Figure 4.17) is piecewise linear.

4.5.2 Time series modelling

To illustrate how neurofuzzy models can be utilised in modelling dynamic systems, we introduce a simple 2-D benchmark time series, which has several

interesting properties such as regional data sparsity, easy visualisation, and a globally attracting limit cycle. Hence this example will be considered by a variety of modelling algorithms throughout the book. The time series is generated by the following equation:

$$\begin{aligned} y(t) = & [0.8 - 0.5 \exp(-y^2(t-1))]y(t-1) - [0.3 + 0.9 \exp(-y^2(t-1))] \\ & y(t-2) + 0.1 \sin(\pi y(t-1)) + N(0, 0.01). \end{aligned} \quad (4.42)$$

Figure 4.18 illustrates a noise free limit cycle version together with the associated modelling surface $\hat{y}(t) = f(y(t-1), y(t-2))$ that represents (4.36) but without the noise term $N(0, 0.01)$.

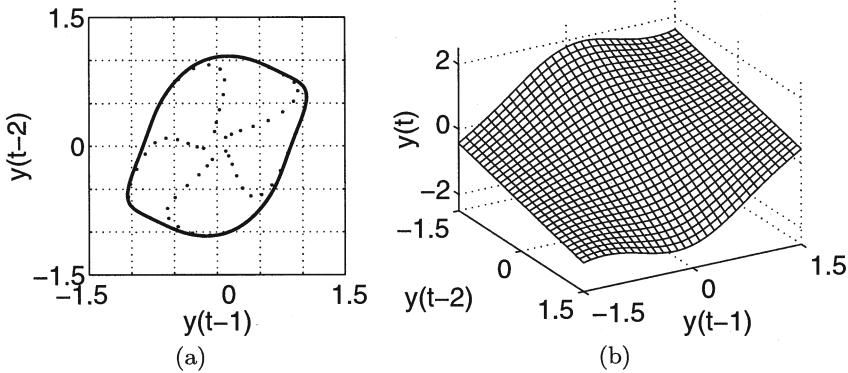


Fig. 4.18. The simulated time series; (a) the noise free limit cycle produced from the initial condition $[0.1, 0.1]^T$ and (b) the noise free output surface

Other data-based modelling algorithms such as radial basis functions [45], multilayer Perceptrons, CMAC [6], and local models [151] have been used to model the same time series. To model this time series, 300 noisy data pairs were generated, producing the noisy limit cycle of Figure 4.19 (a). The simplest model can be obtained from a neurofuzzy algorithm for which 7 conventionally triangular fuzzy sets are defined on each input $y(t-1)$ and $y(t-2)$, corresponding to a B-spline of order 2 with knot vector $[-1.5, -1, -0.5, 0, 0.5, 1, 1.5]^T$ on each axis. These sets/basis functions are shown in Figure 4.19(b). Stochastic normalised least squares [32, 100] and conjugate gradient have been used for training this example. The latter learning has produced a MSE across the training set of $8.2e-2$, suggesting that the model is too flexible, overfitting the data and generating too high a variance, resulting in poor generalisation (see Figure 4.19(d)). Despite this inability to generalise, this very simple model performs well where there is data (see Figure 4.19(c)). This is confirmed by the χ^2 correlation test, the correlation between the residuals $\{e(t), e(t-1)\}$ are shown in Figure 4.20(a). The k -step ahead prediction of the model is produced from a data test set of

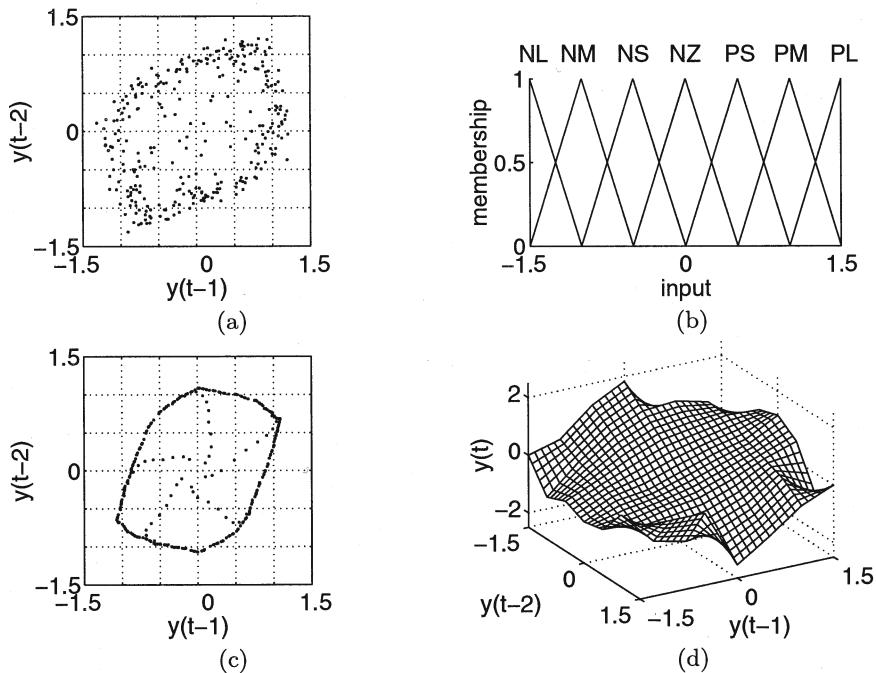


Fig. 4.19. Neurofuzzy modelling of the dynamics of the nonlinear time series, (a) the data set, (b) the fuzzy sets defined on each of the input variables, where the linguistic fuzzy set labels are: P=positive, N=negative, S=small, M=medium, and L=large, (c) the iterated limit cycle produced from the identified neurofuzzy model with the initial condition $[0.1, 0.1]^T$, and (d) the surface produced from this model

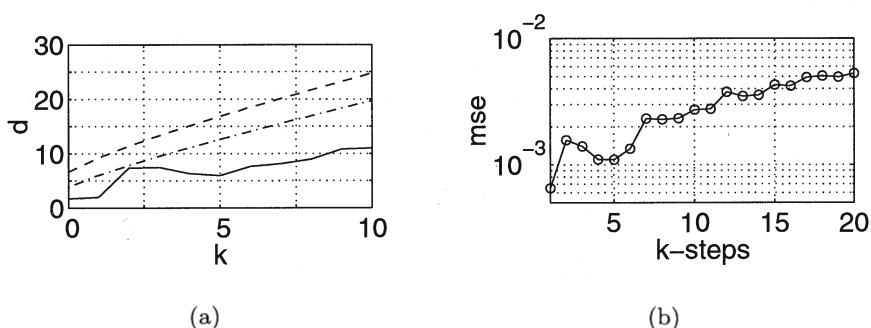


Fig. 4.20. Further time series tests; (a) the χ^2 test for correlation between $e(t)$ and $e(t - 1)$, and (b) the k -step ahead prediction errors across the noise free data set

1000 data points, as shown in Figure 4.20 (b), where the errors are seen to degrade gradually. From the above model, fuzzy rules can be constructed, which qualitatively describe the system's behaviour. Better modelling can be achieved if B-splines of order 3 are used [100], or we note from (4.36) that $y(t) = f(y(t - 1)) + g(y(t - 1), y(t - 2))$ and a model decomposition (construction) algorithm is used to exploit this prior knowledge. In Section 10.7, we compare several of the derived neurofuzzy modelling algorithms on this benchmark example in regard of model accuracy, generalisation ability, transparency and parsimony.

In the following chapters we derive a series automatic neurofuzzy model construction algorithms that determine parsimonious models (see Section 10.6 for modelling algorithm comparisons for this example).

5. Parsimonious neurofuzzy modelling

“The more complex, the sooner dead.” – a version of Parkinson’s third law

5.1 Iterative construction modelling

Whilst neurofuzzy systems have become an attractive powerful data modelling technique, combining well established learning laws of neural networks and the linguistic transparency of fuzzy logic, they do suffer from the curse of dimensionality. A network with two inputs with seven fuzzy sets per input produces a complete rule set of 49 rules, whereas a system with five inputs requires 16,800 ($\simeq 7^5$) rules or weights. Clearly conventional fuzzy and neurofuzzy systems are practically limited to low dimensional modelling. This drawback is a direct consequence of employing a lattice based partitioning strategy on the input space. Algorithms that do not use orthogonal split input lattice structures will potentially avoid this problem, however replacing it with another produces a lack of interpretability or transparency: the prime purpose of using fuzzy/neurofuzzy algorithms. However the modelling capabilities of any modelling technique are heavily dependent on their structure which must be determined similarly to its parameters, when little *a priori* knowledge is available. To solve high dimensional problems and yet retain all the desirable properties of neurofuzzy systems (e.g. linear-in-the-weights, transparency, partition of unity), some form of model complexity reduction must be performed, producing parsimonious models. Hence during model construction the following principles should be employed:

- Principle of data reduction: the smallest number of input variables should be used to explain the maximum amount of information;
- Principle of network parsimony: the best models are obtained using the simplest possible, acceptable structures that contain the smallest number of adjustable weights (so-called Occam’s razor or Einstein’s principle of simplicity).

A model’s accuracy and representational capability are determined by the structure of the model. The B-spline network makes this task easier as *a priori* information can be readily encoded, through choice of the shape (order)

and position of the individual univariate fuzzy sets (B-splines). Often such *a priori* knowledge is not available and the structure has to be determined from the data via some assumed model structure and *off-line* construction algorithms. In this chapter we consider several iterative neurofuzzy modelling algorithms that resolve the curse of dimensionality and yet retain all its desirable attributes.

Model construction algorithms which derive models from training data are typically iterative or stepwise in nature, starting with a simple model that encompasses any prior knowledge. Several refinements or modifications are made to this model, until some minimum in the desired cost function, $V(\cdot)$, is found. At each iteration, a set of candidate refinements are identified, and that refinement which produces the $\min V(\cdot)$ across the training data is used as the basis for further model development. This process is continued until the model of the optimal complexity and data fit is produced. One of the earliest iterative construction algorithm is the group method of data handling (GMDH) algorithm of [117]. GMDH automatically constructs a parsimonious regression model from the training data set. This basic model construction idea is illustrated in Figure 5.1.

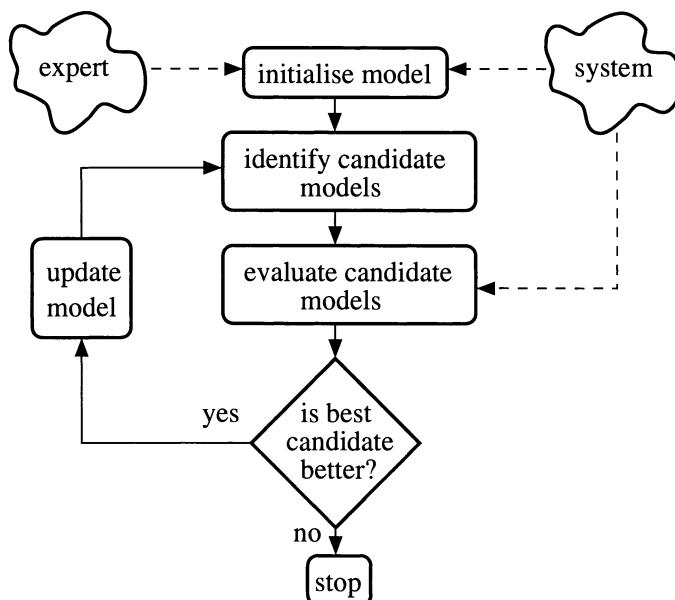


Fig. 5.1. Basic iterative model search algorithm

In general iterative neurofuzzy model construction algorithms have four steps:

1. *Initiate the model.* As a neurofuzzy model can be interpreted as a set of linguistic fuzzy rules, *a priori* expert knowledge of this form can be directly encoded into this initial model. *A priori* knowledge in the form of tables, surfaces, relationships can also usually be encoded into a set of fuzzy rules.
2. *Perform model step refinements.* The current model is refined step by step to produce a new model. Each step refinement changes the complexity and flexibility of the current model in a search aimed at improving its modelling capability. There are two distinct types of step refinement:
 - *Growing.* Usually model construction algorithms are concerned with growing refinements that increase the model complexity by introducing new degrees of freedom through additional parameters. As the algorithm objective is also to determine a parsimonious structure it should minimise the complexity of the structure through:
 - *Pruning* refinements which reduce model complexity by removing redundant or superfluous rules or correcting earlier erroneous refinements.
3. *Model training.* Each of the candidate models produced from the above refinements are trained on the training data set to find the appropriate set of model parameters. As neurofuzzy models are linear-in-the-parameters, this is a linear optimisation task (see Chapter 3).
4. *Model selection.* From the set of candidate models, one must be selected to replace the current model. This selection must achieve a compromise between model accuracy and model complexity (i.e. resolving the bias-variance dilemma). A model's bias can be reduced by increasing model complexity, however this leads to increased model variance so that the expected model error across all possible training set increases. There are several techniques that try to find the balance between the bias and variance of the final model:
 - *Cross validation* is the commonest method, where an independent test data set is generated to provide an accurate estimate of the model prediction risk

$$P_R = \frac{1}{N'} \sum_{i=1}^{N'} (y'(i) - \hat{y}(\mathbf{x}'(i)))^2, \quad (5.1)$$

where $\{\mathbf{x}'(i), y'(i)\}_{i=1}^{N'}$ are new observations independent of the training data. In practice the availability of this extra data is limited, and a test set cannot be realistically generated.

- *Statistical significance metrics* based on information theory (see Section 2.7) such as Akaike information criterion (AIC), Bayesian statistical significance measure (BSSM), and structural risk minimisation (SRM) provide effective heuristics for measuring network parsimony. These metrics combine the usual mean squared error (MSE)

$$MSE = \frac{1}{N} \sum_{t=1}^N (y(t) - \hat{y}(\mathbf{x}(t), \mathbf{w}))^2$$

across the training set with some measure of the model complexity. As an example consider the BSSM given by

$$BSSM = N \ln(MSE) + p \ln(N), \quad (5.2)$$

where p is the number of model free parameters. The Bayesian metric is the only statistical significance measure that does not require the modeller to provide an arbitrary constant or weighting parameters, however it does not make assumptions about the underlying data distribution.

- *Regularisation techniques* are obvious statistical methods to avoid model overfitting (see also Section 2.5). This inability to generalise is caused by the model fitting the noise in the training data and consequent failure to capture the underlying process, through too many degrees of freedom. Regularisation involves adding a constraint to the MSE through

$$V_R(\mathbf{w}) = MSE + \lambda V_{\mathbf{w}}(\mathbf{w}), \quad (5.3)$$

where λ is a hyperparameter or regularisation parameter which controls the degree by which the regularisation term influences the derived model. This constraint is typically a smoothness term related to the second derivative of the model (i.e. $V_{\mathbf{w}}(\mathbf{w}) = E\{d^2\hat{y}(\mathbf{x}, \mathbf{w})/dx^2\}$) or a constraint on the amplitude of the network's weights (i.e. $V_{\mathbf{w}}(\mathbf{w}) = \mathbf{w}^T \mathbf{K} \mathbf{w}$). Regularisation is a good method for controlling the ability of models to generalise, and generally should be included in construction algorithms when training the model.

In this chapter we consider general neurofuzzy model construction and introduce some of the relevant alternative approaches for generating parsimonious neurofuzzy representations which inherently exploit structural redundancy. Whilst the approach is generic, no modelling strategy will be appropriate to every problem. One of the simplest methods for trying to alleviate the curse of dimensionality is to decompose the model into a set of smaller (lower dimensional) submodels on the principle of divide and conquer. This decomposition in its simplest form is either a set of additive (see Section 5.2) or multiplicative (see Section 5.4) submodels.

5.2 Additive neurofuzzy modelling algorithms

An obvious approach to dealing with a multivariate function $f(\mathbf{x})$ is to decompose it into a sum of lower dimensional functions. One such decomposition is the analysis of variance (ANOVA) expansion given by

$$f(\mathbf{x}) = f_0 + \sum_{i=1}^n f_i(x_i) + \sum_{i=1}^{n-1} \sum_{j=i+1}^n f_{ij}(x_i, x_j) + \dots$$

$$+f_{1,2,\dots,n}(x_1, \dots, x_n). \quad (5.4)$$

This decomposition describes the additive decomposition in simpler subfunctions, which solves the curse of dimensionality in two ways: (i) for many functions certain interactions and inputs are redundant; (ii) correlations between many inputs are within the error bars of model approximation (see Section 5.6.4). For neurofuzzy modelling based on a lattice structure, interactions usually should be less than or equal to four input variables to avoid a rule explosion. The ANOVA decomposition reduces a globally partitioned, lattice based neurofuzzy system into a sum of smaller lattice based neurofuzzy systems, retaining network transparency and linearity in the weights for parameter training. The weight vectors of each of the submodels can be concatenated and conventional training algorithms for linear-in-the-parameters networks can be utilised (see Chapter 3). The advantage of the ANOVA decomposition can be easily illustrated by the well known Powell function

$$f(\mathbf{x}) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4. \quad (5.5)$$

It has four inputs (x_1, x_2, x_3, x_4) , so assuming 7 fuzzy sets (basis functions) on each input would require $7^4 \simeq 2400$ premise rules or weights, whereas the ANOVA decomposition produces four 2-D submodels requiring a total of $4 \times 7^2 \simeq 200$ weights for the same model quality, which in turn reduces the required amount of data necessary for parametric identification. Each of these 2-D models are readily interpretable either as surface plots or as a set of rules relating to each conditioned submodel or to “slices” through each plot with its simplified conditioned rule set.

5.3 Adaptive spline modelling algorithm (ASMOD)

A model construction algorithm, adaptive spline modelling of observational data (ASMOD) has been derived by Kavli [129] to identify an ANOVA model decomposition from training data. This iterative algorithm generates a globally partitioned B-spline model, which is directly applicable to neurofuzzy models (see Section 4.2.1 on B-splines). Recent complementary approaches [25] have led to the following improved form of ASMOD.

5.3.1 ASMOD refinements

The ASMOD algorithm is an off-line iterative construction algorithm that uses *growing* and *pruning* in which a more complex model can always reproduce the current model exactly, a distinct advantage that B-splines have over alternatives such as RBFs. Within the improved ASMOD algorithm there are three refinement processes.

(i) Univariate addition/submodel deletion

A prior model is given the ability to additively include a new input variable, by the addition of a new univariate submodel (see Figure 5.2). Univariate submodels are only incorporated into the existing model if the new input is not in the current model. These univariate submodels are drawn from an initial model set or external store, which may be the encoding of prior knowledge about the process being modelled, e.g. output smoothness requiring third order B-splines. A typical example of an external store of univariate submodels is shown in Figure 5.2, each with a different distribution of second-order B-splines (or triangular fuzzy sets). Note that the univariate models can be of differing order to reflect prior knowledge about a particular input variable. For n_s submodels in the external store, with n inputs, the number of possible univariate additions is $n_s(n - n_m)$, for n_m the number of inputs of the current input.

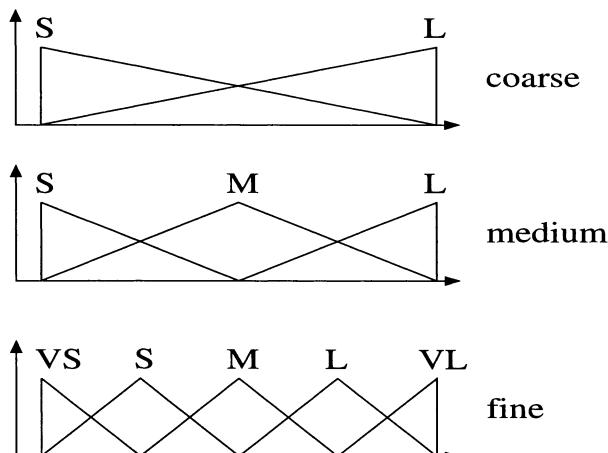


Fig. 5.2. An example of a univariate external store containing three densities of fuzzy sets: coarse, medium and fine

The corresponding pruning step is submodel deletion, where a superfluous submodel is removed from the current model. This is achieved by deleting each of the submodels in the current model, producing a set of candidate evaluation models, each containing one less submodel than the current model. These are trained on the training data set to see if there is any submodel that can be deleted.

(ii) Tensor multiplication/tensor submodel splitting

Input variable interaction necessary to construct model submodels $f(x_1, \dots, x_j)$ are incorporated by tensor multiplication (fuzzy AND) of an existing submodel with a univariate submodel from the external store (see Figure 5.3). This refinement step is computationally expensive as for every iteration there are many different candidate models. If the submodels are tensor multiplied with the different fuzzy set distributions in the external store, the number of possible candidate refinements for replacing an existing submodel with n_u inputs is $n_s(\sum_{i=1}^{n-n_u} C_{n-n_u}^i)$. The original ASMOD algorithm overcame this computational problem by allowing tensor multiplication of *existing* submodels. This approach generates submodels with large numbers of inputs in the early stage of construction as well as preventing the same input occurring in different submodels. This is avoided in the improved ASMOD algorithm by restricting the search so that a model dimension can only increase by one at each iteration, and that all input variables present in the current model, but not present in a given submodel, are considered for tensor multiplication (provided that the candidate model is not redundant). This gives the number of candidate models of the order of $n_s(n_m - n_u)$.

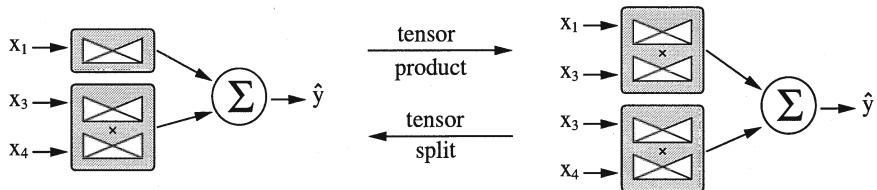


Fig. 5.3. An example of tensor multiplication and tensor splitting. Moving from left to right involves replacing the first submodel with one that has a dependency on x_1 and x_3 . Moving from right to left involves splitting the first submodel into its univariate components and since the new univariate model defined on x_3 is redundant it is removed

The corresponding *pruning* step is to replace a submodel with $n_u \geq 2$ inputs by submodels that depend on combinations of less than n_u inputs. As with tensor multiplication, tensor splitting (see Figure 5.3) leads to a large number of possible splits, however restricting the dimension reduction to one per iteration of any submodel significantly reduces the computational load. After splitting a submodel, any redundant submodels are removed.

(iii) Knot insertion/knot removal

Model flexibility of a submodel can be increased by the introduction of a new fuzzy set (or basis function) to one or more of the inputs (see Figure 5.4). This is achieved by inserting a new knot into one of its univariate knot vectors, creating a new set of fuzzy membership functions for that input variable. The shape of the new fuzzy set is predetermined by the order of

the existing fuzzy sets and their distribution on the domain of the input variable. All fuzzy sets which have non-zero grade of membership at the new knot location are automatically modified to ensure that a partition of unity is preserved. To reduce computation, the new knot is restricted to lie halfway between existing interior knots present in the submodel. The number of new rules produced by knot insertion is equal to the number of multivariate fuzzy sets produced from the tensor product of all the other fuzzy variables in the submodel, i.e. $\prod_{i=1, i \neq j}^{n_u} p_i$, where p_i is the number of fuzzy sets defined on the i th input, and j is the input index on which the new knot is inserted.

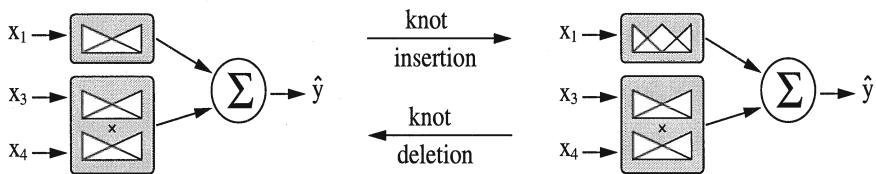


Fig. 5.4. An example of knot insertion and knot deletion. In moving from left to right a knot inserted in the first subnet, while moving from right to left knot deletion is performed

Knot deletion (see Figure 5.4) prunes out redundant submodel flexibility by removal of a knot from one of the submodel knot vectors (or equivalently removing a fuzzy membership function). Every interior knot in the entire ASMOD model is considered for knot deletion, resulting in several redundant multi-dimensional basis functions being removed.

Clearly even the improved ASMOD has potentially serious computational problems, which can be resolved by:

(i) Reducing basis function order

(Sub)models can be simplified by reducing the order of the B-spline membership functions defined on an input. In the limit the order can be reduced to 1, allowing the introduction of crisp decision boundaries, which may be useful in classification/identification applications where a system changes abruptly between operating conditions (e.g. as in a direct fuel injected engine). If there are no interior points, then for order 1 basis functions the dependency of this submodel on this input can be removed.

(ii) Stagewise construction

Restricting the number of refinements and their ordering reduces the number of possible candidate models at each iteration. Also by assuming that the external stores are sufficiently representative knot insertions can be avoided

until the overall model additive structure has been identified. Hence we recommended the following stagewise construction strategy in implementing ASMOD:

ASMOD Construction Modelling Algorithm

1. Identify the additive structure by using the univariate addition, tensor product and tensor splitting refinements (i.e. as in the original ASMOD algorithm).
2. Delete any redundant submodels by performing submodel deletion refinement.
3. Optimise the fuzzy set distributions by insertion and deletion of knots.
4. Check for redundant membership functions by performing the reduced order refinement.

This iterative model construction algorithm allows the user to review models as the refinement proceeds, allowing premature termination if a more complex but acceptable model is generated. The above improved ASMOD algorithm is available commercially as part of *Neuframe* (<http://www.demon.co.uk/neurosciences/>).

5.3.2 Illustrative examples of ASMOD

The first illustrative example is a good example to evaluate ASMOD, as it has been used by a variety of other model construction algorithms, each of the unknown submodels are 1-D or 2-D (i.e. easy to visualise), and the processes contain redundant inputs which would lead to serious overfitting for conventional neurofuzzy algorithm. The second illustrative example is a conventional nonlinear time series modelling problem that illustrates that ASMOD determines the correct underlying model structure, as well as generates a parsimonious representation. The third example, is as with the first, a surface fitting problem which illustrates the importance of failure margins in data modelling.

Example 5.1: 10-D function [73]

The 10-D function

$$\begin{aligned} y = f(\mathbf{x}) = & 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 \\ & + 0x_6 + 0x_7 + 0x_8 + 0x_9 + 0x_{10}, \end{aligned} \quad (5.6)$$

with an inherent additive structure was originated to evaluate the earliest adaptive spline modelling algorithm – MARS [73]. This surface fit problem is a good test for ASMOD as it can be seen that 5 of the 10 inputs (for which data is given) are uncorrelated with the output. A data set of any

100 data pairs were produced, the inputs for which were based on a uniform distribution over the 10-D uni-hypercube. Gaussian noise, $N(0, 1)$, was added to the training set, so that the true function accounts for 95% of the response variance. Model validation was achieved through a test set of 1000 noise free data pairs. The true model surfaces of the individual additive subcomponents of (5.6) are shown in Figure 5.5.

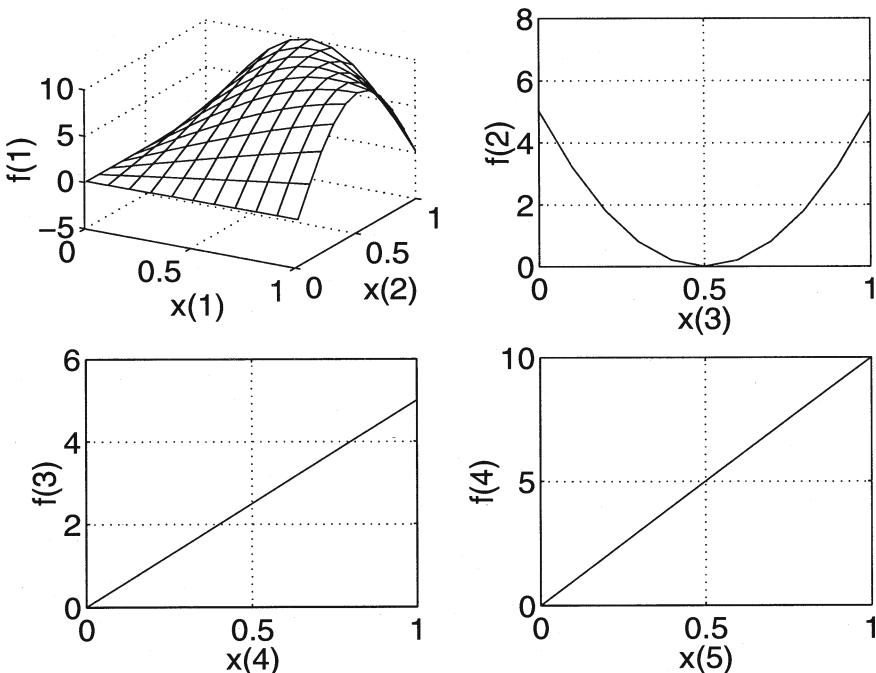


Fig. 5.5. The noise free surfaces of the individual additive components of (5.6)

Utilising the univariate, order 2, store of Figure 5.2, and the improved ASMOD algorithm above together with the MDL model selection criterion (see Section 2.7.1), the resultant ASMOD model is shown in Figure 5.6. This piecewise linear model was constructed after 11 model refinements. The MDL cost functional and associated number of model parameters (degrees of freedom) over these refinements are shown in Figure 5.7.

The 100 data pairs for training form an ideal sparse data set for evaluating ASMOD in the context of the differing model selection criteria of Section 2.7, and the order of external univariate basis functions used in modelling. Table 5.1 illustrates the sensitivities of the various model order selection criteria for sparse data set modelling, for piecewise linear (order 2), quadratic (order 3), and cubic (order 4) basis functions (see store order).

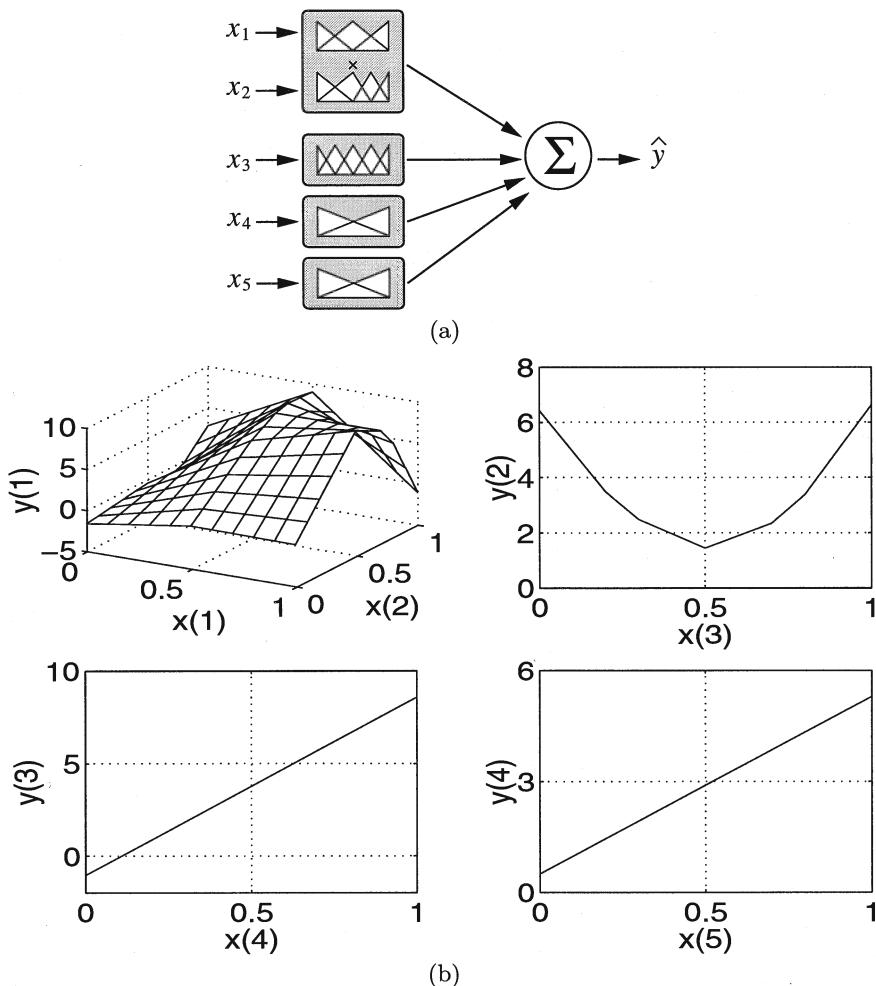


Fig. 5.6. Result from performing standard model identification with piecewise linear stores on the data set for the 10-D function; (a) illustrates the model structure and (b) shows the responses of the individual submodels which should be compared to Figure 5.5

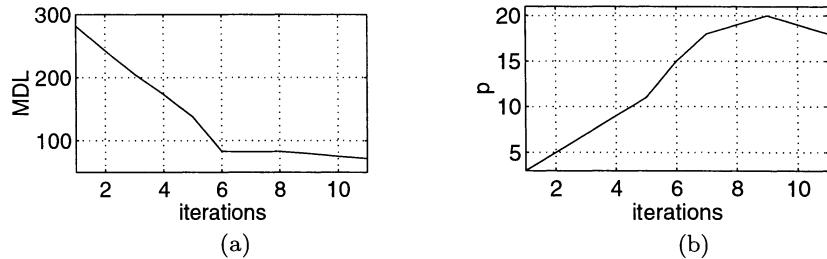


Fig. 5.7. Illustration of the refinement process; (a) shows the MDL and (b) shows the number of degrees of freedom of the current model during model identification

Table 5.1. Summary of the results of identifying additive models for the 10-D example

Selection criteria	Store order	No. of subnets	Degrees freedom	Train MSE	Test MSE	Correct structure
MDL	2	4	18	0.895	0.403	yes
MDL	3	4	13	0.813	0.148	yes
MDL	4	4	13	0.813	0.148	yes
AIC	2	4	97	$1.04e - 3$	$6.28e + 3$	no
AIC	3	4	26	$4.88e - 1$	0.841	no
AIC	4	5	98	$6.04e - 5$	62.3	no
FPE	2	4	35	$4.39e - 1$	3.23	no
FPE	3	5	24	0.596	0.544	no
FPE	4	5	94	$4.8e - 4$	$2.16e + 3$	no
GCV	2	4	23	0.678	0.452	yes
GCV	3	4	16	0.750	0.235	yes
GCV	4	4	19	0.750	0.235	yes
AIC _h	2	4	18	0.895	0.403	yes
AIC _h	3	4	13	0.813	0.148	yes
AIC _h	4	4	13	0.813	0.148	yes

Example 5.2: A nonlinear time series

Consider the time series

$$\begin{aligned} y(t) = & [0.8 - 0.5 \exp(-y^2(t-1))]y(t-1) \\ & -[0.3 + 0.9 \exp(-y^2(t-1))]y(t-2) + u(t-1) \\ & 0.2u(t-2) + 0.1u(t-1)u(t-2) + N(0, 0.01), \end{aligned} \quad (5.7)$$

where $u(t)$ is drawn from a uniform distribution over $[-1, 1]$. One thousand data pairs were generated to identify a model (see Figure 5.8). Using the improved ASMOD algorithm for an initial regressor input vector $\mathbf{x} = [u(t-1), \dots, u(t-4), y(t-1), \dots, y(t-4)]^T$, containing four irrelevant inputs, the relevant regressors and a simple piecewise linear model successfully extracted the process underlying characteristics, possessing the correct additive structure (see Figures 5.9 and 5.10).

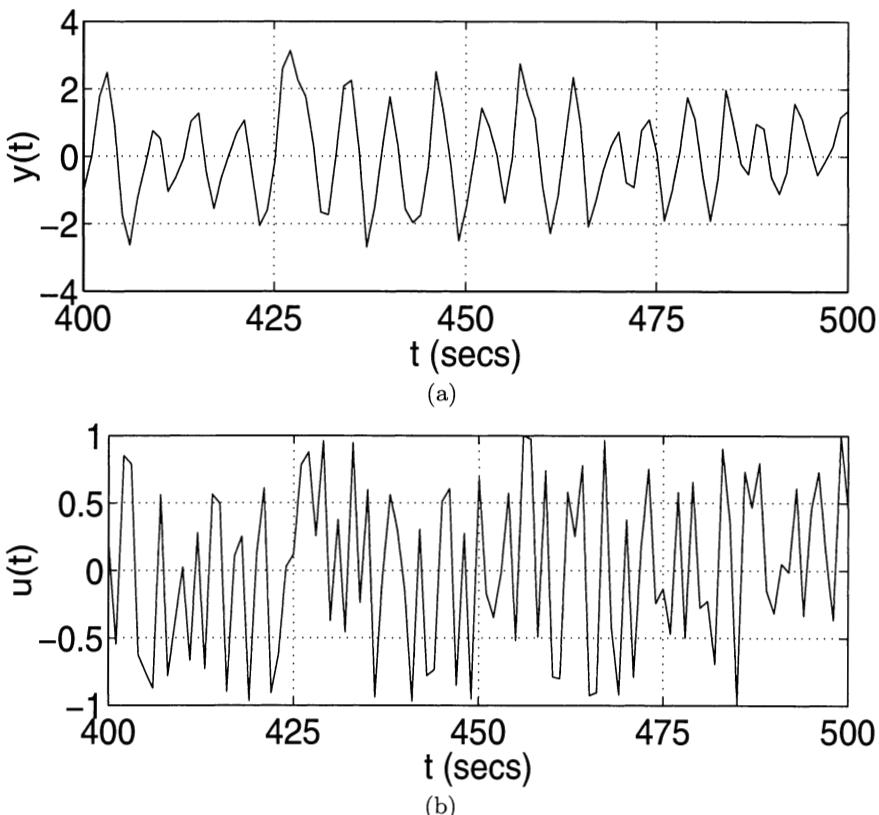


Fig. 5.8. Subsection of the data set produced for the simulated time series; (a) is the output, $y(t)$, and (b) is the random input, $u(t)$

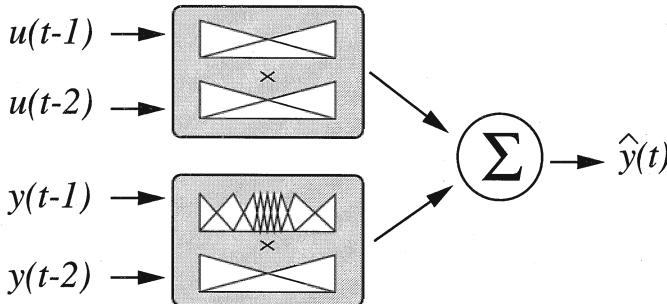


Fig. 5.9. The additive model identified for the nonlinear time series described by (5.7)

To validate the identified model, a noise free test set was generated. Performance of the model across the original training set and test set are shown in Figure 5.11. The MSEs across these data sets were $9.95e - 3$ and $7.99e - 4$, respectively, validating the derived model. As this is a dynamic process, k-step ahead prediction is an essential test criterion. Figure 5.12 illustrates the desired graceful degradation of the MSE of the model.

ASMOD has been extensively used in practical data analysis problems, including

- Autonomous underwater vehicle modelling [26].
- Automobile MPG modelling [34].
- Engine torque modelling [34].
- Al-Zn-Mg-Cu materials modelling [64].
- Car brake systems modelling [69].
- Aircraft flight control [2].
- Gas turbine engine diagnostics [29].
- Breast cancer diagnostics [30].
- Multisensor data fusion [95].
- Driver modelling [3, 96].
- Command and control [93].

ASMOD has the ability to construct parsimonious models from data, generating improved insights to the underlying process (so-called knowledge discovery) through its improved generalisation and interpretation abilities. The latter aspect achieves improved model insight through:

- Identification of the minimum number of important input regressors.
- Determination of simple and important submodel interactions, which, if the number of inputs are one, two, or three, have a direct graphical interpretation.
- Qualitative description of the system behaviour through fuzzy rules.

If fuzzy rules are required as a model output, then for computational reasons and the avoidance of nonsensical rules the model construction has to be

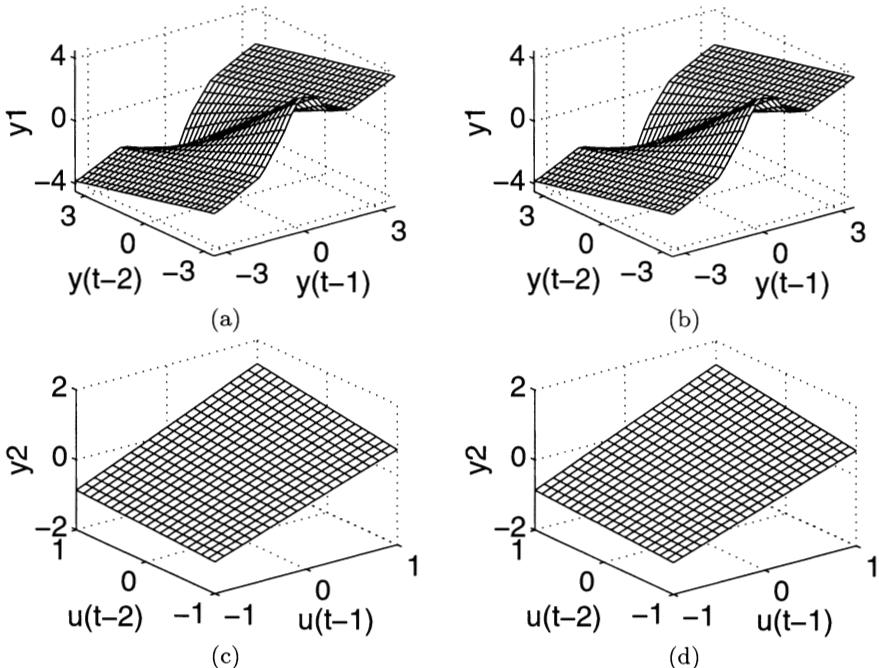


Fig. 5.10. Example 5.2: Comparisons of the subfunctions of the time series and the submodels of identified additive model; (a) and (c) are the true subfunctions and (b) and (d) are the approximations

restricted to user defined fuzzy membership functions (i.e. adaptation of knot positions is prohibited). Here there is a trade-off between model accuracy and user transparency. Any undesired redundancy introduced by this approach can be controlled by regularisation (see Sections 2.5 and 5.6).

A restrictive, one-step ahead, refinement algorithm such as ASMOD and its derivatives may get caught in a local minimum of the cost functional $V(\cdot)$, generating inadequate models. A genetic programming approach to ASMOD with a population of models has been developed [142], but is computationally expensive. An alternative computationally effective method to avoid local minima is to use failure margins. Here when a minimum is found in the ASMOD construction process, refinement continues until there has been a succession of failures, the number of failures being the failure margin.

Example 5.3: A 2-D anti-symmetric function

To illustrate the concept of failure margins in ASMOD, consider a simple 2-D anti-symmetric function

$$y = f(\mathbf{x}) = \sin(\pi x_1 x_2) + N(0, \sigma^2), \quad (5.8)$$

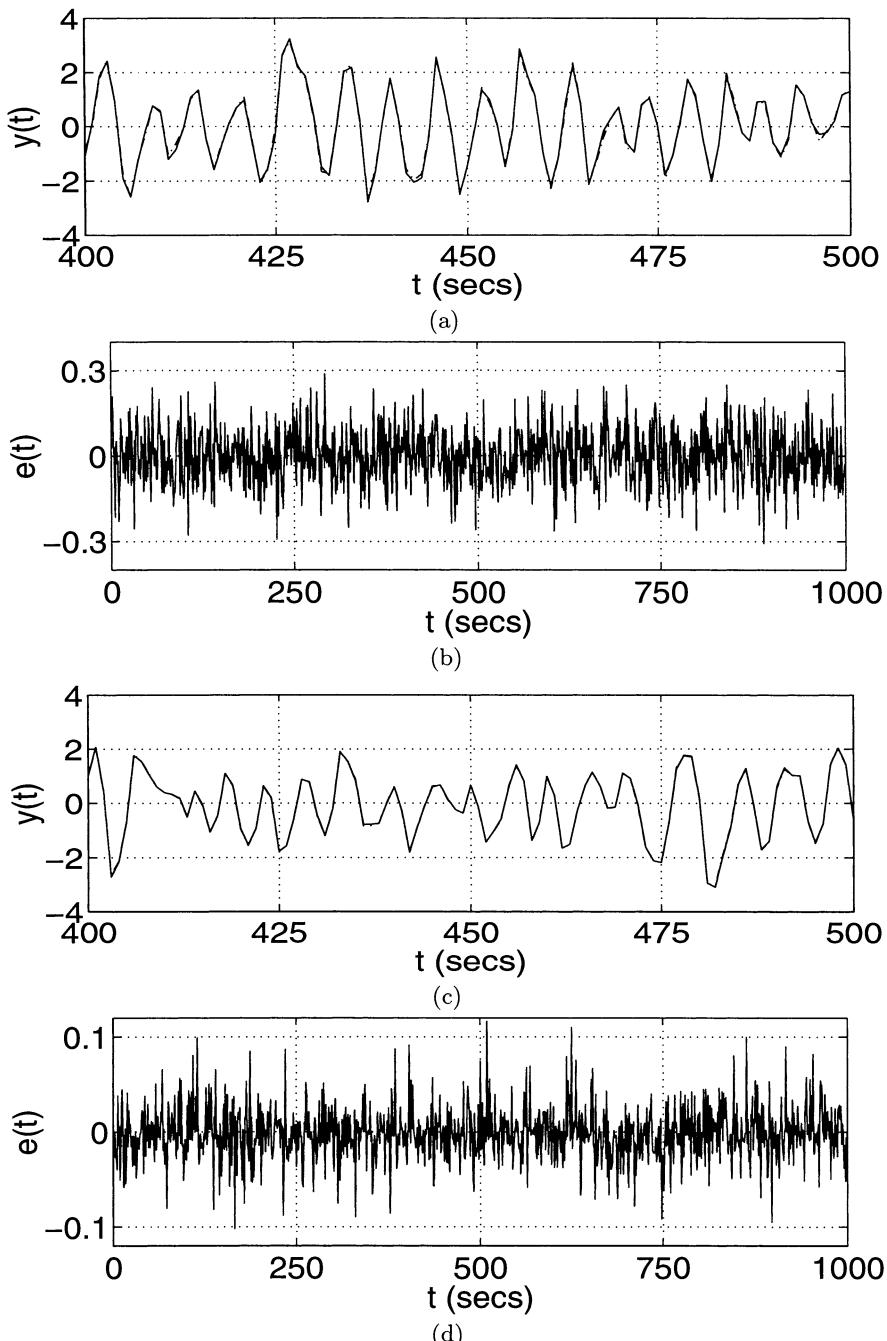


Fig. 5.11. Response of the identified model across both the training and test sets. (a) shows a subsection of the actual, dashed line, and predicted time series, solid line, across the training set, (b) shows the error across the training set, and (c) and (d) show the corresponding graphs for the test set. The sample variances of $e(t)$ across the train and test sets are $9.95e - 3$ and $7.94e - 4$, respectively

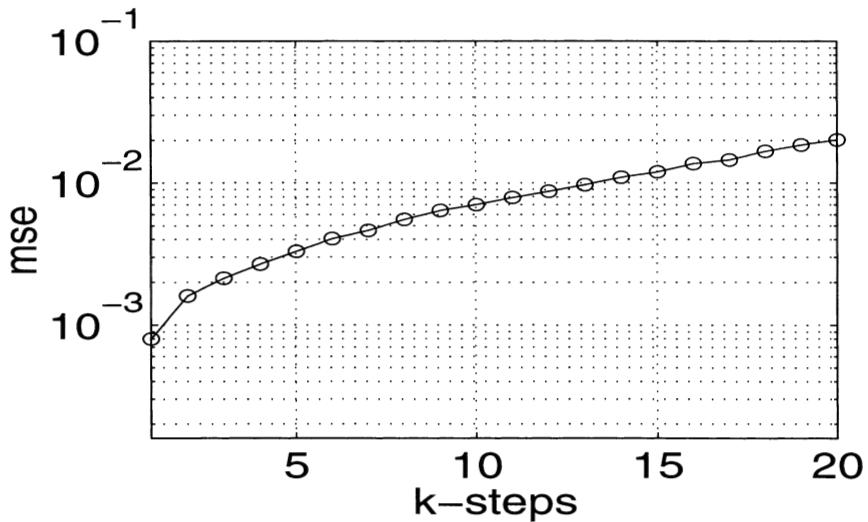


Fig. 5.12. k-Step ahead prediction on the test set

which is illustrated in Figure 5.13. From data drawn uniformly over the (x_1, x_2) plane, assuming no prior knowledge, the initial refinement of ASMOD is either a univariate addition of a submodel on either x_1 or x_2 . As the data is anti-symmetric, the expected value of either model is zero and its choice is dictated by the noise. Similarly for the next refinement, the addition of the other submodel has an expected value of zero and the increased degrees of freedom means that $V(\cdot)$ increases, failing to accept this refinement ensuring that the single univariate model will be pruned back to a bias term with zero mean! However, by allowing failures, the second submodel would be selected and its tensor product selected. This is illustrated in Figure 5.13 for 400 data pairs drawn uniformly on $[-1, 1] \times [-1, 1]$, and the MDL model selection criterion. The model construction history is shown in Table 5.2 where the MSE approaches the variance, $\sigma^2 = 0.01$, of the additive Gaussian noise.

5.4 Extended additive neurofuzzy models

A process which is inherently multiplicative cannot be adequately modelled as an additive model without significant order parameterisation. Analogously to additive models, multiplicative models decompose into several conventional neurofuzzy submodels, whose product forms the overall model. However, parameter identification is by necessity a nonlinear optimisation problem. The output of a multiplicative model is

Table 5.2. The construction history for the anti-symmetric function example. The different refinements are represented by: u/a for univariate addition, t/p for tensor product, k/i for knot insertion and k/d for knot deletion. The number of parameters is represented by p and $V(\cdot)$ represents the MDL model selection criteria

Refinement	p	MSE	$V(\cdot)$
u/a x_2	3	0.3950	-353.6
u/a x_1	6	0.3946	-342.0
t/p	15	0.0411	-1187.0
k/i x_1 @ 0.5	20	0.0201	-1442.3
k/i x_1 @ -0.5	25	0.0113	-1642.2
k/d x_2 @ 0.0	20	0.0114	-1668.0
k/d x_1 @ 0.0	16	0.0116	-1687.4

$$\hat{y}(\mathbf{x}) = f(\mathbf{x}) = \prod_{s=1}^S y_s(\mathbf{x}_s) = \prod_{s=1}^S \sum_{i=1}^{p_s} \mu_{A^i}^s(\mathbf{x}_s) w_i^s, \quad (5.9)$$

where $\mathbf{x}_s \in \mathbf{x}$, S is the number of multiplicative submodels, p_s is the number of multi-dimensional fuzzy membership functions ($\mu_{A^i}^s(\mathbf{x}_s)$) defined on the s th submodel, and w_i^s is the associated weight.

Multiplicative models described by (5.9) can be viewed as a method of constraining the tensor model's behaviour, since (5.9) can be rearranged to

$$\hat{y}(\mathbf{x}) = \sum_{k_1=1}^{p_1} \dots \sum_{k_S=1}^{p_S} \left[\prod_{s=1}^S \mu_{A^{k_s}}^s \right] \left[\prod_{s=1}^S w_{k_s}^s \right], \quad (5.10)$$

producing a tensor model whose weights are constrained to give a product based decomposition. This constrained model can produce a more parsimonious representation than the equivalent tensor model as the parameters are reduced from $\prod_{s=1}^S p_s$ to $\sum_{s=1}^S p_s$. This is a form of regularisation (see Sections 2.5 and 5.6) where the weights of a tensor model are constrained to form a product model.

As the individual submodels of a multiplicative model are conventional neurofuzzy models described by sets of fuzzy rules, the model output is described by the product of these defuzzified rules. This offers only limited transparency. Similarly a complete rule base would include antecedents that include all input variables that appear in the model, leading to redundant rules.

In Section 5.3, additive modelling introduced interacting variables through tensor products, which can be more parsimoniously modelled by multiplicative models. Exploiting the inherent redundancy of these additive neurofuzzy models leads to the generalised *extended additive neurofuzzy model* representation

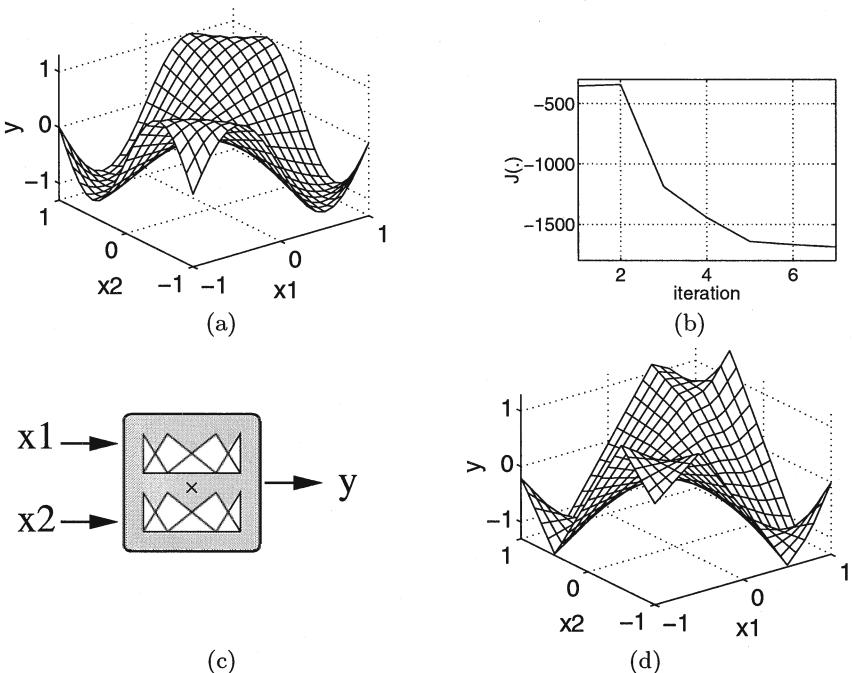


Fig. 5.13. Demonstration of the effectiveness of a failure margin; (a) shows the original anti-symmetric function from which data is generated, (b) is the MDL performance measure ($J(\cdot) = V(\cdot)$) of the model during the construction process, at the 2nd iteration the performance increases but the failure margin allows construction to continue, (c) the resulting model structure and (d) is the resulting piecewise linear surface producing a good approximation to (a)

$$\hat{y}(\mathbf{x}) = \sum_{s=1}^{S_t} y_s^t(\mathbf{x}_s, \mathbf{w}_s^t) + \sum_{s=S_t+1}^{S_t+S_p} y_s^p(\mathbf{x}_s, \mathbf{w}_s^p), \quad (5.11)$$

where the indexes t and p denote tensor and product submodels. This extended additive neurofuzzy model has the following properties:

- *Nonlinear-in-the-weights.* The nonlinear optimisation problem needs to be solved by say truncated Newton's method.
- *Model bias.* Product models, unlike tensor models, do not have a free bias term, so that when $V_N = 0$ an extra bias must be added to the model.
- *Parsimonious models* lead to improved generalisation through reduced degree of freedom.
- *Conventional additive equivalence.* As multiplicative models are equivalent to constrained tensor models, they can be described by more conventional additive decomposition (see (5.11)).
- *Simple model identification,* through a simple extension of the ASMOD algorithm.

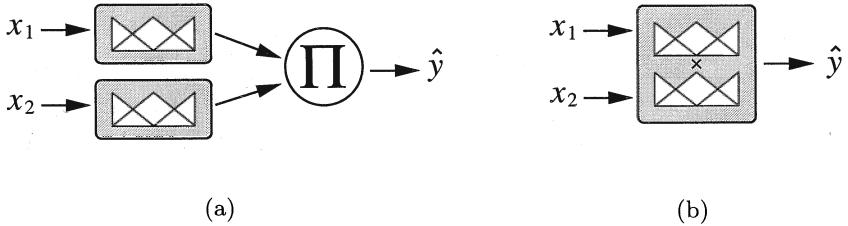


Fig. 5.14. Two potentially equivalent models; (a) is multiplicative model and (b) shows the tensor model which can be used to represent (a)

5.4.1 Weight identification

Given some data set $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$, the task is to find the weight vector that minimises

$$V_N(\mathbf{w}) = \frac{1}{N} \sum_{t=1}^N (y(t) - \hat{y}(\mathbf{x}(t), \mathbf{w}))^2.$$

But as $\hat{y}(\mathbf{x}(t), \mathbf{w})$ is nonlinear in \mathbf{w} , nonlinear optimisation is necessary. The Taylor series expansion of $V_N(\cdot)$ can be approximated by a quadratic in the weight change $\Delta\mathbf{w}$ as

$$V_N(\mathbf{w} + \Delta\mathbf{w}) = V_N(\mathbf{w}) + \Delta\mathbf{w} \frac{dV_N(\mathbf{w})}{d\mathbf{w}} + \frac{1}{2} \Delta\mathbf{w}^T \frac{d^2V_N(\mathbf{w})}{d\mathbf{w}^2} \Delta\mathbf{w},$$

minimising with respect to $\Delta\mathbf{w}$ gives the Newton search direction method [84] as

$$\frac{dV_N(\mathbf{w})}{d\mathbf{w}} = -\frac{d^2V_N(\mathbf{w})}{d\mathbf{w}^2} \cdot \Delta\mathbf{w} \quad or \quad \mathbf{g} = -\mathbf{H}\Delta\mathbf{w}. \quad (5.12)$$

The solution to Newton's method in practice is via a truncated conjugate gradient algorithm to solve (5.12) (for details, see [54]). To solve (5.12) requires the gradient, \mathbf{g} , and Hessian, \mathbf{H} , of the cost function $V_N(\cdot)$. By considering the concatenated weight vector \mathbf{w} , indexed by p and q , these derivatives for the Newton algorithm can be evaluated from $V_N(\cdot)$ as:

$$\frac{\partial V_N(\mathbf{w})}{\partial w_p} = -\frac{2}{N} \sum_{t=1}^N \frac{\partial \hat{y}}{\partial w_p}[y(t) - \hat{y}(\mathbf{x}(t), \mathbf{w})],$$

$$\frac{\partial^2 V_N(\mathbf{w})}{\partial w_p \partial w_q} = \frac{2}{N} \sum_{t=1}^N \left[-\frac{\partial^2 \hat{y}}{\partial w_p \partial w_q} [y(t) - \hat{y}(\mathbf{x}(t), \mathbf{w})] + \frac{\partial \hat{y}}{\partial w_p} \frac{\partial \hat{y}}{\partial w_p} \right],$$

respectively, where from (5.9) the output gradient with respect to the weight is given by

$$\frac{\partial \hat{y}(\mathbf{x}, \mathbf{w})}{\partial w_p} = \left\{ \begin{array}{l} \mu_{A^i}^s \\ \left[\prod_{\substack{s=1 \\ s \neq k}}^S \sum_{i=1}^{p_s} \mu_{A^i}^s w_i^s \right] \mu_{A^j}^k \end{array} \right. \begin{array}{l} \text{where } w_p \text{ refers to} \\ \text{the } i\text{th weight of the} \\ \text{sth tensor model} \end{array}$$

where w_p refers to
the j th weight in the
 k th submodel of one
of the product mod-
els (5.9).

Likewise the output Hessian with respect to the weights is

$$\frac{\partial^2 \hat{y}}{\partial w_p \partial w_q} = \left\{ \begin{array}{l} \left[\prod_{\substack{s=1 \\ s \neq m \neq k}}^S \sum_{i=1}^{p_s} \mu_{A^i}^s w_i^s \right] \mu_{A^j}^k \mu_{A^l}^m \\ 0 \end{array} \right. \begin{array}{l} \text{where } w_p \text{ and } w_q \text{ re-} \\ \text{fer to the } j\text{th weight} \\ \text{of the } k\text{th submodel} \\ \text{and the } l\text{th weight} \\ \text{of the } m\text{th submodel} \\ \text{of the same product} \\ \text{model, respectively.} \\ \text{otherwise.} \end{array}$$

Unfortunately for high dimensional problems in \mathbf{w} , the Hessian is computationally expensive to compute, much of this computation can be avoided if an initial weight vector can be selected close to the minimum that exploits the nonlinear additive structure of the model. Since the nonlinear additive model (5.10) can be represented by an equivalent additive model containing only tensor submodels whose weights can be efficiently computed by conjugate gradient, these weights can be then converted to the weight \mathbf{w} of the extended additive model to give an initial weight vector close to the minimum of $V_N(\cdot)$. In this conversion the weights of the S_t tensor models remain unchanged (except for scaling to ensure a partition of unity), while the weights of the S_p product models are found from several low dimensional nonlinear optimisations. For each product model, a tensor model given by

$$y^t(\mathbf{x}) = \sum_{k_1=1}^{p_1} \dots \sum_{k_S=1}^{p_S} \left[\prod_{s=1}^S \mu_{A^{k_s}}^s \right] \theta_i$$

(for $i = \sum_{s=1}^S (k_s - S + 1)$), needs to be approximated by

$$\hat{y}(\mathbf{x}) = \sum_{k_1=1}^{p_1} \dots \sum_{k_S=1}^{p_S} \left[\prod_{s=1}^{S_t} \mu_{A^{k_s}}^s \right] \left[\prod_{s=1}^{S_p} w_{k_s}^s \right] + b_0,$$

where b_0 is an additional weight used to represent the model's bias. This approximation can be found by identifying the weights $w_{k_s}^s$ and bias b_0 by minimising the cost function

$$V_c(\cdot) = \sum_{k_1=1}^{p_1} \dots \sum_{k_S=1}^{p_S} (w_i - [\prod_{s=1}^{S_p} w_{k_s}^s] - b_0)^2. \quad (5.13)$$

These weights can be then used as initial weights for complete model evaluation. The cost functional $V_c(\cdot)$ of (5.13) can be solved by the truncated Newton algorithm, minimising in the least squares sense, the difference between the tensor and product models. The desired Newton derivatives can be determined from (5.13), similarly to those of $V_N(\cdot)$ above. This technique is only justified if the computational cost of achieving it is compensated by the quality of the resultant initial weight vector and the subsequent saving in computing the overall extended additive model parameters. This is highly likely as the derived initial weights are in the correct region of the weight space to achieve rapid convergence via the truncated Newton algorithm.

5.4.2 Extended additive model structure identification

Similar to the ASMOD refinement principles, an efficient model construction algorithm for extended additive models can be derived. Clearly an iterative search through all extended additive submodels is infeasible, the efficient improved ASMOD algorithm to identify an ANOVA decomposition is used. Assuming that all tensor submodels are used to represent product functions that appear in the data, the ASMOD algorithm is extended to the further refinement of *product split*, as shown in Figure 5.15.

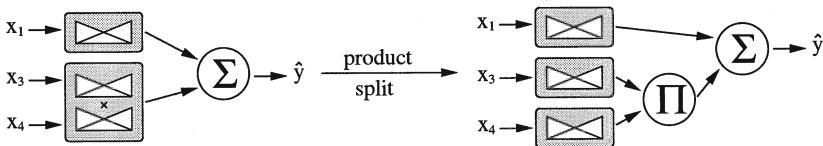


Fig. 5.15. A tensor submodel is split into a product model

To test for product submodels in the ANOVA expansion, existing tensor submodels are replaced by product models (see Figure 5.15). All except univariate submodels are considered for splitting, the number of membership functions defined on each input is unchanged. A tensor submodel with n inputs represented by

$$\hat{y}(\mathbf{x}) = \sum_{k_1=1}^{p_1} \dots \sum_{k_S=1}^{p_S} [\prod_{s=1}^S \mu_{A^{k_s}}^s] w_i,$$

where i labels distinct codes $[k_1, \dots, k_S]$, can be split into $\sum_{j=(n+1)/2}^n C_n^j$ different submodels. This prohibitively large number of submodels is prevented by only allowing tensor models to split into two submodels. Further splitting of the produced product submodel is allowed at further iterations.

Generally model selection criteria are based on the effective number of parameters whose calculation requires matrix inversion, as with ASMOD, the number of degrees of freedom are approximated by

$$p = \dim(\mathbf{w}) + 1 - S_t.$$

So given a nonlinear model structure, it is appropriate to heavily penalise product models unless they contribute significantly to model quality. Alternatively the cost function reduction of converting a tensor product submodel to product submodel relative to the extra computational cost could be used to determine if a product submodel is justified.

Example 5.4: A time series for extended additive modelling

Consider the following nonlinear time series

$$\begin{aligned} y(t) = & 0.5 - 3.5 \exp(-y^2(t-1)) \sin(0.9y(t-2)) + u(t-1) + \\ & 0.2u(t-2) + 0.1u(t-1)u(t-2) + N(0, 0.1). \end{aligned} \quad (5.14)$$

For an input $u(t)$ drawn from a uniform distribution over the interval $[-1, 1]$, a data set of 1,000 data pairs was generated from (5.14). Additionally a noise free test set of 1,000 data pairs was generated to validate derived models. The standard ASMOD algorithm (see Section 5.3) produced the correct regressors as inputs and additive structure as shown in Figure 5.16(a), with a MSE of $6.0e - 3$ across this test set. However by inspecting the derived submodel surfaces, as shown in Figure 5.17(a) and (c), the first submodel generalises poorly. This additive model was then used to initialise an extended submodel, producing the product model of Figure 5.16(b), which has half the number of free parameters of the additive model, with a MSE of $3.6e - 3$ across the test set. Figure 5.17(e) illustrates the improved generalisation capability of this extended additive model. The improved dynamics of this model is further demonstrated in the k-step ahead prediction on the test data of Figure 5.18.

5.5 Hierarchical neurofuzzy models

All the neurofuzzy models discussed have consisted of models which directly map the input variables to the outputs. Hierarchical models consist of several submodels, where the output of one submodel forms the input to another. The models represent *deep* knowledge and can be employed to approximate functions parsimoniously, where the number of rules is a linear function of the number of input variables. Hierarchical models, consisting of 2-D submodels, generate models with the minimum number of rules. Some multi-dimensional functions inherently possess a functional decomposition which can be exploited by such hierarchical models. Consider the following 4-D equation:

$$y = 0.5(x_1x_4)^2 \exp(x_3 \sin(x_1 + x_2)^3),$$

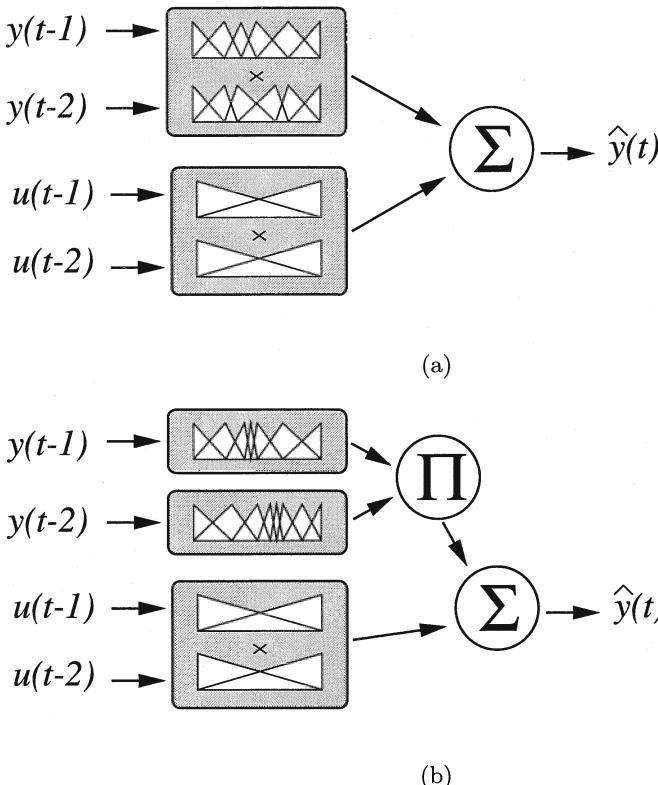


Fig. 5.16. Two identified models for modelling the dynamics of the nonlinear simulated time series; (a) model produced from ASMOD algorithm and (b) is the identified extended additive model

which can be functionally decomposed into the following form:

$$y = f(g_1(x_1, x_4), g_2(x_3, g_3(x_1, x_2))).$$

A functional decomposition of this form could be represented by the hierarchical model illustrated in Figure 5.19. Using the standard seven fuzzy membership functions defined on each input variable, roughly 200 different rule premises are formed to approximate this function. Conventionally the same level of approximation accuracy an additive representation would produce by a 4-D lattice based model, consisting of approximately 2,400. This hierarchical representation not only provides a substantial reduction in the complexity of the system, but also results in rules conveying *deep* system knowledge. The fuzzy rules produced by the conventional additive model would be *shallow*, not truly representing this system.

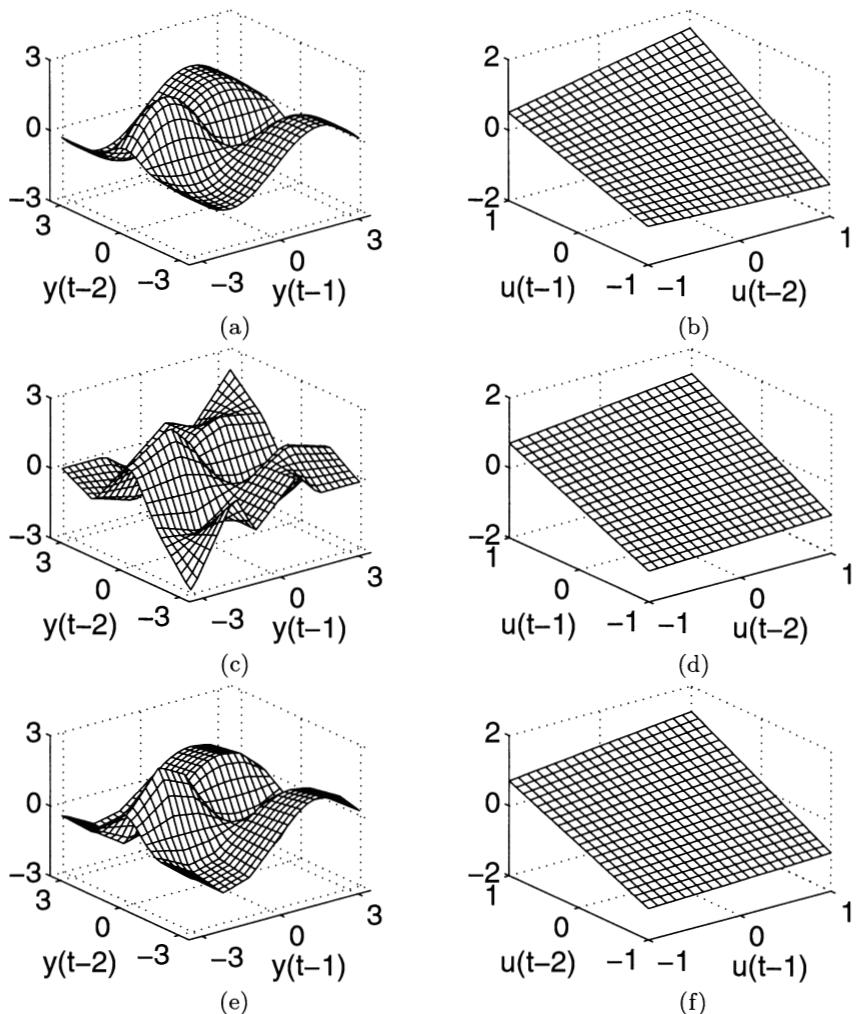


Fig. 5.17. Comparisons of the subfunctions of the time series and the submodels of identified additive models; (a) and (b) are the true subfunctions, (c) and (d) are the approximations produced from the additive neurofuzzy model constructed by ASMOD, and (e) and (f) are submodels of the identified extended additive model

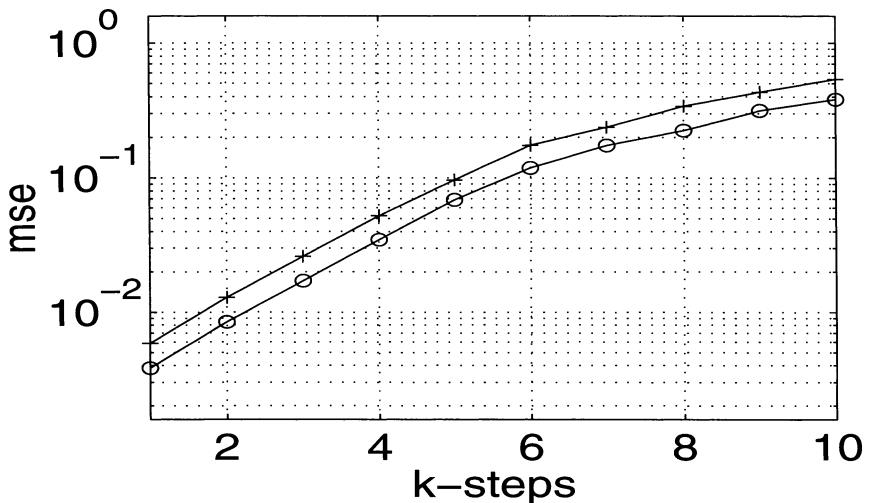


Fig. 5.18. k-Step ahead prediction on the test set, the + line is the conventional additive model and the o line is the extended model

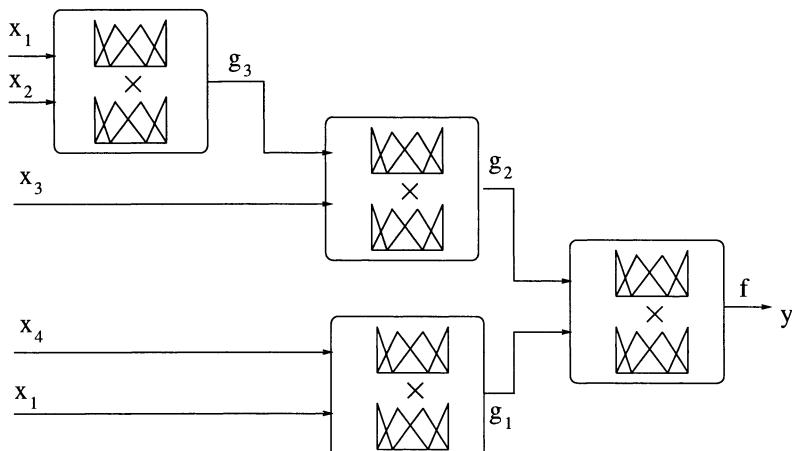


Fig. 5.19. Hierarchical decomposition

5.6 Regularised neurofuzzy models

In Section 2.5 we showed how for finite data sets regularisation produced parsimonious models whilst resolving the bias–variance dilemma. For sparse data sets neurofuzzy model construction algorithms produce restricted model structures with high model variance and concomitant poor generalisation. Hence we develop a form of regularisation where Bayesian estimation produces simple re-estimation formulae which identify a suitable bias–variance balance from the data. This approach is considered as a *post-processing* step to model construction, and an extension of Bayesian modelling technique by MacKay [140] to additive neurofuzzy models. Here by defining multiple weight priors, one for each submodel, traditional global regularisation is extended to local regularisation. One of the major problems in performing regularisation is deciding how much regularisation is necessary, i.e. deciding on the values of the regularisation coefficients or hyperparameters. A Bayesian approach is taken to evaluate them through maximisation of the evidence.

5.6.1 Bayesian regularisation

Given a data set $D_N = \{\mathbf{x}(t), y(t)\}_{t=1}^N$, inferences about the weight vector can be made, where weight identification can be viewed as finding the weight vector that maximises the *pdf* for the weights given the data, i.e. the MAP estimate (see Section 2.4). The *a posteriori pdf* is given by Bayes theorem (see (2.17)) as

$$p(\mathbf{w}|D_N) = \frac{p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \mathbf{w}) p(\mathbf{w})}{p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N)}. \quad (5.15)$$

The inferencing approach here is to find the weight vector that maximises this posterior *pdf*. The *likelihood* function $p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \mathbf{w})$ is the conditional *pdf* of the training set outputs given their associated inputs and weights. Assuming that

$$y(\mathbf{x}) = \hat{y}(\mathbf{x}, \mathbf{w}) + e,$$

where $e \sim N(\mu, \sigma^2)$, then under the central limit theorem for independent observations $(y(1), \dots, y(N))$, the likelihood function is given by

$$\begin{aligned} & p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \mathbf{w}) \\ &= \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left[-\frac{1}{2\sigma^2} \sum_{t=1}^N (y(t) - \hat{y}(\mathbf{x}(t), \mathbf{w}))^2\right]. \end{aligned}$$

The maximum likelihood (ML) identification algorithm finds the parameter vector \mathbf{w} , which maximises this likelihood function. This is formally equivalent to the LMS solution used above in the neurofuzzy model construction algorithm. The prior $p(\mathbf{w})$ serves as a solution to the inadequacy of ML estimation by controlling superfluous parameters, generating better

generalisation. This prior represents the most likely distribution of network weights before data is used. If no prior knowledge about these weights is known, then this ignorance is represented by a prior with a very large variance, and the posterior is dominated by the likelihood function, resulting in a MAP estimate becoming a ML estimate. Common priors of this kind include *pdfs* which enforce small weights and/or small output curvature (i.e. smoothness) that are highly probable (see Section 2.5.2). From Section 2.5, if the prior for the weights is Gaussian

$$p(\mathbf{w}) = \exp(-\alpha V_{\mathbf{w}}(\mathbf{w}))/Z_{\mathbf{w}},$$

and the MAP weight estimate minimises the regularised cost

$$V_R = \frac{\beta}{2N} \sum_{t=1}^N (y(t) - \hat{y}(\mathbf{x}(t), \mathbf{w}))^2 + \alpha V_{\mathbf{w}}(\mathbf{w}), \quad (5.16)$$

for $\beta = \sigma^{-2}$, the ratio $\lambda = \alpha/\beta$ is commonly known as the regulariser coefficient controlling the trade-off between the MSE and the regulariser term. To illustrate the effect of these priors, consider the simple neurofuzzy example of three triangular basis functions, as shown in Figure 5.20, to generate

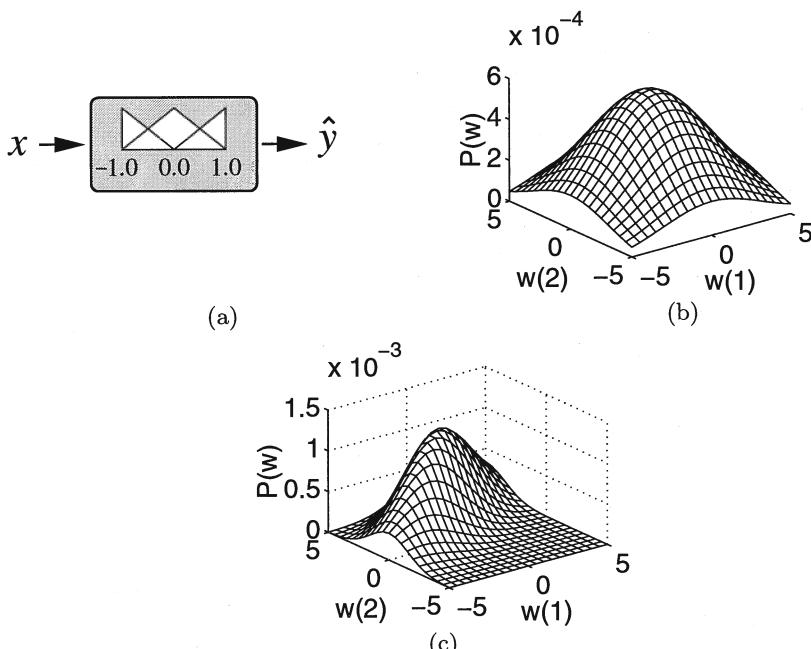


Fig. 5.20. Examples of the different prior distributions; (a) shows the univariate neurofuzzy model, with order 2 B-splines defined on the knot vector $(-1.0, -1.0, 0.0, 1.0, 1.0)^T$, (b) shows the prior with $\alpha = 0.10$ for zero-order regularisation and (c) shows the prior distribution with $\alpha = 0.10$ for second-order regularisation. For both priors w_3 is held constant at five

a piecewise linear output of Figure 5.21 for zero-order regularisation and second-order regularisation. Clearly the prior for zero-order regularisation has high confidence in small weights, whilst for second-order regularisation are those weights leading to a smooth output. The data from which the model was generated hardly excites the third basis function, so that the likelihood function would have little confidence in the weights associated with this basis function. Figure 5.21 clearly illustrates the manipulation of the consequent model by the different type of regulariser.

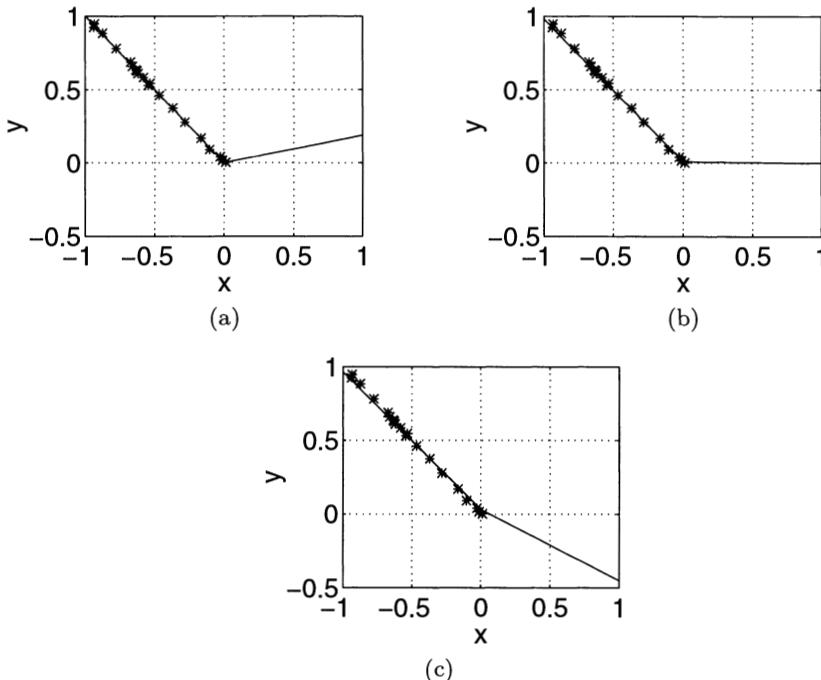


Fig. 5.21. Demonstration of applying the different priors shown in Figure 5.20; (a) shows the data and the model produced from ML estimation, while (b) is the model produced from a MAP estimate using the zero-order regulariser and (c) is the result using a second-order regulariser

The hyperparameters $\{\alpha, \beta\}$ control the balance between the MSE and the prior pdf , determining by how much the network weights are regularised or equivalently determining the consequent model complexity. Also as with model construction they are selected so as to achieve optimal model generalisation capability. In the restricted problem of finding the weight vector which minimises the cost function (5.16), only the single regularisation coefficient or smoothing parameter used, $\lambda = \frac{\alpha}{\beta N}$, has to be found. However in general β

estimates the noise variance and can be used to incorporate prior knowledge about this expected noise level. Equally, α represents the model complexity. Similarly to ML weight determination, the most likely hyperparameters can be found by maximising the posterior *pdf* for the hyperparameters:

$$p(\alpha, \beta | D_N, \mathcal{H}) = \frac{p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \alpha, \beta, \mathcal{H}) p(\alpha, \beta | \mathcal{H})}{p(D_N | \mathcal{H})}, \quad (5.17)$$

where \mathcal{H} is introduced to represent the model architecture/structure, and the type of regulariser used. The prior for the hyperparameters, $p(\alpha, \beta | \mathcal{H})$, can be assumed to be flat, i.e. a normal distribution with very large variance, and as $p(D_N | \mathcal{H})$ is simply a normalising constant, then the MAP estimate for the hyperparameters can be found by maximising the *evidence* given by $p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \alpha, \beta, \mathcal{H})$. This evidence is the normalisation coefficient of the posterior *pdf* for the weight given by

$$\begin{aligned} & p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \alpha, \beta, \mathcal{H}) \\ &= \int_{\mathbf{w}} \text{likelihood} \times \text{prior } d\mathbf{w} \\ &= \int_{\mathbf{w}} p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \mathbf{w}, \beta) p(\mathbf{w} | \alpha) d\mathbf{w}. \end{aligned} \quad (5.18)$$

Maximising the above evidence to find $\{\alpha, \beta\}$ is in general a nonlinear optimisation problem, but nevertheless is attractive, since evidence maximisation implicitly incorporates the principle of parsimony. An alternative approach would be to integrate out (or marginalise) the hyperparameters from the posterior weight *pdf* [36], but this leads to the minimisation of a nonlinear function in determining the MAP weight estimate.

5.6.2 Error bars

Regularisation produces sensible outputs throughout the input space irrespective of the data sparsity or magnitude of measurement noise, giving rise to unfounded confidence in the efficacy of the model. An indication of model belief can be estimated through error bars, derived from the variance of the posterior weight *pdf*, effectively representing confidence in the MAP weight estimates. Since $\text{cov}(\mathbf{w}) = \sigma^2 \mathbf{H}^{-1}$, the inverse Hessian \mathbf{H}^{-1} represents the uncertainty in the MAP weight estimate. For directions in the weight space for which the variance is high, the associated confidence in those weights is low. To provide output rather than weight variance, it needs to be translated from the weight space to the output space. For neurofuzzy models this is trivial since

$$\hat{y}(\mathbf{x}, \mathbf{w}_m) = \psi^T(\mathbf{x}) \mathbf{w}_m.$$

Hence the output variance due to weight uncertainty is:

$$\begin{aligned}\sigma_{\hat{y}}^2 &= \psi^T(\mathbf{x}) \mathbf{H}^{-1} \psi(\mathbf{x}) = \psi^T(\mathbf{x}) U \Lambda^{-1} U^T \psi(\mathbf{x}) \\ &= V^T \Lambda^{-1} V = \sum_{i=1}^p \lambda_i^{-1} v_i^2,\end{aligned}\quad (5.19)$$

where $\mathbf{H} = U \Lambda^{-1} U^T$ denotes singular value decomposition. $V = U^T \psi(\mathbf{x})$ is a rotated version of $\psi(\mathbf{x})$ giving the components parallel to the individual eigenvectors. Decomposing the output variance in this manner determines the contributions to variance by individual eigenvalues. For directions corresponding to large eigenvalues, the variances are small and in these directions there is high confidence in the model approximation. Also for zero-order regularisation ($\mathbf{K} = \alpha I$), the Hessian $\mathbf{H} = \beta \mathbf{A}^T \mathbf{A} + \alpha I$, so that the eigenvalues of \mathbf{H} are $(\lambda_j + \alpha)$, where λ_j are the eigenvalues of $\beta \mathbf{A}^T \mathbf{A} = \beta R$, i.e. the variance of the posterior is combination of the variance of the prior and the likelihood function. Increasing (decreasing) α increases (decreases) the confidence in the prior, moving the MAP estimated weight towards (away) from zero values. Consequent error bars are large when there is little confidence in both the prior and the data determined likelihood function. When there is little prior knowledge, the output variance is given entirely by data through (5.19) as

$$\sigma_{\hat{y}}^2 = \psi^T(\mathbf{x}) [\beta R]^{-1} \psi(\mathbf{x}). \quad (5.20)$$

Example 5.5: Curve fitting regularisation (see also Section 2.5 for bias-variance dilemma illustration with this example)

Consider the noisy nonlinear curve

$$y(x) = \exp(2x - 1) \sin(20(x - 0.6)^2) + N(0, 0.09)$$

modelled by an over-parameterised cubic B-spline with 43 basis functions evenly distributed across $[0, 1]$. The ML fit is shown in Figure 5.22(a), which is deliberately and clearly severely over fitted. Regularisation constrains the 43 weights to a near perfect fit (Figure 5.22(b)). Error bars are found by approximating the posterior *pdf* by the likelihood function (Figure 5.22(c)) via (5.20) or both the likelihood function and the prior (Figure 5.22(d)).

5.6.3 Priors for neurofuzzy models

Consider the additive model

$$\hat{y}(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^S \hat{y}_j(\mathbf{x}_j, \mathbf{w}_j), \quad (5.21)$$

where each $\hat{y}_j(\cdot)$ is a conventional neurofuzzy model. Zero-order regularisation is performed simply by setting $K = I$ in (2.34), constraining the amplitude of the weights (\mathbf{w}). A more powerful method is to choose a prior that produces smooth model outputs for both noisy and sparse data sets. The second-order

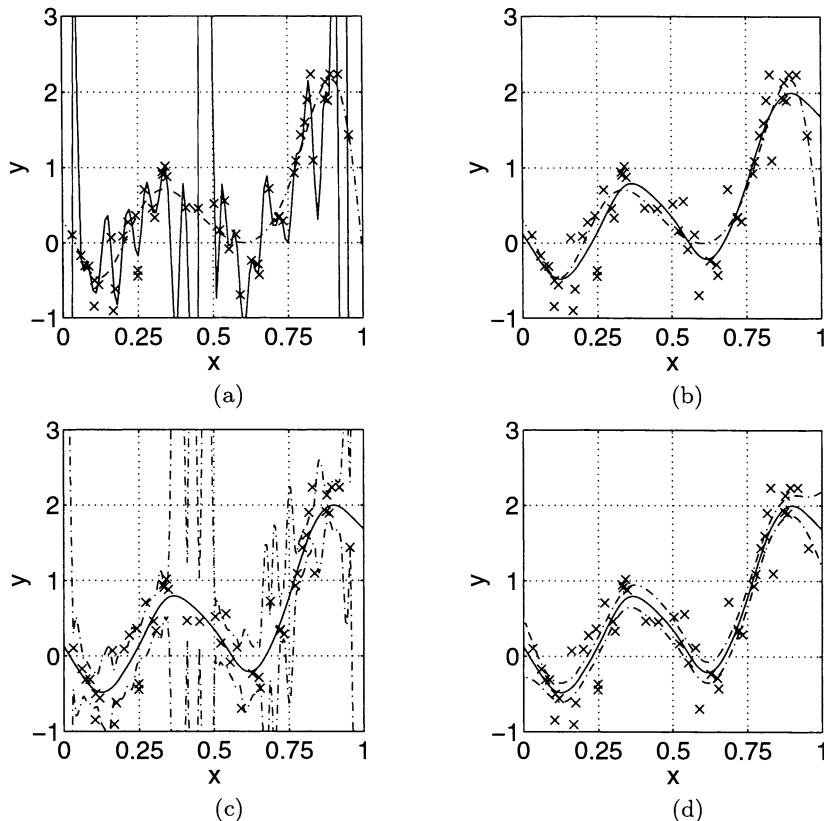


Fig. 5.22. An example demonstrating the different error bars; (a) is the ML fit to the data, where the dashed line represents the true function and (b) is the fit obtained from performing regularisation. (c) and (d) are used to represent the model's error bars, (c) shows those produced from approximating the posterior it pdf by the likelihood function and (d) demonstrates the effect of the prior, in giving more confidence to the model's output

regulariser should represent the expected curvature of the models output across the input space \mathcal{X} , i.e.

$$V_{\mathbf{w}}(\mathbf{w}) = \int_{\mathcal{X}} \left| \frac{d^2 \hat{y}(\mathbf{x}, \mathbf{w})}{d\mathbf{x}^2} \right| p(\mathbf{x}) d\mathbf{x},$$

which can be approximated by the sum of the curvature at the centre of the weight so that

$$V_{\mathbf{w}}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^p \left| \frac{d^2 \hat{y}(\mathbf{c}_i, \mathbf{w})}{d\mathbf{x}^2} \right|^2, \quad (5.22)$$

where \mathbf{c}_i is a n -dimensional vector of the i th multivariate membership function defined in the input space \mathcal{X} , and $p = \sum_{j=1}^S p_j$ with p_j being the number of multivariate basis functions in the j th submodel of (5.21). The curvature at any point in the input space is

$$\frac{d^2 \hat{y}(\mathbf{c}_i, \mathbf{w})}{d\mathbf{x}^2} = \sum_{i=1}^p \frac{d^2 \mu_A^i(\mathbf{c}_i)}{d\mathbf{x}^2} w_i \equiv \mathbf{k}_i^T \mathbf{w}, \quad (5.23)$$

so that (5.22) becomes

$$V_{\mathbf{w}}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^p \mathbf{w}^T \mathbf{k}_i \mathbf{k}_i^T \mathbf{w} = \frac{1}{2} \mathbf{w}^T \mathbf{K} \mathbf{w}, \quad (5.24)$$

where \mathbf{K} represents the square of the sum of the curvature evaluated at the centres of the basis functions. Unfortunately $\text{rank}(\mathbf{K}) = p - S + 1$, and hence \mathbf{K} is nonsingular for $S > 1$, resulting in an improper prior. This is due to the structure of additive submodels where redundant bias terms are introduced with each submodel. A less fundamental, but important problem with the evaluation of \mathbf{K} is that the curvature needs to be computed at p different positions. For B-spline basis functions the required curvature is given by the difference of univariate basis functions (see (4.8)), but for high dimensional spaces the computation becomes prohibitively large. To overcome these problems, an approximation to the regulariser is made as from (5.21) and (5.22)

$$\begin{aligned} V_{\mathbf{w}}(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^p \left[\sum_{j=1}^S \frac{d^2 \hat{y}_j(\mathbf{c}_i, \mathbf{w}_j)}{d\mathbf{x}_j^2} \right]^2 \\ &= \frac{1}{2} \sum_{i=1}^p [\mathbf{k}_j(\mathbf{c}_i) \mathbf{w}_j]^2 \simeq \frac{1}{2} \sum_{i=1}^p \sum_{j=1}^S [\mathbf{k}_j(\mathbf{c}_i) \mathbf{w}_j]^2 \\ &= \frac{1}{2} \sum_{j=1}^S \mathbf{w}_j^T \left(\frac{p}{p_j} \right) \mathbf{k}_j \mathbf{w}_j, \end{aligned} \quad (5.25)$$

where cross product terms in the expansion of the square have been ignored, and \mathbf{k}_j represents the curvature of the j th submodel output squared. This

curvature is calculated at the centres of the p_j submodel (membership) basis functions. As the weight vector \mathbf{w} is the concatenation of the individual weight vector, the regulariser can be expressed in the form of (5.24) with \mathbf{K} a block diagonal matrix given by

$$\mathbf{K} = \begin{vmatrix} (\frac{p}{p_1})\mathbf{k}_1 & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & (\frac{p}{p_S})\mathbf{k}_S \end{vmatrix}.$$

This is attractive since it produces a nonsingular matrix and where the curvature is calculated at $\sum_{i=1}^S p_i$ input positions and is a linear functions of the number of submodel basis functions. To further improve computational efficiency of \mathbf{K} , each submodel curvature is for a n -dimensional model output

$$\frac{d^2\hat{y}_j(\mathbf{x}, \mathbf{w})}{d\mathbf{x}^2} \equiv \sum_{i=1}^n \frac{\partial^2\hat{y}_j(\mathbf{x}, \mathbf{w})}{\partial x_i^2} + [2C_n^2 \text{ cross product terms}],$$

where the number of the terms is exponential in the input dimension. However due to the lattice structure of neurofuzzy models, the cross product terms should be sufficiently regularised by only constraining the curvature parallel to the inputs. Therefore the output's curvature can be approximated by

$$\frac{d^2\hat{y}(\mathbf{x}, \mathbf{w})}{d\mathbf{x}^2} = \sum_{i=1}^n \sum_{b=1}^p \left[\prod_{j=1, j \neq i}^p \mu_{A_{k_j}^{b_j}}(x_j) \right] w_b \frac{\partial^2 \mu_{A_{k_i}^{b_i}}(x_i)}{\partial x_i^2}. \quad (5.26)$$

This approximation is now a linear function of the input.

The above type of second-order regularisation is global, resulting in one prior on the complete weight vector. However having decomposed an input-output model into an additive set of submodels, there is no reason to assume a similar smoothness requirement for each submodel. Alternatively, different priors can be placed on different areas of the input space or weights of the various submodels, giving a more flexible form of regularisation, known as local regularisation.

5.6.4 Local regularised neurofuzzy models

An additive (including extended) model structure is an appropriate format on which local regularisation can be performed, so that different priors can be assigned to different submodels resulting in a cost functional

$$V_R(\mathbf{w}, \alpha, \beta) = \frac{\beta}{2} \sum_{t=1}^N (y(t) - \hat{y}(\mathbf{x}(t), \mathbf{w}))^2 + \sum_{i=1}^S \alpha_i V_i^p(\mathbf{w}_i), \quad (5.27)$$

where each of the S submodels has its own regulariser. The model hyperparameters $\{\alpha, \beta\}$ can be found by the backfitting algorithm of [103] or by a Bayesian inferencing approach based on the *a posteriori pdf*

$$p(\mathbf{w}|D_N, \alpha, \beta) = \frac{p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \mathbf{w}) \prod_{i=1}^S p_i(\mathbf{w}_i | \alpha_i)}{Z(\alpha, \beta)}, \quad (5.28)$$

where $Z(\alpha, \beta) = p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \alpha, \beta)$ is a normalising constant, and the priors are defined as normally distributed *pdfs* given by

$$p_i(\mathbf{w}_i | \alpha_i) = \exp[-\alpha_i V_i^p(\mathbf{w}_i)] / Z_i^p(\alpha_i),$$

producing quadratic regularisers of the form:

$$V_i^p(\mathbf{w}_i) = \frac{1}{2} \mathbf{w}_i^T \mathbf{k}_i \mathbf{w}_i. \quad (5.29)$$

These independent, multiple priors can be augmented into a single prior as

$$\begin{aligned} p(\mathbf{w} | \alpha) &= \prod_{i=1}^S \frac{\exp[-\alpha_i V_i^p(\mathbf{w}_i)]}{Z_i^p(\alpha_i)} \\ &= \frac{\exp[-\sum_{i=1}^S \frac{1}{2} \alpha_i \mathbf{w}_i^T \mathbf{k}_i \mathbf{w}_i]}{\prod_{i=1}^S Z_i^p(\alpha_i)} \\ &= \frac{\exp[-\frac{1}{2} \mathbf{w}^T \mathbf{K} \mathbf{w}]}{Z(\alpha, \beta)}, \end{aligned} \quad (5.30)$$

where $\mathbf{K} = \text{diag}[\alpha_i \mathbf{k}_i]$.

Following Gull [89] and MacKay [140] the hyperparameters (α, β) can be found by maximising their *a posteriori* density $p(\alpha, \beta | D_N)$. Assuming that their prior is uniformly distributed, this is equivalent to maximising their evidence which is the normalising coefficient of (5.28) given by

$$\begin{aligned} p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \alpha, \beta) &= \int p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \mathbf{w}) \prod_{i=1}^S p_i(\mathbf{w}_i | \alpha_i) d\mathbf{w} \\ &= \frac{1}{Z_D(\beta) \prod_{i=1}^S Z_i^p(\alpha_i)} \int \exp[-V_R(\mathbf{w}, \alpha, \beta)] d\mathbf{w}, \end{aligned} \quad (5.31)$$

where $Z_D(\beta)$ represents the normalisation of the likelihood function, and $Z_i^p(\alpha_i)$ is the normalisation of the i th submodel prior. Assuming Gaussian models, they are given respectively by

$$\begin{aligned} Z_D(\beta) &= \int \exp[-\beta V_D(\mathbf{w})] d\mathbf{w} = (\frac{2\pi}{\beta})^{N/2}, \\ Z_i^p(\alpha_i) &= \int \exp[-\alpha_i V_i^p(\mathbf{w}_i)] d\mathbf{w}_i = (\frac{2\pi}{\alpha_i})^{p/2} \det(\mathbf{k}_i)^{-1/2}. \end{aligned}$$

By using a Taylor series expansion for $V_R(\mathbf{w}, \alpha, \beta)$ about the maximum *a posteriori* weight estimate \mathbf{w}_m , the evidence Eq. (5.31) becomes

$$\begin{aligned} p(\{y(t)\}_{t=1}^N | \{\mathbf{x}(t)\}_{t=1}^N, \alpha, \beta) &= \frac{\exp[-V_R(\mathbf{w}_m, \alpha, \beta)] (2\pi)^{p/2} \det(\mathbf{H})^{-1/2}}{Z_D(\beta) \prod_{i=1}^S Z_i^p(\alpha_i)}, \end{aligned} \quad (5.32)$$

where $\mathbf{H} = \beta \mathbf{A}^T \mathbf{A} + \mathbf{K}$ (see smoother matrix in Section 2.4), for $\mathbf{K} = diag[\alpha_i \mathbf{k}_i]$, is the Hessian of the cost functional (5.27). Taking the log of (5.32) yields the equivalent expression for the evidence

$$\begin{aligned} E(\alpha, \beta) &= -V_R(\mathbf{w}_m, \alpha, \beta) - \frac{1}{2} \log \det(\mathbf{H}) + \frac{N}{2} \log \beta \\ &\quad + \sum_{i=1}^S \left[\frac{p_i}{2} \log \alpha_i + \frac{1}{2} \log \det(\mathbf{k}_i) \right]. \end{aligned} \quad (5.33)$$

A simple re-evaluation formulae can be found for the hyperparameters (α, β) from (5.33) by differentiating with respect to α and β :

$$\frac{\partial E(\cdot)}{\partial \beta} = -V_D(\mathbf{w}_m) - \frac{1}{2} \text{tr}(\mathbf{H}^{-1} \mathbf{A}^T \mathbf{A}) + \frac{N}{2\beta},$$

$$\frac{\partial E(\cdot)}{\partial \alpha_i} = -V_i^p(\mathbf{w}_m) - \frac{1}{2} \text{tr}(\mathbf{H}^{-1} \mathbf{k}'_i) + \frac{p_i}{2\alpha_i}.$$

Setting them to zero and assuming $(\mathbf{H}, \mathbf{w}_m)$ are stationary, we then have

$$\beta^{\kappa+1} = \frac{N - \beta^\kappa \text{tr}(\mathbf{H}^{-1} \mathbf{A}^T \mathbf{A})}{2V_D(\mathbf{w}_m)}, \quad (5.34)$$

$$\alpha_i^{\kappa+1} = \frac{p_i - \alpha_i^\kappa \text{tr}(\mathbf{H}^{-1} \mathbf{k}'_i)}{2V_i^p(\mathbf{w}_m)}, \quad (5.35)$$

which converges to a consistent solution, for $\mathbf{k}'_i = \frac{\partial \mathbf{H}}{\partial \alpha_i}$.

The optimum hyperparameters α have a tendency to approach infinity for sparse data, and are therefore constrained via a heuristic so that $[\alpha_i/(N\beta)] \in [1, 0]$, restricting the equivalent smoother values to the range $[1, 0]$.

Note: From Section 2.5, the number of degrees of freedom (2.34), *dof*, of the model given the MAP estimate is $dof = \beta \text{tr}(\mathbf{H}^{-1} \mathbf{A}^T \mathbf{A})$, so that the β which maximises the evidence is the reciprocal of the unbiased noise on the data. Similarly, $\alpha_i \text{tr}(\mathbf{H}^{-1} \mathbf{k}'_i) = dof$ represents the effective number of parameters modelled by the i th prior.

Example 5.6: A revisit to Friedman's problem of Example 5.1

To evaluate the above model regularisation schema for additive models, consider the 5-D function

$$y(\mathbf{x}) = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5,$$

whose structure has been correctly found as

$$y(\mathbf{x}) = y_1(x_1, x_2) + y_2(x_3) + y_3(x_4) + y_4(x_5),$$

but contains many redundant degrees of freedom, so that $y_1(\cdot)$ is a 2-D piecewise quadratic neurofuzzy model with 144 basis functions on a uniform lattice, and $y_2(\cdot)$, $y_3(\cdot)$ and $y_4(\cdot)$ are all univariate piecewise quadratic neurofuzzy models of 12 basis functions each. Figure 5.23(b) shows the true submodels together with ML estimated unregularised submodels (Figure 5.23(a)), clearly illustrating the overfitting to data. Figure 5.24 illustrates results of global regularisation (Figure 5.24(a)) and local regularisation (Figure 5.24(b)). Note that the different offsets in these figures are due to the individual bias of each submodel and are not significant. Finally applying ASMOD directly to the data set of Example 5.1, produces the model response of Figure 5.25 for the globally regularised ASMOD. Table 5.3 illustrates the improvement in generalisation with reduced *dof*, caused by regularisation.

Table 5.3. Improvement in generalisation with reduced *dof* (where (ASMOD+G), (ASMOD+L) denotes ASMOD models regularised by one global regulariser and individual submodel local regularisers, respectively)

Method	Train MSE	Test MSE	<i>dof</i>
ML	$1.27e - 8$	$4.49e + 3$	176^8
global	$5.85e - 1$	$2.11e - 1$	31.8
local	$6.57e - 1$	$1.87e - 1$	25.9
ASMOD	$8.13e - 1$	$1.48e - 1$	13
ASMOD+G	$8.20e - 1$	$1.43e - 1$	12.6
ASMOD+L	$8.32e - 1$	$1.05e - 1$	9.1

Note: Local regularisation can be applied similarly to the extended additive models of (5.11), by defining priors for the output of each submodel. Each product model will have several priors, one for each of its submodels, giving a total of $\sum_{s=1}^{S_p} np_s + S_t$ priors. When each of these priors is defined as a multivariate normal distribution, the regularised cost function is

$$V_R(\cdot) = \frac{N(MSE)}{2} + \sum_s \mathbf{w}_s \mathbf{k}_s \mathbf{w}_s.$$

For this cost function the above theory for linear additive models applies directly. Practical experience shows that the inherent regularisation character of multiplicative models requires little regularisation, so that only tensor models in (5.11) need regularisation.

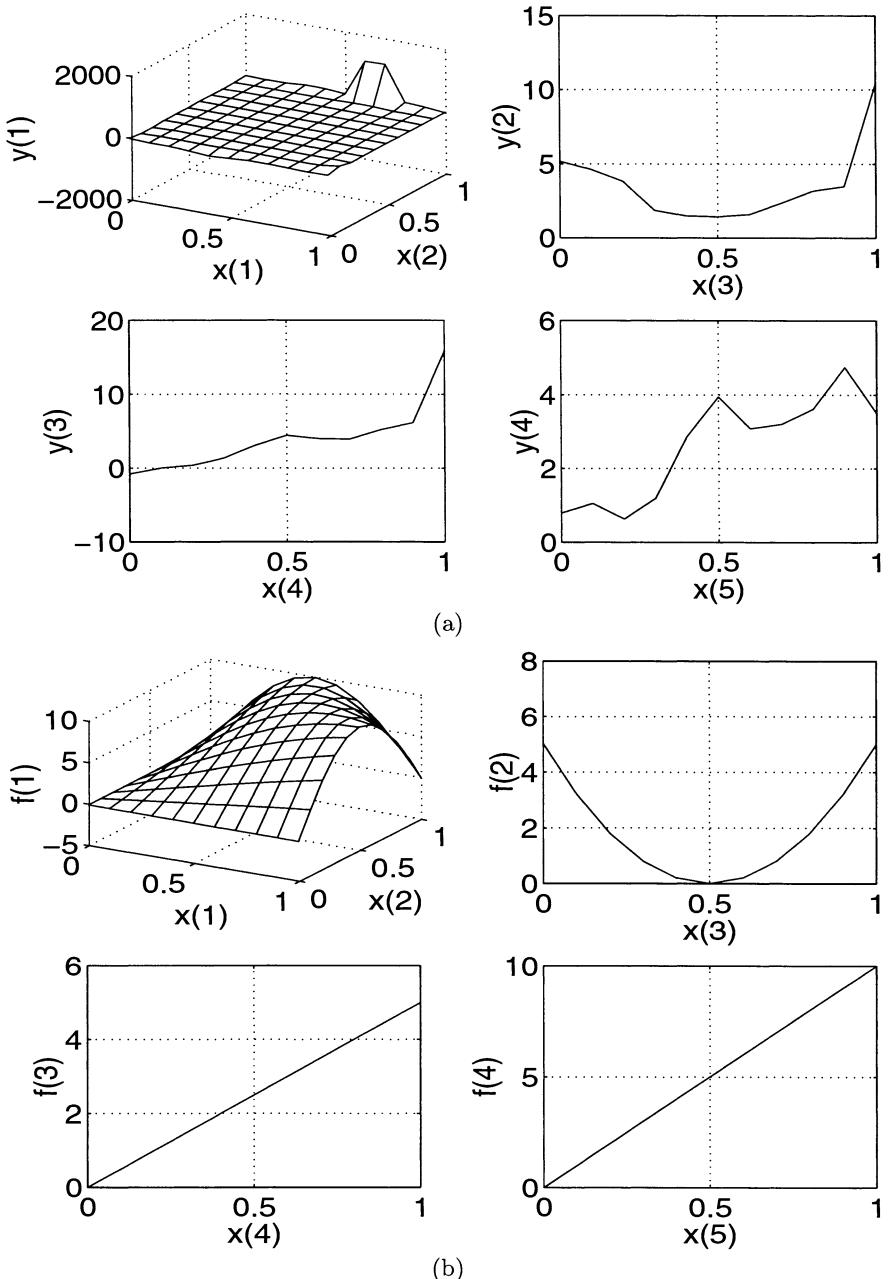


Fig. 5.23. (a) shows the output of the four models, when training is performed without regularisation, i.e. ML estimation. The MSE across the training data is extremely good, $1.27e - 8$, as the model has overfitted the noise, giving poor generalisation. This is reflected in the test set MSE of $4.49e + 3$. (b) shows the output of the four subfunctions of the true function

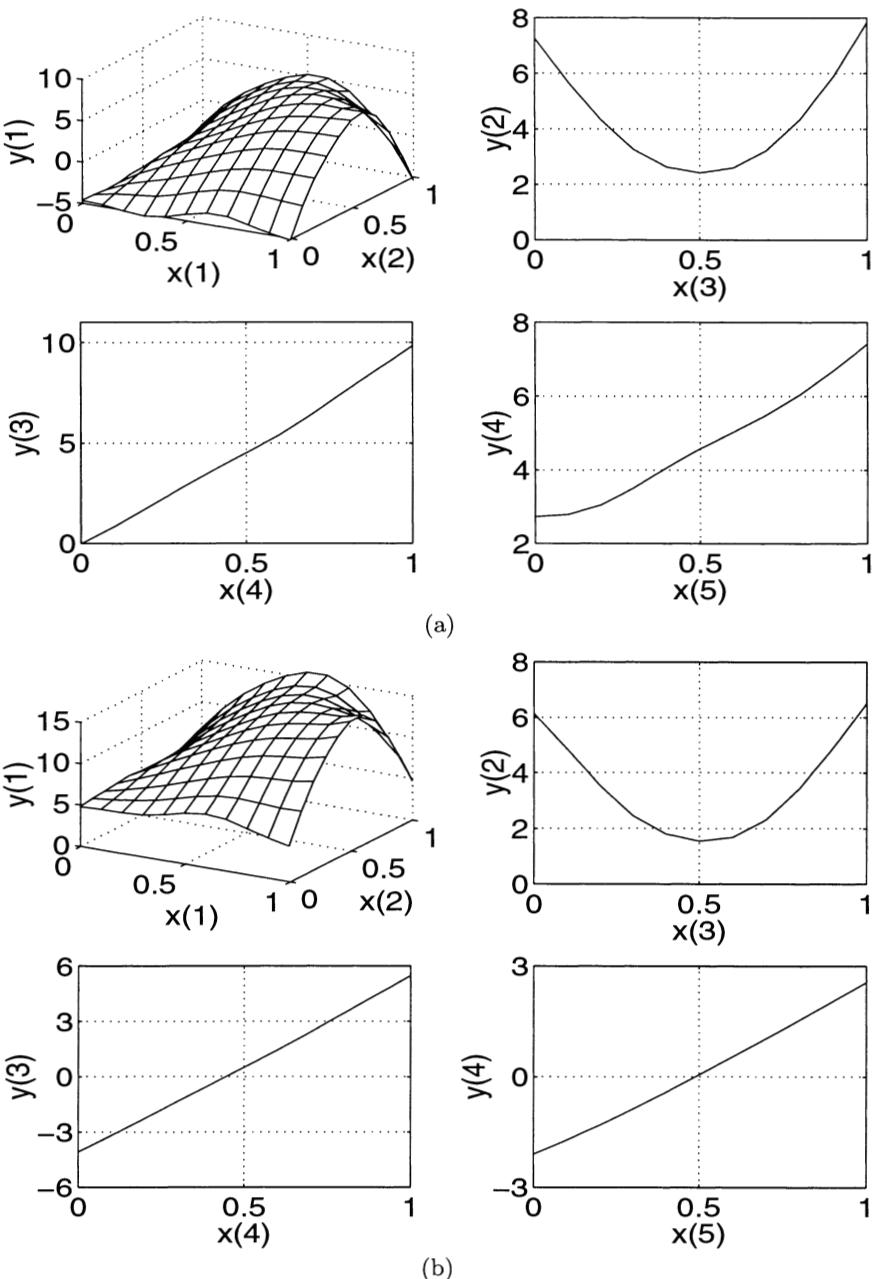


Fig. 5.24. The responses of the submodels resulting from local regularisation; (a) one prior and (b) local regularisation. It can be seen in both cases that the submodel's outputs have been successfully regularised by the various priors. Local regularisation is noticeably better producing the desirable linear response for submodel y_4

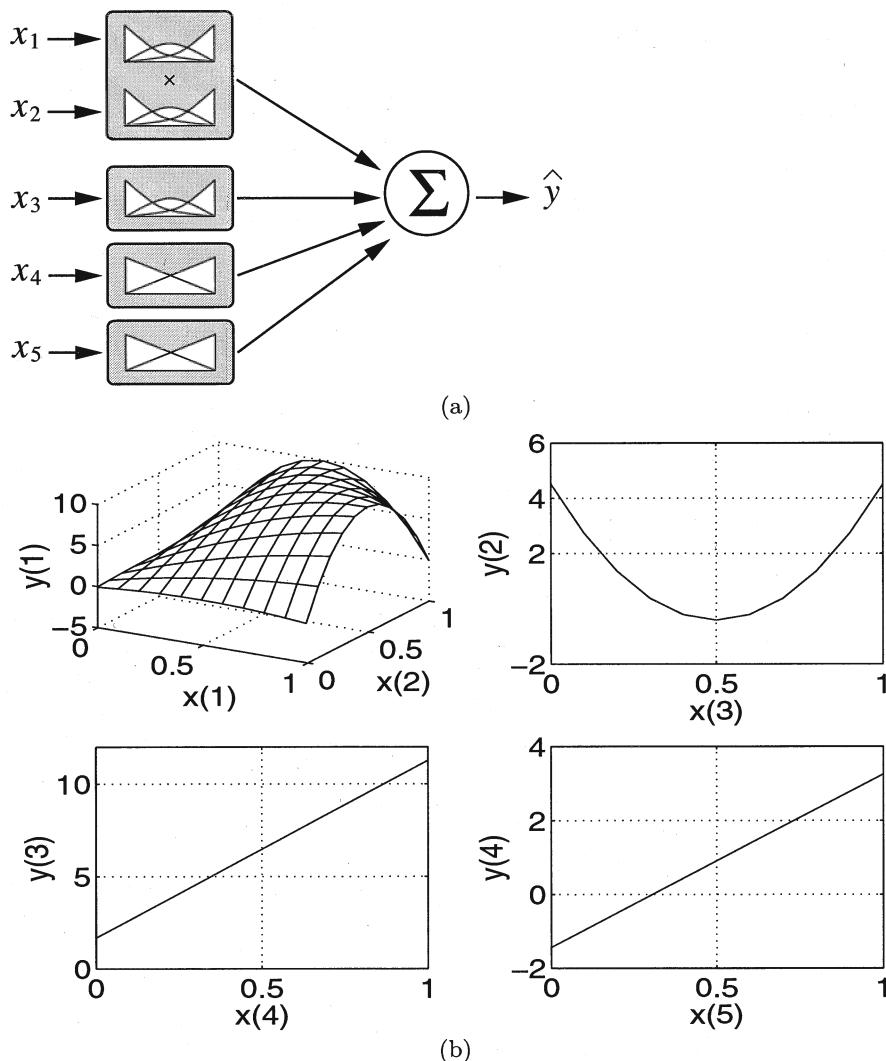


Fig. 5.25. Results from identifying a neurofuzzy model on the data; (a) shows the identified submodels, the sum of which produces the single output and (b) shows the model's response, ML estimation. This model appears to perform adequately throughout as reflected in the MSE across the test set $1.48e - 1$. When local regularisation is applied, this test set MSE is further reduced to $1.05e - 1$

5.7 Complexity reduction through orthogonal least squares

Whilst the above mentioned model construction algorithms are effective in overcoming the curse of dimensionality, in practice a two stage approach is preferable whereby there is a preprocessing stage which reduces the data by selecting high “energy” or variance terms in the data (e.g. SVD or error reduction ratios techniques), followed by a second model construction stage that uses selected model terms. One of the most popular model structure determination techniques is the forward orthogonal least squares (OLS) algorithm [46] which selects model terms based on the corresponding error reduction ratios (see (5.42)). Unfortunately models based on least squares estimates are unsatisfactory for near ill-posed regression matrices. To avoid this problem in OLS, regularisation techniques have been introduced [48, 157], which reduces the variance of parameter estimates, whilst introducing a parameter bias.

For N observations of the process that is modelled by (3.2), the output observation vector $\mathbf{y} = [y(1), \dots, y(N)]^T$ is given by

$$\mathbf{y} = \mathbf{Aw} + \mathbf{e}, \quad (5.36)$$

where $\mathbf{w} = [w_1, \dots, w_p]^T$ is an unknown parameter vector, and $\mathbf{e} = [e(1), \dots, e(N)]^T$ is the model residual vector, and $\mathbf{A} = [\psi(\mathbf{x}(1)), \dots, \psi(\mathbf{x}(N))]^T \in R^{N \times p}$ is the regressor matrix. An orthogonal decomposition of \mathbf{A} is

$$\mathbf{A} = \Phi T, \quad (5.37)$$

where T is a $p \times p$ unit upper triangular matrix and $\Phi = [\phi_1, \dots, \phi_p]$ is a $N \times p$ matrix with orthogonal columns that satisfy

$$\Phi^T \Phi = \text{diag}\{\kappa_1, \dots, \kappa_p\}, \quad (5.38)$$

with $\kappa_j = \phi_j^T \phi_j$, $j = 1, 2, \dots, p$, so that the regression equation (5.36) can be expressed as

$$\mathbf{y} = (\mathbf{AT}^{-1})(T\mathbf{w}) + \mathbf{e} = \Phi \Gamma + \mathbf{e}, \quad (5.39)$$

where $\Gamma = [\gamma_1, \dots, \gamma_p]^T$ is an auxiliary vector. As $e(t)$ is uncorrelated with past outputs, then

$$\gamma_j = \frac{\phi_j^T \mathbf{y}}{\phi_j^T \phi_j}, \quad j = 1, 2, \dots, p. \quad (5.40)$$

As $\phi_i^T \phi_j = 0$ for all $i \neq j$ by the orthogonal property, then multiplying (5.39) by itself and time averaging gives

$$\frac{1}{N} \mathbf{y}^T \mathbf{y} = \frac{1}{N} \sum_{j=1}^p \gamma_j^2 \phi_j^T \phi_j + \frac{1}{N} \mathbf{e}^T \mathbf{e}. \quad (5.41)$$

The total output variance $E(y^2(t)) = \frac{1}{N} \mathbf{y}^T \mathbf{y}$ consists of two elements: (i) $\frac{1}{N} \sum_{j=1}^p \gamma_j^2 \phi_j^T \phi_j$, the output variance explained by the model regressors and

(ii) $\frac{1}{N}\mathbf{e}^T\mathbf{e}$, the unexplained model residual variance. Of the total number of p regressors, a subset with $p'(<< p)$ regressors can be selected according to the error reduction ratio $[ERR]_j$, which is defined as the increment of the overall output variance $E(y^2(t))$ due to each regressor or input variable $\psi_j(t)$ divided by the total output variance, i.e.

$$[ERR]_j = \frac{\gamma_j^2 \phi_j^T \phi_j}{\mathbf{y}^T \mathbf{y}}, \quad j = 1, 2, \dots, p. \quad (5.42)$$

The p' most relevant regressors can be forward selected by this criterion. At the j th selection a candidate regressor is selected as the j th basis of the subset if it produces the largest $[ERR]_j$ from the remaining $(p - j + 1)$ candidates. By setting an appropriate tolerance ρ , generated by Akaike information criterion (AIC) or FPE, which forms a compromise between model performance and model complexity, the variable selection is formulated when

$$1 - \sum_{j=1}^{p'} [ERR]_j < \rho. \quad (5.43)$$

This procedure can automatically select a subset of p' regressors to construct a parsimonious model. The original model parameter vector $\mathbf{w} = [w_1 \dots w_{p'}]^T$ can be computed from $T\mathbf{w} = \Gamma$ through back substitution.

Note: Consider the general nonlinear representation of (2.4), the output observation $y(t)$ can be decomposed into two parts: the noise free part $f(\mathbf{x}(t))$ and the measurement noise $e(t)$. Based on the premise that $y(t)$ is more stochastic than $f(\mathbf{x})$, here the most significant regressors are selected according to their energy level in a forward regression, hence OLS is suboptimal (optimal per regression stage). A global optimal signal reconstruction approach [111] that reconstructs $f(\mathbf{x})$ from data D_N with noisy observations is to use a nonlinear principal component analysis (PCA) [158] (see Section 6.4), where most of the signal energy in the system output is retrieved via a noise free part, $\tilde{y}(t)$, represented by M most significant principal components. Nonlinear PCA prefiltering is global optimal in the sense that maximum information in the regression matrix is retrieved using the smallest number of principal components. Generally PCA transformation obscures physical relationships, and in this context OLS is superior, however the nonlinear PCA is effective in signal smoothing, such as a prefilter to generate additive noise models used for state estimation (see Section 6.4).

5.8 A-optimality neurofuzzy model construction (NeuDec)

Orthogonal least squares (OLS) is an effective algorithm for structure determination and parameter estimation [47]), yet its direct utilisation for

regressor search via orthogonalisation for neurofuzzy models is computationally prohibitive. Since, for example, for seven-input (five univariate basis/axis) system generating $7 \times 5^7 = 546,875$ candidate regressors would require 2.2×10^7 orthogonalisations, and transformation matrices with 546,875 columns to construct a 40 regressors neurofuzzy system! Even utilising the ANOVA expansion for the same system with interaction degree between inputs of two and three generates 3,920 and 34,545 candidate regressors. If by some preprocessing schema the regressors can be reduced to say 500 regressors, then OLS based on the reduced rule base would only require 19220 orthogonalisations, and the associated transformations. Here we introduce a two stage neurofuzzy model construction algorithm, so-called NeuDec, in which a simple preprocessing method finds a reduced subset of regressors (or rules), followed by a fine model detection procedure based on forward OLS to give a parsimonious model with minimal prediction error and parameter variance via a composite cost functional. Central to this new approach is the A-optimality experimental design criteria [9] which is used in both stages. In the preprocessing stage, a lower bound on the A-optimality criteria is used for subset selection, whereas in the second stage, it is incorporated into a composite cost function that seeks minimisation of prediction error and parameter variance.

Optimum experimental designs have been used [9] to construct smooth system response surfaces based on the setting of experimental variables using statistical goodness of experimental design. These criteria try to avoid poor designs with unnecessarily large parameter estimates variances or models with large dimension or parameterisations. Consider (5.36)) for $\mathbf{A} = [\Psi(\mathbf{x}(1)), \dots, \Psi(\mathbf{x}(N))]^T \in \mathbb{R}^{N \times p}$ is formed on the basis function of the complete neurofuzzy model. For example if $n_y + n_u = 6$, and $L_i = 5$ (the number of basis functions used in each input) the model parameter dimension is $p = n \sum_{i=1}^n L_i = 93,750$, or $p = 17,430$ for an ANOVA expansion of degree 3 with no redundant inputs. Generally for large p the complete model contains many redundant terms due to the underlying lattice structure used in a neurofuzzy model, and $\mathbf{A}^T \mathbf{A}$ is generally singular. Here NeuDec initially generates a reduced rule base with $n_w \ll p$ (see Theorem 5.2), then utilies OLS to find a final parsimonious model with $n_f \ll n_w$ regressors [107, 108].

Definition 5.1. A-optimality Design Criterion

The A-optimality experimental design, which can be applied to model selection, is that for a $n_w < p$ minimises the sum of the variance of the parameter estimates vector $\mathbf{w} = [w_1, \dots, w_{n_w}]^T$ through

$$\min\{V_w = \text{tr}[\text{cov}\hat{\mathbf{w}}] = \sigma^2 \sum_{k=1}^{n_w} \frac{1}{\lambda_k}\}, \quad (5.44)$$

where λ_k is the eigenvalues of $\mathbf{A}_k^T \mathbf{A}_k$, where $\mathbf{A}_k = [\mathbf{a}_1^{(k)}, \dots, \mathbf{a}_{n_w}^{(k)}] \in \Re^{N \times n_w}$ denotes the resultant regression matrix, consisting of n_w regressors selected from p regressors in \mathbf{A} .

Definition 5.2. Index vector of \mathbf{A}_k

Each of the selected n_w regressors in \mathbf{A}_k has a previous position in \mathbf{A} , such that

$$\mathbf{a}_j^{(k)} = \mathbf{a}_{o_j}, \quad (5.45)$$

where $j = 1, \dots, n_w$ denotes the new column index, and $o_j \in [1, \dots, p]$ is the previous column index. An index vector of \mathbf{A}_k marks the sequence of the previous column index, and is given by $\mathbf{o} = [o_1, o_2, \dots, o_{n_w}]^T$.

It would appear that the A-optimality criterion requires extensive evaluation of the SVD of $\mathbf{A}_k^T \mathbf{A}_k$ and an exhaustive search of possible regressor combinations. This is clearly unrealistic for large p , but is shown in Theorems 5.1 and 5.2 to be unnecessary as a lower bound on V_w can be established, which can be minimised.

Theorem 5.1. From Definition 5.1, a lower bound of V_w is $n_w^2 \sigma^2 (\text{tr} [\mathbf{A}_k^T \mathbf{A}_k])^{-1}$.

Proof. From Definition 5.1, $\mathbf{A}_k^T \mathbf{A}_k$ is a positive matrix, whose eigenvalues λ_i , $i = 1, \dots, n_w$ are all real and non-negative. From (5.44)

$$\begin{aligned} V_w &= \sigma^2 \sum_{k=1}^{n_w} \frac{1}{\lambda_k} \\ &\geq n_w \sigma^2 \sqrt[n_w]{\frac{1}{\lambda_1} \frac{1}{\lambda_2} \cdots \frac{1}{\lambda_{n_w}}} \\ &= n_w \sigma^2 (\sqrt[n_w]{\lambda_1 \lambda_2 \cdots \lambda_{n_w}})^{-1}. \end{aligned} \quad (5.46)$$

Since by the arithmetic-geometric mean inequality for non-negative, real number λ_i , $i = 1, \dots, n_w$

$$\frac{\lambda_1 + \lambda_2 + \dots + \lambda_{n_w}}{n_w} \geq \sqrt[n_w]{\lambda_1 \lambda_2 \cdots \lambda_{n_w}}.$$

Applying this inequality again to (5.46) and noting that for an $(n \times n)$ matrix \mathbf{A} , $\sum_{i=1}^n \lambda_i = \text{tr} \mathbf{A}$, it follows that with eigenvalues λ_i

$$\begin{aligned} V_w &\geq n_w^2 \sigma^2 \left[\sum_{k=1}^{n_w} \lambda_k \right]^{-1} \\ &= n_w^2 \sigma^2 (\text{tr} [\mathbf{A}_k^T \mathbf{A}_k])^{-1}. \end{aligned} \quad (5.47)$$

□

Theorem 5.2. Given some finite number $n_w < p$, sort the first n_w largest regressors into magnitude order from the initial p regressor candidates, and rank them in descending order of magnitude, such that $\|\mathbf{a}_{o_1}\| \geq \|\mathbf{a}_{o_2}\| \geq \|\mathbf{a}_{o_{n_w}}\| \geq \|\mathbf{a}_j\|, j \notin \{o_1, o_2, \dots, o_{n_w}\}$, for an index vector $\mathbf{o} = [o_1, o_2, \dots, o_{n_w}]$. The resultant subset \mathbf{A}_k , based on Definition 5.2, minimises the lower bound $n_w^2 \sigma^2 (\text{tr} [\mathbf{A}_k^T \mathbf{A}_k])^{-1}$, of V_w .

Proof. For fixed n_w , the minimisation of V_w is equivalently the maximisation of $\text{tr} [\mathbf{A}_k^T \mathbf{A}_k]$, hence

$$\begin{aligned} \text{tr} [\mathbf{A}_k^T \mathbf{A}_k] &= \sum_{j=1}^{n_w} [\mathbf{a}_j^{(k)}]^T \mathbf{a}_j^{(k)} \\ &= \sum_{j=1}^{n_w} \mathbf{a}_{o_j}^T \mathbf{a}_{o_j}. \end{aligned} \quad (5.48)$$

It is clear from (5.48) that the bound of V_w is minimised by sorting the first n_w largest regressors in magnitude, followed by ranking them in descending order of magnitude, such that $\|\mathbf{a}_{o_1}\| \geq \|\mathbf{a}_{o_2}\| \geq \|\mathbf{a}_{o_{n_w}}\|$, for its index vector $\mathbf{o} = [o_1, o_2, \dots, o_{n_w}]$. \square

In practice, a n_w is preselected as a few hundred so as to form a moderate candidate initial regressor base size for further refinement via OLS. It is assumed that n_w is sufficiently large so that the estimation set has adequate modelling capability to approximate the underlying data generating process. The most relevant (or significant) n_f ($\ll n_w$) regressors can be determined by the forward OLS scheme of Section 5.7 with $n_f = p'$, $n_w = p$.

Alternatively the A-optimality design criterion (5.44) can be used to select the n_f regressors, so as to minimise the variance of the parameter estimate vector $\mathbf{w} = [w_1, \dots, w_{n_f}]^T$ ensuring model adequacy. Minimising this cost function does not by itself ensure good model approximation, hence V_w needs an amended or composite cost functional

$$V_f = \frac{1}{N} (\mathbf{y}^T \mathbf{y} - \sum_{k=1}^{n_f} \gamma_k^2 \kappa_k) + \alpha \sum_{k=1}^{n_f} \frac{1}{\kappa_k}, \quad (5.49)$$

for some small positive α , where the regression matrix $\mathbf{A}_k = \Phi T$ has been decomposed into an upper triangular matrix T and matrix Φ with orthogonal columns that satisfy $\Phi^T \Phi = \text{diag}\{\kappa_1, \dots, \kappa_{n_w}\}$, and which defines an auxillary vector $\Gamma = [\gamma_1, \dots, \gamma_{n_w}]^T = T \mathbf{w}$ (see also Section 5.7). Equation (5.49) can be incorporated into the forward OLS algorithm to select the most relevant k th regressor at the k th forward regression stage, expressed as

$$V^{(k)} = V^{(k-1)} - \frac{1}{N} \gamma_k^2 \kappa_k + \frac{\alpha}{\kappa_k}. \quad (5.50)$$

At the k th regression stage, a candidate regressor is selected if it produces the smallest $V^{(k)}$ and a further reduction in $V^{(k-1)}$. The procedure terminates if $V^{(k)} \geq V^{(k-1)}$ at the derived model size n_f . Consequently the NeuDec algorithm automatically detects and determines a parsimonious model from data.

Notes

(i) The composite cost function (5.49) appears similar to model regularisation (see Section 2.5), where large parameter values are penalised or the model is forced to be smooth in area of data sparsity. Here the most appropriate model structure is determined whilst minimising parameter variance.

(ii) If $\alpha = 0$, the above algorithm reduces to the conventional forward OLS algorithm. An arbitrarily small α will detect near singularity of the regression matrix during modelling avoiding selection of an ill-posed model. Also the error tolerance ρ used in OLS is no longer required for model selection for $\alpha \neq 0$.

(iii) The decremental cost function (5.50) contains two elements; (a) $\frac{1}{N}\gamma_k^2\kappa_k$, representing model error reduction due to inclusion of a new regressor; and (b) $\frac{\alpha}{\kappa_k}$, which represents increase in model parameter variance due to this new regressor. This latter term helps to distinguish model pseudo terms that have large parametric values γ_k^2 , but significantly small κ_k values (norm of an orthogonal basis). α should therefore be sufficiently selected so that $\alpha < \frac{1}{N}\gamma_k^2\kappa_k^2$. Violating this inequality terminates model selection. In practice, α can be selected heuristically or by a statistical method such as GCV or AIC (see Section 2.7).

Example 5.7: Gas furnace model (Series J of [27])

A gas furnace which mixes air and methane are combined to produce CO₂ on combustion has a constant air feed rate, but variable methane feed rate $u(t)$ (see Figure 5.26) (of 0.60 – 0.04(t) cu ft/min). The furnace CO₂ output is shown in Figure 5.26 and 296 data points have been used to fit a linear model [27] given by

$$\begin{aligned} y(t) = & \frac{-(0.53 + 0.37q^{-1} + 0.51q^{-2})}{1 - 0.57q^{-1}} u(t-3) \\ & + \frac{1}{1 - 1.53q^{-1} + 0.63q^{-2}} e(t), \end{aligned} \quad (5.51)$$

with the $\sigma_e^2 = 0.0561$.

This same data has been used with the NeuDec algorithm for $n_y = n_u = 5$, i.e. $\mathbf{x}(t) = [y(t-1), \dots, y(t-5), u(t-1), \dots, u(t-5)]^T$. ANOVA was used to set up an initial model basis by setting the input component interaction degree up to two. Basis function knot vectors were defined for $y(t-i), i = 1, \dots, 5$, as

$$\{16, 30, 53, 76, 90\}$$

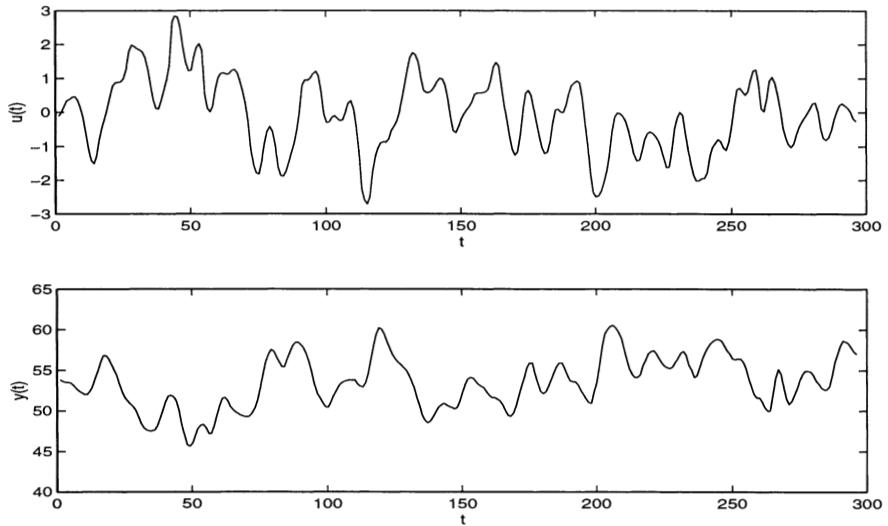


Fig. 5.26. Gas furnace data; ($u(t)$: coded input methane gas input feed rate; $y(t)$: CO_2 concentration of the furnace output)

as and for $u(t - i), i = 1, \dots, 5$ as

$$\{-8, -4, 0, 4, 8\}$$

using piecewise linear B-spline ($m = 1$) for all the input components. The initial ANOVA neurofuzzy model had 4,350 basis functions.

Starting NeuDec with $n_w = 400$ and a range of α . NeuDec generated (i) a 24 basis function model with $\sigma_e^2 = 0.048$ for $\alpha = 0.16$ (ii) a 32 basis function model with $\sigma_e^2 = 0.042$ for $\alpha = 0.04$, in both cases superior to the linear model (5.51). For the derived 24 basis function model validation test and model prediction over the output data set are illustrated in Figures 5.27 and 5.28. Figure 5.29 plots the derived model parameter variance sum $\sigma^2 \sum_{k=1}^{n_f} \frac{1}{\kappa_k}$ for the conventional OLS algorithm and the NeuDec algorithm for this example, illustrating significant superiority of NeuDec over the OLS algorithm. (Other examples see [107] demonstrate similar computational advantages for NeuDec over OLS.) The NeuDec algorithm is applicable to any linear in the parameter network including B-spline, RBF's and Bézier-Bernstein networks (see also Chapter 7). Equally it can be used to extract a parsimonious set of linguistic rules to describe the resultant model [109].

So far data-based neurofuzzy modelling has been focused on decomposition techniques that essentially generate global submodels of reduced input dimension. An alternative approach is to produce local low dimensional models together with submodel integration techniques to generate global models – this is pursued in Chapters 6 and 7.

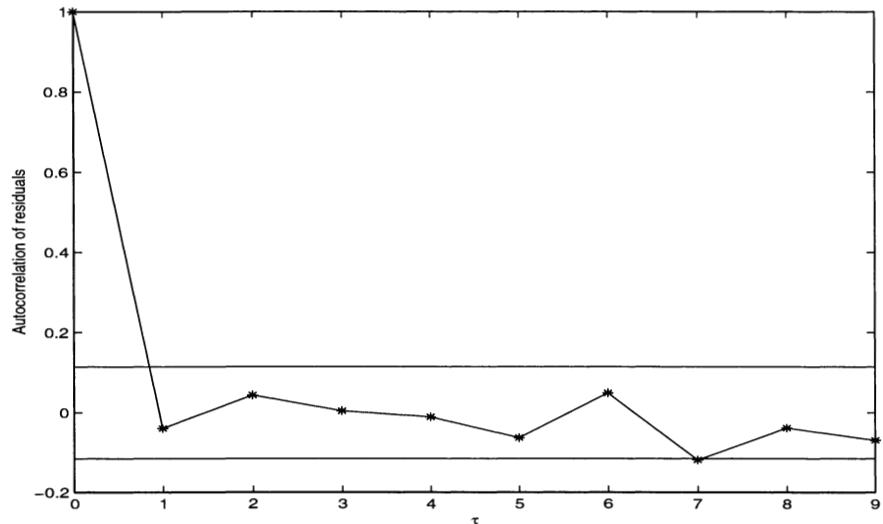


Fig. 5.27. Model validity test for the derived model with 24 basis function ($\alpha = 0.16$)

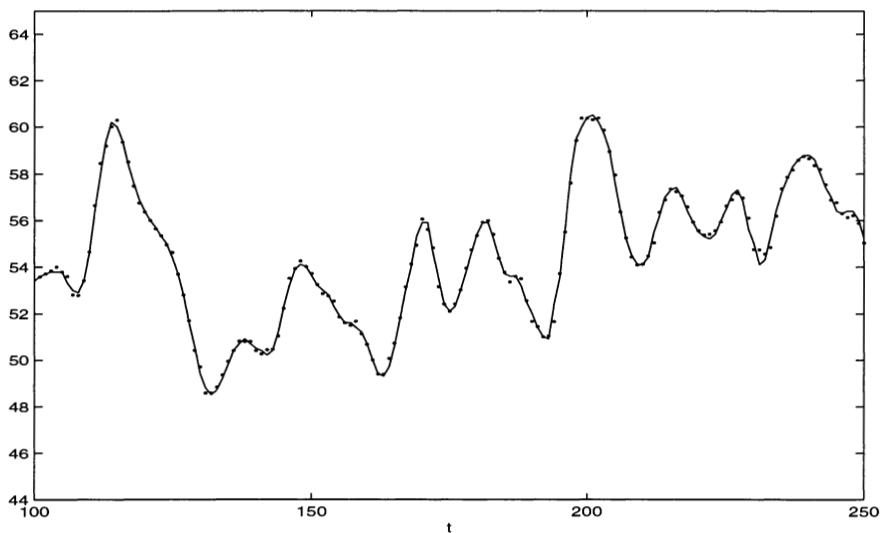


Fig. 5.28. Model predictions of output $y(t)$ based on the derived model with 24 basis function($\alpha = 0.16$); (solid line: actual measurements; and dotted line: model predictions)

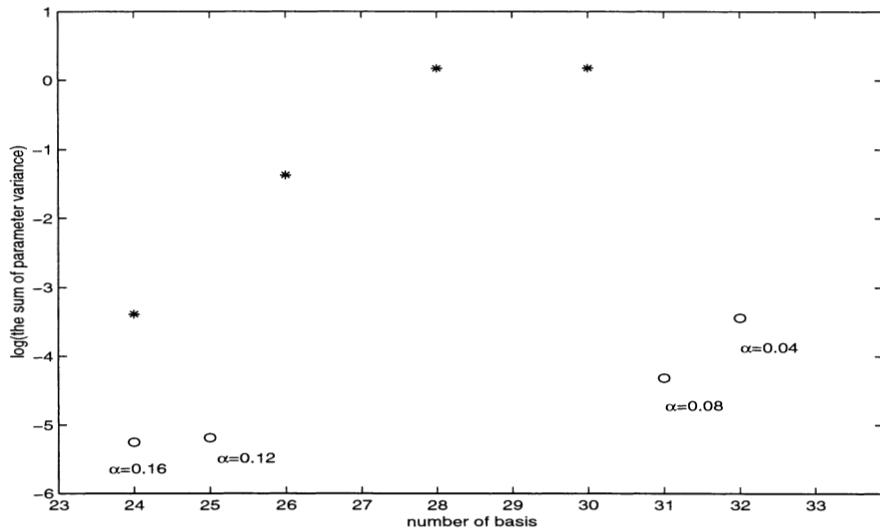


Fig. 5.29. Improvement in parameter variance by the proposed approach over the conventional OLS method; ('o' denotes the automatic model construction termination point based on a predetermined α value; and '*' denotes the result of conventional OLS ($\alpha = 0$))

6. Local neurofuzzy modelling

6.1 Introduction

The previously described model construction algorithms of Chapter 5 resolved model complexity by utilising the divide and conquer strategy, by decomposing high dimensional problems into a number of lower dimensional submodels each with variable dependencies whose composite solution yields the original complex problem. Central to the conventional neurofuzzy approach is an orthogonal axis partitioning of the input space, which is the main cause of the curse of dimensionality. Clearly, other decompositions or partitioning are possible including irregular simplexes (see Figure 6.1(a)) and data clustering with centred Gaussians (see Figure 6.1(b)).

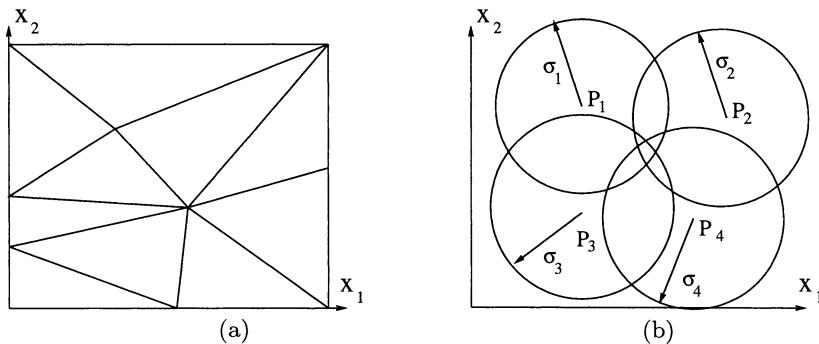


Fig. 6.1. Input space partitioning; (a) by irregular simplexes and (b) by centred Gaussians

Many dynamical processes have locally linear behaviour, yet have global nonlinear characteristics. Therefore the modelling task here is to find the decomposition of the operating space of the unknown nonlinear dynamic process into the minimum number of local (linear) models parameterised by the known or measured operating point conditions. For many practical processes the operating point conditions vary as a function of independent parameters (such as Mach number and altitude of an aircraft) or more usually as a function of the observed system states. Often these parametric relationships are

complex, unknown, and nonlinear, but may be measurable or at least inferred from other sensory or database knowledge. The philosophy behind operating point/region modelling (and control) is that in each operating region system behaviour is simpler than that exhibited globally. Additionally simple models are easy to understand, inherently easier to incorporate *a priori* quantitative and qualitative knowledge, easier to train, and are (if linear) directly applicable to classical control and estimator design techniques. Well known examples [195] of operating point dynamics include aircraft and gas turbine dynamics as a function of Mach number and altitude as operating point conditions; auto-car dynamics as a function of velocity, which is in turn a function of gear position and engine throttle angle; chemical reactors as a function of dissolved oxygen and temperature; paper processing as a function of paper moisture content; or ship dynamics as a function of sea state and forward velocity. Similar to the various model construction algorithms of Chapter 5, operating point modelling is an iterative or evolutionary search process that includes:

- Characterisation of the operating space by definition of the operating point variables (which usually have to be measurable or known).
- Decomposition of the operating space into a minimum set of (overlapping) operating regimes, each with an *a priori* determined local model approximation error.
- Selection of local model structures, usually these are dictated by analytical tractability, desired purpose of the model (e.g. control, tracking, or diagnostics), and prior knowledge about local behaviour. Note that the model structure may be different in different operating regions reflecting local knowledge.
- Mechanisms for fusion or integration of local models into a global model, together with overall model behaviour tests.

Each of these stages is highly dependent on each other, hence the need for iterative search algorithms for model decomposition and construction. Usually there is a trade-off between the number and size of operating regions and the complexity of the local models. A fundamental question is, how does local modelling capability translate with appropriate fusion algorithms into global modelling capability? This problem is simple if the local models are polynomial functions defined on a compact domain, since they can approximate any global nonlinear continuous function with arbitrary uniform accuracy [195].

Characterisation of the operating space is usually dictated by the underlying phenomenology of the problem (e.g. Mach number and altitude). Similarly generation of operating space regions may be determined by the physics of the problem (e.g. gear changes in a car engine), or by data clusters [192], or by automatic decomposition algorithms (such as the mixture of experts, see Section 6.5). Local model structures are almost inevitably selected as linear due to their ease of understanding, use of conventional engineering design/implementation, and amenability to analysis

and global integration. Global integration of the local models is dependent upon both the partitioning of the operating space as well as the desired smooth transition between operating regimes. The partitions may be hard/discontinuous/stochastic/overlapping. Whatever is used in practice it is highly desirable to have smooth transition between regions. Therefore usually some fuzzy or probabilistic weighting is used between local models or regions.

There are of course many other approaches to local neurofuzzy modelling, principle amongst which are:

(i) *Fuzzy data clustering algorithms* [192] approximate a nonlinear system by a set of locally linear models which are valid for a certain range of operating conditions and an interpolate scheme for integrating local model outputs to generate a global model. Fuzzy data clustering and the Gustafson–Kessel algorithm are used to construct a series of fuzzy IF–THEN rules of a particular structure: Takagi–Sugeno (T–S) models (see Section 4.4), which utilise triangular membership functions due to their analytic simplicity. Fuzzy clustering algorithms are based on the premise that the identification data D_N is a representative sample of the underlying regression surface ($(X \times Y) \in \Re^{n+1}$) defined by the unknown nonlinear function $y = f(\mathbf{x})$. The objective is to find the parameters of the T–S models (4.37) such that it approximates this hypersurface by a piecewise series of linear hyperplanes. Data is partitioned by the Gustafson–Kessel algorithm with an adaptive distance measure into prototype adjoining hyperellipsoid clusters. For each data cluster fuzzy covariance matrices are computed, whose eigenvectors are used to determine the T–S rules directly. The number of T–S rules must be prespecified, with more rules leading to greater accuracy. For *a priori* unknown processes a high initial number is assumed and cluster merging techniques together with validity measures akin to AIC are utilised to prune the rule set to fit the acquired data. This approach works well for low dimensional processes, subject to low measurement noise and adequate amounts of data. For high dimensional problems, the data inevitably becomes sparse, increasing the number of clusters exponentially with the input space dimension.

(ii) *Adaptive-networks-based fuzzy inference systems* (ANFIS) [122]. ANFIS is a group of neurofuzzy architectures based on three general types of fuzzy system models: the Tsukamoto model, the Mamdani model and the Takagi–Sugeno model. The premise parts of the rules in these models are identical, but the consequent part differs according to the model used. Figure 6.2 illustrates the differences in the rule consequences of the three models.

The most commonly used model is based on the T–S model, in which ANFIS uses normalised Gaussian basis functions. The ANFIS architectures embody the same dual interpretation of B-spline networks, enabling them to be trained as neural networks and be interpreted as fuzzy systems. The

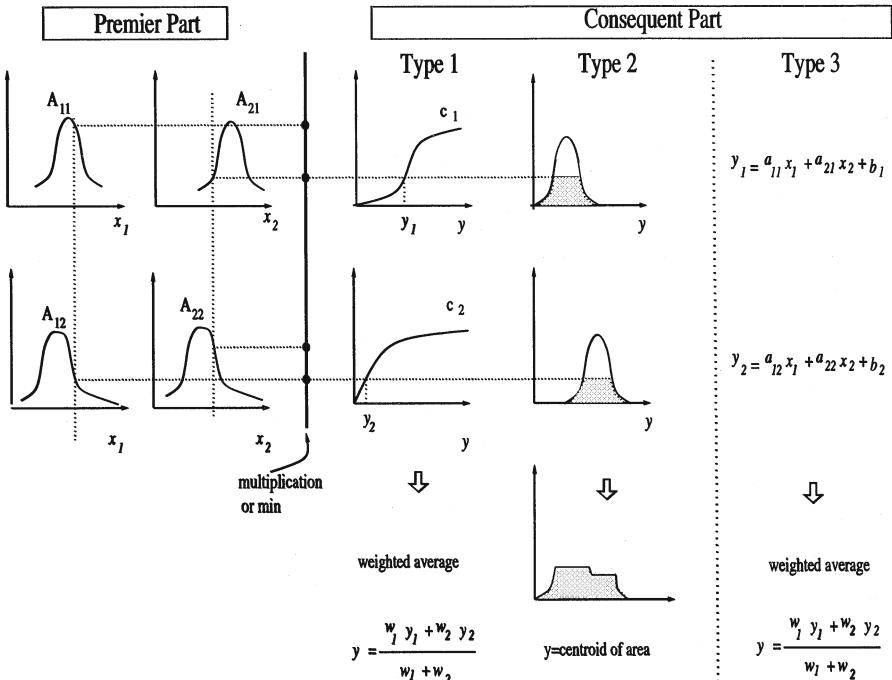


Fig. 6.2. Different membership functions in three models

technique uses least squares to find the initial structure of the ANFIS model, and then uses gradient descent to determine the model weights (parameters). The ANFIS model construction algorithm is as follows:

ANFIS Model Construction Algorithm

1. Specify number of inputs m , number of membership functions and shape (order).
2. Divide the data into two sets (D_N^t for training, D_N^v for validation).
3. For $i = 1$ to C_m^m
 - Estimate parameters for model m_i using D_N^t
 - End.
4. Select model as the model m_i with minimum MSE performance on D_N^v .
5. Update basis functions in model using gradient descent.

The ANFIS algorithm generates a complete rule base and therefore suffers from the curse of dimensionality. Unlike ASMOD, the ANFIS architecture does not exploit the inherent transparency and parsimony of semi-local basis functions such as B-splines. ANFIS is restricted to models with a small number of inputs, which allow some transparency. ANFIS has been improved by

using input selection to identify and isolate redundant or irrelevant inputs from the final model, by retaining the ANFIS model with the least RMSE after each epoch of training on the assumption that it has the best potential for achieving an improved performance after further training. Clearly there are problems with this approach if after an epoch no improvement in modelling is achieved. Use of failure margins readily overcomes this problem (see Section 5.3). Also, unlike ASMOD, the number of inputs, shape and number of fuzzy membership functions on each input axis have to be prespecified. A software implementation for ANFIS is available within the MATLAB fuzzy tool box. ANFIS modelling attributes are compared with ASMOD and locally partitioned algorithms in the next section.

6.2 Local orthogonal partitioning algorithms

In Chapter 5 we introduced the concept of generalised additive neurofuzzy model construction algorithms that overcome the curse of dimensionality and simultaneously retain network transparency. If transparency is not critical then alternative strategies for avoiding an orthogonal input space partitioning are possible. Equally additive neurofuzzy models are still liable to produce a redundant structure unless local regularisation is employed. The inherent lattice structure adopted by conventional neurofuzzy systems is restrictive for some problems in that it fails to model local input space functional behaviour parsimoniously. This drawback can be overcome by exploiting a more local representation by using a strategy that locally partitions the input space. Popular non-global lattice representation include $k - d$ trees (Figure 6.3(a)), Quad-trees (Figure 6.3(b)), and hierarchical multiresolution model structures (Figure 6.3(c) and (d)), all of which have axis orthogonal partitions, introducing hyper-rectangular regions, which represent the support of multivariate fuzzy sets ensuring transparent representations for these local partitions.

6.2.1 $k - d$ Trees

$k - d$ Trees partition an n -dimensional input space by a succession of axis-orthogonal splits made across the entire domain of an existing partition. Each split produces two new regions which are candidates for future splits. Friedman [73] used $k - d$ trees to produce high dimensional spline approximation, based on the product of truncated spline basis functions defined on the resulting $k - d$ tree partition. More relevant to our interest in neurofuzzy modelling, Kavli [129] extended Friedman's partitioning methods to the ASMOD algorithm for identification of locally partitioned neurofuzzy models. Unfortunately this modified form of ASMOD has several practical problems [25] such as slow convergence, poor generalisation as well as a lack of transparency, making it unsuitable for neurofuzzy modelling.

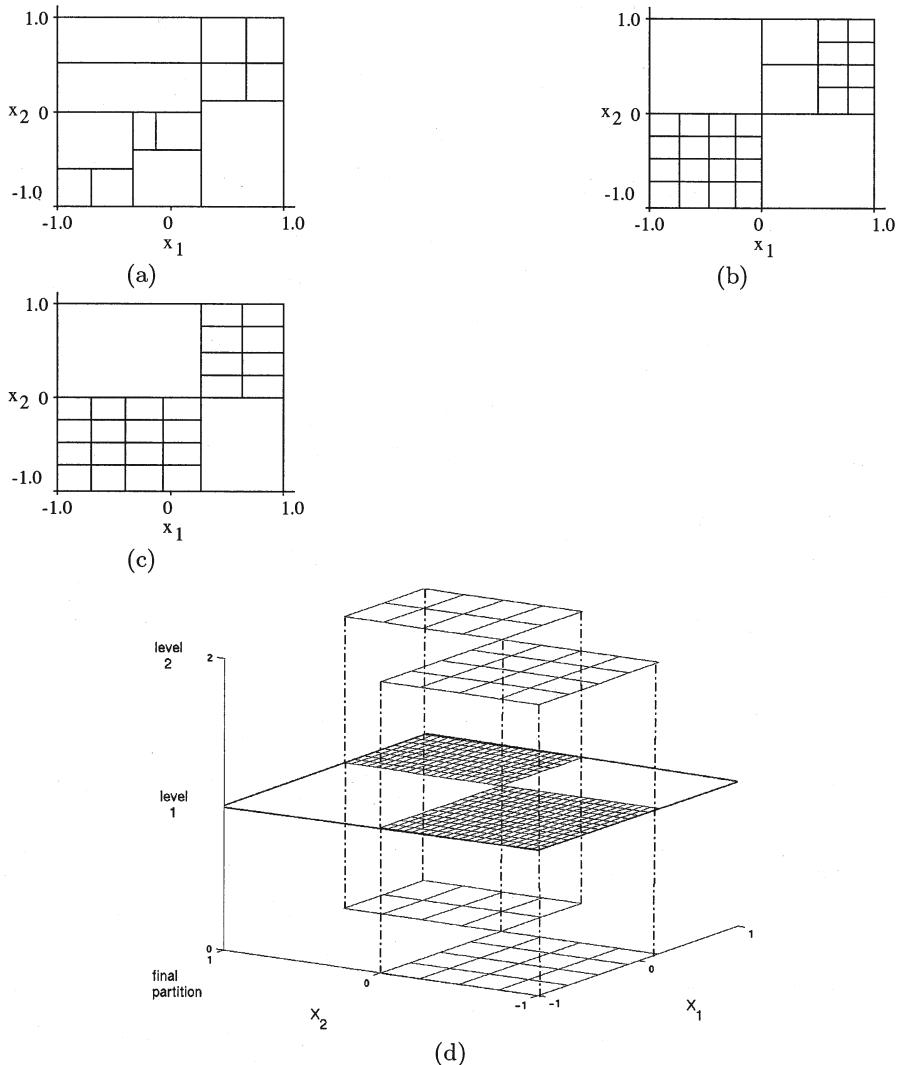


Fig. 6.3. Non-lattice partition: (a) $k-d$ tree; (b) quad-trees; and (c) the input space partition generated by a hierarchical multiresolution model structure of (d)

The local linear model tree (LOLIMOT) approach [154] is a piecewise linear modelling algorithm that partitions the input space using a $k - d$ tree structure into hyper-rectangulars. Each hyper-rectangulars region is modelled by a separate local linear model, as with the ANFIS approach. A LOLIMOT model with Gaussian weighting function is structurally similar to the T-S fuzzy system and ANFIS model with Gaussian membership functions. Figure 6.4 shows how a two-input model could be decomposed into seven submodels. At each stage the algorithm computes the approximation improvement by inserting in each half a hyper-rectangular in each input dimension.

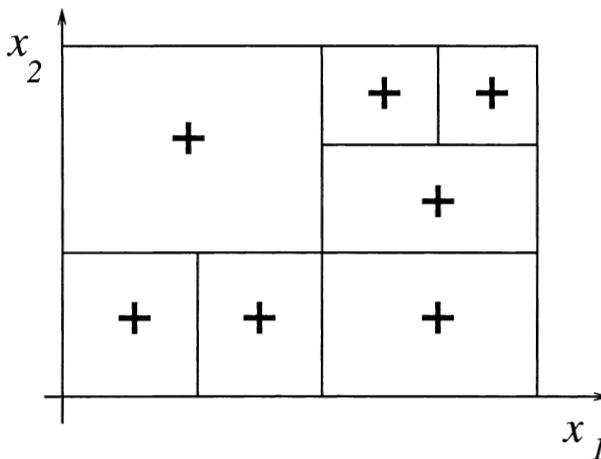


Fig. 6.4. An example of the LOLIMOT ($k - d$ tree) input space partitioning, where + denotes the centre of Gaussian weighting functions

At each iteration, the local linear model that contributes most to the overall model error is subdivided into two new ones. Cuts in all dimensions are evaluated and the one with the best performance improvement is selected for further refinements. LOLIMOT uses the structural risk minimisation (SRM) metric (see Sections 1.3 and 8.2) to determine the most appropriate model structure. However due to input space partitioning it has a coarser parameter resolution than ASMOD ($n + 1 : LOLIMOT$, $1 : ASMOD$). Similarly to ANFIS, LOLIMOT permits every data input to contribute to the model, making no attempt to eliminate redundant inputs. Also LOLIMOT contains no (automatic) iteration management schedule, relying on the user to terminate the algorithm. As with ASMOD, LOLIMOT has a construction and pruning or elimination phase, eliminating any previously made partition if it does not significantly contribute to the model's overall error measure. The LOLIMOT algorithm can be described as follows:

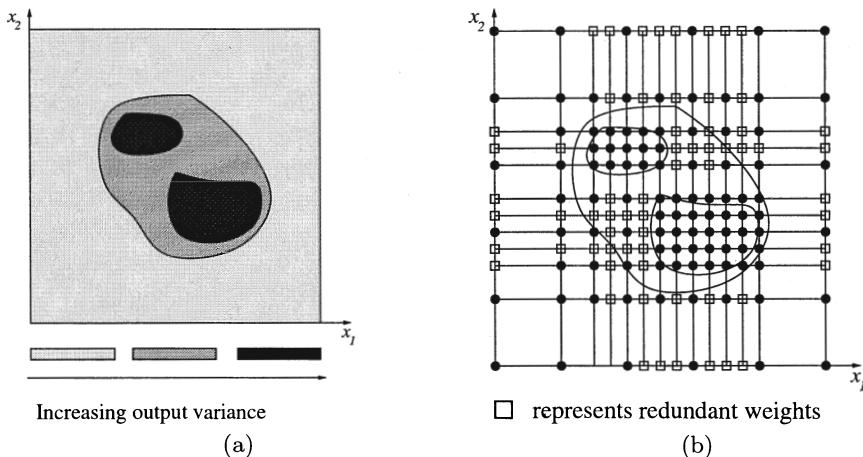


Fig. 6.5. A 2-D surface consisting of regions of different output variances. (a) illustrates the 2-D surface, and (b) shows the inadequate partitioning of the input space produced by global partitioning, where a box represents a redundant rule premise

Table 6.1. Comparison of three construction algorithms

	ASMOD	ANFIS	LOLIMOT
Transparency	Good	Average	Poor
Isolation of redundant inputs	Yes	Partially	No
Basis function neighbourhood	Semi-local	Local	Local
Basis function selection	Automatic	Semi-automatic	Automatic
Input space partition	Semi-lattice	Lattice	$k - d$ tree
Resolves curse of dimensionality	Yes	No	Yes
Generalisation capability	Good	Poor	Average
Regulariser included for noisy or sparse data sets	Yes	No	No

The Local Linear Model Tree (LOLIMOT) Construction Algorithm

1. Set the initial hyper-rectangle to enclose all data.
2. Estimate a global linear model for it.
3. Do
 - For $i = 1$ to No. of hyper-rectangulars.
 - For $j = 1$ to n Bisect the hyper-rectangulars along dimension j .
 - Place a weighting function at the centre of the two hyper-rectangulars, with standard deviations proportional to the dimensions of the hyper-rectangulars.
 - Estimate the parameters of the local linear models for each half.
 - Calculate the new global approximation error.
 - End.
 - End.
 - Select the bisection which led to the smallest approximation error.
 - While (hypothesis measure of model is decreasing).

The improved ASMOD, ANFIS and LOLIMOT have been extensively benchmarked [34] against a series of data modelling problems including simulated nonlinear time series problems, auto-car-miles per gallon data set (<ftp://ics.uci.edu/pub/machine-learning-databases/auto-mpg>), Box Jenkins gas furnace data set (<http://www.maths.monash.edu.au/~hyndman/tseries/industry.html>), and an industrial engine torque data set (m1brown@uk.ibm.com). A summary of the comparative aspects of these construction algorithms is given in Table 6.1. ANFIS generalisation capability is generally worse than either ASMOD or LOLIMOT, due to the reduced data density through splitting data sets into test and validation data sets $\{D_N^t, D_N^v\}$. Both ANFIS and LOLIMOT could be further improved by post-processing regularisation. Despite the drawbacks of ANFIS and LOLIMOT both have found widespread usage in industry in diverse applications including car engine control, materials classification and hot metal rolling. An alternative approach to input space partitioning, which leads to multiresolutional models, is to use quad-trees.

6.2.2 Quad-trees

Quad-trees decomposition of a n -dimensional space is less structured than $k - d$ trees. This is achieved by successfully partitioning cells in the n -dimensional space by n axis orthogonal splits, one for each axis, producing 2^n new cells. Berger [17] exploits this partitioning strategy to adaptively produce piecewise linear models whose output is piecewise linear across each of the hypercubes. These models are quad-trees on which order 2 B-splines are defined. Quad-trees have also been exploited in the fuzzy literature by Sun [184] where they have been used to position fuzzy membership functions, producing the so-called *fuzzy boxtrees*.

A related more flexible approach is based on the multiresolution partitioning into quad-trees by Moody [149], which generates models composed of a hierarchy of local conventional neurofuzzy models. The resulting model is a hierarchy of lattice based neurofuzzy models of different resolutions combining the generalisation properties of coarse models with the accuracy of fine ones. Existing at the top of the hierarchy is a large coarse model, called the *base submodel*, which models the general trade of the system, aiding generalisation. At the bottom are small finely partitioned *specialised submodels* which model local behaviours. The advantage of such a model can be demonstrated by considering the 2-D function illustrated in Figure 6.5(a). The input space is separated into regions of different output variance. Regions of low output variance can be modelled by coarse resolution neurofuzzy models, while regions with high output variance can only be modelled accurately by fine resolution neurofuzzy models. If a single lattice model is employed the partitioning of the input space illustrated in Figure 6.5(b) would be produced. Due to the regions of different output variance this strategy produces many redundant rules. Hierarchical local partitioning would produce a partition of the input space shown in Figure 6.6(a), consisting of four different lattice based submodels (s_0, s_1, s_2, s_3). The hierarchical structure of this model is illustrated in Figure 6.6(b).

The hierarchy of multiresolution models is represented by a tree structure, which is used as an efficient mechanism for matching the input to a relevant submodel. This is achieved by ascending this tree until the input is found to lie in the domain of one of the submodels. The output of this submodel is the output of the overall network, as only one submodel contributes to the overall output. One of the main difficulties of this type of representation is maintaining continuity in the output surface. This is achieved by weight sharing, with weights translated down the hierarchy. The weights at the boundaries of the individual specialised submodels are inherited from their parent's weight vector to ensure output continuity. In each submodel, a particular weight falls into one of three categories: directly inherited, indirectly inherited, and free. These three different types of weights are shown in Figure 6.6. Training is achieved by teaching each of the submodels individually, starting at the top of the hierarchy and working down. The base model is first taught on the complete training data set. Next the submodels of the next level of the hierarchy are taught. First all the boundary weights are inherited, either directly or indirectly, and then the free weights of these models can be taught on a limited size training set producing a more accurate model of the output surface in the designated region. The submodels are iteratively trained in this fashion until all the free weights of the complete model have been identified.

Effectively the inherited weights and those of coarse models on regions of the input space covered by finer models (as represented by the shaded regions in Figure 6.6(b)) are redundant. Despite this redundancy they provide a basis

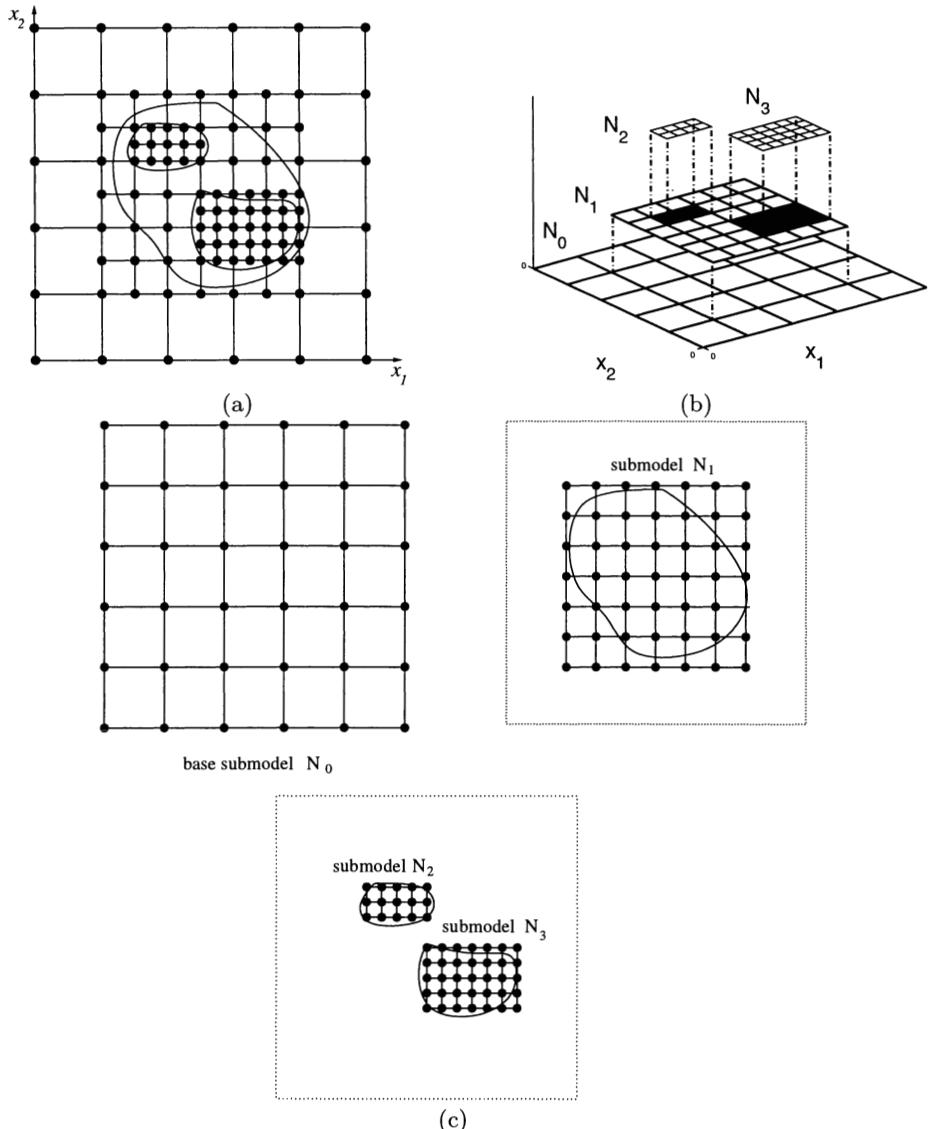


Fig. 6.6. The hierarchical model representing the 2-D surface shown in figure: (a) shows the resulting partition of the input space, (b) illustrates the hierarchy of models, and (c) shows the individual submodels

by which a complete fuzzy rule base for each individual submodel can be constructed.

An algorithm that constructs these models from a representative training set is required. These hierarchical models should be used, where required, to locally partition submodels of a globally partitioned model represented by the ANOVA decomposition when required. This additive decomposition can be identified by the ASMOD algorithm, resulting in a parsimonious globally partitioned models consisting of lattice based submodels. As the output of the system may possess local behaviour in the input domains of these submodels, they can be locally refined to produce models of the form described above.

Parallels can be drawn between this approach and wavelet networks [210] which form a model as a sum of a hierarchy of local basis functions with different smoothness or resolution characteristics.

6.3 Operating point dependent neurofuzzy models

In this section we develop a class of local neurofuzzy models that have a linear structure but are parameterised by a measurable operating dependent vector \mathbf{O}_t . This operating point vector represents for example the Mach number and altitude of an aircraft, which parameterises its dynamics, so that for fixed \mathbf{O}_t , the dynamics are constant and frequently linear for perturbations around \mathbf{O}_t . Hence the following model representations linear structure enables linear estimation and control algorithms to be used, yet knowledge of the behaviour of \mathbf{O}_t over the input space effectively provides a form of global gain scheduling. Local operating point neurofuzzy models are directly equivalent to Takagi–Sugeno (T–S) fuzzy models with local linear models, yet have more amenable mathematical properties, nor are limited to order 2 fuzzy basis functions. In the following we establish some of the mathematical properties of such models prior to their use in estimation problems. Consider the general discrete time, SISO, nonlinear stochastic model of (2.4):

$$y(t) = f[y(t-1), \dots, y(t-n_y), u(t-1), \dots, u(t-n_u)] + e(t), \quad (6.1)$$

where $f(\cdot)$ is *a priori* unknown nonlinear function, n_y and n_u ($n_y + n_u = n$, $n_y \geq n_u$) are positive integers representing the system known lags, and $e(t)$ is observational noise which is assumed zero mean and variance σ^2 . Define the system observational vector

$$\mathbf{x}(t) = [x_1, \dots, x_n]^T = [y(t-1), \dots, y(t-n_y), u(t-1), \dots, u(t-n_u)]^T.$$

Then system (6.1) can be represented by a quasi-linear model of the form [195]:

$$\begin{aligned} y(t) &= a_1(\mathbf{O}_t)y(t-1) + \dots + a_{n_y}(\mathbf{O}_t)y(t-n_y) \\ &\quad + b_1(\mathbf{O}_t)u(t-1) + \dots + b_{n_u}(\mathbf{O}_t)u(t-n_u) + \Delta f(\mathbf{O}_t, \mathbf{x}), \end{aligned} \quad (6.2)$$

where $\{a_i(\mathbf{O}_t), b_j(\mathbf{O}_t)\}$ are *a priori* unknown functions of the measurable operating point \mathbf{O}_t , and $\Delta f(\mathbf{O}_t, \mathbf{x})$ is model mismatch error. The existence of the representation (6.2) depends on the differentiability of $f(\cdot)$ with respect to its arguments [22, 173]. This is an assertion since $f(\cdot)$ is *a priori* unknown, however for most practical problems, $f(\cdot)$ is sufficiently smooth for this to apply. Various special cases of (6.2) have been investigated, including:

- (i) Additive and separable nonlinear decomposition models:

$$f(\cdot) = \sum_{i=1}^{n_y} f_i(y(t-i)) + \sum_{j=1}^{n_u} g_j(u(t-j)),$$

for all $f_i(\cdot)$ and $g_j(\cdot)$, first order differentiable [195].

- (ii) Models related to classical statistical time series representations [164]:
- $\{a_i(\cdot)\}$ are constants and

$$b_j(\cdot) = c_j + \sum_{i=1}^n b_{ij}y(t-i), \quad j = 1, \dots, n_u,$$

for c_j , b_{ij} constants, then (6.2) is a bilinear model.

- $\{a_i(\cdot)\}$ and $\{b_j(\cdot)\}$ are constants, then (6.2) is an autoregressive moving average (ARMA) process.
- $\{a_i(\cdot)\}$ and $\{b_j(\cdot)\}$ are dependent on t , then (6.2) is linear and nonstationary.

(iii) If \mathbf{O}_t is bounded and smooth in a subregion of the observational space, then the unknown coefficients $\{a_i(\mathbf{O}_t), b_j(\mathbf{O}_t)\}$ can be approximated to arbitrary accuracy by neurofuzzy networks as

$$a_i(\mathbf{O}_t) = \sum_{k=1}^p w_{ik}^{(a)} N_k(\mathbf{O}_t), \quad i = 1, \dots, n_y, \quad (6.3)$$

$$b_j(\mathbf{O}_t) = \sum_{k=1}^p w_{jk}^{(b)} N_k(\mathbf{O}_t), \quad j = 1, \dots, n_u, \quad (6.4)$$

where $N_k(\mathbf{O}_t)$ are multivariate B-splines formed by the tensor product of univariate basis functions $B_{i,k_i}(O_t^i)$, i.e. $N_k(\mathbf{O}_t) = \prod_{i=1}^n B_{i,k_i}(O_t^i)$, where O_t^i is the i th element of the measured vector \mathbf{O}_t . In this case we have a series of Takagi–Sugeno models. The number of parameters $\{w_{ik}^{(a)}, w_{jk}^{(b)}\}$ used in expansions (6.3) and (6.4) can be pruned by various automatic construction algorithms such as ASMOD (see Section 5.3). In practice this reduces the number of parameters from exponential in n ($n = n_u + n_y$) to almost linear.

Problems may arise when neural networks are used to estimate the parameters of (6.2) as these networks are defined on a bounded input space, requiring the measurable operating points to lie in a uniformly bounded set. Three possibilities can occur:

- The measurable operating point is independent of the system input and output (or $\mathbf{x}(t)$).
- The measurable operating point is a subset of $\mathbf{x}(t)$.
- The measurable operating point is composed of independent variables and a subset of $\mathbf{x}(t)$.

Assuming that the coefficient expansions (6.3) and (6.4) are applicable and that for some finite truncation M in the series expansion, such that the error of approximation generated by each $\{a_i(\cdot), b_j(\cdot)\}$ neurofuzzy network is upper bounded by δ , then the model mismatch δ_1 is bounded by

$$|\Delta f[\mathbf{x}(t)]| \leq (n_y + n_u)\delta \|\mathbf{x}(t)\| = \delta_1. \quad (6.5)$$

Therefore (6.2) can be expressed as

$$\hat{y}(t) = \psi^T(t-1)\mathbf{w} + \Delta f[\mathbf{x}(t-1)], \quad (6.6)$$

where

$$\begin{aligned} \mathbf{w} = & [w_{11}^{(a)}, \dots, w_{1p}^{(a)}, \dots, w_{n_y 1}^{(a)}, \\ & \dots, w_{n_y p}^{(a)}, w_{11}^{(b)}, \dots, w_{1p}^{(b)}, \dots, w_{n_u 1}^{(b)}, \dots, w_{n_u p}^{(b)}]^T \end{aligned} \quad (6.7)$$

$$\begin{aligned} \psi = & [N_1 y(t-1), \dots, N_p y(t-1), \dots, N_1 y(t-n_y), \\ & \dots, N_p y(t-n_y), N_1 u(t-1), \dots, N_p u(t-1), \\ & \dots, N_1 u(t-n_u), \dots, N_p u(t-n_u)]^T \in R^{p(n_y+n_u)}. \end{aligned} \quad (6.8)$$

Due to the existence of the model mismatch nonlinear uncertainty term $\Delta f(\cdot)$ in (6.6), the conventional recursive least squares (RLS) or normalised least mean squares (NLMS) algorithms, (3.42) and (3.28) respectively, have to be modified [195], by introducing a normalised signal $\rho(t)$ and output error dead zone $v(t)$ to the algorithm.

Define this normalised signal $\rho(t)$ as

$$\rho(t) = \alpha\rho(t-1) + (1-\alpha) \max\{1, \|\mathbf{x}(t-1)\|\}, \quad (6.9)$$

where $\alpha \in (0, 1)$ and $\rho(0) \neq 0$. Denote

$$\begin{aligned} y_\rho(t) &= y(t)/\rho(t) \\ \psi_\rho(t) &= \psi(t)/\rho(t) \\ \Delta_\rho f[\mathbf{x}(t)] &= \Delta f[\mathbf{x}(t)]/\rho(t). \end{aligned} \quad (6.10)$$

Then (6.6) can be redefined in normalised form as

$$y_\rho(t) = \psi_\rho^T(t-1)\hat{\mathbf{w}} + \Delta_\rho f[\mathbf{x}(t)], \quad (6.11)$$

where $|\Delta_\rho f[\mathbf{x}(t)]| \leq \delta_1$, which can be applied to the modified RLS:

$$\begin{aligned} \Delta \hat{\mathbf{w}}(t) &= \hat{\mathbf{w}}(t) - \hat{\mathbf{w}}(t-1) \\ &= \frac{\eta(t)\mathbf{P}(t-2)\psi_\rho(t-1)e_\rho(t)}{1 + \psi_\rho^T(t-1)\mathbf{P}(t-2)\psi_\rho(t-1)} \end{aligned} \quad (6.12)$$

$$e_\rho(t) = y_\rho(t) - \hat{y}_\rho(t) \quad (6.13)$$

$$\hat{y}_\rho(t) = \psi_\rho^T(t-1)\hat{\mathbf{w}}(t-1) \quad (6.14)$$

$$\mathbf{P}^{-1}(t-1) = \mathbf{P}^{-1}(t-2) + \eta(t)\psi_\rho^T(t-1)\psi_\rho(t-1), \quad (6.15)$$

where

$$\eta(t) = \begin{cases} 1 & |e_\rho(t)| > v(t), \\ 0 & \text{otherwise,} \end{cases} \quad (6.16)$$

and

$$v(t) = \frac{\delta_1}{[1 + \psi_\rho^T(t-1)\mathbf{P}(t-2)\psi_\rho(t-1)]^{1/2}}, \quad (6.17)$$

and $\hat{\mathbf{w}}(0)$, $\mathbf{P}(0)$ are initial values. Convergence of the above modified RLS algorithm follows similarly to the conventional RLS proof [87]:

Theorem 6.1: When the modified RLS algorithm (6.12)–(6.17) is applied to identify the model (6.6) using the data set D_N , then

$$\lim_{t \rightarrow \infty} \frac{e_\rho^2(t)}{1 + \psi_\rho^T(t-1)\mathbf{P}(t-2)\psi_\rho(t-1)} = \delta_1^2 \quad (6.18)$$

$$\lim_{t \rightarrow \infty} \|\hat{\mathbf{w}}(t) - \hat{\mathbf{w}}(t-k_0)\| = 0, \quad (6.19)$$

where k_0 is a positive integer.

Proof: From the Lyapunov function

$$L(t) = [\hat{\mathbf{w}}^T(t) - \mathbf{w}^T]\mathbf{P}^{-1}(t-1)[\hat{\mathbf{w}}(t) - \mathbf{w}],$$

the equality

$$\begin{aligned} \Delta L(t) &= L(t) - L(t-1) \\ &= \begin{cases} -\frac{e_\rho^2(t)}{1 + \psi_\rho^T(t-1)\mathbf{P}(t-2)\psi_\rho(t-1)} + \delta_1^2 & |e_\rho(t)| > v(t) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \quad (6.20)$$

holds. Therefore $\Delta L(t)$ is nonpositive and hence $L(t)$ has a lower bound and is a nonincreasing series with time, hence $\lim_{t \rightarrow \infty} \Delta L(t) = 0$ exists. \square

If there is no model mismatch ($\delta = 0$), then the conventional RLS algorithm applies, and Theorem 6.1 has $\delta_1 = 0$ and as $\|\psi(t-1)\| < \infty$, then

$$\lim_{t \rightarrow \infty} [y(t) - \hat{y}(t)] = 0.$$

Notes:

- (i) The above operating point dependent neurofuzzy algorithm has been successfully applied to paper processing fault detection [195], adaptive neurofuzzy control [67], gas turbine engine sensor management [29], K-step ahead control and pole placement control [196].

(ii) Figure 6.7 illustrates the structure of this operating point dependent neurofuzzy model, all external signals $\{\mathbf{O}_t, y(t), u(t)\}$ are measurable. The network consists of a set of ‘coefficient’ neurofuzzy networks with the same input \mathbf{O}_t . The network has two hidden layers; the weights in the first layer (\mathbf{w}) are updated by the above modified RLS algorithm and the weights in the second layer take the time delayed values of the observed inputs and outputs $\{y(t), u(t)\}$.

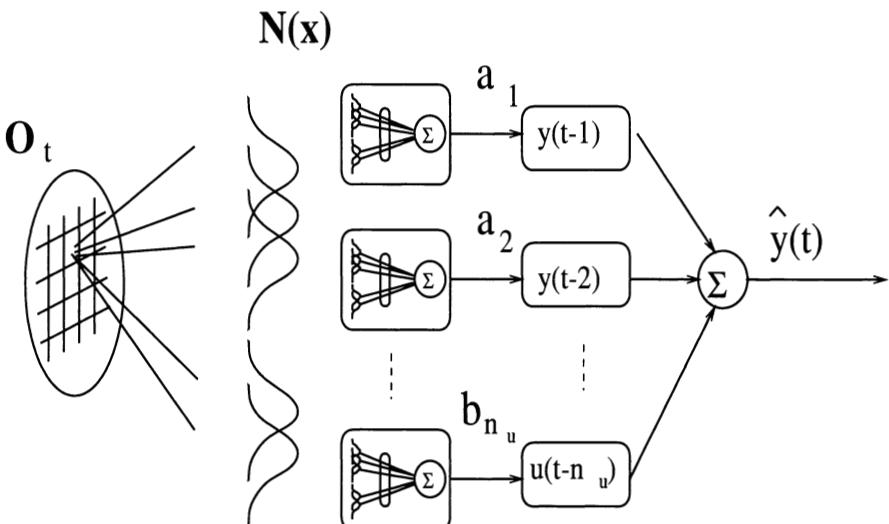


Fig. 6.7. Operating point dependent neurofuzzy model

6.4 State space representations of operating point dependent neurofuzzy models

The above neurofuzzy network can be applied to state estimation, as by the Taken–Whitney theorem [186] the appropriately dimensioned observation vector $\mathbf{x}(t)$ contains the state vector of the underlying process that generates the input–output data set D_N . A state space representation for the model (6.2) is readily available for the use in the Kalman filter (see Sections 2.2 and 8.5). The coefficients of (6.2), $\{a_i(\mathbf{O}_t), b_j(\mathbf{O}_t)\}$, can be viewed as function of time, so that (6.2) is a time-varying ARMA type process, which can be represented in a canonical controllable form

$$\begin{aligned} \mathbf{z}(t+1) &= A(t)\mathbf{z}(t) + B\tilde{u}(t) + \mathbf{v}(t) \\ y(t) &= C(t)\mathbf{z}(t) + \tilde{u}(t) + \xi(t), \end{aligned} \quad (6.21)$$

with state vector $\mathbf{z}(t) = [y(t-1), y(t-2), \dots, y(t-n_y)]^T$, $\tilde{u}(t) = \sum_{j=1}^{n_u} b_j(\mathbf{O}_t) u(t-j)$ and

$$A(t) = \begin{bmatrix} a_1(\mathbf{O}_t) & a_2(\mathbf{O}_t) & \dots & a_{n_y-1}(\mathbf{O}_t) & a_{n_y}(\mathbf{O}_t) \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \in \mathbb{R}^{n_y \times n_y}$$

$$B = [1 \ 0 \ \dots \ 0]^T \in \mathbb{R}^{n_y}$$

$$C = [a_1(\mathbf{O}_t) \ \dots \ a_{n_y}(\mathbf{O}_t)] \in \mathbb{R}^{n_y},$$

and $\mathbf{v}(t)$ represents process noise and model mismatch. The Kalman filter can be applied directly to (6.21) (see Section 8.5), assuming that $\{\mathbf{v}, \xi\}$ are Gaussian with zero mean, and are uncorrelated.

There are three basic approaches: the indirect, direct state estimation methods [207] and state estimation using prefiltering.

(i) Neurofuzzy Indirect Estimation

In the indirect method, system identification and state estimation by a Kalman filter [87] are separated as shown in Figure 6.8. First the neurofuzzy network of Figure 6.7 is used to identify the nonlinear system model whose parameters are used to instantiate a Kalman filter to estimate the system states indirectly.

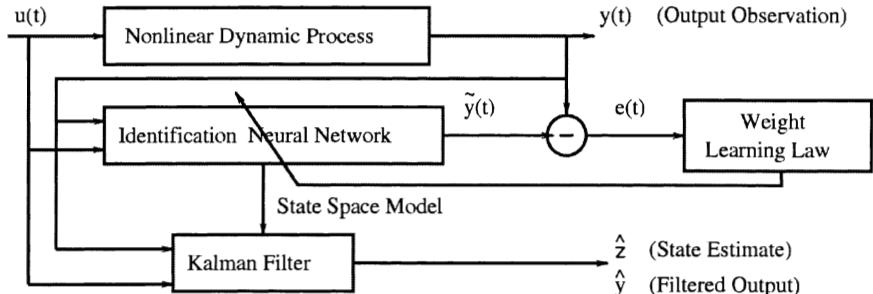


Fig. 6.8. Indirect state estimation

In its indirect form, the input $\mathbf{x}(t) = \{x_i\} = [y(t-1), \dots, y(t-n_y), u(t-1), \dots, u(t-n_u)]^T \in \mathbb{R}^n$ is represented to the network of Figure 6.7 for some measured operating point vector \mathbf{O}_t . In the forward pass the network calculates the network output, denoted by $\hat{y}(t)$, from (6.2). For an error of $e(t) = y(t) - \hat{y}(t)$, as there are no free parameters in the second layer to be trained, this error is propagated back through the second layer to the output

of the first layer. Therefore the errors in $\{a_i, b_j\}$ normalised by $\mathbf{x}^T(t)\mathbf{x}(t)$ are given by

$$e_{(a,b)}(t) = \frac{x_i(t)e(t)}{\mathbf{x}^T(t)\mathbf{x}(t)}, \quad i = 1, \dots, n, \quad (6.22)$$

which are used to update the weights in the first layer via the NLMS algorithm:

$$\Delta \hat{\mathbf{w}}^{(a,b)}(t) = \eta \frac{\mathbf{N}(t)x_i(t)e(t)}{[c + \mathbf{N}^T(t)\mathbf{N}(t)\mathbf{x}^T(t)\mathbf{x}(t)]}, \quad (6.23)$$

for $0 < \eta < 2$, c an arbitrary small positive constant, and $\mathbf{N}(t) = [N_1(\mathbf{O}_t), \dots, N_p(\mathbf{O}_t)]^T$. These derived parameters are then substituted into the Kalman filter of the state representation (6.21).

(ii) Direct Neurofuzzy Estimation

In the direct method, the identification process and Kalman filter algorithms are bootstrapped together, as shown in Figure 6.9, to generate the state estimates directly. Here the first layer is the same as that of Figure 6.7, but the second layer is a Kalman filter which uses the autoregressive parameters derived in the first layer to directly perform state estimation.

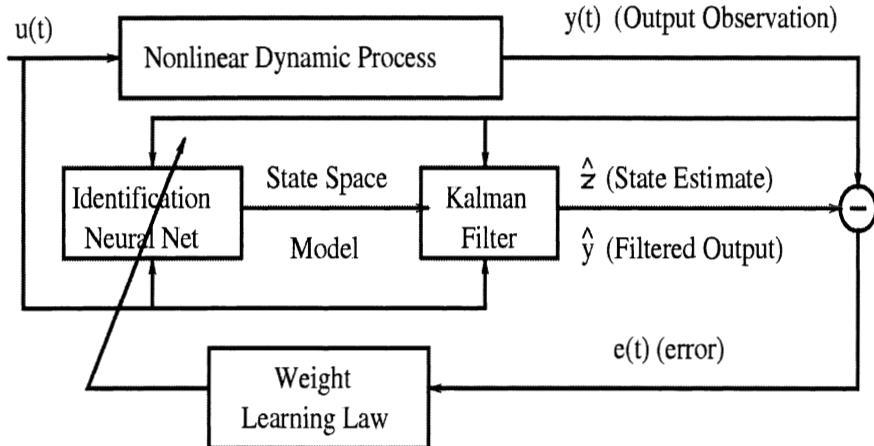


Fig. 6.9. Direct state estimation

The direct neurofuzzy state estimator of Figure 6.10 can also be viewed as a two layered network, in which the second layer is a Kalman filter that does not require training, the first layer being identical to the indirect method.

For this implementation, in order to use the NLMS algorithm (6.22) and (6.23) together with the Kalman filter, the state vector and observational vector have to be redefined as

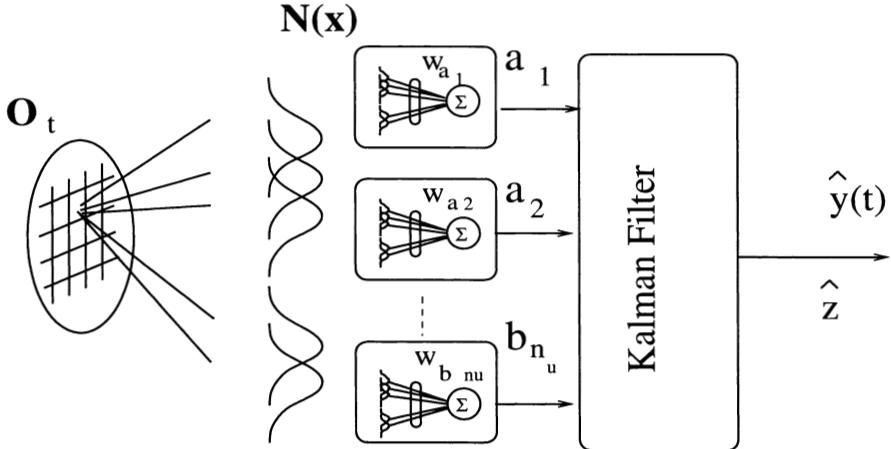


Fig. 6.10. The neural network for direct Kalman filtering

$$\mathbf{z}(t) = [\hat{y}(t-1), \dots, \hat{y}(t-n_y)]^T \quad (6.24)$$

$$\mathbf{x}'(t) = [\hat{y}(t-1), \dots, \hat{y}(t-n_y), u(t-1), \dots, u(t-n_u)]^T, \quad (6.25)$$

with $e(t) = y(t) - \hat{y}(t)$, for $\hat{y}(t)$ the filtered output at time t .

(iii) *Neurofuzzy State Estimation by Prefiltering through Principle Component Analysis*

A problem associated with the indirect state estimation is that due to the fact that system measurement $y(t)$ is subject to noise, the parameter identification of the state space form would incur a bias [111]. In practice, a filtered system signal, such as state vector given by (6.24), or alternatively, $\mathbf{z}(t) = [z(t-1), \dots, z(t-n_y)]^T$, where $z(t) = f(\mathbf{x})$, is representative of underlying process state vector and should be used. To achieve this, a prefilter can be used [111] that reconstructs $z(t) = f(\mathbf{x})$ from data D_N with noisy observations by using a prefiltering procedure based upon principal component analysis (PCA) [158], without prior knowledge of the actual system's structure. The basic idea is to identify an operating point dependent neurofuzzy model

$$\begin{aligned} y(t) &= a_1(\mathbf{O}_t)z(t-1) + \dots + a_{n_y}(\mathbf{O}_t)z(t-n_y) \\ &\quad + b_1(\mathbf{O}_t)u(t-1) + \dots + b_{n_u}(\mathbf{O}_t)u(t-n_u) + e(t), \end{aligned} \quad (6.26)$$

followed by its transformation into state space form of (6.21). Since $z(t)$ is unknown, it can be replaced by a prefiltered signal $\tilde{y}(t)$. Linear PCA is based on the singular value decomposition (SVD) of the covariance matrix $E[\mathbf{y}\mathbf{y}^T]$ as

$$E[\mathbf{y}\mathbf{y}^T] = \mathbf{V}\Lambda\mathbf{V}^T \quad (6.27)$$

where $\mathbf{V} \in \Re^{N \times N}$ is an orthonormal matrix consisting of N orthonormal eigenvectors $\mathbf{v}_j \in R^N$, $j = 1, 2, \dots, N$, such that $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N]$ and $\Lambda = \text{diag}\{\lambda_1, \dots, \lambda_N\} \in R^{N \times N}$, for λ_j , $j = 1, 2, \dots, N$, represent eigenvalues in decreasing order of magnitude, i.e. $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_N$. Construct a vector $\tilde{\mathbf{y}}$ using the first M principal components as follows:

$$\tilde{\mathbf{y}} = \sum_{j=1}^M c_j \mathbf{v}_j = \sum_{j=1}^M \langle \mathbf{y}, \mathbf{v}_j \rangle \mathbf{v}_j, \quad (6.28)$$

where $c_j^2 = \lambda_j$. Denoting $\tilde{\mathbf{y}}$ as $[\tilde{y}(1), \dots, \tilde{y}(N)]^T \in R^N$, then $\tilde{\mathbf{y}}$ is optimal for fixed M principal components in that the Euclidean norm $\|\mathbf{y} - \tilde{\mathbf{y}}\|$ is minimised. The above linear PCA is carried out on the output vector covariance, which is inappropriate for models of the form (5.36) where it has to be carried out on the regression matrix $\mathbf{A} \in R^{N \times p}$ [199]. Hence a subset of the eigenvectors of \mathbf{A} are sought as a basis for approximating the output vector \mathbf{y} . The SVD of \mathbf{A} is

$$\mathbf{A} = \mathbf{V} \Lambda \mathbf{U}, \quad (6.29)$$

where $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_p] \in \Re^{N \times p}$ is an orthonormal matrix consisting of p orthonormal eigenvectors $\mathbf{v}_j \in R^N$, $j = 1, 2, \dots, p$, $\Lambda = \text{diag}\{\lambda_1, \dots, \lambda_p\} \in \Re^{p \times p}$, where λ_j , $j = 1, 2, \dots, N$, represent eigenvalues in decreasing order of magnitude, i.e. $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p$, and $\mathbf{U} \in \Re^{p \times p}$ is also an orthonormal matrix. As the SVD decomposition is based on the regression matrix \mathbf{A} and not on $E[\mathbf{y}\mathbf{y}^T]$, then $c_j^2 \neq \lambda_j$, $\forall j$, implying that the optimal set of M eigenvectors used to construct $\tilde{\mathbf{y}}$ is not necessarily the first M principal eigenvectors of \mathbf{A} . However, a procedure based on the magnitude of the projection of the output vector \mathbf{y} on each eigenvector

$$c_j = \langle \mathbf{y}, \mathbf{v}_j \rangle, \quad j = 1, 2, \dots, p \quad (6.30)$$

uses a subset of M of the eigenvectors with largest magnitude of projections to construct the filtered output through

$$\tilde{\mathbf{y}} = \sum_{j=1}^M \tilde{c}_j \tilde{\mathbf{v}}_j, \quad (6.31)$$

where

$$\left. \begin{array}{l} k = \arg \{ \max \{ |c_i|, \text{ for } i = j, j+1, \dots, p \} \} \\ \tilde{c}_j = c_k \\ \tilde{\mathbf{v}}_j = \mathbf{v}_k \\ c_k = c_j \\ \mathbf{v}_k = \mathbf{v}_j \end{array} \right\} j = 1, 2, \dots, M. \quad (6.32)$$

The associated model residual vector is $\tilde{\mathbf{e}} = \mathbf{y} - \tilde{\mathbf{y}}$. As the orthonormal conditions $\tilde{\mathbf{v}}_i^T \tilde{\mathbf{v}}_j = 0$, $i \neq j$ and $\tilde{\mathbf{v}}_i^T \tilde{\mathbf{v}}_j = 1$, $i = j$ hold, hence

$$\mathbf{y}^T \mathbf{y} = \sum_{j=1}^M \tilde{c}_j^2 + \tilde{\mathbf{e}}^T \tilde{\mathbf{e}}. \quad (6.33)$$

An appropriate M can be determined from the residual sequence $\{\tilde{\mathbf{e}}(t)\}$ via some model validity model. Also from (6.32) and (6.33), $\tilde{c}_1^2 \geq \tilde{c}_2^2 \geq \dots \geq \tilde{c}_M^2$, implying that nonlinear PCA prefiltering is optimal in the sense that maximum information in the regression matrix is retrieved using the smallest number of principal components. Generally PCA transformation obscures physical relationships (see comparison note with OLS in Section 5.7) and in this context OLS is superior in system identification, however this method is effective in signal smoothing, such as being used here as a prefilter to generate additive noise models used for state estimation.

Example 6.1. Consider the nonlinear system

$$\begin{aligned} z(t) &= \frac{2.5z(t-1)z(t-2)}{1+z^2(t-1)+z^2(t-2)} + 0.3 \cos(0.5(z(t-1) + z(t-2))) \\ &\quad + 0.5u(t-1) \\ y(t) &= z(t) + \xi(t), \end{aligned} \quad (6.34)$$

where $u(t) = \sin(\pi * t/20)$ and $\xi(t) \sim N(0, 0.05^2)$ is used to generate a sequence of system output observation $y(t)$, $t = 1, 2, \dots, 1,000$ (see Figure 6.11). The first 500 samples $\{u(t), y(t)\}$ were used as an training or estimation data set. An order-2 B-spline neurofuzzy network was used to approximate the system (6.34), with input vector $\mathbf{x}(t) = [y(t-1), y(t-2), u(t-1)]^T$, i.e. $n_y = 2$, $n_u = 1$. On each input five 1-D basis functions were defined with knot vectors $\{-1.2, -1.1, -0.9, 0, 0.9, 1.1, 1.2\}$ and $\{-0.6, -0.5, -0.1, 0.65, 1.4, 1.8, 1.9\}$ for $u(t)$ and $y(t)$ respectively. As $z(t)$ is not directly available to achieve unbiased estimates the nonlinear PCA prefiltering approach was utilised to construct $\tilde{y}(t)$. The performance of this prefiltering is shown in Figure 6.12 showing that the residual is uncorrelated after prefiltering. The prefiltered signal is used in state space identification based on operating point neurofuzzy model of (6.26). The modelled state space model training performance is shown in Figure 6.13 demonstrating network convergence. Figure 6.14 is the result of applying the neurofuzzy Kalman filtering algorithm to the validation set $t = 501, 502, \dots, 1,000$ (a further example of this technique can be found in [111]).

6.5 Mixture of experts modelling

The mixture of experts network (MEN) data-based modelling architecture has attracted considerable interest [152] following the original work of Jacobs et al. [118]. The mixture approach is attractive in modelling dynamical systems since local models are formulated in a global form usually based on

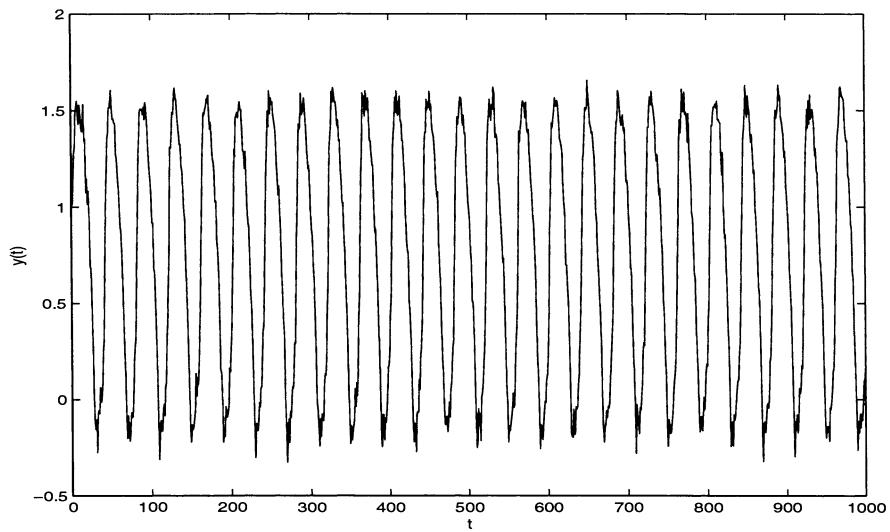


Fig. 6.11. The output data $y(t)$ in Example 6.1

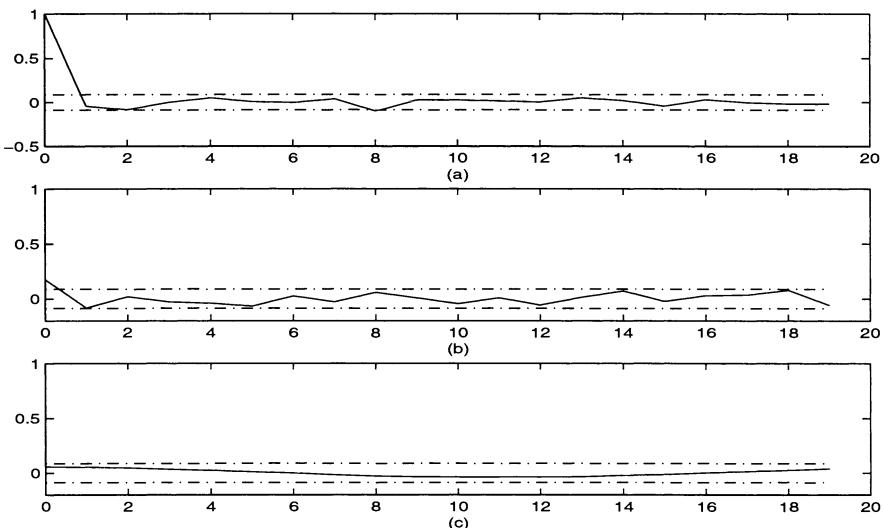


Fig. 6.12. Model validity tests in Example 6.1; (a) model validity test $\rho_{\tilde{e}\tilde{e}}(\tau)$; (b) model validity test $\rho_{(y\tilde{e})\tilde{e}^2}(\tau)$ and (c) model validity test $\rho_{(y\tilde{e})u^2}(\tau)$ (see (2.60) for model validity tests)

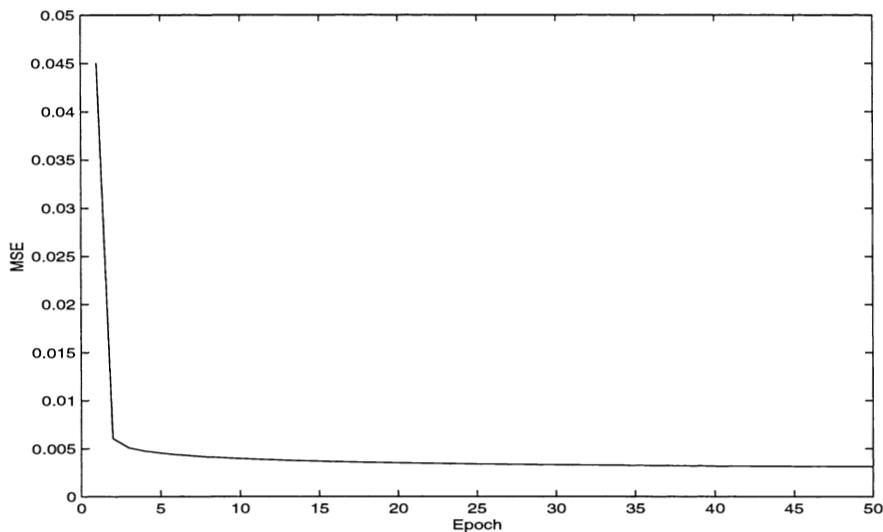


Fig. 6.13. The MSE in the estimation set in Example 6.1

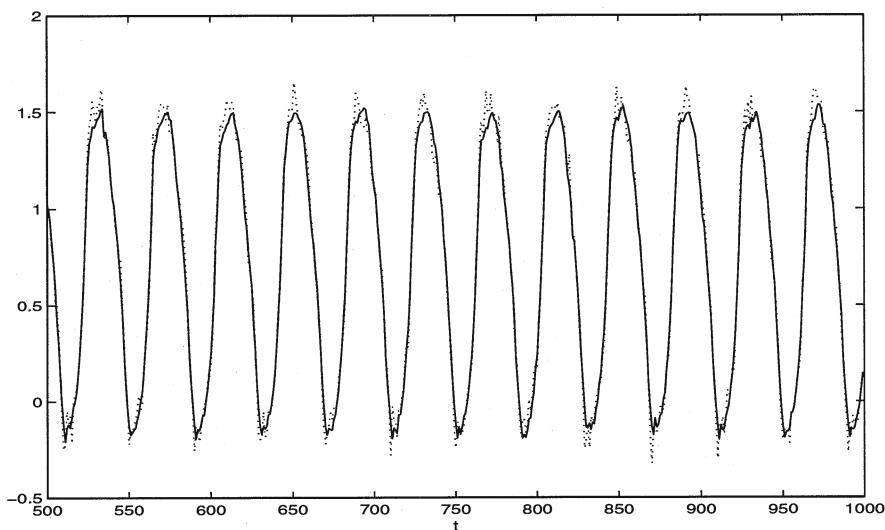


Fig. 6.14. The state estimator on the validation set using Kalman filter algorithm in Example 6.1; (solid line: filtered state variable $z(t)$ and dotted line: system observations $y(t)$)

probabilistic models allowing effective uncertainty representation as well as the ability to be trained adaptively and on line. Generally for a data set $D_N = \{u(t), y(t)\}_{t=1}^N$, the mixture model assumes that it has been generated from the composite density

$$p(u(t), y(t)) = \sum_{i=1}^K p(u(t), y(t)|i, \mathbf{w}_i)p(i),$$

where \mathbf{w}_i is the parameter of the i th submodel, K is the number of local probabilistic models. Given a particular input u , then the conditional density

$$p(y|u) = \sum_{j=1}^K p(j|u, \mathbf{w}_j)p(y|u, \mathbf{w}_j)$$

describes the input-output mapping estimated by the model, and $p(j|u, \mathbf{w}_j)$ is the probability that u belongs to the j th local region of the input space. This probability density plays the role of partitioning the input space, frequently it is selected as Gaussian. $p(y|u, \mathbf{w}_j)$, $j = 1, 2, \dots, K$, are the outputs from each expert. If each expert is assumed to have a Gaussian distribution, then the expectation-maximisation (EM) algorithm [126] can be used as an effective training mechanism. We will return to the MEN/EM algorithm in the context of neurofuzzy linearisation for state estimation in Section 8.3.

In keeping with linear-in-the-parameters modelling the MEN architecture can be re-expressed as

$$p(y|u, \mathbf{w}) = \sum_{j=1}^K g_j p(y|u, \mathbf{w}_j), \quad (6.35)$$

with $g_j \geq 0$, $\sum_{j=1}^K g_j = 1$, and \mathbf{w} the parameters set $\{g_j, \mathbf{w}_j\}_{j=1}^K$, where $g_j \equiv p(j|u, \mathbf{w}_j)$ are gating terms. If the individual experts or submodels have been predefined or evaluated as Gaussian with mean $\hat{y}_j(u, y) = \hat{y}_j(\mathbf{x})$, then the expected output of the model

$$\hat{y}(\mathbf{x}) = \sum_{j=1}^K g_j \hat{y}_j(\mathbf{x}), \quad (6.36)$$

which is a linear-in-the-parameters model with convex constraints on the gate combination parameters (see Figure 6.15).

The submodels $\{\hat{y}_j(\mathbf{x})\}$ can be viewed as a library of possible primitive models that when combined in an optimal manner fit the underlying data. The MEN submodels can be local, semi global as well as global in the coverage of the state space. Formally if the gating parameters is independent of either the operating point or the input vector then the MEN approach is a global multiple modelling approach (equally if the gating parameter is operating point dependent, then the MEN algorithm is a form of multiple model selection). The library of models can include linear, polynomial and radial

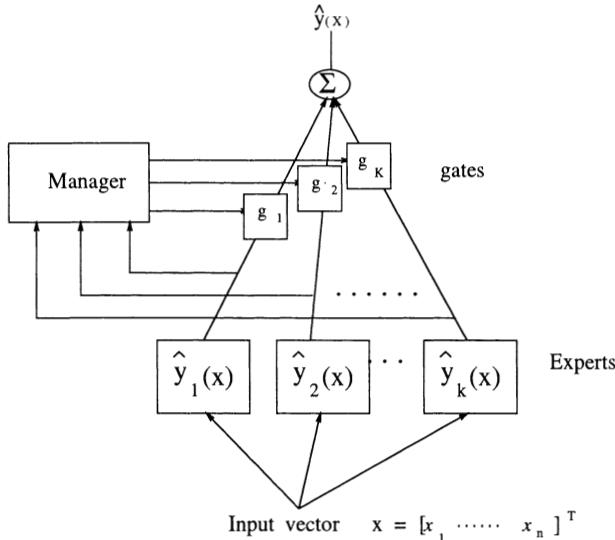


Fig. 6.15. Mixture of experts model network

basis functions, etc. The MEN problem is then the selection of the minimum number of gating functions g_j and submodels that fit $y(\mathbf{x})$ in some optimal sense. For general linear-in-the-parameters models without constraints in the parameters g_i , a forward regression method based upon orthogonal least squares (OLS) [47] has been implemented to solve the structure selection problem. Given the full set of candidate submodels/bases, the OLS selects the most significant bases one by one in a forward regression manner, terminating when a subset of bases satisfies some cost criteria. However due to the convex constraints on g_j , OLS is not applicable to the construction of a MEN architecture, instead a new structure recursive construction algorithm [106] based on a forward constrained regression (FCR) procedure selects the most significant model base one by one to minimise the overall system approximation error, whilst the gate combination parameters, g_j , are automatically adjusted to satisfy the constraint conditions. The MEN system is sequentially constructed such that as a new expert (submodel) is included, the updated model is formulated as a two subsystems convex combination of the original MEN and the new submodel/expert.

Suppose that there are K submodels ranked in a sequence, \hat{y}_j , $j = 1, 2, \dots, K$, and at the r th forward regression, the overall model is constructed using the first r submodels, so that

$$\hat{y}^{(r)}(\mathbf{x}) = \sum_{j=1}^r g_j^{(r)} \hat{y}_j(\mathbf{x}), \quad (6.37)$$

where gate combination coefficients $g_j^{(r)}$ satisfy $g_j^{(r)} \geq 0$, $\sum_{j=1}^r g_j^{(r)} = 1$, for $r = 2, 3, \dots, K$. Initially only one submodel (the most significant in terms of fit) is used, i.e.

$$\hat{y}^{(1)}(\mathbf{x}) = \hat{y}_1(\mathbf{x}) \quad (6.38)$$

at the r th step, the MEN construction of the next model by including the r th submodel is

$$\hat{y}^{(r)}(\mathbf{x}) = \lambda_{r-1} \hat{y}^{(r-1)}(\mathbf{x}) + (1 - \lambda_{r-1}) \hat{y}_r(\mathbf{x}), \quad (6.39)$$

where $0 \leq \lambda_{r-1} \leq 1$, $\forall r$.

Lemma 6.1: The MEN system constructed by FCR satisfies the convex constraint conditions for the gate combination parameters, i.e. $g_j^{(r)} \geq 0$, $\sum_j g_j^{(r)} = 1$, for $r = 2, 3, \dots, K$.

Proof: By induction, substituting (6.38) into (6.39) for $r = 2$ gives $\hat{y}^{(2)}(\mathbf{x}) = \lambda_1 \hat{y}_1(\mathbf{x}) + (1 - \lambda_1) \hat{y}_2(\mathbf{x})$, so that $g_1^{(2)} = \lambda_1$, $g_2^{(2)} = 1 - \lambda_1$, hence the convex constraint conditions are satisfied. Suppose then at the r th step, the convex conditions on $g_j^{(r)}$ are satisfied, then at step $(r+1)$, substituting (6.37) into (6.39) where r is replaced by $(r+1)$, the resultant MEN is constructed by

$$\begin{aligned} \hat{y}^{(r+1)}(\mathbf{x}) &= \lambda_r \hat{y}^{(r)}(\mathbf{x}) + (1 - \lambda_r) \hat{y}_{r+1}(\mathbf{x}) \\ &= \sum_{j=1}^r \lambda_r g_j^{(r)} \hat{y}_j(\mathbf{x}) + (1 - \lambda_r) \hat{y}_{r+1}(\mathbf{x}). \end{aligned} \quad (6.40)$$

Expanding (6.37) with r replaced by $(r+1)$ yields

$$\hat{y}^{(r+1)}(\mathbf{x}) = \sum_{j=1}^r g_j^{(r+1)} \hat{y}_j(\mathbf{x}) + g_{r+1}^{(r+1)} \hat{y}_{r+1}(\mathbf{x}). \quad (6.41)$$

Comparing (6.40) with (6.41) gives

$$\begin{aligned} g_j^{(r+1)} &= \lambda_r g_j^{(r)}, \quad j = 1, 2, \dots, K \\ g_{r+1}^{(r+1)} &= 1 - \lambda_r. \end{aligned} \quad (6.42)$$

Assuming that $\sum_{j=1}^r g_j^{(r)} = 1$, then

$$\sum_{j=1}^{(r+1)} g_j^{(r+1)} = \lambda_r \sum_{j=1}^r g_j^{(r)} + (1 - \lambda_r) = 1. \quad (6.43)$$

Clearly, $g_j^{(r+1)} \geq 0$, for $j = 1, 2, \dots, r$ follows by applying $g_j^{(r)} \geq 0$, and $\lambda_r > 0$ in (6.42). This establishes Lemma 6.1. \square

The MEN model construction algorithm is a supervised training procedure, based on a training data set of input-output data pairs $D_N =$

$\{\mathbf{x}(t), y(t)\}_{t=1}^N$, via the FCR procedure together with a forward selection criterion to optimise the system model approximation error:

$$V_N^{(r)} = \frac{1}{N} \sum_{t=1}^N [y(t) - \hat{y}^{(r)}(\mathbf{x})]^2, \quad (6.44)$$

at the r th regression step. By minimising this cost, the procedure aims to add the most suitable submodel to the existing MEN system, and correspondingly find the optimal combination parameter λ_{r-1} for the chosen submodel. Similarly the gate combination parameter, $g_j^{(r)}$, can be computed based on similar derivatives in the FCR procedure.

The MEN Modelling Algorithm

The MEN modelling process is in two stages: construction and pruning. The construction stage includes the following steps:

1. MEN is initialised with a single submodel $y^{(1)}(\mathbf{x})$ from amongst the K available submodels, and is selected as the submodel with least approximation error from:

$$j_1 = \arg \min_j \{V_N^{(j)} = \frac{1}{N} \sum_{t=1}^N [y(t) - \hat{y}_j(\mathbf{x})]^2, \forall j = 1, 2, \dots, K\}. \quad (6.45)$$

The initial model output is then (6.38).

2. At the r th step ($r = 2, 3, \dots$), the MEN system is composed of r submodels, the remaining submodels $j = r, \dots, K$ form a candidate set for the new submodel to be selected. The cost $V_N^{(r)}$ of (6.44) is computed for each candidate and meanwhile whilst varying the combination parameter λ_{r-1} in the range $[0, 1]$, i.e. via direct search, the resultant MEN with least approximation error and corresponding λ_{r-1} is selected from:

$$\begin{aligned} & \{\lambda_{r-1}, j_r\} \\ &= \arg \min_{j, \lambda} \{V_N^{(j)} = \frac{1}{N} \sum_{t=1}^N [y(t) - \lambda \hat{y}^{(j-1)}(\mathbf{x}) - (1 - \lambda) \hat{y}_j(\mathbf{x})]^2\} \\ & \forall \lambda \in [0, 1], \forall j = r, r + 1, \dots, K. \end{aligned} \quad (6.46)$$

The j_r th submodel is permuted with the r th submodel and chosen as the new submodel in the MEN overall model at the r th step, whose output is given by (6.39). The gate parameters of the MEN model is computed from the recursion (6.42) with r replaced by $(r - 1)$.

3. Set $r = r + 1$ and the procedure (ii) repeated until $r = K$ or some termination criteria such as approximation error or number of submodels is achieved.

In the pruning stage, to ensure overall model parsimony, some means for excluding or pruning out say the i th ($i < r$) submodel is necessary. The resultant pruned MEN model can be formed with the remaining $(r - 1)$ submodels through:

$$\hat{y}_{-i}^{(r-1)}(\mathbf{x}) = \sum_{j=1, j \neq i}^r \frac{g_j^{(r)}}{1 - g_i^{(r)}} \hat{y}_j(\mathbf{x}). \quad (6.47)$$

It can be clearly seen that $\sum_{j=1, j \neq i}^r g_j^{(r-1)} = 1$, $g_j^{(r-1)} \geq 0$. Recursive backward submodel pruning starts with a full MEN model consisting of K submodels identified as $\hat{y}^{(K)}(\mathbf{x}) = \sum_{j=1}^{(K)} g_j^{(K)} \hat{y}_j(\mathbf{x})$, then at $(K - r + 1)$ step, $r = K, K - 1, \dots$, based on (6.47), constructs the $(r - 1)$ MEN model by excluding one submodel (i th) from the existing r submodels which are potential candidates for pruning. For each excluded candidate submodel, the resultant MEN model system with least approximation error is computed from:

$$i_r = \arg \min_i \{V_N^{(-r)} = \frac{1}{N} \sum_{t=1}^N [y(t) - \hat{y}_{-i}^{(r-1)}(\mathbf{x})]^2, i = 1, \dots, K\}. \quad (6.48)$$

The i_r th submodel is chosen as the submodel to be pruned at the $(K - r - 1)$ th step, the remaining $(r - 1)$ submodels in the MEN model are then relabelled from 1 to $(r - 1)$. Backward pruning continues by setting $r = r - 1$ at each step until $V_N^{(-r)}$ violates a threshold or the desired number of submodels is reached.

Note: The algorithm solves a low dimensional nonlinear optimisation over a small range $[0,1]$ for λ_{r-1} via direct search.

Example 6.2: Variable star brightness modelling (<http://www-personal.buseco.monash.edu.au/hyndman/TSDL/index.htm>)

This time series is shown in Figure 6.16. Assume the maximum lag of the global system as $n_y = 24$. The input vector was predetermined as $[y(t - 1), \dots, y(t - 24)]^T$. The assumed set of prior submodels is 23 linear autoregressive models of the form:

$$\hat{y}_r(t) = w_{r1}y(t - 1) + w_{r2}y(t - j), \quad r = 1, \dots, 23, \quad j = 2, \dots, 24. \quad (6.49)$$

where $\mathbf{w}_r = [w_{r1}, w_{r2}]^T \in R^2$ are weights estimated from the data by LMS. Also 10 radial basis function (RBF) models based on thin plate spline basis functions were generated as submodel candidates. Five centres were also predetermined across the input distribution range (see Figure 6.17). Each RBF network uses three centres, from any three centre combination of the five possible centres ($C_5^3 = 10$). Different combinations were used to construct 10 distinctive RBF submodels:

$$\hat{y}_r(t) = \psi_r^T \mathbf{w}_r, \quad r = 24, \dots, 33, \quad (6.50)$$

where $\psi_r \in R^3$ are RBF functions and parameters $\mathbf{w}_r \in R^3$ are estimated by LMS.

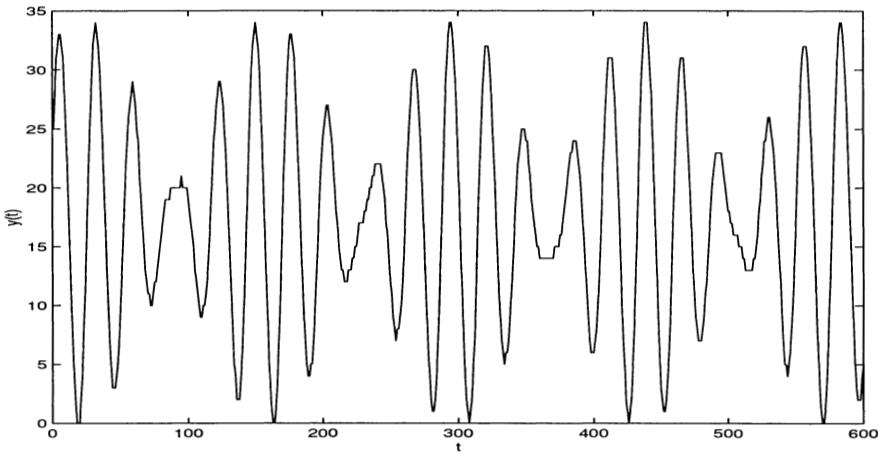


Fig. 6.16. The daily brightness of a variable star

All above 33 submodels were used to construct the MEN network, the first model selected was linear model ($r = 2$):

$$\hat{y}_2(t) = 1.452y(t-1) - 0.4698y(t-3),$$

with a MSE $V_N = 0.6129$, as this submodel had least noise. The second submodel selected was a RBF network ($r = 29$) whose individual MSE was 52.14. At the third iteration the submodel selected was a linear model ($r = 18$)

$$\hat{y}_{18}(t) = 0.9346y(t-1) - 0.0864y(t-19),$$

whose individual MSE was 2.7766. As each submodel is added the global MSE is reduced, as shown in Figure 6.18. The finally selected MEN model had just three submodels with a MSE of 0.3683 (NB: less than any individual submodel MSE). The gate parameters used at each iteration are shown in Table 6.2. The one-step ahead prediction from the consequent MEN system is illustrated in Figure 6.19 over $t = 200 \sim 400$.

This MEN algorithm can be directly used in other modelling schemes such as Takagi–Sugeno submodels, neurofuzzy or probabilistic submodels (as in the original scheme of Jordan [118]). The prime weakness of the MEN approach is the choice of primitive submodels. However problem domain knowledge makes this task relatively straightforward in practice. One means of resolving this problem is by constraining the choice of local submodels and selecting

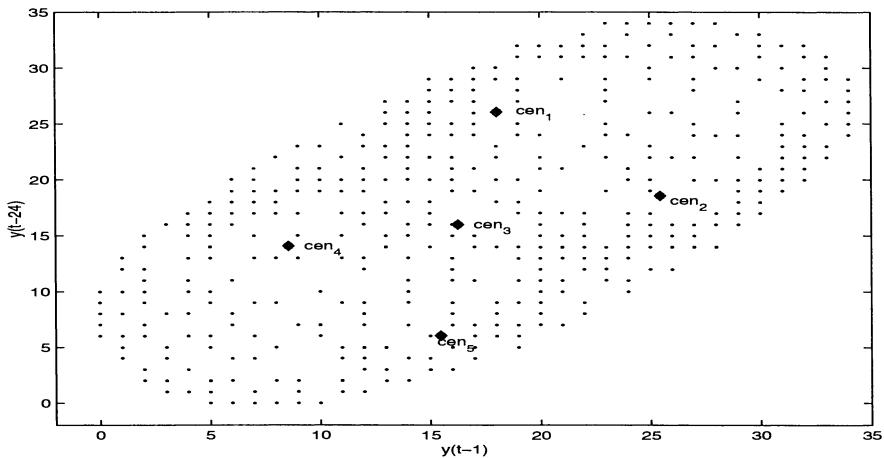


Fig. 6.17. Five centres used in the construction of RBF network experts

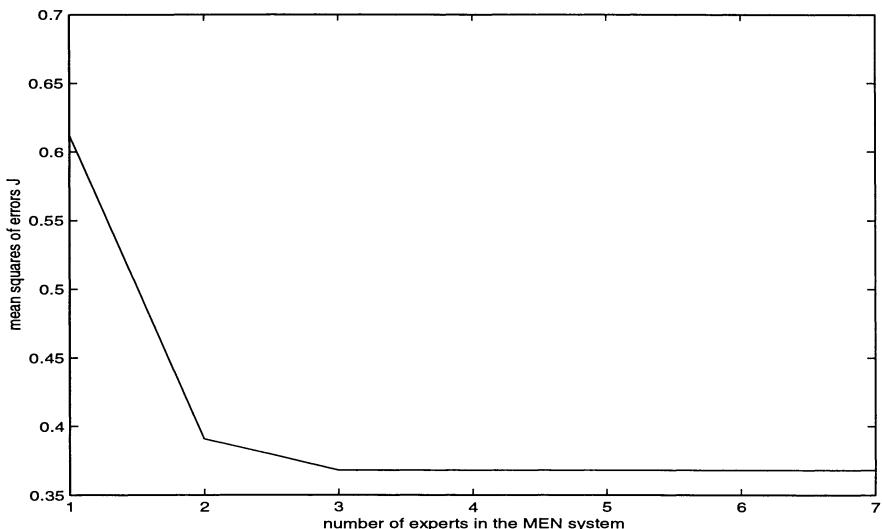


Fig. 6.18. The evolution of the MSE of the MEN model

Table 6.2. Gate parameters

Step r	Chosen expert j	λ_{r-1}	g_2	g_{29}	g_{18}
1	2	1	1		
2	29	0.9400	0.9400	0.0600	
3	18	0.9100	0.8554	0.0546	0.0900

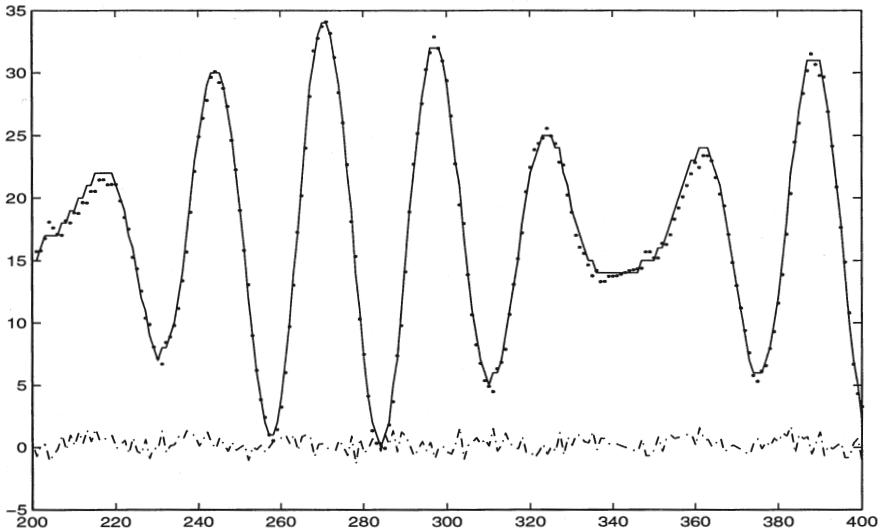


Fig. 6.19. System actual output (solid line), model one-step ahead prediction (dotted line), and model residual (dashdotted line)

the gate parameters so as to maximise the sensitivity of the composite model output to the chosen expert submodel, to this end, a fast parsimonious parallel modelling algorithm [99] can be readily derived which exploits the dual attributes of local linear neurofuzzy models and the MEN architecture. From (6.36) assuming that each expert y_j is independent, it follows that

$$\frac{\partial \hat{y}(\mathbf{x})}{\partial \hat{y}_j(\mathbf{x})} = g_j. \quad (6.51)$$

i.e. the gate parameters in the MEN system measure the sensitivity of the composite model output with respect to associated submodel or expert. A good modelling strategy would be to rank the expert or submodel gate parameters in the order of the greatest sensitivity and then to utilise the first K experts with greatest sensitivity, whilst normalising the associated gate parameter through

$$g_j^{new}(t) = \frac{N_{k_j}(t)}{\sum_{j=1}^K N_{k_j}(t)}, \quad (6.52)$$

where k_j denotes index of the selected local submodels, ensuring that for any selected K , $g_j^{new}(t) \geq 0$, $\sum_1^K g_j^{new}(t) = 1$, so that the system output is given as

$$\hat{y}(t) = \sum_{j=1}^K g_j^{new}(t) \hat{y}_{k_j}(\mathbf{x}(t)). \quad (6.53)$$

Rather than leave the selection of the submodels $\hat{y}_{k_j}(\mathbf{x}(t))$ open, we select them as Takagi–Sugeno submodels or linear in the observations with unknown parameter vector $\hat{\mathbf{w}}_{k_j}$, i.e.

$$\hat{y}_{k_j}(\mathbf{x}(t)) = \hat{\mathbf{w}}_{k_j}^T \mathbf{x}(t). \quad (6.54)$$

Substituting into (6.53), gives

$$\hat{y}(t) = \mathbf{p}(t)^T \hat{\mathbf{w}}^{old}(t), \quad (6.55)$$

where

$$\begin{aligned} \mathbf{p}(t) &= [g_1^{new}(t)x_1, \dots, g_K^{new}(t)x_1, \dots, \\ &\quad g_1^{new}(t)x_n, \dots, g_K^{new}(t)x_n]^T \in \Re^{Kn} \end{aligned}$$

and

$$\begin{aligned} \hat{\mathbf{w}}^{old}(t) &= [\hat{w}_{1k_1(t)}(t-1), \dots, \hat{w}_{1k_K(t)}(t-1), \\ &\quad \dots, \hat{w}_{nk_1(t)}(t-1), \dots, \hat{w}_{nk_K(t)}(t-1)]^T \end{aligned}$$

The model parameters $\hat{\mathbf{w}}(t)$ can be found from a modified version of the normalised least squares learning law

$$\hat{\mathbf{w}}_{k_j}(t) = \hat{\mathbf{w}}_{k_j}(t-1) + \eta N_{k_j}(\mathbf{x}(t)) \frac{\mathbf{x}(t)e(t)}{\mathbf{x}(t)^T \mathbf{x}(t) + c}, \quad j = 1, \dots, K, \quad (6.56)$$

for $e(t) = y(t) - \hat{y}(t)$, $\eta \in (0, 2)$ is the learning rate, c is a positive arbitrarily small constant, which is a reduced parameter vector, as the MEN output sensitivity term (6.52) limits terms in the model (6.55) to K times the dimension of the input vector $\mathbf{x}(t)$. Also each learning law or rule (6.56) is effectively independent, allowing the additional advantage of parallel processing. The convergence property of this algorithm can be easily studied if we assume that the training data is compatible with the model structure used for estimation. That is, without loss of generality, it is assumed that the data is generated by the model

$$y(t) = \Psi^T(t)\mathbf{w}, \quad (6.57)$$

where \mathbf{w} is the true system parameter vector to be estimated, and

$$\begin{aligned} \Psi(t) &= [N_1(\mathbf{x}(t))x_1, \dots, N_1(\mathbf{x}(t))x_n, \dots, N_p(\mathbf{x}(t))x_1, \dots, N_p(\mathbf{x}(t))x_n]^T \\ &= [N_1(\mathbf{x}(t))\mathbf{x}(t)^T, \dots, N_p(\mathbf{x}(t))\mathbf{x}(t)^T]^T \in \Re^{pn}, \end{aligned} \quad (6.58)$$

for p , the total number of possible fuzzy rules for model completeness.

Theorem 6.2: Applying the parallel learning algorithm (6.52)–(6.56) to data generated by (6.57), then

- (i) $\|\hat{\mathbf{w}}(t) - \mathbf{w}\| \leq \|\hat{\mathbf{w}}(t-1) - \mathbf{w}\| \leq \|\hat{\mathbf{w}}(0) - \mathbf{w}\|, \quad t \geq 1$
- (ii) $\lim_{t \rightarrow \infty} \frac{e(t)}{\sqrt{c + \mathbf{x}(t)^T \mathbf{x}(t)}} = 0$
- (iii) $\lim_{t \rightarrow \infty} \|\hat{\mathbf{w}}(t) - \hat{\mathbf{w}}(t-k)\| = 0 \quad \text{for any finite } k$

Proof. Define $\tilde{\mathbf{w}}(t) = \hat{\mathbf{w}}(t) - \mathbf{w}$ and $V(t) = \tilde{\mathbf{w}}(t)^T \tilde{\mathbf{w}}(t) = \|\tilde{\mathbf{w}}(t)\|^2$. Applying (6.52-6.57), gives (on setting $k_j = j$, $K = p$) $e(t) = y(t) - \sum_{j=1}^p N_j(\mathbf{x})$ $\hat{\mathbf{w}}_j(t)^T \mathbf{x}(t) = \Psi(t)^T \mathbf{w} - \Psi(t)^T \hat{\mathbf{w}}(t-1) = -\Psi(t)^T \tilde{\mathbf{w}}(t-1)$.

Combining (6.56) (on setting $k_j = j, K = p$) as np -dimensional vector equation

$$\begin{aligned}\hat{\mathbf{w}}(t) &= \hat{\mathbf{w}}(t-1) + \eta \frac{[N_1(\mathbf{x})\mathbf{x}(t)^T, \dots, N_p(\mathbf{x})\mathbf{x}(t)^T]^T e(t)}{\mathbf{x}(t)^T \mathbf{x}(t) + c} \\ &= \hat{\mathbf{w}}(t-1) + \eta \frac{\Psi(t)e(t)}{\mathbf{x}(t)^T \mathbf{x}(t) + c},\end{aligned}\quad (6.59)$$

whereas from the definition of $V(t)$

$$V(t) - V(t-1) = \chi(t) \frac{\eta e^2(t)}{\mathbf{x}(t)^T \mathbf{x}(t) + c}, \quad (6.60)$$

where

$$\begin{aligned}\chi(t) &= -2 + \frac{\eta \Psi^T(t) \Psi(t)}{\mathbf{x}(t)^T \mathbf{x}(t) + c} \\ &= -2 + \frac{\eta \sum_{j=1}^p N_j^2(\mathbf{x}) \mathbf{x}^T(t) \mathbf{x}(t)}{\mathbf{x}(t)^T \mathbf{x}(t) + c} \leq -\gamma < 0.\end{aligned}\quad (6.61)$$

Since $c \geq 0$, $0 \leq \eta \leq 2$, and $0 \leq \sum_{j=1}^p N_j^2(\mathbf{x}) \leq 1$; whence property (i) follows.

Also from (6.60)

$$V(t) = V(0) + \sum_{k=1}^t \chi(k) \frac{\eta e^2(k)}{\mathbf{x}(k)^T \mathbf{x}(k) + c}, \quad (6.62)$$

hence

$$\sum_{k=1}^t \frac{\eta e^2(k)}{\mathbf{x}(k)^T \mathbf{x}(k) + c} \leq \frac{1}{\gamma} [V(0) - V(t)], \quad (6.63)$$

then since $0 \leq V(t) \leq V(0)$, then property (ii) follows.

From (6.59)

$$\begin{aligned}&\|\hat{\mathbf{w}}(t) - \hat{\mathbf{w}}(t-1)\|^2 \\ &= \frac{\eta^2 \Psi^T(t) \Psi(t) e^2(t)}{[\mathbf{x}(t)^T \mathbf{x}(t) + c]^2} \\ &= \frac{\eta^2 e^2(t)}{\mathbf{x}(t)^T \mathbf{x}(t) + c} \left(1 - \frac{c}{\mathbf{x}(t)^T \mathbf{x}(t) + c}\right) \sum_{j=1}^p N_j^2(\mathbf{x}(t)) \rightarrow 0\end{aligned}\quad (6.64)$$

as $t \rightarrow \infty$, following Property (ii), $c > 0$ and $0 \leq \sum_{j=1}^p N_j^2(\mathbf{x}(t)) \leq 1$. Hence

$$\begin{aligned}\|\hat{\mathbf{w}}(t) - \hat{\mathbf{w}}(t-k)\|^2 &= \left\| \sum_{i=1}^k [\hat{\mathbf{w}}(t) - \hat{\mathbf{w}}(t-i)] \right\|^2 \\ &\leq \sum_{i=1}^k \|\hat{\mathbf{w}}(t-i+1) - \hat{\mathbf{w}}(t-i)\|^2 \rightarrow 0\end{aligned}\quad (6.65)$$

as $t \rightarrow \infty$, for finite k ; establishing the theorem. \square

Note: The parallel learning algorithm (6.53)–(6.56) does not use the full regression vector $\Psi(t) \in \mathbb{R}^{pn}$ used in the above proof, but only the input vector $\mathbf{x}(t) \in \mathbb{R}^n$ ($n \ll pn$)

Example 6.3: Consider the chemical model benchmark process [44]

$$\begin{aligned}y(t) = & \frac{2.5y(t-1)y(t-2)}{1 + y^2(t-1) + y^2(t-2)} + 0.3 \cos(0.5(y(t-1) + y(t-2))) \\ & + 1.2u(t-1) + \xi(t),\end{aligned}\quad (6.66)$$

where the system input is deterministic and given by $u(t) = \frac{1}{2}[\sin(\pi * t/20) + \sin(\pi * t/50)]$, and the plant noise sequences $\xi(t) \sim N(0, 0.05^2)$. One thousand data samples $\{y(t), u(t)\}$ were derived (see Figure 6.20 for the output sequence). The first 500 were used for model estimation, the second 500 for model validation. An order-3 B-spline neurofuzzy network was selected together with an input vector $\mathbf{x}(t) = [y(t-1), y(t-2), u(t-1)]^T$, i.e. $n_y = 2$, $n_u = 1$ (this is the only prior plant knowledge being used). Knot vectors for the output and input respectively were defined as

$$\{-1.2, -1.1, -1.0, -0.4, 0.2, 0.8, 1.4, 2.0, 2.6, 2.7, 2.8\},$$

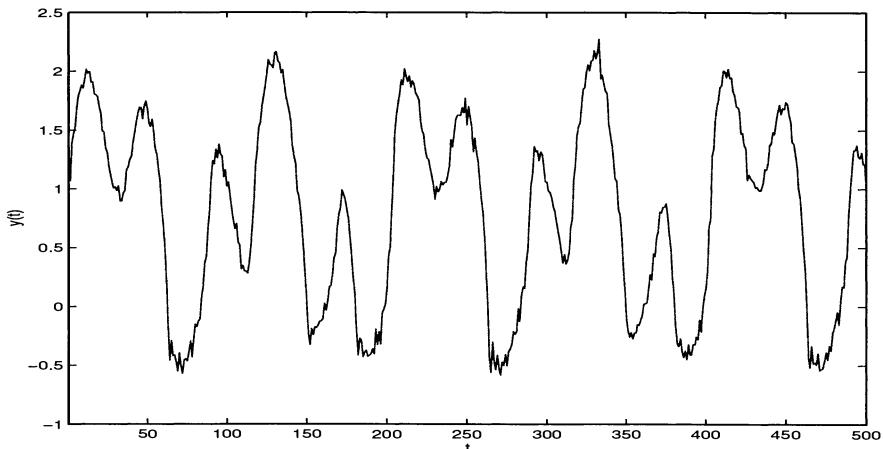


Fig. 6.20. The output sequence in Example 6.3

and

$$\{-1.3, -1.2, -1.1, -0.8, -0.4, 0, 0.4, 0.8, 1.1, 1.2, 1.3\},$$

resulting in nine 1-D basis functions for each network input. The complete 512 multi-dimensional basis functions were formed in the usual manner using the tensor product of 1-D basis functions; each of these 512 basis functions is a potential gate parameter of the MEN system based on the connection of the neurofuzzy system with MEN system. Models with $K = 1, \dots, 10$ experts (out of the total 512 experts) were developed using a learning rate $\eta = 0.3$ for 50 training epochs to ensure adequate convergence of the MSE over the training data set. Figure 6.21 illustrates the MSE convergence with regard to the number of experts and training epochs. A final model with $K = 8$ experts results in a model with only 24 parameters and a MSE of 0.058². Model validation indicates that the model residual is uncorrelated (Figure 6.22), and the validation data set for $K = 8$ model has a MSE of 0.07². Figure 6.23 shows the one-step ahead prediction of the model over the validation set.

6.6 Multi-input–Multi-output (MIMO) modelling via input variable selection

In this section we introduce a multivariable neurofuzzy modelling algorithm that achieves automatic parsimonious structured piecewise linear submodels (akin to Takagi–Sugeno models) through input variable selection by utilising a version of the Gram–Schmidt orthogonal least squares algorithm.

6.6.1 MIMO NARX neurofuzzy model decomposition

Consider the general multi-input–multi-output NARX model

$$\mathbf{y}(t) = \mathbf{f}(\mathbf{y}(t-1), \dots, \mathbf{y}(t-n_y), \mathbf{u}(t-1), \dots, \mathbf{u}(t-n_u)) + \Xi(t) \quad (6.67)$$

where $\mathbf{y}(t) = [y_1(t), y_2(t), \dots, y_m(t)]^T \in \mathbb{R}^m$, $\mathbf{u}(t) = [u_1(t), u_2(t), \dots, u_r(t)]^T \in \mathbb{R}^r$ represent observable multi-dimensional system outputs and inputs, respectively, and $\Xi(t) \in \mathbb{R}^m$ is model residual error. Define the observation vector $\mathbf{x}(t) \in \mathbb{R}^n$, for $n = mn_y + rn_u$, as

$$\begin{aligned} \mathbf{x}(t) &= [x_1(t), x_2(t), \dots, x_n(t)]^T \\ &= [y_1(t-1), \dots, y_1(t-n_y), \dots, y_m(t-1), \dots, y_m(t-n_y), \\ &\quad u_1(t-1), \dots, u_1(t-n_u), \dots, u_r(t-1), \dots, u_r(t-n_u)]^T. \end{aligned} \quad (6.68)$$

Assuming that (6.68) contains all the terms to identify (6.67), then the modelling task is to find the most parsimonious input variable selection or subset of $\mathbf{x}(t)$, that generates a rule base that provides a good statistical fit to training data and has good generalisation. (6.67) can be readily decomposed into m scalar MISO NARX models

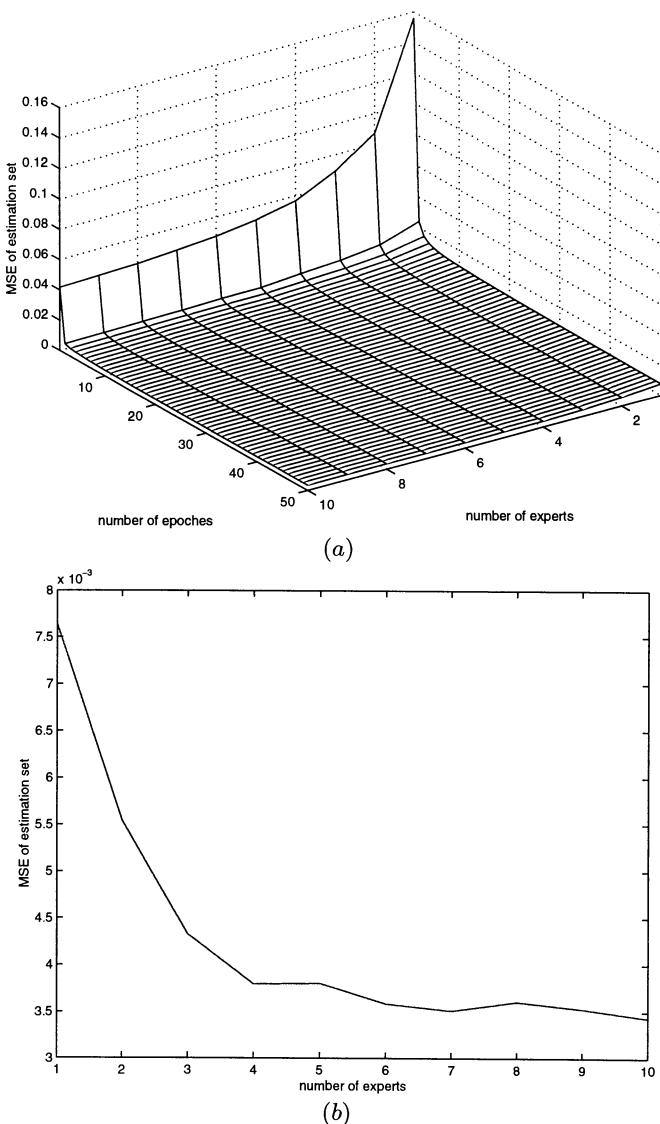


Fig. 6.21. Simulation results of Example 6.3; (a) the MSE of models with respect to number of experts (model complexity) and training epochs and (b) the MSE of models with respect to number of experts (model complexity) for training epoch 50

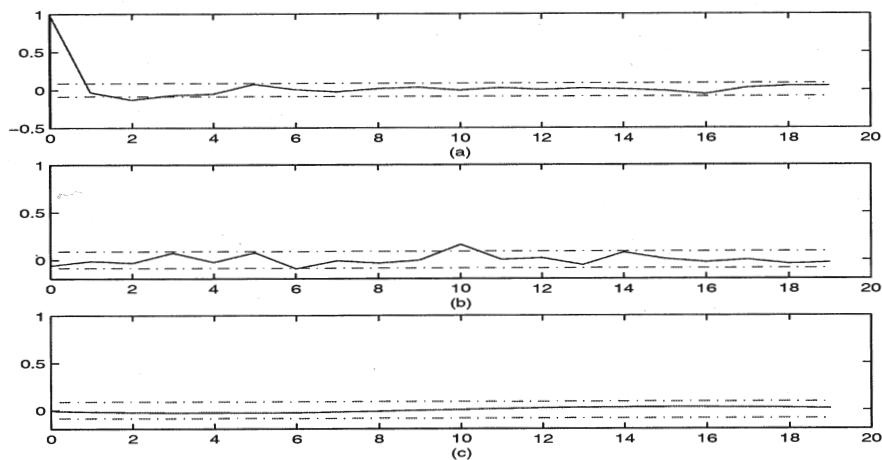


Fig. 6.22. Model validity tests for $K = 8$ experts system in Example 6.3; (a) model validity test $\rho_{ee}(\tau)$; (b) model validity test $\rho_{(ye)e^2}(\tau)$ and (c) model validity test $\rho_{(ye)u^2}(\tau)$ (see (2.60) for model validity tests)

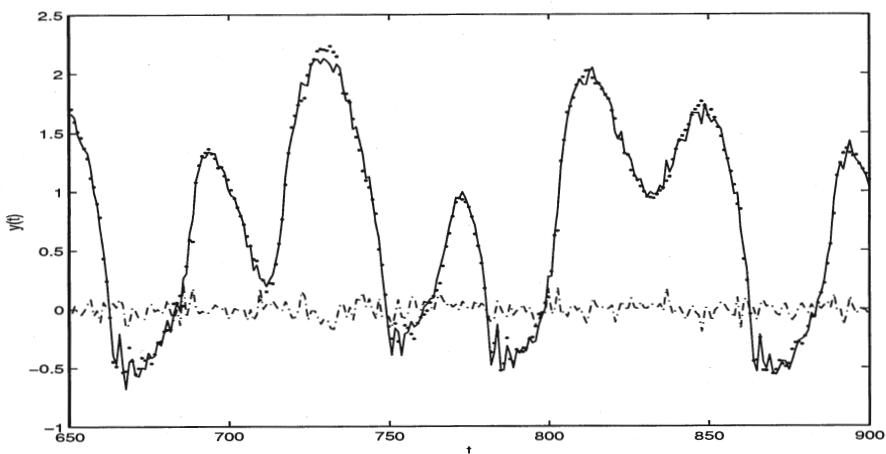


Fig. 6.23. The output sequence over the validation data in Example 6.3; (solid line: system outputs; dotted line: model predictions and broken line: model residuals)

$$\begin{aligned} y_{i_y}(t) &= f_{i_y}(\mathbf{y}(t-1), \dots, \mathbf{y}(t-n_y), \mathbf{u}(t-1), \dots, \mathbf{u}(t-n_u)) + \xi_{i_y}(t) \\ &= f_{i_y}(\mathbf{x}(t)) + \xi_{i_y}(t), \quad i_y = 1, 2, \dots, m. \end{aligned} \quad (6.69)$$

A large number of dynamical processes have local linear behaviour around an operating point $\mathbf{x}(t_0)$, where it can be assumed that (6.69) is dominated by its first order derivative evaluated at $\mathbf{x}(t) = \mathbf{x}(t_0)$, such as the Takagi–Sugeno model of (4.36) is effective

$$IF (x_1 \text{ is } A_1^{j_1} \text{ AND } \dots \text{ AND } x_n \text{ is } A_n^{j_n}) \text{ THEN } (y_{i_y} \text{ is } \hat{y}_j(\mathbf{x})), \quad (6.70)$$

where $j = 1, 2, \dots, p$ is an index corresponding to the sequence $\{j_1, \dots, j_n\}$, with $j_i = 1, 2, \dots, m_i$, $p = \prod_{i=1}^n m_i$. For simplicity, $\hat{y}_j(\mathbf{x})$ is represented as a linear combination of the observed inputs:

$$\hat{y}_j(\mathbf{x}) = \hat{w}_{1j}x_1 + \hat{w}_{2j}x_2 + \dots + \hat{w}_{nj}x_n \quad (j = 1, 2, \dots, p). \quad (6.71)$$

Defining $\hat{\mathbf{w}}_i = [\hat{w}_{i1}, \dots, \hat{w}_{ip}]^T$, $i = 1, \dots, n$, $\hat{\mathbf{w}} = [\hat{\mathbf{w}}_1^T, \dots, \hat{\mathbf{w}}_n^T]^T \in \Re^{pn}$ is an estimate of the system parameter vector \mathbf{w} . By using B-splines for the membership functions $A_i^{j_i}$ and algebraic operators, the real output of the fuzzy system (6.69) is from (4.40)

$$\hat{y}_{i_y}(\mathbf{x}) = \sum_{j=1}^p N_j(\mathbf{x})\hat{y}_j(\mathbf{x}), \quad (6.72)$$

where $N_j(\mathbf{x})$ is given as in (4.40). Substitute (6.71) into (6.72), yielding

$$y_{i_y}(t) = \sum_{k=1}^n \theta_k(\mathbf{x}(t))x_k(t) + \xi_{i_y}(t) \quad (6.73)$$

where

$$\begin{aligned} \theta_k(\mathbf{x}(t)) &= \hat{\mathbf{w}}_k^T \mathbf{N}(\mathbf{x}(t)) \\ &= \sum_{j=1}^p \hat{w}_{jk} N_j(\mathbf{x}(t)) \end{aligned} \quad (6.74)$$

The ‘local’ model (6.73) requires for $n=6$ and $m_i = 5$ (for $i = 1, \dots, n$), $np = n \prod_{i=1}^n m_i = 93,750$ free parameters to be derived! Fortunately a significant number are redundant due to the underlying lattice structure used in conventional neurofuzzy modelling or redundant input variable. In practice a model with $n_w \ll p$ significant input variables is adequate, leading to the model

$$y_{i_y}(t) = \sum_{k=1}^{n_w} \theta_k(\mathbf{x}'(t))x_k(t) + \xi_{i_y}(t) \quad (6.75)$$

where

$$\mathbf{x}'(t) = [x_1(t), x_2(t), \dots, x_{n_w}(t)]^T \in \Re^{n_w} \quad (6.76)$$

is the vector of observed inputs selected from n possible variables defined by (6.68). So for the above example with $n=6$, and $m_i = 5$, if $n_w=3$, the total number of free parameters is now $n_w \prod_{i=1}^{n_w} m_i = 375$.

6.6.2 Feedforward Gram–Schmidt OLS procedure for linear systems

Clearly to construct the reduced neurofuzzy model (6.75) requires some form of preprocessing of input variables or selection of input variables are required. In Section 5.7, orthogonal least squares (OLS) was introduced for input selection for linear-in-the-parameters models. The forward OLS procedure can be extended to nonlinear system input variable selection if piecewise locally linear models such as (6.75) are utilised. One obvious disadvantage of using piecewise linear modelling is that the global model may exhibit poor approximation through the derived input space partitioning inducing output discontinuities through local interpolation. An obvious approach to overcome this problem is to use a MEN (see Section 6.5) model with fuzzy gating functions (such as (6.52)) to ensure piecewise smooth interpolation. In the following we develop the forward OLS procedure based on the modified Gram–Schmidt algorithm of Section 5.7 for input variable selection for piecewise locally linear fuzzy models defined on appropriately derived subregions of the input space. In this approach [110], the most significant input node is selected in a forward regression manner so as to maximise the increment of explained output variance, or the reduction of the model residual variance (see (5.41)), subject to the existence of previously selected input nodes. The selected input node is then used to fit locally linear models for further system approximation. The obtained model residual is then used at the next iteration for input node selection. In using the forward OLS structure detection of Section 5.7 based on the Gram–Schmidt algorithm, the required regressor matrix for model (6.73) with an additional bias w_0 , is

$$\mathbf{A} = \begin{bmatrix} 1 & x_1(1) & \dots & x_n(1) \\ 1 & x_1(2) & \dots & x_n(2) \\ \dots & \dots & \dots & \dots \\ 1 & x_1(N) & \dots & x_n(N) \end{bmatrix},$$

whose orthogonal decomposition is $\mathbf{A} = \Phi T$, where T is a $(n + 1) \times (n + 1)$ upper triangular matrix, and $\Phi = [\phi_0, \phi_1, \dots, \phi_n]^T$ is an $N \times (n + 1)$ matrix with orthogonal columns that satisfy $\Phi^T \Phi = \text{diag}\{\kappa_0, \kappa_1, \dots, \kappa_n\}$, with $\kappa_k = \phi_k^T \phi_k$, $k = 0, 1, \dots, n$, so that the regression model (6.73) or in matrix form for all its observations ($t = 1, 2, \dots, N$) (see also Section 5.7)

$$\begin{aligned} \mathbf{y} &= \mathbf{Aw} + \mathbf{e} \\ &= (\mathbf{AT}^{-1})(\mathbf{T}\mathbf{w}) + \mathbf{e} = \Phi\Gamma + \mathbf{e} \end{aligned} \tag{6.77}$$

where $\Gamma = \mathbf{T}\mathbf{w} = [\gamma_0, \gamma_1, \dots, \gamma_n]^T$ is an auxiliary vector, for $\mathbf{y} = [y(1), \dots, y(N)]^T$, $\mathbf{e} = [e(1), \dots, e(N)]^T$, and $\mathbf{w} = [w_0, w_1, \dots, w_n]^T$. (Note: the MISO model (6.73) subscript i_y has been omitted here and the associate noise term ξ_{i_y} is replaced with $e(t)$ for simplicity of nomenclature.) Denote $\mathbf{A} = [\mathbf{a}_0, \dots, \mathbf{a}_n]$, The modified Gram–Schmidt orthogonalisation procedure [47] computes T row by row and simultaneously orthogonalizes \mathbf{A} .

Starting from $k = 0$, the columns \mathbf{a}_j of \mathbf{A} , $k + 1 \leq j \leq n$ are made orthogonal to the $(k+1)$ th column at the $(k+1)$ th stage; This operation is repeated for $0 \leq k \leq n - 1$. Specifically, denoting $\mathbf{a}_j^{(-1)} = \mathbf{a}_j$, $0 \leq j \leq n$, then for $k = 0, 1, \dots, n - 1$

$$\begin{aligned}\phi_k &= \mathbf{a}_k^{(k-1)} \\ \alpha_{kj} &= \frac{\phi_k^T \mathbf{a}_j^{(k-1)}}{\phi_k^T \phi_k}, \quad k + 1 \leq j \leq n \\ \mathbf{a}_j^{(k)} &= \mathbf{a}_j^{(k-1)} - \alpha_{kj} \phi_k, \quad k + 1 \leq j \leq n,\end{aligned}\tag{6.78}$$

where α_{kj} 's are components of T . The last stage of the procedure is simply $\phi_n = \mathbf{a}_n^{(n-1)}$. The elements of the auxiliary vector Γ are computed by transforming $\mathbf{y}^{(-1)} = \mathbf{y}$ in a similar way. For $0 \leq k \leq n$

$$\begin{aligned}\gamma_k &= \frac{\phi_k^T \mathbf{y}^{(k-1)}}{\phi_k^T \phi_k} \\ \mathbf{y}^{(k)} &= \mathbf{y}^{(k-1)} - \gamma_k \phi_k.\end{aligned}\tag{6.79}$$

This orthogonalisation scheme can be used to derive a simple and efficient algorithm for selecting subset models. Introducing the definition of $\mathbf{A}^{(k-1)}$ as

$$\begin{aligned}\mathbf{A}^{(k-1)} &= [\phi_0, \phi_1, \dots, \phi_{k-1}, \mathbf{a}_k^{(k-1)}, \dots, \mathbf{a}_n^{(k-1)}] \\ &= [\mathbf{a}_0^{(k-1)} \dots \mathbf{a}_n^{(k-1)}].\end{aligned}\tag{6.80}$$

If some of the columns $\mathbf{a}_k^{(k-1)}, \dots, \mathbf{a}_n^{(k-1)}$ in $\mathbf{A}^{(k-1)}$ have been interchanged, this will still be referred to as $\mathbf{A}^{(k-1)}$ for notational convenience. The $(k+1)$ th stage of the forward regression selection procedure is given below.

Forward Linear Regression Selection Algorithm

1. For $k \leq j \leq p$, compute

$$\begin{aligned}\gamma_k^{(j)} &= \frac{(\mathbf{a}_j^{(k-1)})^T \mathbf{y}^{(k-1)}}{(\mathbf{a}_j^{(k-1)})^T \mathbf{a}_j^{(k-1)}} \\ [ERR]_k^{(j)} &= \frac{(\gamma_k^{(j)})^2 (\mathbf{a}_j^{(k-1)})^T \mathbf{a}_j^{(k-1)}}{\mathbf{y}^T \mathbf{y}}.\end{aligned}\tag{6.81}$$

2. Find

$$[ERR]_k = [ERR]_k^{(j_k)} = \max\{[ERR]_k^{(j)}, \quad k \leq j \leq n\}.\tag{6.82}$$

Then the $(j_k + 1)$ th column of $\mathbf{A}^{(k-1)}$ is interchanged with the $(k+1)$ th column of $\mathbf{A}^{(k-1)}$ and the $(j_k + 1)$ th column of T up to the (k) th row is interchanged with the $(k+1)$ th column of T . This effectively selects the $(j_k + 1)$ th candidates as the $(k+1)$ th regressor in the subset model.

3. Perform the orthogonalisation as indicated in (6.78) to transform $\mathbf{A}^{(k-1)}$ into $\mathbf{A}^{(k)}$ and to derive the $(k+1)$ th row of T . $\mathbf{y}^{(k-1)}$ is then updated into $\mathbf{y}^{(k)}$ in accordance with (6.79).
4. The selection is terminated at the $(n_w + 1)$ th stage when the criterion (5.43) is met and this achieves a subset model containing n_w (excluding the constant term if it has been selected) or $(n_w + 1)$ significant input variables (if the constant term has not been selected).

6.6.3 Input variable selection via the modified Gram–Schmidt OLS for piecewise linear submodels

The above Gram–Schmidt OLS procedure for determining the structure of linear processes is modified to consider nonlinear systems which are modelled as a series of piecewise locally linear models. Geometrically the output vector \mathbf{y} , at step $(k+1)$, is projected onto a set of orthogonal basis vectors $\{\phi_0, \phi_1, \dots, \phi_k\}$. The model residual is decreased by projecting the output vector \mathbf{y} onto a new basis ϕ_k at this step. Interpreting $\mathbf{y}^{(k)}$ as the model residual vector at $(k+1)$ th step, (6.79) can be regarded as a global 1-D linear fitting process, so that the previous step (k th step) model residual vector $\mathbf{y}^{(k-1)}$ is fitted using

$$\mathbf{a}_k^{(k-1)} = [a_k^{(k-1)}(1), a_k^{(k-1)}(2), \dots, a_k^{(k-1)}(N)]^T,$$

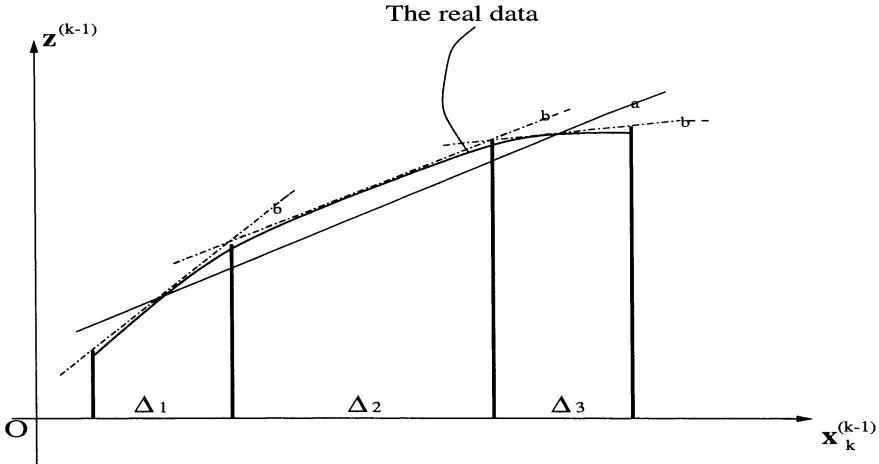
which has been selected using (6.83). The idea here, illustrated in Figure 6.24, is that the model residual $\mathbf{y}^{(k-1)}$ is fitted using the selected variable $\mathbf{a}_k^{(k-1)}$, but as a locally linear model at each regression step. Denote the input domain of the selected variable $a_k^{(k-1)}(t)$ as $\Delta = [\min(a_k^{(k-1)}(t), \forall t), \max(a_k^{(k-1)}(t), \forall t)]$. Divide the input domain Δ into \mathcal{N}_k disjoint but adjacent intervals as $\Delta = [\Delta_1, \Delta_2, \dots, \Delta_{\mathcal{N}_k}]$. For instance, if the obtained input domain for $a_k^{(k-1)}(t)$ is obtained as $\Delta = [1, 5]$, this can then be divided into a predetermined $\mathcal{N}_k = 3$ intervals as $\Delta_1 = [1, 2]; \Delta_2 = (2, 4.5]; \Delta_3 = (4.5, 5]$. It may be necessary that each subrange is excited with sufficiently many data points for accurate parameter estimation. The present approach is advantageous in overcoming the curse of dimensionality because only the most relevant subranges are formed using the selected significant input variables.

Rearrange the position of the components according to the amplitude in $\mathbf{a}_k^{(k-1)}$ to form the vector

$$\tilde{\mathbf{a}}_k^{(k-1)} = [a_k^{(k-1)}(t_1), a_k^{(k-1)}(t_2), \dots, a_k^{(k-1)}(t_N)]^T, \quad (6.83)$$

so that $a_k^{(k-1)}(t_i) \leq a_k^{(k-1)}(t_{i+1})$, where $1 \leq t_i \leq N$, $i = 1, 2, \dots, N$, denote the previous position of $a_k^{(k-1)}(t_i)$, the i th component in $\tilde{\mathbf{a}}_k^{(k-1)}$, in $\mathbf{a}_k^{(k-1)}$. $\tilde{\mathbf{a}}_k^{(k-1)}$ is then divided into \mathcal{N}_k subvectors as

$$\tilde{\mathbf{a}}_k^{(k-1)} = [(a_{(k, \Delta_1)}^{(k-1)})^T, (a_{(k, \Delta_2)}^{(k-1)})^T, \dots, (a_{(k, \Delta_{\mathcal{N}_k})}^{(k-1)})^T]^T, \quad (6.84)$$



a: Global linear fitting

b: Piecewise locally linear fitting

Fig. 6.24. Piecewise locally linear fitting at regression step k ©2001 IEEE

such that $\mathbf{a}_{(k,\Delta_j)}^{(k-1)}$ falls into $\Delta_j, j = 1, 2, \dots, \mathcal{N}_k$, respectively. Denote the data points in each subrange Δ_j as $N^{(\Delta_j)}$, then $\mathbf{a}_{(k,\Delta_j)}^{(k-1)} \in \Re^{N^{(\Delta_j)}}$. Correspondingly, the previous step model residual vector $\mathbf{y}^{(k-1)}$ can be rearranged in accordance with the time index sequence t_1, t_2, \dots, t_N and divided into a set of \mathcal{N}_k subvectors as

$$\begin{aligned}\tilde{\mathbf{y}}^{(k-1)} &= [y^{(k-1)}(t_1), y^{(k-1)}(t_2), \dots, y^{(k-1)}(t_N)]^T \\ &= [(\mathbf{y}_{\Delta_1}^{(k-1)})^T, (\mathbf{y}_{\Delta_2}^{(k-1)})^T, \dots, (\mathbf{y}_{\Delta_{\mathcal{N}_k}}^{(k-1)})^T]^T,\end{aligned}\quad (6.85)$$

where $\mathbf{y}_{\Delta_j}^{(k-1)} \in \Re^{N^{(\Delta_j)}}, j = 1, \dots, \mathcal{N}_k$. (6.83–6.85) effectively enable data samples to be readily utilised for local linear model fitting as the training data are suitably rearranged according to the position in subranges other than actual time series. A piecewise locally linear model can be fitted on each interval respectively using least squares

$$\gamma_{(k,\Delta_j)} = \frac{(\mathbf{a}_{(k,\Delta_j)}^{(k-1)})^T \mathbf{y}_{\Delta_j}^{(k-1)}}{(\mathbf{a}_{(k,\Delta_j)}^{(k-1)})^T \mathbf{a}_{(k,\Delta_j)}^{(k-1)}}, \quad j = 1, 2, \dots, \mathcal{N}_k. \quad (6.86)$$

$\gamma_{(k,\Delta_j)}$'s can be interpreted as the auxiliary coefficients of \mathcal{N}_k local linear models (compared to (6.79) which involves only one auxiliary coefficient γ_k for each stage)

Then $\mathbf{y}_{\Delta_j}^{(k-1)}, j = 1, 2, \dots, \mathcal{N}_k$ is updated based on each subregion as

$$\mathbf{y}_{\Delta_j}^{(k)} = \mathbf{y}_{\Delta_j}^{(k-1)} - \gamma_{(k, \Delta_j)} \mathbf{a}_{(k, \Delta_j)}^{(k-1)}, \quad j = 1, 2, \dots, \mathcal{N}_k. \quad (6.87)$$

\mathcal{N}_k subvectors $\mathbf{y}_{\Delta_j}^{(k)}$, $j = 1, 2, \dots, \mathcal{N}_k$ are then merged and position rearranged according to $t = 1, 2, \dots, N$ to form a new model residual vector $\mathbf{y}^{(k)}$ for next step variable selection. Assuming that the decrement of residual vector, denoted as $(\mathbf{y}^{(k-1)} - \mathbf{y}^{(k)})$, is uncorrelated with the model residual vector $\mathbf{y}^{(k)}$ due to the orthogonality between the basis vectors ϕ_i and ϕ_j for $i \neq j$, the contribution of the selected variable in model residual reduction, $[ERR]_k$, is updated accordingly as

$$[ERR]_k = \frac{[\mathbf{y}^{(k-1)} - \mathbf{y}^{(k)}]^T [\mathbf{y}^{(k-1)} - \mathbf{y}^{(k)}]}{\mathbf{y}^T \mathbf{y}}, \quad (6.88)$$

as initially $[ERR]_k$ was computed based on global linear modelling, but not piecewise local modelling. The subranges may need to be adjusted so as to achieve a bigger value of $[ERR]_k$. It is noted that global linear model can be regarded as a special case of piecewise local linear model, if at some stage no improvement, compared to global linear fitting, can be achieved through piecewise linear fitting, then the input domain can remain undivided for the stage.

This piecewise locally linear fitting procedure can be incorporated into the forward modified Gram-Schmidt OLS algorithm used in linear system modelling described previously, but now for a nonlinear system.

Forward Regression Selection Algorithm with Piecewise Locally Linear Models

- At the $(k + 1)$ th stage of the forward global linear regression selection, for $k \leq j \leq M$, compute

$$\begin{aligned} \gamma_k^{(j)} &= \frac{(\mathbf{a}_j^{(k-1)})^T \mathbf{y}^{(k-1)}}{(\mathbf{a}_j^{(k-1)})^T \mathbf{a}_j^{(k-1)}} \\ [ERR]_k^{(j)} &= \frac{(\gamma_k^{(j)})^2 (\mathbf{a}_j^{(k-1)})^T \mathbf{a}_j^{(k-1)}}{\mathbf{y}^T \mathbf{y}}. \end{aligned} \quad (6.89)$$

- Find

$$[ERR]_k = [ERR]_k^{(j_k)} = \max\{[ERR]_k^{(j)}, \quad k \leq j \leq n\}.$$

Then the j_k th column of $\mathbf{A}^{(k-1)}$ is interchanged with the k th column of $\mathbf{A}^{(k-1)}$ and the j_k th column of T up to the $(k-1)$ th row is interchanged with the k th column of T . This effectively selects the j_k th candidates as the k th regressor in the subset model.

3. Perform the orthogonalisation as indicated in (6.78) to transform $\mathbf{A}^{(k-1)}$ into $\mathbf{A}^{(k)}$ and to derive the k th row of T . The update of model residual vector $\mathbf{y}^{(k-1)}$ into $\mathbf{y}^{(k)}$ is performed using piecewise locally linear modelling illustrated by (6.83)–(6.88). Note that $\mathbf{y}^{(k)}$ is formed by merging the local model residuals, \mathcal{N}_k subvectors $\mathbf{y}_{\Delta_j}^{(k)}$'s, $j = 1, 2, \dots, \mathcal{N}_k$, in accordance with the order of time index t .
4. The selection is terminated when the criterion (5.43) is met to obtain an input variable number n_w . Or the model selection can be terminated by a predetermined number of variables n_w through a few iterations. Note that at each step $[ERR]_k$ was computed twice in step 2, and step 3 respectively. Initially $[ERR]_k$ was computed based on global linear modelling followed by the local linear fitting which will further increase $[ERR]_k$, and the update of $[ERR]_k$ is computed using (6.88) in step 3.
5. The neurofuzzy operating dependent model (6.75) is then formed using the selected input variables. The network can be trained using a normalised least mean squares (NLMS) algorithm (see Chapter 3).

Example 6.4: Consider the illustrative MIMO nonlinear system

$$\begin{aligned} y_1(t) &= 0.4y_1(t-1) - 0.1y_2(t-1)^3 + \xi_1(t) \\ y_2(t) &= 0.5y_1(t-1) + u(t-1) + \xi_2(t) \\ y_3(t) &= \text{rand}(0, 1) \\ y_4(t) &= \sin(y_2(t-1)), \end{aligned} \tag{6.90}$$

where $\text{rand}(0, 1)$ is a random number uniformly distributed over $(0, 1)$. The system input is given as $u(t) = \sin(\pi t/25)$, and the measurement noise terms are Gaussian distributed with $\xi_1(t) \sim N(0, 0.01^2)$, $\xi_2(t) \sim N(0, 0.02^2)$. The output observational data sequences are shown in Figure 6.25, from which a 1000 data points split into two equal data sets and are used for estimation and validation. Given $n_y = 3$, $n_u = 2$, the full input vector for subsystem $y_1(t)$ was

$$\begin{aligned} \mathbf{x}(t) &= [\text{constant}, y_1(t-1), y_1(t-2), y_1(t-3), y_2(t-1), y_2(t-2), \\ &\quad y_2(t-3), y_3(t-1), y_3(t-2), y_3(t-3), y_4(t-1), \\ &\quad y_4(t-2), y_4(t-3), u(t-1), u(t-2)]^T. \end{aligned}$$

The input variable selection procedure described above was applied over the estimation set. Note that the input region setting for piecewise locally linear modelling is predetermined, here at each step, the selected variable was divided into three regions each containing [25%, 50%, 25%] of all the data points respectively, by assuming three local linear regions with a larger region around the data distribution center, and two smaller regions at the data distribution edges. Two input variables were selected by the proposed algorithm as $\mathbf{x}(t) = [y_1(t-1), u(t-1)]^T$ with $[ERR]_k$'s of 0.942 and 0.016

respectively. This indicates that 94.2% of system output variance can be explained by the first term $y_1(t - 1)$ and 95.8% of system output variance can be explained by a combination of two terms, $y_1(t - 1)$ and $u(t - 1)$. (Note that the input $u(t - 1)$ was selected instead of $y_2(t - 1)$ because this is the causal input of $y_2(t)$.) An order-2 B-spline neurofuzzy network was consequently used to approximate the system. The knot vectors for the input domain were defined as

$$\{-0.3, -0.25, -0.15, 0, 0.15, 0.25, 0.3\}$$

for $y_1(t)$ and

$$\{-1.2, -1.1, -0.8, 0, 0.8, 1.1, 1.2\}$$

for system input $u(t)$, resulting in 5 univariate basis functions for $y_1(t)$ and $u(t)$ respectively. The 25 multi-dimensional basis functions are formed by the tensor product of univariate basis functions. The neurofuzzy network of (6.75) was formed and trained over the estimation data set for 15 epochs with a learning rate of 0.15 for the NLMS algorithm.

For the subsystem $y_2(t)$, given $n_y = 3$, $n_u = 2$, the full input vector using all the subsystem's variables was

$$\begin{aligned} \mathbf{x}(t) = & [constant, y_1(t - 1), y_1(t - 2), y_1(t - 3), y_2(t - 1), \\ & y_2(t - 2), y_2(t - 3), y_3(t - 1), y_3(t - 2), y_3(t - 3), \\ & y_4(t - 1), y_4(t - 2), y_4(t - 3), u(t - 1), u(t - 2)].^T \end{aligned}$$

The input variable selection procedure described above was applied over the estimation set. At each step, the selected variable was divided into three regions containing [25%, 50%, 25%] of all the data points respectively. Two input variables were selected as $\mathbf{x}(t) = [u(t - 1), y_1(t - 1)]^T$ with $[ERR]_k$'s of 0.9985 and 0.0005 respectively. This indicates that 99.85% of system output variance can be explained by the first term $u(t - 1)$, and 99.9% of system output variance can be explained by a combination of two terms, $u(t - 1)$ and $y_1(t - 1)$.

An order-2 B-spline neurofuzzy network was used to approximate the system. The knot vectors for the input domain were defined as

$$\{-1.2, -1.1, -0.8, 0, 0.8, 1.1, 1.2\}$$

for system input $u(t)$ and

$$\{-0.3, -0.25, -0.15, 0, 0.15, 0.25, 0.3\}$$

for $y_1(t)$, resulting in five 1-D basis functions for $y_1(t)$ and $u(t)$ respectively. The 25 multi-dimensional basis functions were formed using a direct multiplication of 1-D basis function. The neurofuzzy network of (6.75) was formed and trained over the estimation data set for 15 epochs with a learning rate of 0.15 using the NLMS algorithm.

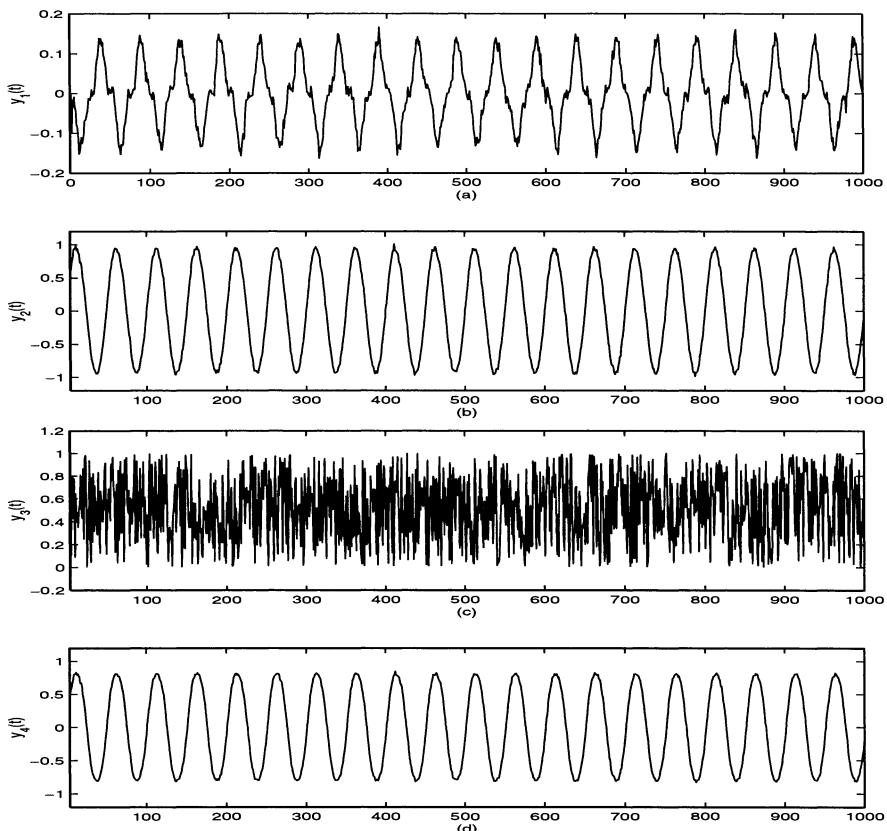


Fig. 6.25. The output data $y_i(t)$'s in Example 6.4; (a) $y_1(t)$; (b) $y_2(t)$; (c) $y_3(t)$ and (d) $y_4(t)$ ©2001 IEEE

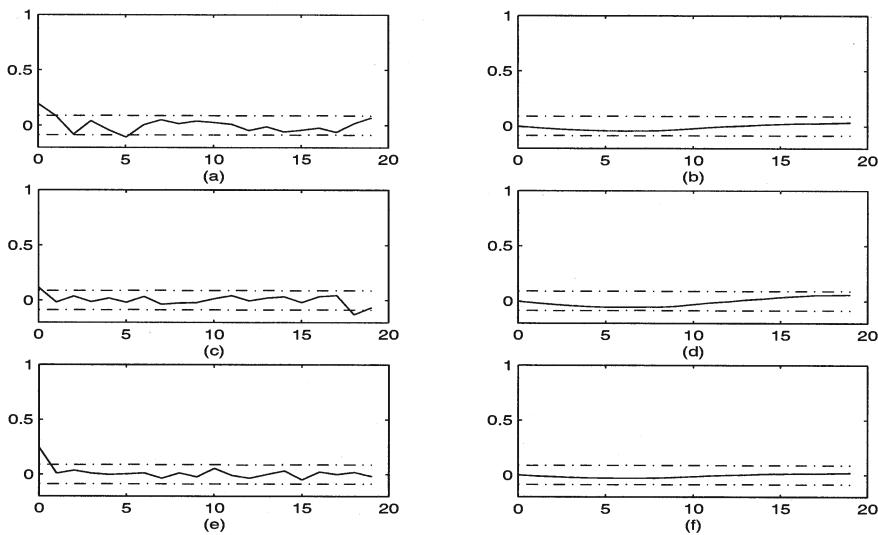


Fig. 6.26. Model validity tests in Example 6.4; (a) global model validity test – 1; (b) global model validity test – 2; (c) model validity test $\rho_{(y_1 \xi_1)} \xi_1^2(\tau)$ (first subsystem); (d) model validity test $\rho_{(y_1 \xi_1) u^2}(\tau)$ (first subsystem); (e) model validity test $\rho_{(y_2 \xi_2)} \xi_2^2(\tau)$ (second subsystem); (f) model validity test $\rho_{(y_2 \xi_2) u^2}(\tau)$ (second subsystem) ©2001 IEEE

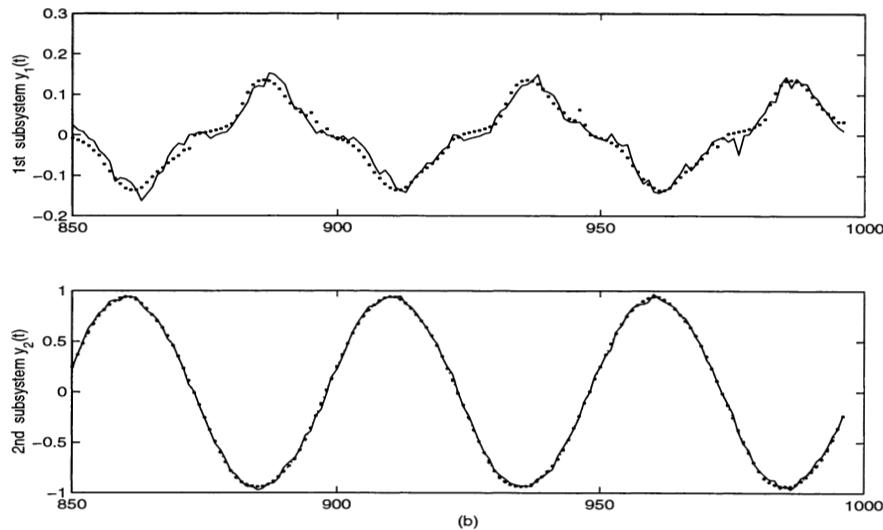


Fig. 6.27. Five-step ahead predictions over the validation data set in Example 6.4 (solid line: measurements and dotted line: 5-step ahead model predictions); (a) first subsystem $y_1(t)$; (b) second subsystem $y_2(t)$ ©2001 IEEE

Based on the assumption that if the model structure and the estimated parameters are correct then the residual sequence $\xi_{i_y}(t)$ should be unpredictable from all linear and nonlinear combinations of past inputs $u(t)$ and outputs $y(t)$, model validity tests (see [19, 20], and the references within) can be applied to detect the inadequacy of a fitted model. Correlation based validation involves computing correlation functions which are composed of model residuals and system inputs and testing if these satisfy certain conditions given in the form of confidence intervals. The system output can be included to form the new higher order correlation tests [19]. The global MIMO nonlinear model validity tests [20] was developed by using a sequence of the mean square of subsystem residuals. The global MIMO nonlinear model validity tests and model validity tests [20] for each subsystem [19] are used and the results, plotted in Figure 6.26, show that both subsystems are appropriate. The model was then applied to ℓ -step ahead prediction, where the past system output term $y_1(t + \ell - 1)$, were iteratively replaced by model predictions $\hat{y}_1(t + \ell - 1|t - 1)$, and system input $u(t + \ell - 1)$ is assumed known. The MSE of (1–6) step ahead prediction over the validation data set for the subsystem $y_1(t)$ were $(0.013^2, 0.015^2, 0.016^2, 0.019^2, 0.024^2, 0.024^2)$ respectively. The mean square error of (1–6) step ahead predictions over the validation data set for the subsystem $y_2(t)$ were $(0.021^2, 0.022^2, 0.023^2, 0.023^2, 0.023^2, 0.023^2)$ respectively. The five-step ahead predictions are plotted in Figure 6.27, demonstrating that the derived models using the selected variables are appropriate.

7. Delaunay input space partitioning modelling

7.1 Introduction

We have already demonstrated in previous chapters that data-based modelling is a complex and demanding process especially if there is little prior knowledge about the underlying process. In practice there is usually some process structural knowledge that can be exploited in model construction, additionally specific model structures lend themselves to subsequent uses such as a control design and estimation (see Chapter 8). For many processes the derived models are only valid across a limited domain (for empirical data modelling this is usually determined by the data gathering process), and are often implicitly represented in the model. In this regard the concept of local models is a promising technique since they combine conventional system theory approaches (especially for locally linear models – see Chapter 6) with adaptive learning algorithms. In the local model approach a system $y = f(\mathbf{x})$, where y denotes system output, \mathbf{x} a system input vector of regressors, and $f(\cdot)$ the hypothesised model, is approximated by

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^K g_i(\mathbf{x}) \hat{f}_i(\mathbf{x}), \quad (7.1)$$

where $g_i(\mathbf{x})$ is a validity or gating function (in the MEN context of Chapter 6) that qualifies the application of the determined local models $\hat{f}_i(\mathbf{x})$ (frequently selected as linear), and provides smooth or switched interpolation between the local models. Note that (7.1) is similar to the concept of gain scheduling in control whereby the controller action varies as \mathbf{x} migrates across the operating envelope of the state space. So for a given input space $\mathcal{X} : \mathbf{x} \in \mathcal{X} \in \Re^n$, a local model network consists of a set of K local models $\hat{f}_i(\mathbf{x})$, which exists on a set of K operating regions such that

$$\mathcal{X}^{(1)} \cup \mathcal{X}^{(2)} \cup \dots \cup \mathcal{X}^{(K)} = \mathcal{X} \in \Re^n, \quad (7.2)$$

and

$$\mathcal{X}^{(j)} \cap \mathcal{X}^{(k)} = \emptyset \quad \text{when } j \neq k, \quad (7.3)$$

where \emptyset denotes an empty set. If the local models are linear, then

$$\hat{f}_i(\mathbf{x}) = [1 \quad \mathbf{x}^T] \hat{\mathbf{w}}^{(i)}, \\ \text{IF } \mathbf{x} \in \mathcal{X}^{(i)}, \quad i = 1, \dots, K, \quad (7.4)$$

where $\hat{\mathbf{w}}^{(i)} \in \Re^{n+1}$ denotes the estimated parameter vector used in the i th local linear model. In practice, to avoid discontinuous behaviours associated with crisp sets, noncrisp or fuzzy membership functions are used to ensure smooth interpolation between operating regions. A variety of researchers have concentrated on local linear modelling schemes as they enable standard linear control and estimation schemes to be used (e.g. pole placement to H_∞ for control and Kalman filters for state estimation – Chapter 8), see for example [68, 151]. The MEN algorithms of Chapter 6 exploit this input space decomposition. In Chapter 5 we introduced data-based modelling algorithms that generated global *additive* (linear and nonlinear) neurofuzzy submodels to overcome the curse of dimensionality whilst retaining network transparency. Basis function networks form the building blocks of both local modelling approaches and global additive models (such as ASMOD); generally in local linear modelling networks [151], the basis functions are used to interpolate between the individual models, whereas in additive models the sub-networks are typically selected as basis function networks. Clearly they can be joined together into a single approach which uses both local linearisation and basis function networks to form data generated models – this is the approach developed in Chapter 8. However in this chapter we consider nonorthogonal decomposition of the input space based upon simplices or triangulations instead, to produce parsimonious and smooth piecewise linear models which are directly applicable to linear control or estimation schemes. Significantly this new data-based method automatically decomposes the input space into a minimum set of locally linear models.

7.2 Delaunay triangulation of the input space

The majority of local model decomposition approaches use lattice grid structuring of the input space, either orthogonal splits as in fuzzy logic, or quad-trees, or $k - d$ trees (see Section 6.2), however when the grid structure is dropped and arbitrary positions of the interpolation models are utilised (see Figure 6.1(a)) based on local information such as local data density/variance, multi-dimensional mappings can be realised without the concomitant growth of memory, weights, rules, or data which is inherent in lattice based structures. This increased modelling flexibility comes at a price since it is accompanied by loss of topological information and transparency that is implicit in the grid/lattice input space decomposition, i.e. using a set of distributed data points as the position of nodes for the input space does not imply any topological information. Additional neighbourhood relationships have to be explicitly defined. One obvious way of achieving this is to connect $(n + 1)$

neighbouring nodes by lines generating a n -dimensional *simplex* or Delaunay triangular decomposition of the input space [33]. Any simplices used to define a network structure must satisfy conditions (7.2) and (7.3), in which the condition (7.3) ensures that an input vector \mathbf{x} always activates the same subset of nodes, otherwise if there were intersecting simplices the selection of the active nodes would be ambiguous. The condition (7.2) is established to ensure that the set of all simplices cover the input space avoiding extrapolation. Delaunay triangles (a triangle in 2-D, a tetrahedron in 3-D, ...) have been effectively used in computational geometry [138] to decompose the n -dimensional input space as a set of simplices satisfying (7.2) and (7.3). They can be formally defined by:

Definition 7.1 A simplex $\mathcal{X}^{(i)}$ consisting of $(n + 1)$ nodes located at vertices (positions) $\mathbf{b}_k^{(i)}$, $k = 0, 1, \dots, n$ in \Re^n is a Delaunay simplex if and only if the embedding n -dimensional hypersphere does not contain any other node.

That is, the operating regions $\mathcal{X}^{(i)}$ of the input space are formed as $\mathcal{X}^{(i)} = \mathcal{H}\{\mathbf{b}_0^{(i)}, \mathbf{b}_1^{(i)}, \dots, \mathbf{b}_n^{(i)}\}$, where \mathcal{H} denotes convex hull, $\{\mathbf{b}_0^{(i)}, \mathbf{b}_1^{(i)}, \dots, \mathbf{b}_n^{(i)}\}$ denote the vertices of $\{\mathcal{X}^{(i)}\}$, by allowing some points freely within or on the hull of the input space to form these vertices. Each Delaunay triangulation connects $n + 1$ vertices in the n -dimensional space, as shown in Figure 7.1 for $n = 2$.

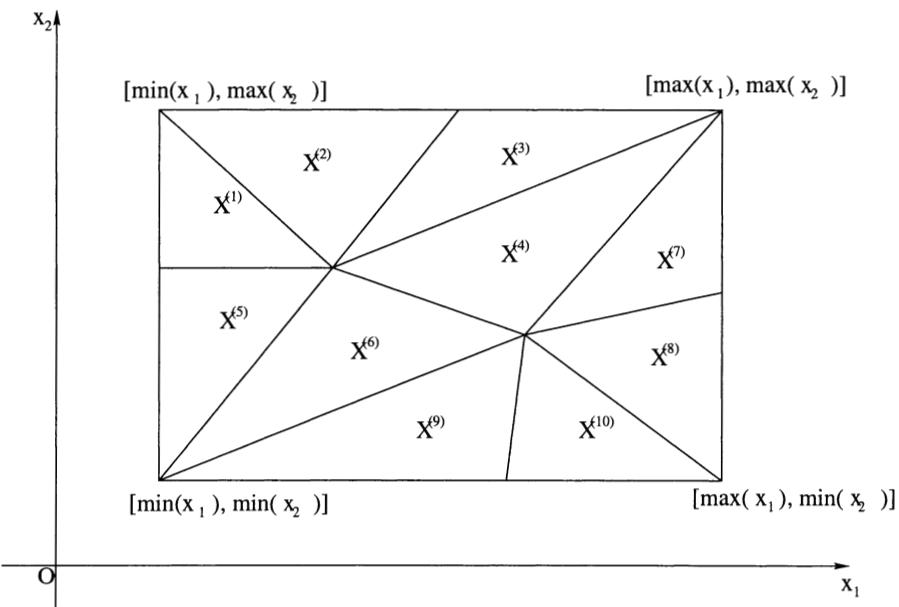


Fig. 7.1. Delaunay partition of the input space

The major benefit of using Delaunay simplices is that they are non-intersecting, further it can be shown that if linear interpolation is used, the worst case approximation error using Delaunay triangulation is least compared with any other triangulation of the data set [156]. The problem of selecting the nodes for a given input \mathbf{x} is equivalent to finding the simplex whose convex hull contains \mathbf{x} . This latter problem is resolved by means of the concept of *barycentric coordinates* of \mathbf{x} relative to the simplices $\{\mathcal{X}^{(i)}\}$.

Definition 7.2: Barycentric coordinates

The barycentric coordinates $\mathbf{B}^{(i)}(\mathbf{x}) = [B_0^{(i)}, \dots, B_n^{(i)}]^T$ of a vector \mathbf{x} in relation to a simplex $\mathcal{X}^{(i)}$ are the $(n+1)$ weights which move the $\mathcal{X}^{(i)}$ centre of gravity to the point \mathbf{x} , i.e.

$$\mathbf{x} = \sum_{k=0}^n B_k^{(i)} \mathbf{b}_k^{(i)}, \quad (7.5)$$

with $B_k^{(i)} \geq 0$, $\sum_{k=0}^n B_k^{(i)} = 1$.

Equation (7.5) contains $(n+1)$ unknowns, the above normalising constraint on the barycentric coordinates ensures that a linear set of equations exist for the solution of the barycentric coordinates. Clearly the computed barycentric coordinates can be used as a validity function in (7.4) by checking if the solution satisfies these constraints or not (see Section 7.3).

Delaunay triangular partitioning of the input space is appealing in obtaining a parsimonious and smooth piecewise linear model, since it avoids the use of redundant nodes caused by the orthogonal splits inherent in a lattice based input space structure. Also piecewise local linear models based on Delaunay partition can overcome the discontinuity in conventional lattice partitioned, locally piecewise linear models (as illustrated in Figure 7.2). Because a neurofuzzy network can also be interpreted as a locally piecewise polynomial model which inherently has a stronger approximation capability than local piecewise linear models, it is highly desirable that neurofuzzy networks based on Delaunay partitioning of the input space can be developed, so that all the advantages associated with the neurofuzzy network can be extended to the case of Delaunay input space partitioning.

7.3 Delaunay input space partitioning for locally linear models

By utilising Definition 7.2, a decision rule can be derived [101] for the gate control in training each local model, if the locally linear model is formed as a mixture of experts network (MEN) (see Section 6.5). From Figure 6.15 of the MEN, the manager monitors and evaluates all the reports to form a

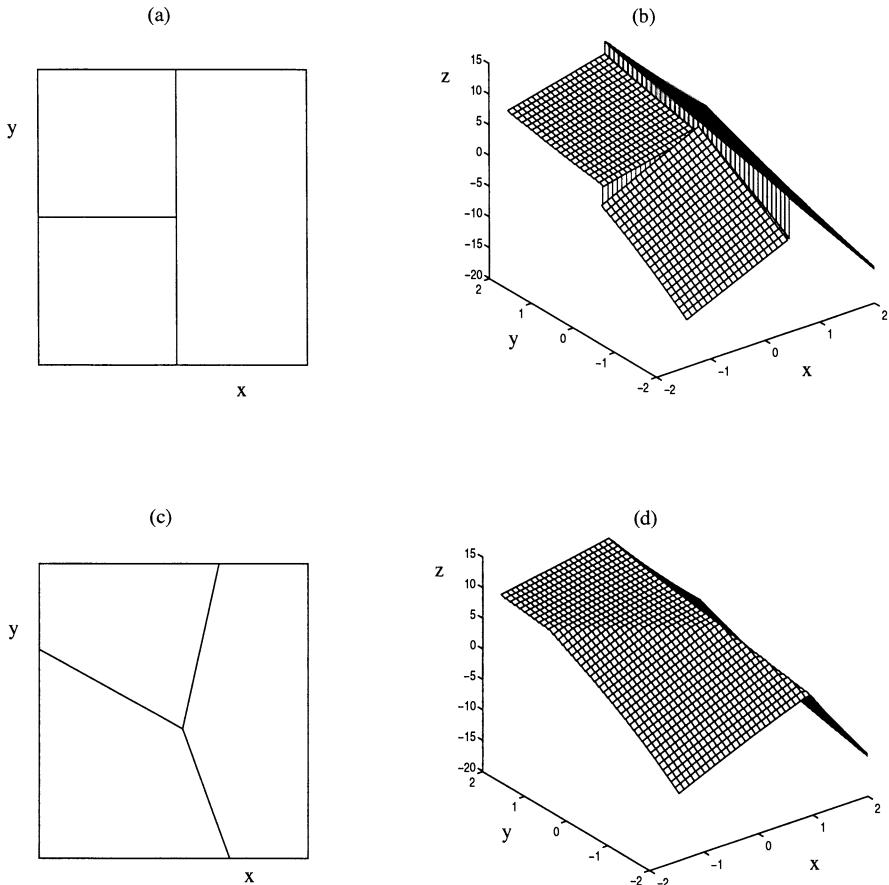


Fig. 7.2. Input space partitions and the corresponding piecewise linear surfaces in modelling $z = f(x, y)$; (a) orthogonal/lattice decomposition; (b) resultant locally piecewise linear models of (a); (c) Delaunay partitioning; (d) resultant locally piecewise linear models of (c) ©2000 IEEE

final opinion of the system output and how the network is to be trained, by operating the gate g_i of each expert using some decision rules. In the piecewise linear modelling problem, the experts correspond to a set of local linear models. The decision rule is based on the solutions of the barycentric coordinates of \mathbf{x} with respect to Delaunay triangles $\mathcal{X}^{(i)}$, as shown in Figure 7.1, that form the whole input space.

From (7.5), denote $A^{(i)} = [B_1^{(i)}, \dots, B_n^{(i)}]^T \in \Re^n$ and substitute $B_0^{(i)} = 1 - \sum_{k=1}^n B_k^{(i)}$ into (7.5), yields

$$\Lambda^{(i)} = \left[\tilde{V}^{(i)} \right]^{-1} (\mathbf{x} - \mathbf{b}_0^{(i)}), \quad (7.6)$$

where $\tilde{V}^{(i)} = [\mathbf{b}_1^{(i)} - \mathbf{b}_0^{(i)}, \dots, \mathbf{b}_n^{(i)} - \mathbf{b}_0^{(i)}] \in \Re^{n \times n}$.

The IF statement in (7.4) can be represented through K experts opinions given by (7.6). The manager chooses the gate g_i by using the decision rule that checks the solutions given by K set of $\Lambda^{(i)}$, $i = 1, \dots, K$ and selects the only one that satisfies the constraints $B_k^{(i)} \in [0, 1]$ for $1 \leq k \leq n$ and $\sum_{j=1}^n B_j^{(i)} = 1$, and then let $g_i = 1$.

The training of local linear models can be carried out adaptively in an on-line manner by using the mixture of experts network and is summarised as follows [101].

Delaunay Input Space Partitioned for Locally Linear Modelling Algorithm

1. Determine the input space domain (the convex set \mathcal{X}) constrained within the maximum and minimum value of the components of the input vector \mathbf{x} by using data preprocessing. Given an appropriate number of points which are located within or on the hull of the input space to form vertices of a set of K subranges, denoted as $\mathcal{X}^{(i)} = \mathcal{X}^{(i)}\{\mathbf{b}_0^{(i)}, \mathbf{b}_1^{(i)}, \dots, \mathbf{b}_n^{(i)}\}$. Corresponding to each $\mathcal{X}^{(i)}$, given a set of initial parameters for K local linear models $\hat{\mathbf{w}}^{(i)} \in \Re^{n+1}$, $i = 1, 2, \dots, K$.
2. At time step t , the training sample is given as $\{\mathbf{x}(t), y(t)\}$. Each expert opinion is formed using the solution of (7.5) (where \mathbf{x} is replaced by $\mathbf{x}(t)$) and reports to the manager. The manager selects the i th model that satisfies the constraints $B_k^{(i)} \in [0, 1]$ for $1 \leq k \leq n$ and $\sum_{k=1}^n B_k^{(i)} = 1$, and then lets $g_i = 1$, $g_j = 0$ for $j \neq i$.
3. The output of the network is produced and the network weights are adjusted using the NLMS algorithm (see Section 3.4) or least squares algorithm.
4. The overall performance of the network can be measured using a mean square error (MSE) as $V = \sum_{t=1}^N e^2(t)$. The network output is a highly complicated nonlinear function of the positions of the vertices which determines how the input space is partitioned. The optimization of V with respect to the vertices position to obtain an optimal piecewise linear model can be realised using the very fast simulated reannealing (VFSR) algorithm [101, 115] or any other nonlinear optimisation algorithm such as genetic algorithms.

Example 7.1a Consider the standard benchmark nonlinear autoregressive (NAR) time series introduced in Section 4.5

$$y(t) = (0.8 - 0.5 \exp(-y^2(t-1)))y(t-1)$$

$$\begin{aligned} & -(0.3 + 0.9 \exp(-y^2(t-1)))y(t-2) \\ & + 0.1 \sin(3.1415926y(t-1)) + e(t), \end{aligned} \quad (7.7)$$

where the noise $e(t)$ was a Gaussian white sequence with mean zero and variance 0.02. One thousand data points were generated and the first 500 points were used as an estimation data set. The remaining data were used as a validation data set. The input vector is set as $\mathbf{x} = [y(t-1), y(t-2)]^T$. The input space partition is shown in Figure 7.3, where the rectangle ABCD represents the bounds of a slightly enlarged input space \mathcal{X} . Four points E, F, G, H are allowed to move along each side of the rectangular ABCD and point K is a free point within the input space, forming $K = 8$ triangulations $\mathcal{X}^{(i)}$'s that can be determined using a free parameter vector \mathbf{w} of dimension size of six. The VFSR algorithm was applied to determine the position of points E, F, G, H and K, as plotted in Figure 7.3. At identification stage, the local linear models were trained using ordinary least squares to reduce computation time. Eight local linear models of (7.4) were obtained as

$$\begin{aligned} \hat{y}_1(\mathbf{x}(t)) &= -0.0605 + 0.6845y(t-1) - 0.6636y(t-2), \text{ IF } \mathbf{x}(t) \in \mathcal{X}^{(1)} \\ \hat{y}_2(\mathbf{x}(t)) &= -0.4503 + 1.1478y(t-1) - 0.7507y(t-2), \text{ IF } \mathbf{x}(t) \in \mathcal{X}^{(2)} \\ \hat{y}_3(\mathbf{x}(t)) &= -0.4463 + 1.3563y(t-1) - 0.9220y(t-2), \text{ IF } \mathbf{x}(t) \in \mathcal{X}^{(3)} \\ \hat{y}_4(\mathbf{x}(t)) &= -0.1022 + 0.9262y(t-1) - 1.1197y(t-2), \text{ IF } \mathbf{x}(t) \in \mathcal{X}^{(4)} \end{aligned}$$

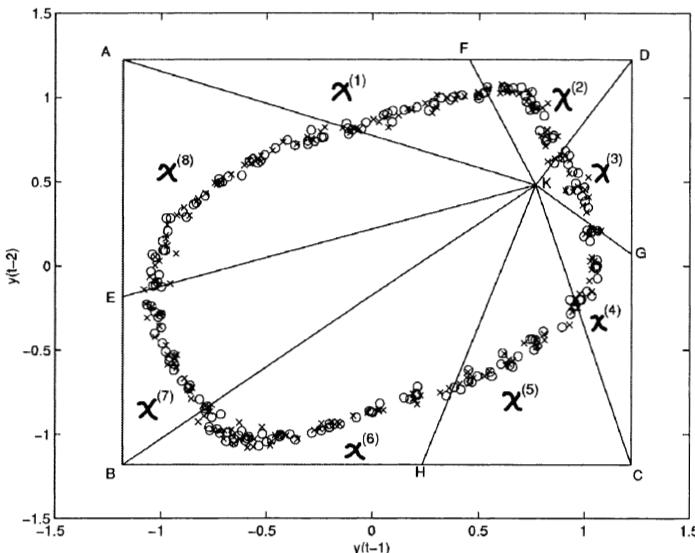


Fig. 7.3. Piecewise locally linear modelling for Example 7.1a (circles: system measurements and crosses: one-step ahead predictions)

$$\begin{aligned}
\hat{y}_5(\mathbf{x}(t)) &= -0.1336 + 0.2907y(t-1) - 1.0535y(t-2), \text{ IF } \mathbf{x}(t) \in \mathcal{X}^{(5)} \\
\hat{y}_6(\mathbf{x}(t)) &= 0.2329 + 0.8593y(t-1) - 0.6843y(t-2), \text{ IF } \mathbf{x}(t) \in \mathcal{X}^{(6)} \\
\hat{y}_7(\mathbf{x}(t)) &= 0.2200 + 1.2104y(t-1) - 1.0669y(t-2), \text{ IF } \mathbf{x}(t) \in \mathcal{X}^{(7)} \\
\hat{y}_8(\mathbf{x}(t)) &= 0.1853 + 0.2381y(t-1) - 1.0057y(t-2), \text{ IF } \mathbf{x}(t) \in \mathcal{X}^{(8)}.
\end{aligned} \tag{7.8}$$

The results of one-step ahead prediction using these local linear models are plotted in Figure 7.3 demonstrating the good predictive performance of the model. The model validity test and the one-step ahead prediction over the validation data set $t = 800 \sim 900$ are plotted in Figure 7.4 demonstrating the model (7.8) is appropriate. The MSE of the validation data set $t = 501 \sim 1000$ is $(0.023)^2$.

Clearly the model (7.8) is directly amenable to local linear controller design for some input $u(t)$ based upon the model appropriate to the region $\mathcal{X}^{(i)}$ of current occupation, global control is then produced by using the same validity or gating function as in (7.1) for modelling but now for controller integration. Overall closed loop stability can be then determined by the method of linear matrix inequalities [68]. Also the Takagi–Sugeno (T–S) fuzzy model in the form of rules follows directly from (7.8).

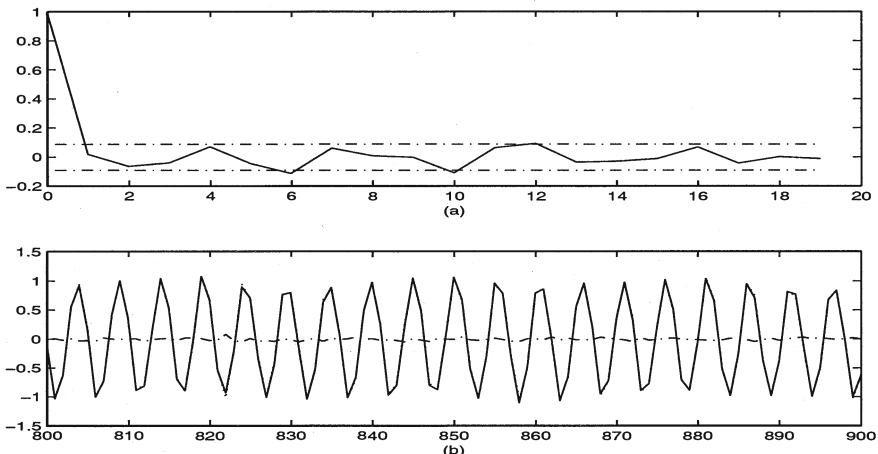


Fig. 7.4. Model validation for Example 7.1a; (a) model validity test $\rho_{ee}(\tau)$ and (b) one-step ahead prediction over the validation data set (*solid line*: system measurements, *dotted line*: one-step ahead prediction and *dash-dot line*: model residual)

7.4 The Bézier–Bernstein modelling network

Fundamental to neurofuzzy network modelling is the idea of building a bridge between fuzzy logic and quantitative adaptive numeric/data processing via a concept of fuzzification by using a fuzzy membership function. The normalising constraint (the partition of unity) on the barycentric coordinates suggests their applicability as fuzzy membership functions. The concept of barycentric coordinates and the de Casteljau algorithm have been widely used in Bézier–Bernstein (BB) curve and surface design in the field of computer aided geometric design [63]. Generalised neurofuzzy network modelling algorithms using Bézier–Bernstein polynomial functions based on an additive decomposition approach have been developed by the authors [105]. A neurofuzzy network based on the expansion of Bézier–Bernstein polynomial function will maintain most of the properties held by the conventional B-spline expansion, such as non-negativity of the basis functions, the partition of unity, approximation capability, but with the additional advantages of structural parsimony and Delaunay input space partition. The generalised neurofuzzy network modelling algorithms using Bézier–Bernstein polynomial functions are based on an additive decomposition approach via the well known analysis of variance (ANOVA) expansion [24]), as shown in Figure 7.5

$$f(\mathbf{x}) = f_0 + \sum_{i=1}^n f_i(x_i) + \sum_{i_1=1}^n \sum_{j_1=i_1+1}^n f_{i_1 j_1}(x_{i_1}, x_{j_1}) + e(\mathbf{x}), \quad (7.9)$$

where $f(\mathbf{x})$ is a general multivariate nonlinear mapping for input $\mathbf{x} = [x_1, x_2, \dots, x_n]^T \in \Re^n$. $f_i(x_i)$ and $f_{i_1 j_1}(x_{i_1}, x_{j_1})$ are univariate and bivariate nonlinear submodels of $f(\cdot)$. For almost all practical problems the ANOVA expansion (7.9) can be truncated at the bivariate submodel level with minimal truncation error $e(\mathbf{x})$.

The Bézier–Bernstein neurofuzzy network construction algorithm for the modelling of the univariate term $f_i(x_i)$, and the bivariate terms $f_{i_1 j_1}(x_{i_1}, x_{j_1})$ can be interpreted as an inverse procedure of the de Casteljau algorithm used in Bézier–Bernstein *curve* and *surface* design, respectively.

7.4.1 Neurofuzzy modelling using Bézier–Bernstein function for univariate term $f_i(x_i)$ and bivariate term $f_{i_1, j_1}(x_{i_1}, x_{j_1})$

Univariate Bézier–Bernstein polynomials are used as basis functions for univariate term $f_i(x_i)$. Univariate Bernstein polynomial basis $B_j^{(d)}(s)$ are terms in the expansion of $[s + (1 - s)]^d$, defined by

$$B_j^{(d)}(s) = \frac{d!}{j!(d-j)!} s^j (1-s)^{(d-j)}, \quad (7.10)$$

where j and d are non-negative integer numbers, $j \leq d$ over the region $s \in [0, 1]$. The total number of the d th order univariate Bernstein polynomials $B_j^{(d)}(s)$ is $(d + 1)$. A set of different order Bernstein polynomial basis

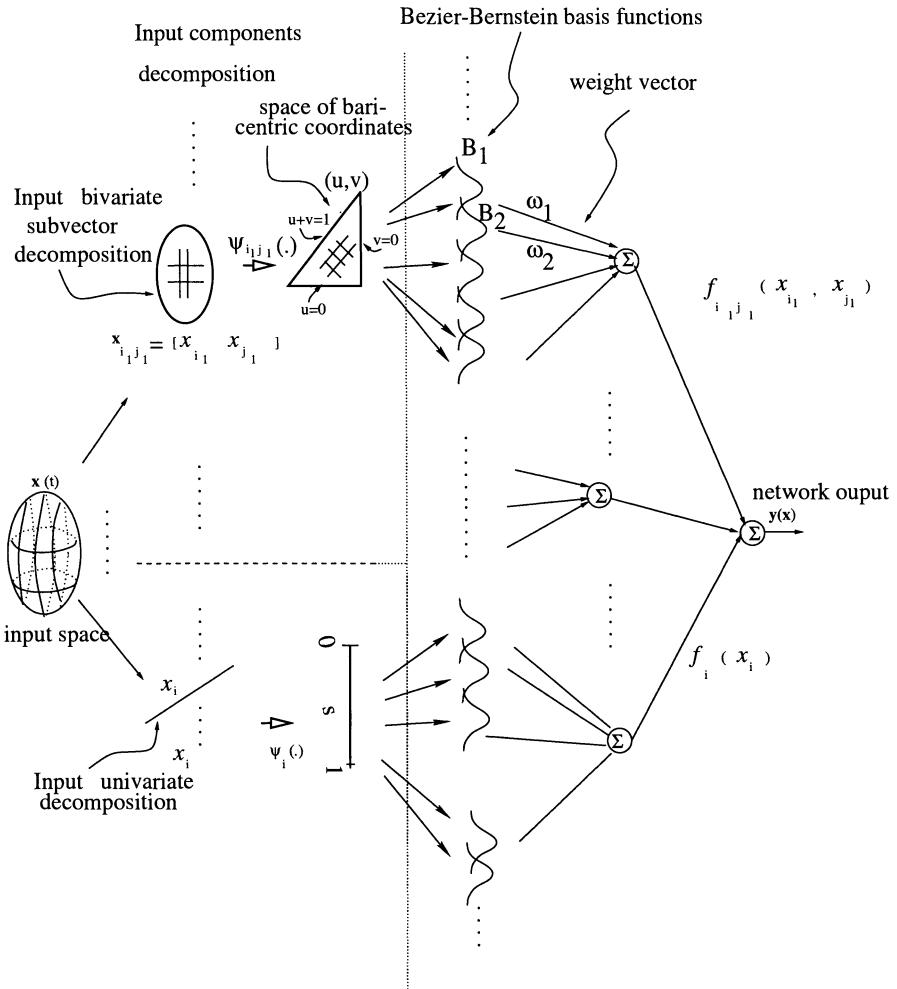


Fig. 7.5. Bézier–Bernstein polynomial function based neurofuzzy network via the additive approach (*up-left part*: modelling of the bivariate terms $f_{i_1j_1}(x_{i_1}, x_{j_1})$, by using barycentric coordinates from bivariate decomposition of input vector; *down-left part*: modelling of the univariate terms $f_i(x_i)$, by using univariate decomposition of input vector; and *right part*: addition of all modelled term based on (7.9))

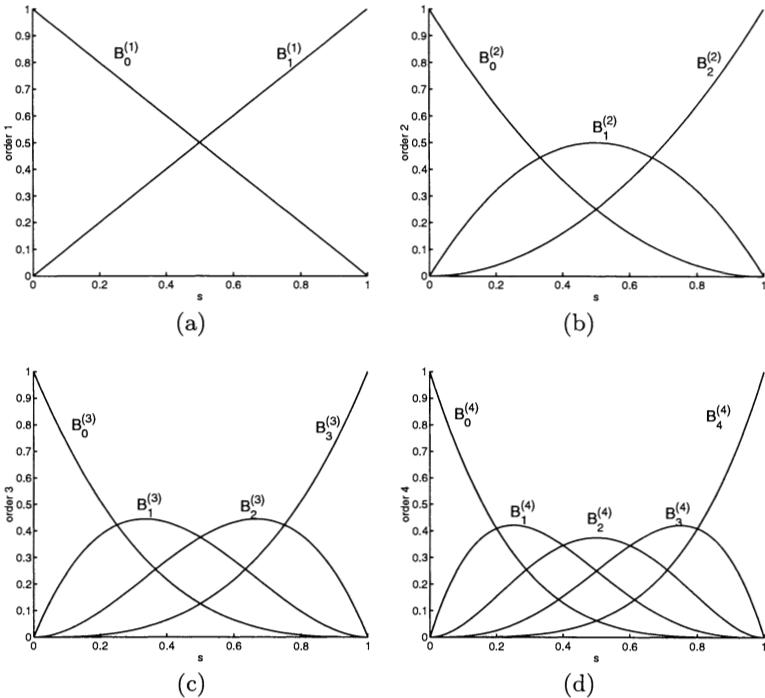


Fig. 7.6. A set of univariate Bernstein polynomials of different orders defined over $s = [0, 1]$; (a) order 1, linear; (b) order 2, quadratic; (c) order 3, cubic; and (d) order 4, quartic

functions are shown in Figure 7.7. Clearly they are very similar to conventional fuzzy sets and B-spline polynomials. Also in a Bézier–Bernstein polynomial function based neurofuzzy network, these polynomials can be used as fuzzy membership functions as they satisfy $B_j^{(d)}(s) \in [0, 1]$ (representing fuzzy membership function) and $\sum_{j=0}^d B_j^{(d)}(s) = 1$ (a partition of unity).

Given knots positions, there exists a one-to-one mapping between a data point (located within the range determined by knots) to s . As an illustration, a set of different order Bernstein polynomial basis functions are shown in Figure 7.6 as functions of data points ranging from 1 to 10, based on known knot positions.

It can be shown [63] that the Bernstein polynomials also satisfy the following simple recursion:

$$B_j^{(d)}(s) = (1 - s)B_j^{(d-1)}(s) + sB_{j-1}^{(d-1)}(s). \quad (7.11)$$

Suppose that the $f_i(x)$ term ((7.9)) is modelled as

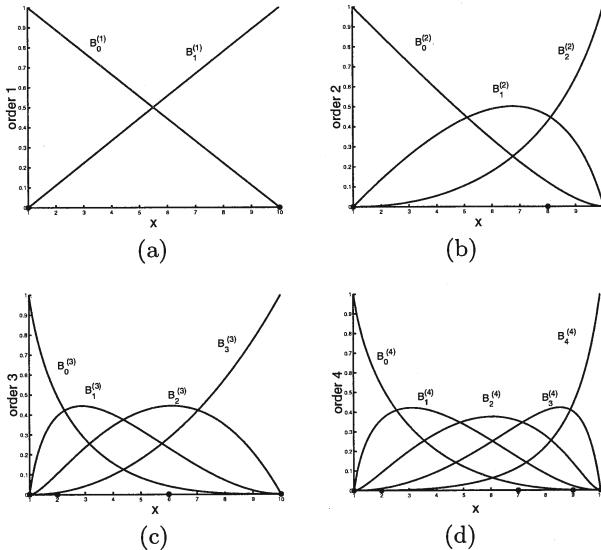


Fig. 7.7. A set of univariate Bernstein polynomials of different orders defined over $x=[1 \ 10]$ (circle denotes knot position); (a) order 1 with knots [1,10], linear; (b) order 2 with knots [1 8 10], quadratic; (c) order 3 with knots [1 2 6 10], cubic; and (d) order 4 with knots [1 2 7 9 10], quartic

$$f_i(x_i) = \sum_{j=0}^d B_j^{(d)}(s(x_i))w_j, \quad (7.12)$$

where $x_i \in \Re^1$ is the i th component of the input vector $\mathbf{x}(t)$, $B_j^{(d)}(s(x_i))$'s are d th order Bernstein polynomials and w_j 's are weights determined from input-output data. In order to derive these basis functions, it is necessary to find a one-to-one mapping: $\Psi_i(\cdot) : x_i \in [\min(x_i), \max(x_i)] \rightarrow s(x_i) \in [0, 1]$. It is well known [63] that the univariate de Casteljau algorithm which has been used in Bézier curve construction, can realise a mapping $s \in [0, 1] \rightarrow x_i(s) \in [\min(x_i), \max(x_i)]$ with respect to a set of predetermined knots. The de Casteljau algorithm, as illustrated in Figure 7.8, is described next.

(i) The de Casteljau Algorithm

Suppose that there are a set of plane knots $\mathbf{a}_j = [b_j, a_j]^T \in \Re^2$, $j = 1, \dots, d$, and $s \in [0, 1]$, the de Casteljau algorithm is a recursion defined on these knots by

$$\mathbf{a}_j^{(r)}(s) = (1 - s)\mathbf{a}_j^{(r-1)}(s) + s\mathbf{a}_{j+1}^{(r-1)}(s), \quad (7.13)$$

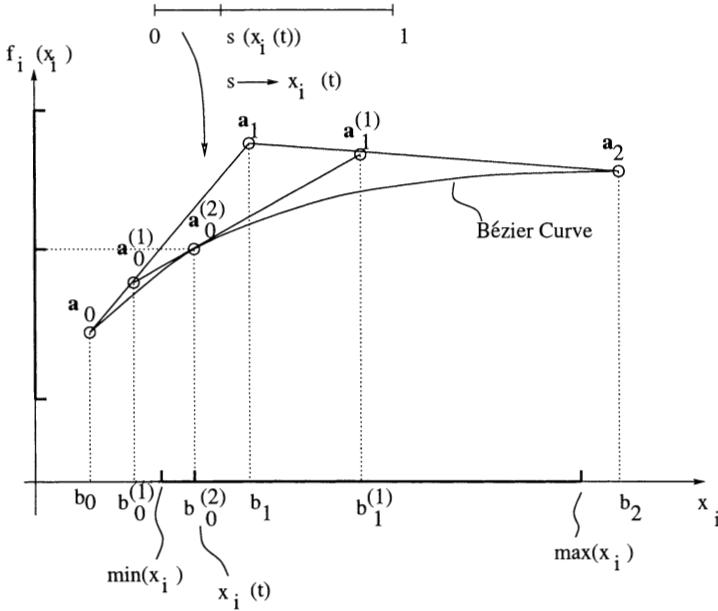


Fig. 7.8. Bézier curve generation using de Casteljau algorithm

where $\mathbf{a}_j^{(0)}(s) = \mathbf{a}_j$, $r = 1, 2, \dots, d$ and $j = 0, \dots, d - r$. Denote $\mathbf{a}_j^{(r)}(s) = [b_j^{(r)}(s), a_j^{(r)}(s)]^T$, (7.13) can be written in a decomposed form and only the first component is used here for derivation as

$$\mathbf{b}_j^{(r)}(s) = (1 - s)\mathbf{b}_j^{(r-1)}(s) + s\mathbf{b}_{j+1}^{(r-1)}(s), \quad (7.14)$$

for $\mathbf{b}_j^{(0)}(s) = \mathbf{b}_j(s)$, $r = 1, 2, \dots, d$ and $j = 0, \dots, d - r$. $b_0^{(d)}(s)$ is the first component of the point $\mathbf{a}_j^{(0)}(s)$ on the Bézier curve with parameter s . It can be shown [63] that $b_0^{(d)}$ (denoted as $\hat{x}_k(s)$) can be written as

$$\hat{x}_k(s) = b_0^{(d)}(s) = \sum_{j=0}^d b_j(s) B_j^{(d)}(s). \quad (7.15)$$

□

It is clear that the de Casteljau algorithm as a decomposed form in (7.14), is a recursive algorithm which realises the mapping from $s(x_i) \in [0, 1]$ to $\hat{x}_i(s) \in [\min(x_i) \max(x_i)] \in [b_0, b_d]$, with known supporting knot components b_j . To model $f_i(x_i)$ using a neurofuzzy network construction algorithm based on Bézier–Bernstein functions, the objective is to learn the system characteristics from only input–output data such that an inverse procedure of de Casteljau algorithm is utilised. In the following, the proposed inverse

de Casteljau algorithm for univariate terms $f_i(x_i)$ attempts to construct the mapping $\Psi_i : x_i \in [b_0, b_d] \rightarrow s \in [0, 1]$ using backpropagation type algorithm. Note that the present stage only involves the formation of basis functions and not the precise identification of $f_i(x_i)$; and also only the first component b_j of knots $\mathbf{a}_j = [b_j, a_j]^T$ need to be predetermined and not the second component a_j . Given an input vector component $x_i(t)$, the method aims to obtain a desired s by the minimisation of the loss function $\|x_i(t) - \hat{x}_i(s)\|$. By taking into account the decomposed form of the de Casteljau algorithm given in (7.14), the well-known backpropagation algorithm can be applied iteratively to obtain the new inverse procedure of de Casteljau algorithm for the univariate case, as below.

(ii) The Univariate Inverse de Casteljau Algorithm

Denote the iteration step in the minimisation procedure as superscript (m) .

1. Predetermine a set of knot components $b_j, j = 0, \dots, d$, satisfying $[\min(x_i) \max(x_i)] \in [b_0, b_d]$. Initially set $m = 1$, s as random number $s^{(m)}$, satisfying the constraints $0 < s^{(m)} < 1$.
2. Calculate the corresponding first components of Bézier curve points using the de Casteljau recursive formula (7.14) until $r = d$.

$$\hat{x}_i(s^{(m)}) = (1 - s^{(m)})b_j^{(d-1)}(s^{(m)}) + s^{(m)}b_{j+1}^{(d-1)}(s^{(m)}). \quad (7.16)$$

3. A new point is generated by using the backpropagation rule as

$$\tilde{x}_i(s^{(m)}) = \hat{x}_i(s^{(m)}) + \alpha[x_i(t) - \hat{x}_i(s^{(m)})], \quad (7.17)$$

where α is a very small positive number representing the learning rate $0 < \alpha \ll 1$. Note that α should be sufficiently small such that $\tilde{x}(s^{(m)})$ is bounded in the range: $\tilde{x}_i(s^{(m)}) \in [b_0^{(d-1)}(s^{(m)}), b_1^{(d-1)}(s^{(m)})]$.

4. The desired solution of s at iteration step $(m+1)$ is determined as the solution of s such that $\tilde{x}_i(s^{(m)})$ is the first order (linear) Bézier point with respect to two end knots as $b_0^{(d-1)}(s^{(m)})$ and $b_1^{(d-1)}(s^{(m)})$, and determined by

$$\tilde{x}_i(s^{(m)}) = (1 - s)b_0^{(d-1)}(s^{(m)}) + sb_1^{(d-1)}(s^{(m)}). \quad (7.18)$$

The solution is thus given by

$$s^{(m+1)} = \frac{\tilde{x}_i(s^{(m)}) - b_0^{(d-1)}(s^{(m)})}{b_1^{(d-1)}(s^{(m)}) - b_0^{(d-1)}(s^{(m)})}. \quad (7.19)$$

5. The procedure continues while the cost function $\|x_i(t) - \hat{x}_i(s^{(m)})\|$ is monitored to see if $\|x_i(t) - \hat{x}_i(s^{(m)})\| \leq \varepsilon$, where ε is a positive number

close to zero. If $\|x_i(t) - \hat{x}_i(s^{(m)})\| \leq \varepsilon$, then set $s = s^{(m)}$ and the iteration loop stops. Otherwise set $m = m + 1$, go to step 2.

□

The modelling of the bivariate term $f_{i_1j_1}(x_{i_1}, x_{j_1})$ used in the general ANOVA expansion of (7.9) is based on the *bivariate* de Casteljau algorithm which has been widely used in computer aided geometric surface design [63]. As a neurofuzzy network has to be constructed from only input–output data, the construction algorithm for the bivariate term $f_{i_1j_1}(\cdot)$ based on the Bézier–Bernstein function is an inverse procedure of the de Casteljau algorithms that are used in Bézier surface geometric design [63].

Definition 7.3: Bivariate Bernstein polynomial basis functions $B_{i,j,k}^{(d)}(u, v, w)$, (with $u + v + w = 1$) are terms in the expansion of $(u + v + w)^d$, defined by

$$B_{i,j,k}^{(d)}(u, v, w) = \frac{d!}{i!j!k!} u^i v^j w^k, \quad i + j + k = d, \quad (7.20)$$

where i, j, k are non-negative integers. The total number of $B_{i,j,k}^{(d)}(u, v, w)$ is $\frac{1}{2}(d+1)(d+2)$.

It can be shown [63] that the generated Bernstein polynomials satisfy the following recursion :

$$\begin{aligned} B_{i,j,k}^{(d)}(u, v, w) &= u B_{i-1,j,k}^{(d-1)}(u, v, w) + v B_{i,j-1,k}^{(d-1)}(u, v, w) \\ &\quad + w B_{i,j,k-1}^{(d-1)}(u, v, w), \end{aligned} \quad (7.21)$$

where u, v, w satisfies the constraints: $0 \leq u, v, w \leq 1$, and $u + v + w = 1$. Bernstein polynomials $B_{i,j,k}^{(d)}(u, v, w)$ are well suitable as a fuzzy membership function candidates as $B_{i,j,k}^{(d)}(u, v, w) \in [0, 1]$ (representing fuzzy membership function) and $\sum_{i+j+k=d} B_{i,j,k}^{(d)}(u, v, w) = 1$ (a unity of partition). Consider the construction problem for the bivariate term $f_{i_1j_1}(x_{i_1}, x_{j_1})$. Denote a system input subvector $\mathbf{x}_{i_1j_1} = [x_{i_1}, x_{j_1}]^T \in \Re^2$, where x_{i_1} and x_{j_1} , ($i_1 \neq j_1$) denote the i_1 th and j_1 th components of the input vector $\mathbf{x}(t)$. Suppose $f_{i_1j_1}(x_{i_1}, x_{j_1})$ is modelled using the neurofuzzy system with Bernstein polynomials $B_{i,j,k}^{(d)}(u, v, w)$ as basis functions as

$$f_{i_1j_1}(x_{i_1}, x_{j_1}) = \sum_{i+j+k=d} w_{i,j,k} B_{i,j,k}^{(d)} [u(\mathbf{x}_{i_1j_1}(t)), v(\mathbf{x}_{i_1j_1}(t)), w(\mathbf{x}_{i_1j_1}(t))], \quad (7.22)$$

where $u(\mathbf{x}_{i_1j_1}(t)) + v(\mathbf{x}_{i_1j_1}(t)) + w(\mathbf{x}_{i_1j_1}(t)) = 1$, and $0 \leq u(\mathbf{x}_{i_1j_1}(t)), v(\mathbf{x}_{i_1j_1}(t)), w(\mathbf{x}_{i_1j_1}(t)) \leq 1$. The total number of terms in (7.22) is $\frac{1}{2}(d+1)(d+2)$.

The de Casteljau algorithm based surface design [63] is based on the fundamental concept of the Bézier triangle patch defined as

Defintion 7.4: Bézier triangle patch

A Bézier triangle patch of polynomial degree d can be constructed based on a polygon \mathcal{X} terraced by d^2 triangles formed by $\frac{1}{2}(d+1)(d+2)$ vertices labelled as $\mathbf{b}_{i,j,k}$, with $i+j+k = d$. In the cubic case, as shown in Figure 7.9, the control net consists of vertices

$$\begin{aligned} & \mathbf{b}_{3,0,0} \\ & \mathbf{b}_{2,1,0} \mathbf{b}_{2,0,1} \\ & \mathbf{b}_{1,2,0} \mathbf{b}_{1,1,1} \mathbf{b}_{1,0,2} \\ & \mathbf{b}_{0,3,0} \mathbf{b}_{0,2,1} \mathbf{b}_{0,1,2} \mathbf{b}_{0,0,3} \end{aligned}$$

The de Casteljau algorithm involves the construction of a point $\mathbf{b}_{0,0,0}^{(d)}(u, v, w)$ on a Bézier triangle patch (cubic polynomial surface if $d=3$) by using a recursive linear interpolation. Given a triangle array of points $\mathbf{b}_{i,j,k}$, with $i+j+k = d$ and a given barycentric coordinates $\{u, v, w\}$, with $u+v+w = 1$, $0 \leq u, v, w \leq 1$.

$$\begin{aligned} \mathbf{b}_{i,j,k}^{(r)}(u, v, w) = & u \mathbf{b}_{i+1,j,k}^{(r-1)}(u, v, w) + v \mathbf{b}_{i,j+1,k}^{(r-1)}(u, v, w) \\ & + w \mathbf{b}_{i,j,k+1}^{(r-1)}(u, v, w) \end{aligned} \quad (7.23)$$

for $r = 1, 2, \dots, d$, where $i+j+k = d-r$, $\mathbf{b}_{i,j,k}^{(0)}(u, v, w) = \mathbf{b}_{i,j,k}$. It can be shown that, by using the recursive formula (7.21) and (7.23), the point on

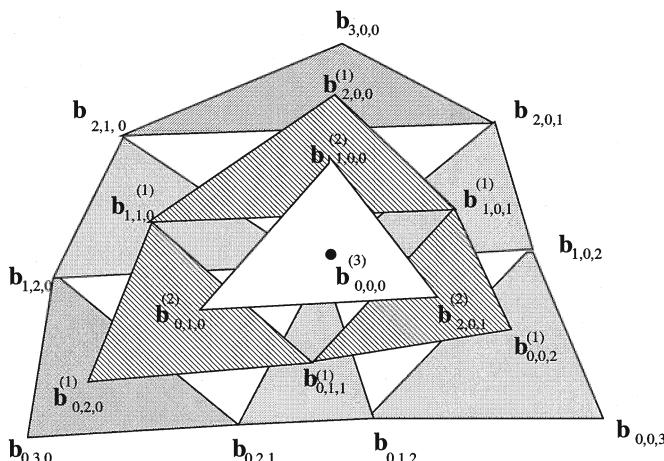


Fig. 7.9. Bézier surface generation using de Casteljau algorithm

the resultant Bézier triangle patch $\mathbf{b}_{0,0,0}^{(d)}(u, v, w)$ (denoted as $\hat{\mathbf{x}}_o(u, v, w)$) can be written as

$$\begin{aligned}\hat{\mathbf{x}}_o(u, v, w) &= \mathbf{b}_{0,0,0}^{(d)}(u, v, w) \\ &= \sum_{i+j+k=d} \mathbf{b}_{i,j,k}(u, v, w) B_{i,j,k}^{(d)}(u, v, w).\end{aligned}\quad (7.24)$$

Denote the range of the resultant Bézier triangle patch formed using polygon \mathcal{X} as $\tilde{\mathcal{X}}$. The de Casteljau algorithm realises the mapping $\Psi_o^{-1} : \{u, v\} \in \text{triangle}\{u = 0, v = 0, u + v = 1\} \rightarrow \mathbf{x}_o(u, v) \in \tilde{\mathcal{X}}$. An important property called endpoint interpolation exists for the Bézier surface, that is, $\hat{\mathbf{x}}_o(1, 0, 0) = \mathbf{b}_{d,0,0}^{(d)}$, $\hat{\mathbf{x}}_o(0, 1, 0) = \mathbf{b}_{0,d,0}^{(d)}$ and $\hat{\mathbf{x}}_o(0, 0, 1) = \mathbf{b}_{0,0,d}^{(d)}$.

Definition 7.5: Extended barycentric coordinates

A natural extension of the Definition 7.2 is to interpret the $\{u, v, w\}$ as the barycentric coordinates of $\hat{\mathbf{x}}_o \in \mathcal{X}$ with respect to \mathcal{X} , if the location of $\hat{\mathbf{x}}_o \in \tilde{\mathcal{X}}$ can be expressed in terms of barycentric coordinates $\{u, v, w\}$ with respect to the polygon \mathcal{X} with vertices $\mathbf{b}_{i,j,k}$, $i + j + k = d$ as

$$\hat{\mathbf{x}}_o(u, v, w) = \sum_{i+j+k=d} \mathbf{b}_{i,j,k}(s) B_{i,j,k}^{(d)}(u, v, w).\quad (7.25)$$

In order to construct the bivariate term $f_{i_1 j_1}(\cdot)$ based on (7.22) it is necessary to find a one-to-one mapping: $\Psi_{i_1 j_1} : \mathbf{x}_{i_1 j_1}(t) \in \mathbb{R}^2 \rightarrow \{u(\mathbf{x}_{i_1 j_1}(t)), v(\mathbf{x}_{i_1 j_1}(t))\} \in \text{triangle } \{u = 0, v = 0, u + v = 1\}$, or it is necessary to relate system input subvector $\mathbf{x}_{i_1 j_1}(t)$ with the parameters $\{u(t), v(t), w(t)\}$. It follows from Definition 7.4 that in the neurofuzzy system (7.10), shown in Figure 7.5, the bivariate term $f_{i_1 j_2}(\cdot)$ can be constructed based on Bézier–Bernstein polynomial basis functions given in (7.22), if $u(\mathbf{x}_{i_1 j_1}(t)), v(\mathbf{x}_{i_1 j_1}(t))$ are extended barycentric coordinates of $\mathbf{x}_{i_1 j_1}(t)$ with respect to \mathcal{X} , a polygon which is formed using a set of predetermined knots $\mathbf{b}_{i,j,k}$. Based on this basic idea, a construction algorithm has been introduced [98] which aims to construct the function mapping $\Psi_{i_1 j_1} : \mathbf{x} \in \tilde{\mathcal{X}} \rightarrow \{u, v\} \in \text{triangle}\{u = 0, v = 0, u + v = 1\}$. The endpoint interpolation property of a Bézier surface can be applied to predetermine the knots $\mathbf{b}_{i,j,k}$ using the input data range. The input data domain should be bounded within the Bézier triangle patch $\tilde{\mathcal{X}}$, formed from \mathcal{X} using the de Casteljau algorithm. It is convenient to predetermine the boundary knots such that \mathcal{X} should be sufficiently larger than input data domain due to the endpoint interpolation property.

The de Casteljau algorithm is a recursive algorithm which realises the mapping from $\{u, v\} \in \text{triangle}\{u = 0, v = 0, u + v = 1\} \rightarrow \mathbf{x}_{i_1 j_1} \in \tilde{\mathcal{X}}$, with respect to the known supporting knots $\mathbf{b}_{i,j,k}$. The neurofuzzy network construction algorithm based on Bézier–Bernstein function is different from the de Casteljau algorithm based surface design method in that the objective is to learn the system characteristics only from input–output data, so that

an *inverse* procedure of the de Casteljau algorithm is necessary to find the extended barycentric coordinates. So in the proposed algorithm, the Bézier–Bernstein functional network basis is formed by using the barycentric coordinates of an input subvector, which in turn are obtained by using an inverse de Casteljau procedure based on the backpropagation algorithm. The network weights can then be trained by the least squares methods as in other linear-in-the-weights artificial networks. The principle of the new inverse de Casteljau algorithm is as follows: given input subvector data $\mathbf{x}_{i_1 j_1} \in \tilde{\mathcal{X}}$, the proposed inverse de Casteljau algorithm attempts to construct the mapping $\Psi : \mathbf{x}_{i_1 j_1} \in \tilde{\mathcal{X}} \rightarrow \{u, v\} \in \text{triangle}\{u = 0, v = 0, u + v = 1\}$ using backpropagation, where $\{u, v, 1 - u - v\}$ are the barycentric coordinates of input subvector $\mathbf{x}_{i_1 j_1}$ with respect to the known supporting knots. Note that the de Casteljau algorithm can be viewed as a special multi-layer network structure, illustrated by Figure 7.9 and the recursive formula (7.21). For an input subvector $\mathbf{x}_{i_1 j_1}(t)$, a method of obtaining a desired $\{u, v, 1 - u - v\}$ is by the minimisation of the loss function $\|\mathbf{x}_{i_1 j_1}(t) - \hat{\mathbf{x}}_o(u, v)\|$ via backpropagation. In the backpropagation procedure, the traditional backpropagation method can be extended here for this special multi-layer network structure by utilising the matrix inversion (7.6) which derives barycentric coordinates of an input data point with respect to a triangle (within which the input data is bounded). It is advantageous that the output error $\|\mathbf{x}_{i_1 j_1}(t) - \hat{\mathbf{x}}_o(u, v)\|$ can be easily propagated to dynamically adjust the values for the desired $\{u, v, 1 - u - v\}$, so as the mapping $\Psi : \mathbf{x}_{i_1 j_1} \in \tilde{\mathcal{X}} \rightarrow \{u, v\} \in \text{triangle}\{u = 0, v = 0, u + v = 1\}$ can be achieved.

(iii) The Bivariate Inverse de Casteljau Algorithm

Denote the iteration step in the minimisation procedure as superscript (m) .

- Initially set $m = 1$, $\{u, v\}$ as random number $\{u^{(m)}, v^{(m)}\}$, satisfying the constraints $0 < u^{(m)} < 1$, $0 < v^{(m)} < 1$ and $0 < u^{(m)} + v^{(m)} < 1$.
- Calculate the corresponding point on the Bézier triangle patch using the de Casteljau algorithm using the recursive formula (7.23) until $r = d$.

$$\begin{aligned} \hat{\mathbf{x}}_o(u^{(m)}, v^{(m)}, w^{(m)}) = & u^{(m)} \mathbf{b}_{1,0,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}) \\ & + v^{(m)} \mathbf{b}_{0,1,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}) \\ & + w^{(m)} \mathbf{b}_{0,0,1}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}). \end{aligned} \quad (7.26)$$

- A new point is generated by using backpropagation rule through

$$\begin{aligned} \tilde{\mathbf{x}}_{i_1 j_1}(u^{(m)}, v^{(m)}, w^{(m)}) = & \hat{\mathbf{x}}_o(u^{(m)}, v^{(m)}, w^{(m)}) \\ & + \alpha [\mathbf{x}_{i_1 j_1}(t) - \hat{\mathbf{x}}_o(u^{(m)}, v^{(m)}, w^{(m)})], \end{aligned} \quad (7.27)$$

where α is a very small positive number representing learning rate $0 < \alpha \ll 1$. Note that α should be sufficiently small such that $\tilde{\mathbf{x}}_{i_1 j_1}(u^{(m)}, v^{(m)}, w^{(m)})$ is bounded within the triangle with vertices $\{\mathbf{b}_{1,0,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}), \mathbf{b}_{0,1,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}), \mathbf{b}_{0,0,1}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)})\}$.

4. The desired solution of $\{u, v\}$ at iteration step $(m + 1)$ is determined as the barycentric coordinates of $\tilde{\mathbf{x}}_{i_1 j_1}(u^{(m)}, v^{(m)}, w^{(m)})$ with respect to the triangle with vertices $\{\mathbf{b}_{1,0,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}), \mathbf{b}_{0,1,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}), \mathbf{b}_{0,0,1}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)})\}$. Applying (7.24) (by definition) with w replaced by $1 - u - v$, yields

$$\begin{aligned}\tilde{\mathbf{x}}_{i_1 j_1}(u^{(m)}, v^{(m)}, w^{(m)}) &= u\mathbf{b}_{1,0,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}) \\ &\quad + v\mathbf{b}_{0,1,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}) \\ &\quad + (1 - u - v)\mathbf{b}_{0,0,1}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}).\end{aligned}\tag{7.28}$$

The solution $\{u, v\}$ can be solved from (7.6) as

$$\begin{vmatrix} u^{(m+1)} \\ v^{(m+1)} \end{vmatrix} = \tilde{B}_{(m)}^{-1} \left[\tilde{\mathbf{x}}_{i_1 j_1}(u^{(m)}, v^{(m)}, w^{(m)}) - \mathbf{b}_{0,0,1}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}) \right],$$

where

$$\begin{aligned}\tilde{B}_{(m)} = & \left[\mathbf{b}_{1,0,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}) - \mathbf{b}_{0,0,1}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}), \right. \\ & \left. \mathbf{b}_{0,1,0}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}) - \mathbf{b}_{0,0,1}^{(d-1)}(u^{(m)}, v^{(m)}, w^{(m)}) \right].\end{aligned}\tag{7.29}$$

5. The procedure continues while the cost function $\|\mathbf{x}_{i_1 j_1}(t) - \hat{\mathbf{x}}_o(u, v)\|$ is monitored to see if $\|\mathbf{x}_{i_1 j_1}(t) - \hat{\mathbf{x}}_o(u^{(m)}, v^{(m)})\| \leq \varepsilon$, where ε is a positive number close to zero. If $\|\mathbf{x}_{i_1 j_1}(t) - \hat{\mathbf{x}}_o(u, v)\| \leq \varepsilon$, then set $u = u^{(m)}$ and $v = v^{(m)}$ and the iteration loop stops. Otherwise set $m = m + 1$, go to step 2.

□

7.4.2 The complete Bézier–Bernstein model construction algorithm

The neurofuzzy modelling using Bézier–Bernstein polynomial function via additive decomposition, shown in Figure 7.5, is summarised as an off-line version as below.

The Bézier–Bernstein Model Construction Algorithm

Given input–output data sets $\{\mathbf{x}(t), y(t)\}_{t=1}^N$, predetermine a polynomial degree as d .

1. For each $x_i(t)$, the i th component of the input vector $\mathbf{x}(t)$, predetermine b_0, \dots, b_d , first components of $(d+1)$ knots as shown in Figure 7.8, under the constraints that $[\min(x_i(t)), \max(x_i(t))] \in [b_0, b_d]$.
2. For all t , applying the univariate inverse de Casteljau algorithm to find out the solution $s(t)$ for $x_i(t)$.
3. Apply (7.10) and (7.12) to form a basis subvector $\mathbf{B}(x_i(t)) \in \mathbb{R}^{d+1}$. The objective of procedures 1 – 3 is to form the basis functions in the univariate term $f_i(\cdot)$. The total number of the univariate basis functions is therefore $n(d+1)$.
4. For every combination using $x_{i_1}(t)$ and $x_{j_1}(t)$, ($i_1 \neq j_1$) two distinctive components of the input vector $\mathbf{x}(t)$, predetermine the position of $\frac{1}{2}(d+1)(d+2)$ knots $\mathbf{b}_{i,j,k}$ ranked to form the polygon \mathcal{X} shaped as plotted in Figure 7.9. This can be done by graphically inputting via mouse in a graph window in which the input data range had been plotted. The polygon \mathcal{X} should be sufficiently larger than the input range to ensure all the input data are bounded within the generated d polynomial degree Bézier surface which formed using a series of triangles formed by $\mathbf{b}_{i,j,k}$'s.
5. For all t , applying the bivariate inverse de Casteljau algorithm to find the $u(t), v(t)$, and $w(t) = 1 - u(t) - v(t)$ from the input subvector $[x_{i_1}(t), x_{j_1}(t)]$.
6. Applying (7.20) and (7.22) to form the basis vector $\mathbf{B}(\mathbf{x}_{i_1 j_1}(t)) \in \mathbb{R}^{\frac{1}{2}(d+1)(d+2)}$ for each input subvector $[x_{i_1}(t), x_{j_1}(t)]$. The objective of procedures 4 – 6 is to from the basis function in the bivariate term $f_{i_1 j_1}(\cdot)$. The total number of the bivariate basis functions is therefore $\frac{1}{4}n(n-1)(d+1)(d+2)$.
7. Substitute (7.12) and (7.22) into (7.9). The total number of the basis functions excluding the constant term is $p = n(d+1) + \frac{1}{4}n(n-1)(d+1)(d+2)$. Least squares is then used, via the minimisation of $\sum_{t=1}^N (y(t) - \hat{y}(t))^2$ to obtain the weights vector $\mathbf{w} = [..., w_j, ..., w_{i,j,k}, ...]^T$ in (7.9).

□

7.4.3 Numerical examples

Example 7.1b. The standard benchmark nonlinear autoregressive time series of Section 4.5 (using the same data set as in Example 7.1a)

A 2-D Bézier–Bernstein network (using only one bivariate term) has been applied [98]. The input vector is set as $\mathbf{x}(t) = [y(t-1), y(t-2)]^T$. Set $d = 4$; the number of the knots (number of basis functions) is $p = \frac{1}{2}(d+1)(d+2) = 15$. The knots $\mathbf{b}_{i,j,k}$'s were input to a graph window to form the polygon \mathcal{X} and plotted in Figure 7.10, where the input data $y(t-1)$ with respect to $y(t-2)$ is also shown and covered by the polygon. The mean square error (MSE) of the resultant network over the validation data set was obtained as $\sigma^2 = 0.0207^2$, compared to $\sigma^2 = 0.022^2$ for example 7.1a. The one-step ahead prediction

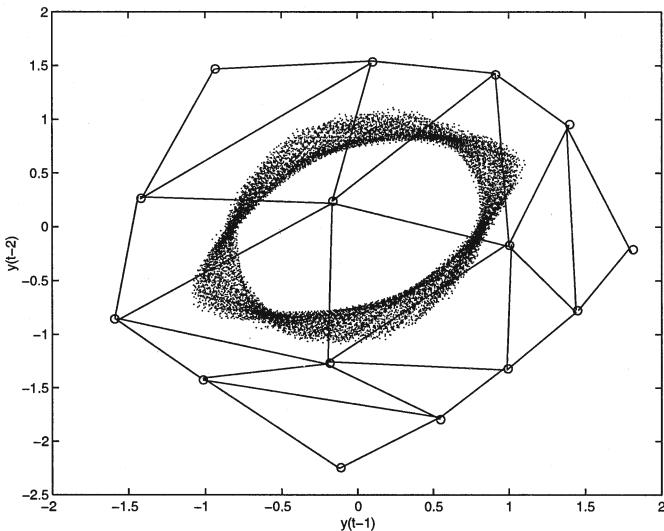


Fig. 7.10. The Delaunay input space partition using predetermined knots for Example 7.1b

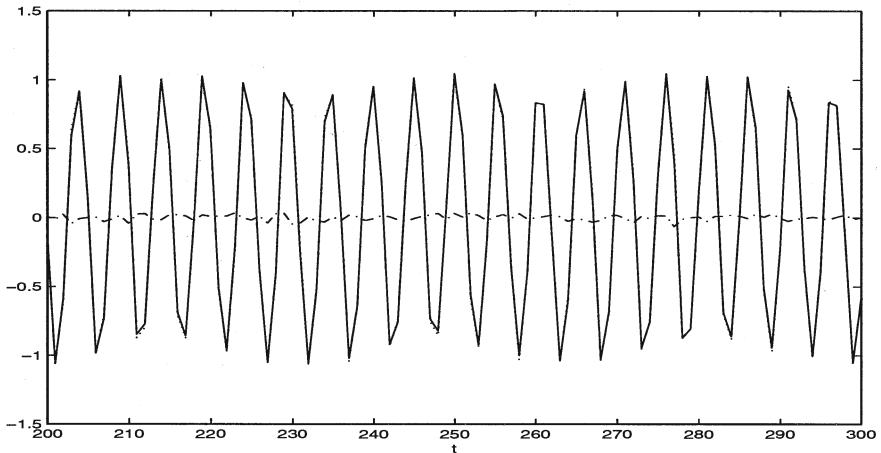


Fig. 7.11. System actual output (solid line), model one-step ahead predictions (dotted line) and model residual (dash-dotted line) for Example 7.1b

using the obtained network over $t = 200 - 300$ is demonstrated in Figure 7.11 to indicate system performance.

Example 7.2. Consider a nonlinear dynamical system given by

$$y(t) = \frac{y^2(t-1) + u_s(t-2)}{1 + y^2(t-1)} + 0.4\xi(t-1) + \xi(t), \quad (7.30)$$

where the system input sequence $u_s(t)$ is a uniformly distributed random excitation signal within range $[-1, 1]$ and the noise sequence $\xi(t) \sim N(0, 0.1^2)$. Five hundred data samples were generated for estimation set. Suppose that system structure is known to have a linear noise term

$$y(t) = f(\mathbf{x}(t)) + c\xi(t-1) + \xi(t), \quad (7.31)$$

where the input vector $\mathbf{x} = [y(t-1), u_s(t-1), u_s(t-2)]$. Set polynomial degree $d = 4$ for both univariate basis function and bivariate basis function. Corresponding to the additive expansion based on (7.9). The system input vector is decomposed into six terms based on $y(t-1)$, $u_s(t-1)$, $u_s(t-2)$, $[y(t-1), u_s(t-1)]$, $[y(t-1), u_s(t-2)]$ and $[u_s(t-1), u_s(t-2)]$ respectively. For each of these terms, the knots are inputs to a graph window as shown in Figure 7.12. The knots are subject to constraints such that, for the univariate term, the input region $[\min(\xi(t)), \max(\xi(t))] \in [b_0, b_d]$, and for the bivariate terms, a polygon \mathcal{X} formed by the knots $\mathbf{b}_{i,j,k}$'s should sufficiently cover the input region. In this example the knots for univariate terms $f(y(t-1))$ are plotted as in Figure 7.12(a), the knots for univariate terms $f(u_s(t-1))$ and $f(u_s(t-2))$ (set to be the same) are plotted in Figure 7.12(b). The knots for bivariate terms $f(y(t-1), u_s(t-1))$, $f(y(t-1), u_s(t-2))$ and $f(u_s(t-1), u_s(t-2))$ are plotted in Figure 7.12(c), (d) and (e), respectively.

For all data samples, the univariate inverse de Casteljau algorithm is applied to find the $s(t)$ from the three input components $y(t-1)$, $u_s(t-1)$ and $u_s(t-2)$ respectively. Each basis subvector $\mathbf{B}(x_i(t)) \in \mathbb{R}^{d+1}$ is formed by applying (7.10) and (7.12). The number of basis function in univariate terms is therefore $3(d+1) = 15$.

Then for all data samples, the bivariate inverse de Casteljau algorithm is applied to find the $u(t)$, $v(t)$, and $w(t) = 1 - u(t) - v(t)$ from 3 input subvectors $[y(t-1), u_s(t-1)]$, $[y(t-1), u_s(t-2)]$ and $[u_s(t-1), u_s(t-2)]$ respectively. By applying (7.20) and (7.22), the bivariate basis vector $\mathbf{B}(\mathbf{x}_{i_1 j_1}(t)) \in \mathbb{R}^{\frac{1}{2}(d+1)(d+2)}$ for each subvector $[y(t-1), u_s(t-1)]$, $[y(t-1), u_s(t-2)]$ and $[u_s(t-1), u_s(t-2)]$ is formed. The total number of the basis functions in bivariate terms is therefore $\frac{3}{2}(d+1)(d+2) = 45$.

For the parameter estimation of system (7.31), the weight vector \mathbf{w} in $f(\mathbf{x}(t))$ is finally obtained by using the extended least squares due to the noise term in the model structure of (7.31) and the term $\xi(t-1)$ can be substituted by $(y(t-1) - \hat{y}(t-1|t-2))$, where $\hat{y}(t-1|t-2)$ is model predictions via several times of iterations. The MSE of the network is obtained as $\sigma^2 = 0.092^2$. The autocorrelation coefficients of the model residual, plotted in Figure 7.13(a), is used for model validation, showing that the model residual is uncorrelated. A comparison of the actual system output with one-step ahead prediction using the obtained network is demonstrated in Figure 7.13(b) to indicate the excellent system performance. It has been shown again by this more complex function approximation example that the Bézier–Bernstein polynomial function based neurofuzzy system possesses excellent modelling capabilities.

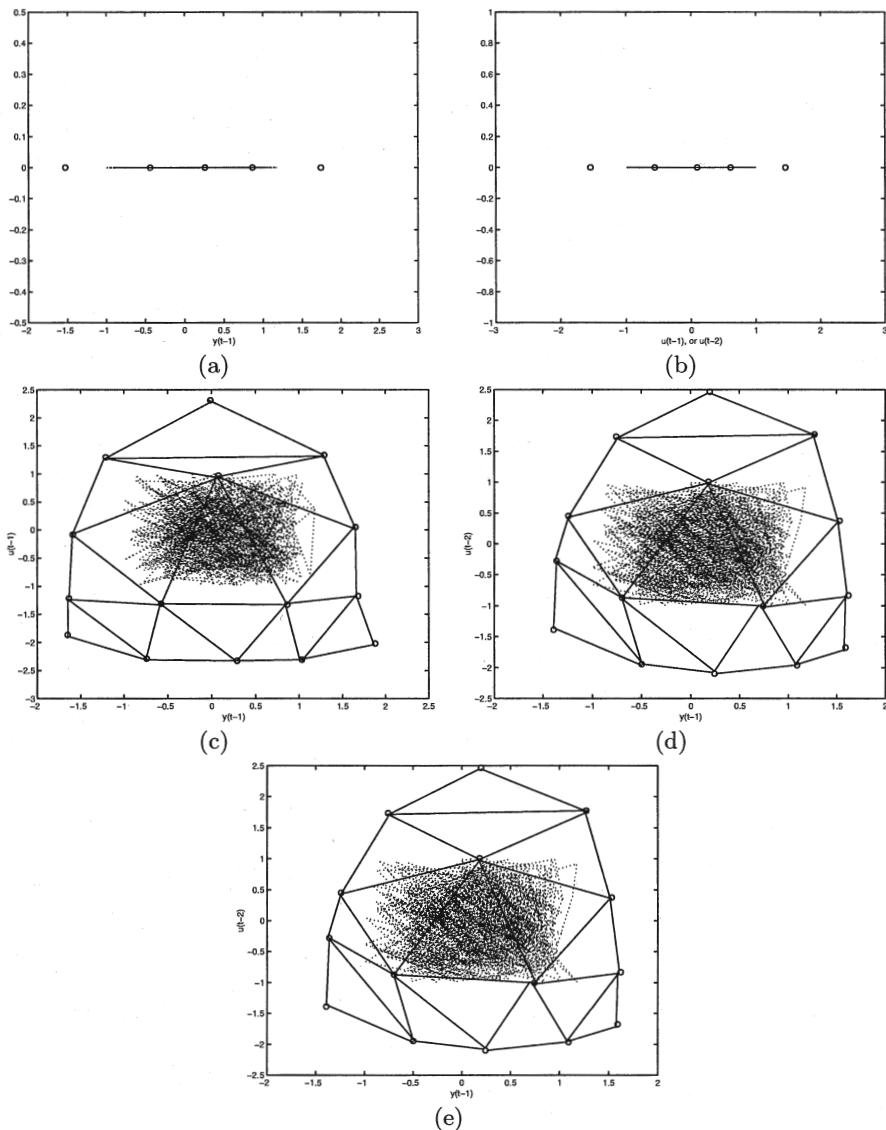


Fig. 7.12. Input space partition using predetermined knots in Example 7.2; (a) knots for univariate term $f(y(t - 1))$; (b) knots for univariate term $f(u_s(t - 1), f(u_s(t - 2)))$; (c) knots for bivariate term $f(y(t - 1), u_s(t - 1))$; (d) knots for bivariate term $f(y(t - 1), u_s(t - 2))$; and (e) knots for bivariate term $f(u_s(t - 1), u_s(t - 2))$

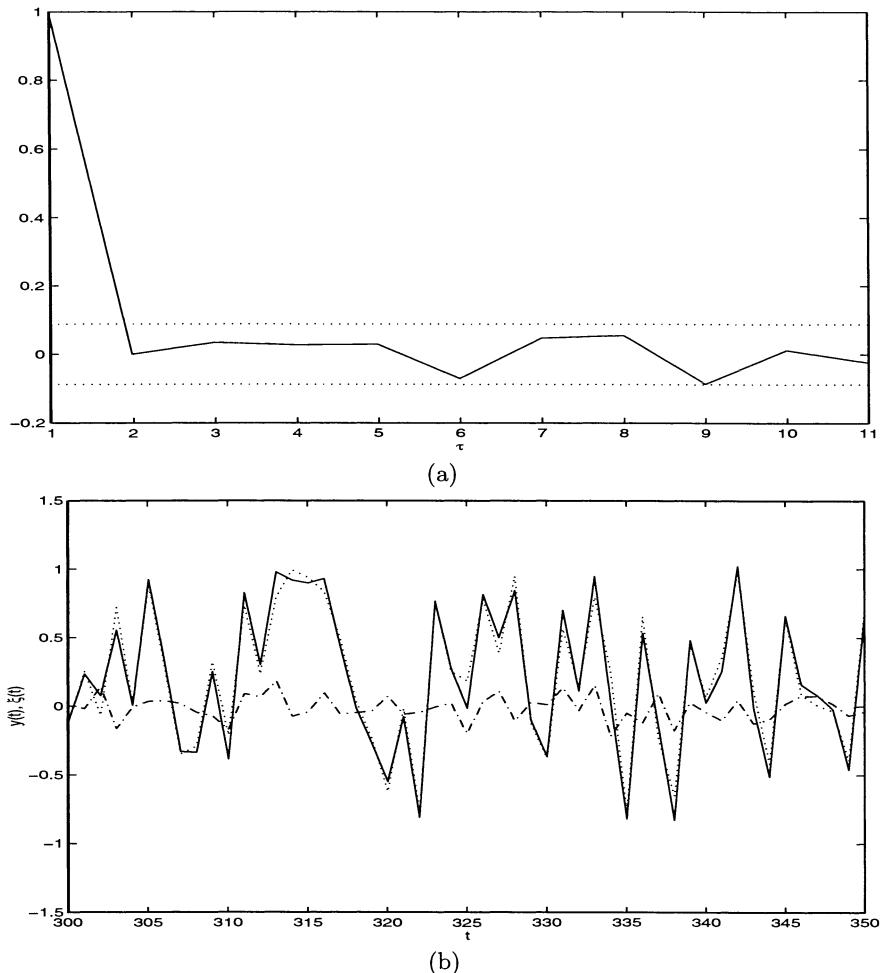


Fig. 7.13. Simulation results in Example 7.2; (a) model validation test using autocorrelation coefficients of the residual sequence; (b) system actual output (*solid line*), model one-step ahead predictions (*dotted line*) and model residual (*dash-dotted line*)

8. Neurofuzzy linearisation modelling for nonlinear state estimation

8.1 Introduction to linearisation modelling

In developing model-based methods for state estimation or control of *a priori* unknown dynamic processes, the first step is to establish plant models from available observational data and/or expert process knowledge. Except for the usual requirement of the model approximation ability, it is also required that the model structure is well suited for applications in the consequent state estimation and control algorithms.

Nonlinear process (or system) modelling is generally a difficult and complex task, frequently subject to a variety of problems such as data sparsity, parametric overfitting/underfitting and the curse of dimensionality (see Chapter 1). Divide-and-conquer is a principle widely used to attack complex problems by dividing a complex problem into simpler problems whose solutions can be combined to yield a solution to the complex problem. There are three critical issues in divide-and-conquer methods: (i) the decomposition of the complex problem, or the partitioning of the input or operating space in the case of nonlinear process modelling; (ii) the design of corresponding local models to generate global models that generalise well; (iii) the combination of the local models (see Chapters 6 and 7). Neurofuzzy local linearisation (NFLL) has recently emerged as a useful divide-and-conquer approach to nonlinear process modelling, it distinguishes itself from traditional piecewise linearisation by fuzzy (soft) input space partitioning (see also Chapter 6). Although new methods have been developed for crisp input space partition in piecewise linear modelling [101], their applications are restricted due to the inherent ambiguity or fuzziness in the input space partitioning based upon local linearities. NFLL provides a potential way to resolve this problem.

Most early work in local linearisation has been based on the first-order Tagagi-Sugeno (T-S) fuzzy model [185], in which local linear models are constructed on local regions generated from the input space partition by fuzzy sets and are combined by membership function weighting. Both the membership functions that define the fuzzy sets and then the corresponding local linear models need to be identified by using neural learning algorithms based on optimisation techniques and observational data and/or fuzzy rules. Whilst least squares (LS) and least mean squares (LMS) are widely used conventional methods for optimisation (see Chapter 3), expectation–maximisation

(EM) is a general technique for maximum likelihood or maximum *a posteriori* estimation that has become an alternative to LS and LMS techniques in solving many estimation problems [53, 74, 83, 125, 126, 208]. LS, LMS and EM are frequently used for local model identification. For input space partitioning, growing-pruning algorithms based on some optimal criteria, such as structural risk minimisation (SRM), have been developed (see Chapters 1 and 10). For instance, the adaptive spline modelling (ASMOD) algorithm (see Chapter 5) based on the analysis of variance (ANOVA) decomposition has been widely used in spline models for combating the problem of curse of dimensionality in high dimensional system modelling. In the original paper on fuzzy local linearisation by Takagi and Sugeno [185], an algorithm was developed to identify the membership functions based on fuzzy rules. Although in almost all papers subsequently triangular membership functions have been used as their piecewise linear structure aid analysis, Jang [121] and Johansen and Foss [124] used Gaussians as membership functions, with their means and variances identified by gradient-descent algorithms or set manually based on observational data. Wang et al. [196] investigated a local linearisation (operating-point dependent) model for adaptive nonlinear control, using B-splines as membership functions which were selected manually by making use of the inherent transparency of B-splines. Wu and Harris [207] applied a neurofuzzy local linearisation model to nonlinear state estimation of single-input-single-output (SISO) dynamic systems, in which B-splines were used as membership functions and similarly constructed manually (see also Chapter 6 under direct and indirect state estimation). In all the above algorithms, the local linear models are identified by LS or by the LMS algorithm. An important issue that has not been well addressed in the above methods is how to automatically partition the input space using fuzzy sets in an effective (parsimonious) manner. This is a major weakness in any data-based modelling paradigm, where there is a requirement to determine both model structure and parameterisation automatically. Aiming at resolving this problem, Gan and Harris [76, 79] decomposed a neurofuzzy local linearisation model into submodels in an ANOVA form and developed a modified ASMOD (MASMOD) algorithm for automatic fuzzy partitioning of the input space, in which B-splines, as membership functions, were automatically constructed, that is, the number of decomposed submodels, the number of B-splines required in each submodel and their positions and shapes were automatically determined by the MASMOD algorithm based on observational data and other *a priori* process knowledge. This model construction algorithm essentially decomposes the underlying nonlinear process into an additive set of low dimensional submodels which are individually parameterised, avoiding the curse of dimensionality.

It is possible to give NFLL a probabilistic interpretation, because the relation between the input vector \mathbf{x} and the output vector \mathbf{y} of a nonlinear system can be described by multiple probabilistic models. The conditional

probability of \mathbf{y} given \mathbf{x} can be constructed as the weighted sum of local probabilistic models as in the mixture of experts algorithm (see Section 6.5). The local probabilistic models can be defined as Gaussians, with linear functional means and covariance matrices determined by learning algorithms. The weights are defined as probability functions associated with local models, whose parameters also need to be determined by learning. Jordan and Jacobs [126] and Xu and Jordan [208] used the EM algorithm to identify the parameters in both the weight functions and the local probabilistic models. The difficulty of the Jordan's EM algorithm lies in the selection of the following hyperparameters: the number of local models and the learning rate for updating the parameters in the weight functions. If the whole probabilistic model has been identified, the expected value of \mathbf{y} given \mathbf{x} can be calculated from the weighted sum of the linear functional means of the local probabilistic models, the formulation of which is similar to the NFLL scheme. This observation motivates the idea of combining the method for membership function identification in the neurofuzzy local linearisation method with the method for identifying local probabilistic models in the probabilistic model approach to improve the performance of NFLL modelling and local probabilistic modelling or to provide covariance information about the model mismatch [74], which is helpful for the consequent state estimation.

Feedback linearisation [116, 173, 179, 193] is one of the major techniques for nonlinear system analysis and control and has also drawn extensive research interest in recent years [71, 77]. Feedback linearisation is a well known technique to force a nonlinear plant to behave linearly by a state coordinate transformation and a feedback control law. There exists a considerable body of theoretical results on state-feedback linearisation of continuous-time nonlinear systems, mainly based on Lie derivatives [173, 193]. If the underlying nonlinearities are exactly known, the state transformation and the feedback control law can be mathematically constructed by using Lie derivatives of the nonlinear functions. Otherwise, the nonlinearities have to be approximated by some data-based modelling methods such as neural networks. There are two approaches to this modelling problem. The first is to directly approximate the nonlinear functions of the system, based on which the Lie derivatives are calculated to construct the state transformation and the feedback control law [72, 173]. However, it is a hard problem to accurately estimate high-order derivatives based on a neural network approximation of the nonlinear functions [77]. These theoretical results have not been translated into usable algorithms. An alternative way to deal with uncertain or unknown systems is to use neural networks to directly approximate the feedback term in the feedback linearised canonical form [43, 44, 119, 120, 127, 130, 131, 136, 143, 144, 145, 146, 172, 209]. The problem in the latter approach is that the state vector in the canonical form is usually unobservable without knowledge of the state coordinate transformation, leading to difficult learning in the neural network. In practice, therefore,

this approach can only be applied to systems described by nonlinear ARMA models, such as Lagrangian control systems, which are equivalent to canonical state-space models with states defined as delayed outputs. However, most realistic nonlinear systems are more complex and are totally or partially unknown, demanding the development of new approximate feedback linearisation techniques using neural networks that have a potential for widespread industry applicability. Gan and Harris [77] have developed an adaptively constructed recurrent neurofuzzy network and its learning scheme for feedback linearisation (NFFL for short), which enlarges the class of nonlinear systems that can be feedback linearised using neurofuzzy networks.

Linearisation has become an important tool in solving nonlinear problems and one of the most interesting research areas in nonlinear system analysis and control, because linearisation makes it possible to utilise well-developed linear techniques for state estimation and adaptive control in nonlinear systems. The neurofuzzy local linearisation and feedback linearisation (NFLL and NFFL) models of a nonlinear system can be easily reformulated as a time-varying linear system model whose time-varying coefficients depend on both membership functions and the parameters in local linear models. If the underlying nonlinear system is locally or feedback linearisable, then the time-varying coefficients will change slowly with the system state vector, and linear techniques for state estimation and adaptive control, such as Kalman filter, can be directly applicable to the NFLL or NFFL model, resulting in neurofuzzy state estimators [76, 79, 207].

This chapter is devoted to neurofuzzy approaches to local linearisation and feedback linearisation and their applications in nonlinear state estimation. Section 8.2 presents neurofuzzy local linearisation (NFLL) based on the MASMOD algorithm. A hybrid learning scheme for NFFL, which combines the MASMOD and EM algorithms, is introduced in Section 8.3. Section 8.4 describes the neurofuzzy method for neurofuzzy feedback linearisation (NFFL). The formulation on state estimators based on NFLL or NFFL models is given in Section 8.5. Finally, an example of nonlinear trajectory estimation using the above methods is given in Section 8.6.

8.2 Neurofuzzy local linearisation and the MASMOD algorithm

While coping with an *a priori* unknown nonlinear process, the initial problem is how to approximate the nonlinear functions that describe the process using observational data. For simplicity, let us first consider a general nonlinear function described by the input-output relation: $y(t) = f[\mathbf{x}(t)]$, where $f[.]$ represents a nonlinear function, y is the output, and $\mathbf{x}(t) = [x_1(t) \ x_2(t) \ \dots \ x_n(t)]^T$ represents the input vector. If the nonlinear process is dynamic, the input vector will include both observed external inputs and previous output feedback, for example, $\mathbf{x}(t) = [y(t-1) \ y(t-2) \ \dots \ y(t-n_y) \ u(t) \ u(t-1) \ \dots \ u(t-n_u)]^T$.

One of the most popular methods widely used in function approximation or modelling is the B-spline expansion [32, 183], expressed as follows:

$$\hat{f}(\mathbf{x}) = \sum_{k=1}^K N_k(\mathbf{x}) a_k, \quad (8.1)$$

with $f(\mathbf{x}) = \hat{f}(\mathbf{x}) + \Delta f(\mathbf{x})$, where a_k represent expansion coefficients, $N_k(\mathbf{x})$ are B-splines which can be interpreted as membership functions in a neurofuzzy system and $\Delta f(\mathbf{x})$ represents the model mismatch. It has been previously shown (see Section 4.3 and [32]) that if B-splines are used for fuzzy membership functions, algebraic sum/production operators are used for union, intersection and inference, and the centre of area defuzzification is employed, then (8.1) represents a wide class of fuzzy logic systems which not only provide good approximation capability but also transparency or interpretation through fuzzy rules. This class of representations is neurofuzzy when the weights, a_k , are adjustable (through some linear optimisation scheme), as such they reflect the confidence or belief in a set of fuzzy rules being true. If the constant coefficients a_k in (8.1) are replaced by linear functions of the inputs, then (8.1) is converted into an NFLL model:

$$\hat{f}(\mathbf{x}) = \sum_{k=1}^K N_k(\mathbf{x})(a_{k0} + \mathbf{a}_k^T \mathbf{x}), \quad (8.2)$$

where

$$N_k(\mathbf{x}) = \prod_{i=1}^n B_{i,k_i}(x_i), \quad k = 1, 2, \dots, K, \quad (8.3)$$

for $B_{i,k_i}(x_i)$ are B-splines over the variable x_i , defined by the recursive equation (4.8) in which $\mu_{A_j^k}(x)$ is replaced with $B_{i,k_i}(x_i)$. Obviously, the curse of dimensionality problem is severe for B-spline expansion when the dimension of the input space is high (say, $n > 4$). In order to resolve this problem, the ANOVA decomposition of Chapter 5 is often adopted (as in ASMOD, see Section 5.3) to decompose a high dimensional model into a sum of low dimensional submodels. By using the ANOVA decomposition in the NFLL model, (8.2) is reformed as follows:

$$\hat{f}(\mathbf{x}) = f_0 + \sum_{s=1}^S f_s(\mathbf{x}_s) = f_0 + \sum_{s=1}^S \sum_{k=1}^{K_s} N_k^{(s)}(\mathbf{x}_s)(a_{k0}^{(s)} + \mathbf{a}_k^{(s)T} \mathbf{x}_s), \quad (8.4)$$

where s is the index of B-spline submodels, $\mathbf{x}_s = [x_{s1} \ x_{s2} \ \dots \ x_{sn_s}]^T$ represent low-dimensional subspaces of the input space, $\mathbf{a}_0^{(s)} = [a_{10}^{(s)} \ a_{20}^{(s)} \ \dots \ a_{K_s}^{(s)}]^T$ and $A^{(s)} = [a_{ki}^{(s)}]_{K_s \times n_s}$ are local model parameters and $N_k^{(s)}(\mathbf{x}_s)$ are B-splines used as membership functions of the fuzzy sets that partition the s th subspaces. A special case of (8.4) is the more familiar ANOVA decomposition of (5.4):

$$\hat{f}(\mathbf{x}) = f_0 + \sum_{i=1}^n f_i(x_i) + \sum_{i=1}^{n-1} \sum_{j=i+1}^n f_{ij}(x_i, x_j) + \dots + f_{1,2,\dots,n}(\mathbf{x}), \quad (8.5)$$

in which a construction algorithm retains the univariate and bivariate sub-models in preference to higher dimensional models to overcome the curse of dimensionality associated with high dimensional inputs.

The model described by (8.4) is linear-in-the-parameters $a_{ki}^{(s)}$ and can be rearranged as

$$\hat{f}(\mathbf{x}) = f_0 + \sum_{s=1}^S \sum_{k=1}^{K_s} \sum_{i=0}^{n_s} [N_k^{(s)}(\mathbf{x}_s) \cdot x_{si}] a_{ki}^{(s)} = f_0 + \sum_{l=0}^{K'} N'_l(\mathbf{x}) w_l, \quad (8.6)$$

where, $x_{s0} = 1$, $K' = \sum_{s=1}^S K_s \times (n_s + 1)$, $w_l = a_{ki}^{(s)}$, $l = i + (k - 1) \times (n_s + 1) + \sum_{j=1}^{s-1} K_j \times (n_j + 1)$. It is very interesting to note that NFLL can be interpreted as a special form of a local basis function expansion (LBFE), with some constraints imposed on the local basis functions. With $\{\mathbf{x}(t), y(t)\}_{t=1}^N$ as training data pairs, by setting $\hat{\mathbf{y}} = [\hat{f}(\mathbf{x}(1)) \dots \hat{f}(\mathbf{x}(N))]^T$, (8.6) can be represented in a matrix form $\hat{\mathbf{y}} = \mathbf{N}'\mathbf{w}$. This is a typical equation which is directly suitable for the LS algorithm to identify the free parameters w_l .

With (8.4) as the model representation, after some modifications, the ASMOD algorithm for LBFE model construction can be applied to identify the NFLL structure and free parameters. For the B-spline construction, an *SRM* index is used as the optimisation criterion [79, 129], which is a function of root-mean-square (*RMS*) of model estimation errors and the degrees of freedom (*dof*) of the NFLL model, and the number of training samples (*N*). The *SRM* index is defined as follows:

$$SRM = \begin{cases} \frac{RMS}{\sqrt{1-PN}} & \text{if } PN < 1, \\ \infty & \text{otherwise,} \end{cases} \quad (8.7)$$

with

$$PN = \beta_1 \sqrt{[dof \cdot \log(2N) - \log(dof!) - \beta_2]/N}, \quad (8.8)$$

where, $RMS = \sqrt{\sum_{t=1}^N \{y(t) - \hat{f}[\mathbf{x}(t)]\}^2/N}$, *dof* is defined as the rank of the matrix \mathbf{N}' , and β_1 and β_2 are set as 0.9 and 4.8 respectively in our studies. The optimisation criterion for local linear model identification is modified as follows to accommodate process model (8.4):

$$V(\mathbf{a}_0^{(s)}, A^{(s)}) = \frac{1}{2} \sum_{t=1}^N \{y(t) - f_0 - \sum_{s=1}^S [(\mathbf{a}_0^{(s)} + A^{(s)} \mathbf{x}_s(t))^T \mathbf{N}^{(s)}(\mathbf{x}_s(t))] \}^2, \quad (8.9)$$

where $\{\mathbf{x}_s(t), y(t)\}_{t=1}^N$ are training data pairs. The LMS algorithm for updating the local linear models is described by:

$$\varepsilon(t) = y(t) - f_0 - \sum_{s=1}^S [(\mathbf{a}_0^{(s)} + A^{(s)} \mathbf{x}_s(t))^T \mathbf{N}^{(s)}(\mathbf{x}_s(t))], \quad (8.10)$$

$$\mathbf{a}_0^{(s)}(t) = \mathbf{a}_0^{(s)}(t-1) + \gamma(t)\varepsilon(t)\mathbf{N}^{(s)}(\mathbf{x}_s(t)), \quad (8.11)$$

$$A^{(s)}(t) = A^{(s)}(t-1) + \gamma'(t)\varepsilon(t)\mathbf{N}^{(s)}(\mathbf{x}_s(t))\mathbf{x}_s^T(t), \quad (8.12)$$

where $\gamma(t)$ and $\gamma'(t)$ are learning gain coefficients. Just like the training process in the back-propagation neural network, the training data should be represented repeatedly and the updating described by (8.10)–(8.12) should be repeated accordingly until a convergence criterion is met.

The original ASMOD algorithm is based on the local basis function expansion (LBFE) model representation described by (8.1). If the NFLL model representation described by (8.2) and (8.4) is used, some modifications introduced by (8.6) and (8.9)–(8.12) should be considered while applying the ASMOD algorithm to partition the input space. The modified ASMOD (MASMOD) algorithm can be described by the following steps.

The Modified ASMOD (MASMOD) Modelling Algorithm

1. Initialise the model, i.e. set up an empty model or a minimum sized model, set the initial order of B-splines and initial values of the free parameters in (8.4).
2. Update parameter vector $\mathbf{a}_0^{(s)}$ and matrix $A^{(s)}$ using (8.10)–(8.12) or using the least squares algorithm, based on the input-output data pairs.
3. Compute the optimisation criterion. If it is satisfied, exit.
4. Refine the model by constructing the B-splines. The refinement includes adding new univariate submodels, inserting new knots in the existing submodels, forming tensor product submodels, pruning knots or submodels (for details, refer to Section 5.3).
5. Go to Step 2.

The block diagram of the MASMOD algorithm is shown in Figure 8.1. For more details about the MASMOD algorithm, the Matlab codes for implementing MASMOD can be found via <ftp://ftp.isis.ecs.soton.ac.uk/pub/users/qg/masmod/>.

Several advantages of the MASMOD over the original ASMOD are achieved by using the NFLL model representation: for the same approximation accuracy, the number of B-splines (or the number of fuzzy rules) required is significantly reduced, also the order of B-splines is reduced, thus the total number of knots required for modelling is reduced [76]. For example, to approximate a 1-D system described by $f(x) = x^2 + 1$, $-1 \leq x \leq 3$, the NFLL model constructed by the MASMOD algorithm uses four second-order B-splines to partition the input space as shown in Figure 8.2, while the local basis function expansion (LBFE) model constructed by the ASMOD

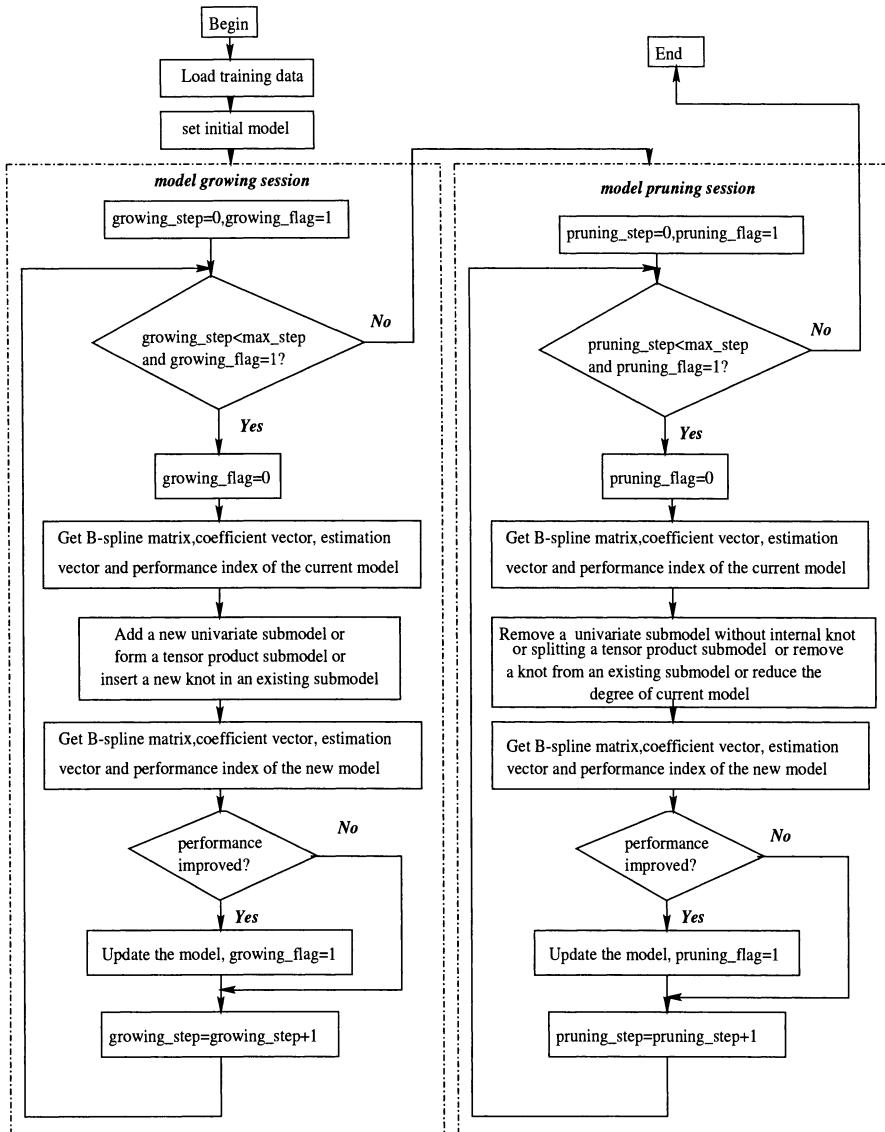


Fig. 8.1. Block diagram 1 of the MASMOD algorithm ©1999 IEEE

algorithm needs 16 second-order B-splines as shown in Figure 8.3, or four third-order B-splines as shown in Figure 8.4. Except for a direct mathematical analysis, this can also be explained in an intuitive way. As shown in Figure 8.5, in the NFLL model a nonlinear function is approximated by a combination of multiple local linear models defined by a_{ki} . Whereas in the LBFE model the same nonlinear function is approximated by a combination of special local linear models defined by a_k , i.e. horizontal lines, as shown in Figure 8.6. Intuitively, the number of local models needed in NFLL is fewer than the number of local models needed in LBFE in order to achieve the same level of approximation accuracy. Another very important advantage is that well-developed linear techniques, such as the standard Kalman filter algorithm, are directly applicable to this kind of locally linearised neurofuzzy models for state estimation and control [79, 196, 207], yet no differentiation process has been utilised (unlike the extended Kalman filter).

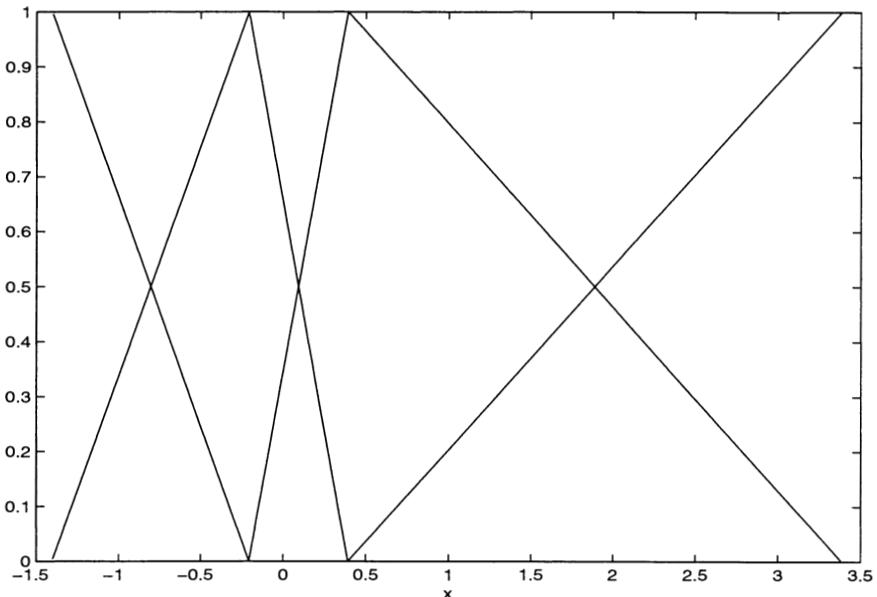


Fig. 8.2. B-splines used for 1-D input space partition by NFLL ©1999 IEEE

It is noteworthy that ANFIS [121] (see also Section 6.2) is a similarly successful model based on the first-order Tagagi-Sugeno model with Gaussian functions or bell-shaped functions as membership functions. In ANFIS, the input space is partitioned by the centres of Gaussian functions, which are determined by a gradient-descent based algorithm. There exists a potential problem of divergence and the design of centres is not so transparent as in the design of B-spline knots. The curse of dimensionality is also a fundamental

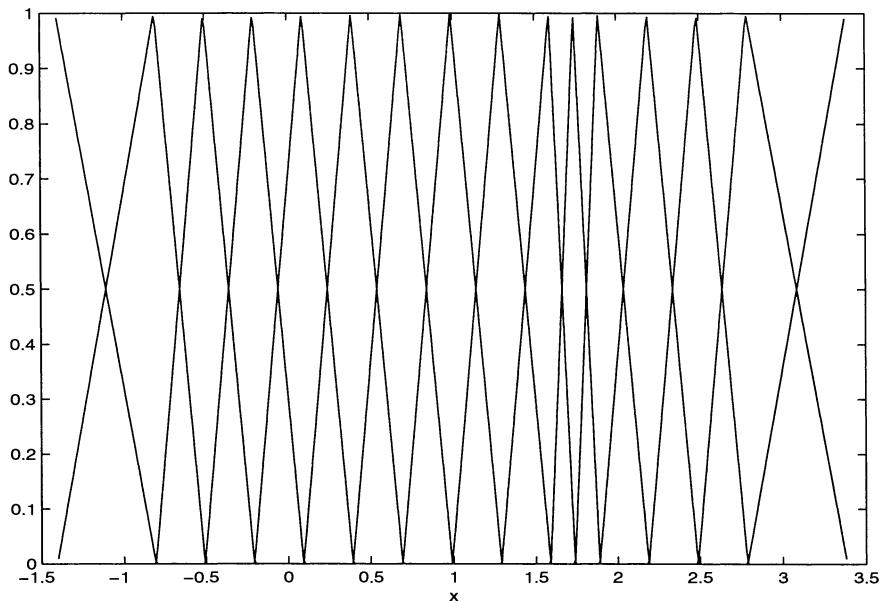


Fig. 8.3. B-splines used for 1-D input space partition by LBFE (second-order) ©1999 IEEE

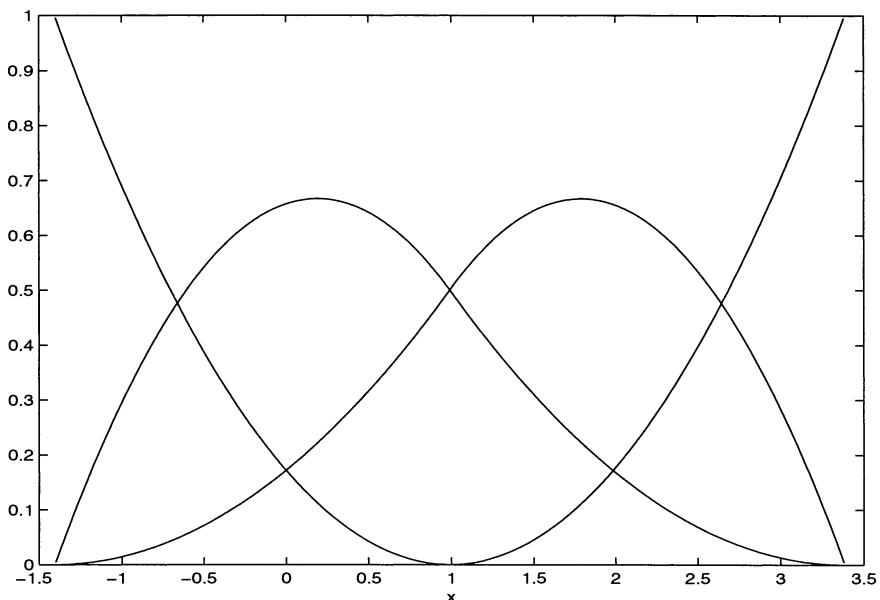


Fig. 8.4. B-splines used for 1 -D input space partition by LBFE (third-order) ©1999 IEEE

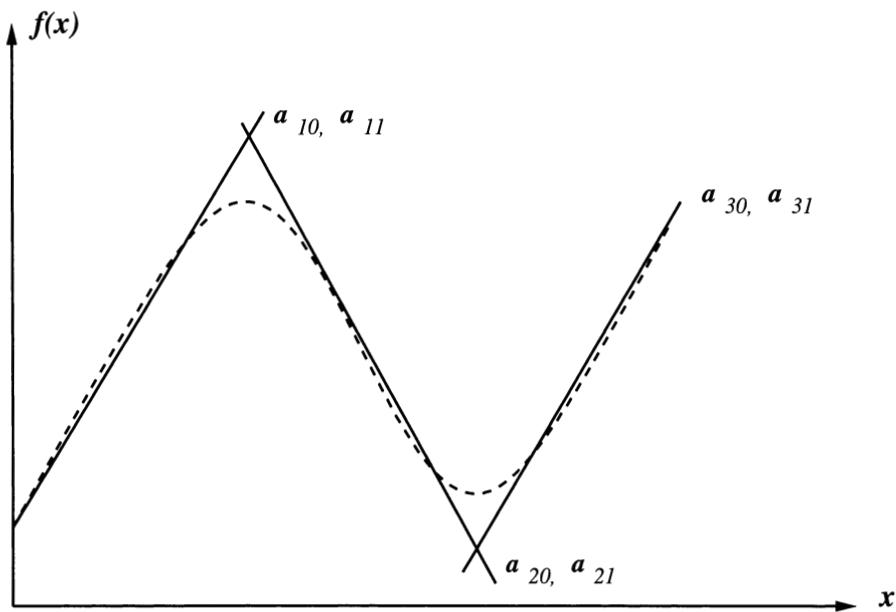


Fig. 8.5. Intuitive description of NFLL modelling ©1999 IEEE

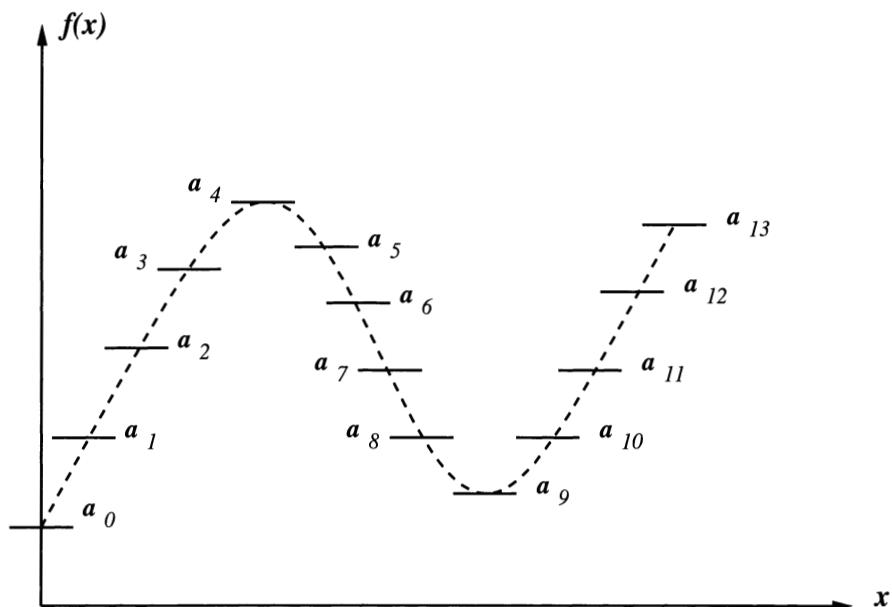


Fig. 8.6. Intuitive description of LBFE modelling ©1999 IEEE

problem and has not been addressed in the ANFIS method. In the MASMOD algorithm the transparency property of B-spline functions is fully employed to obtain automatically a parsimonious network structure.

8.3 A hybrid learning scheme combining MASMOD and EM algorithms for neurofuzzy local linearisation

The conditional probability density of output y given input vector \mathbf{x} can be constructed as follows:

$$p(y|\mathbf{x}) = \sum_{k=1}^K p(k|\mathbf{x}, \mathbf{a}'_k) p(y|\mathbf{x}, \mathbf{a}_k, \Sigma_k), \quad (8.13)$$

where \mathbf{a}'_k , \mathbf{a}_k and Σ_k represent free parameter vectors, K is the number of local probabilistic models, $p(k|\mathbf{x}, \mathbf{a}'_k)$ is the probability that \mathbf{x} belongs to the k th local region of the input space, satisfying the following conditions: $p(k|\mathbf{x}, \mathbf{a}'_k) \geq 0$ and $\sum_{k=1}^K p(k|\mathbf{x}, \mathbf{a}'_k) = 1$, and $p(y|\mathbf{x}, \mathbf{a}_k, \Sigma_k)$ represents the k th local probabilistic model which is usually defined as a Gaussian:

$$p(y|\mathbf{x}, \mathbf{a}_k, \Sigma_k) = \frac{1}{(2\pi)^{1/2} |\Sigma_k|^{1/2}} \exp\left\{-\frac{1}{2}[y - f_k(\mathbf{x}, \mathbf{a}_k)]^T \Sigma_k^{-1}\right\}, \quad (8.14)$$

with a covariance Σ_k and a functional mean $f_k(\mathbf{x}, \mathbf{a}_k) = a_{k0} + \sum_{i=1}^n a_{ki}x_i$.

The probability $p(k|\mathbf{x}, \mathbf{a}'_k)$ in (8.13) plays the role of partitioning the input space and can be designed in various ways. For instance, the following “softmax” function was chosen and studied in the mixture of expert architecture by Jordan et al. [126, 125]:

$$p(k|\mathbf{x}, \mathbf{a}'_k) = g_k(\mathbf{x}, \mathbf{a}'_k) = \exp(\mathbf{a}'_k^T \mathbf{x}) / \left[\sum_{l=1}^K \exp(\mathbf{a}'_l^T \mathbf{x}) \right]. \quad (8.15)$$

Given a training data set $\{\mathbf{x}(t), y(t)\}_{t=1}^N$, the total probability (or likelihood) of the training data set can be calculated as follows:

$$L(\mathbf{w}) = \prod_{t=1}^N p(y(t)|\mathbf{x}(t)) = \prod_{t=1}^N \sum_{k=1}^K g_k(\mathbf{x}(t), \mathbf{a}'_k) p(y(t)|\mathbf{x}(t), \mathbf{a}_k, \Sigma_k), \quad (8.16)$$

where $\mathbf{w} = \{\mathbf{a}'_1, \dots, \mathbf{a}'_K, \mathbf{a}_1, \dots, \mathbf{a}_K, \Sigma_1, \dots, \Sigma_K\}$. Maximum likelihood estimators can be used to identify the parameters in \mathbf{w} by maximising $L(\mathbf{w})$, or more conveniently, by maximising the log likelihood

$$l(\mathbf{w}) = \ln L(\mathbf{w}) = \sum_{t=1}^N \ln \left[\sum_{k=1}^K g_k(\mathbf{x}(t), \mathbf{a}'_k) p(y(t)|\mathbf{x}(t), \mathbf{a}_k, \Sigma_k) \right]. \quad (8.17)$$

If the probabilistic model described by (8.13) has been identified, then the expected value of y given \mathbf{x} can be calculated as follows:

$$\hat{y} = \hat{f}(\mathbf{x}) = E[y|\mathbf{x}] = \int yp(y|\mathbf{x})dy = \sum_{k=1}^K p(k|\mathbf{x}, \mathbf{a}'_k) f_k(\mathbf{x}, \mathbf{a}_k). \quad (8.18)$$

Here, $p(k|\mathbf{x}, \mathbf{a}'_k)$ can be interpreted as a membership function and $f_k(\mathbf{x}, \mathbf{a}_k)$ a local linear model in a probabilistic sense. Note that (8.18) and (8.2) are in a similar form if $p(k|\mathbf{x}, \mathbf{a}'_k)$ are chosen as B-spline functions (B-splines satisfy the conditions imposed on $p(k|\mathbf{x}, \mathbf{a}'_k)$, with \mathbf{a}'_k representing the knot and order parameters). This observation motivates the idea of combining the method used in NFLL modelling for membership function identification with the method for identifying local models in the probabilistic model.

Gradient-descent algorithms may be directly applied to maximise the log likelihood defined in (8.17) for identifying all the free parameters in (8.13). However, in many estimation problems the likelihood is a highly complicated nonlinear function of the parameters, which makes it difficult to obtain the optimal parameters that maximise the likelihood by direct use of the gradient-descent algorithm. The EM algorithm is an alternative approach that has been widely used for maximum likelihood estimation. Instead of maximising the observational data log likelihood, the EM algorithm constructs a new likelihood function by augmenting the observational data with a set of additional variables, which is easier to be maximised than the observational data likelihood. In the EM algorithm, the observational data set, denoted as \mathbf{Y} , is referred to as “incomplete-data”. A “complete-data” set, denoted as $\mathbf{Z} = \{\mathbf{Y}, \mathbf{Y}_{mis}\}$, is constructed by augmenting \mathbf{Y} with a set of “missing” variables \mathbf{Y}_{mis} . The resulting “complete-data” log likelihood is given by $l_c(\mathbf{w}) = \ln p(\mathbf{Y}, \mathbf{Y}_{mis}|\mathbf{w})$, with $p(\mathbf{Y}, \mathbf{Y}_{mis}|\mathbf{w})$ satisfying the following condition:

$$p(\mathbf{Y}|\mathbf{w}) = \int p(\mathbf{Y}, \mathbf{Y}_{mis}|\mathbf{w})d\mathbf{Y}_{mis}. \quad (8.19)$$

Note that the “complete-data” log likelihood is a function of the “missing” random variables \mathbf{Y}_{mis} , and cannot be maximised directly. The strategy of the EM algorithm is to maximise the conditionally expected log likelihood: $E_{\mathbf{Y}_{mis}}[\ln p(\mathbf{Y}, \mathbf{Y}_{mis}|\mathbf{w})|\mathbf{Y}, \mathbf{w}^{(\kappa)}]$, where $\mathbf{w}^{(\kappa)}$ represents the current estimate of \mathbf{w} .

The Expectation-Maximisation Algorithm

The algorithm consists of two steps:

1. The expectation (E) step, which calculates the following conditional expectation of the log likelihood given \mathbf{Y} and $\mathbf{w}^{(\kappa)}$:

$$\begin{aligned} Q(\mathbf{w}|\mathbf{w}^{(\kappa)}) &= E_{\mathbf{Y}_{mis}}[\ln p(\mathbf{Y}, \mathbf{Y}_{mis}|\mathbf{w})|\mathbf{Y}, \mathbf{w}^{(\kappa)}] \\ &= \int \ln p(\mathbf{Y}, \mathbf{Y}_{mis}|\mathbf{w})p(\mathbf{Y}_{mis}|\mathbf{Y}, \mathbf{w}^{(\kappa)})d\mathbf{Y}_{mis}, \end{aligned} \quad (8.20)$$

where $\mathbf{w}^{(\kappa)}$ is the estimate of the parameter vector \mathbf{w} at iteration κ .

2. The maximisation (M) step, which computes

$$\mathbf{w}^{(\kappa+1)} = \arg \max_{\mathbf{w}} Q(\mathbf{w}|\mathbf{w}^{(\kappa)}). \quad (8.21)$$

Obviously, the difficulty in using the EM algorithm lies in the selection of the “missing” variables and the definition of the “complete-data” log likelihood. Jordan and Xu [125] chose the following set of indicator random variables as the “missing” data: $\mathbf{Y}_{mis} = \{I_k^{(t)}, k = 1, 2, \dots, K; t = 1, 2, \dots, N\}$, with

$$I_k^{(t)} = \begin{cases} 1 & \text{if } y(t) \text{ is generated from the } k\text{th local model,} \\ 0 & \text{otherwise,} \end{cases}$$

and defined the “complete-data” likelihood as follows:

$$p(\mathbf{Y}, \mathbf{Y}_{mis}|\mathbf{w}) = \prod_{t=1}^N \prod_{k=1}^K [g_k(\mathbf{x}(t), \mathbf{a}'_k)p(y(t)|\mathbf{x}(t), \mathbf{a}_k, \Sigma_k)]^{I_k^{(t)}}. \quad (8.22)$$

Combining (8.20) and (8.22), we have

$$\begin{aligned} & Q(\mathbf{w}|\mathbf{w}^{(\kappa)}) \\ &= E_{\mathbf{Y}_{mis}} [\ln p(\mathbf{Y}, \mathbf{Y}_{mis}|\mathbf{w})|\mathbf{Y}, \mathbf{w}^{(\kappa)}] \\ &= \sum_{t=1}^N \sum_{k=1}^K h_k^{(\kappa)}(t) \ln [g_k(\mathbf{x}(t), \mathbf{a}'_k)p(y(t)|\mathbf{x}(t), \mathbf{a}_k, \Sigma_k)], \end{aligned} \quad (8.23)$$

with

$$\begin{aligned} h_k^{(\kappa)}(t) &= E[I_k^{(t)}|\mathbf{Y}, \mathbf{w}^{(\kappa)}] = p(k|\mathbf{x}(t), y(t)) \\ &= \frac{g_k(\mathbf{x}(t), \mathbf{a}'_k)p(y(t)|\mathbf{x}(t), \mathbf{a}_k^{(\kappa)}, \Sigma_k^{(\kappa)})}{\sum_{l=1}^K g_l(\mathbf{x}(t), \mathbf{a}'_l^{(\kappa)})p(y(t)|\mathbf{x}(t), \mathbf{a}_l^{(\kappa)}, \Sigma_l^{(\kappa)})}. \end{aligned} \quad (8.24)$$

Comparing (8.23) and (8.17), we note that the log of the sum of functions in (8.17) has turned into the sum of log functions in (8.23). This makes the partial derivatives of $Q(\mathbf{w}|\mathbf{w}^{(\kappa)})$ with respect to the free parameters simpler than the partial derivatives of $l(\mathbf{w})$. However, it is proved [53] that an iteration of the EM algorithm not only increases the “complete-data” log likelihood $l_c(\mathbf{w})$ but also increases the “incomplete-data” log likelihood $l(\mathbf{w})$, that is, $l_c(\mathbf{w}^{(\kappa+1)}) \geq l_c(\mathbf{w}^{(\kappa)})$ leads to $l(\mathbf{w}^{(\kappa+1)}) \geq l(\mathbf{w}^{(\kappa)})$.

From $\partial Q/\partial \Sigma_k = 0$ and $\partial Q/\partial \mathbf{a}_k = 0$, equations for updating the parameters Σ_k and \mathbf{a}_k can be obtained explicitly as follows:

$$\Sigma_k^{(\kappa+1)} = \frac{\sum_{t=1}^N \{h_k^{(\kappa)}(t)[y(t) - f_k(\mathbf{x}(t), \mathbf{a}_k^{(\kappa)})]^2\}}{\sum_{t=1}^N h_k^{(\kappa)}(t)}, \quad (8.25)$$

$$\begin{aligned} \mathbf{a}_k^{(\kappa+1)} &= [\sum_{t=1}^N h_k^{(\kappa)}(t) X^{(t)} (\Sigma_k^{(\kappa)})^{-1} X^{(t)T}]^{-1} \\ &\quad \times \sum_{t=1}^N h_k^{(\kappa)}(t) X^{(t)} (\Sigma_k^{(\kappa)})^{-1} y(t), \end{aligned} \quad (8.26)$$

where $X^{(t)} = [\mathbf{x}(t)^T \ 1]^T$.

The equation for updating the parameters \mathbf{a}'_k can not be explicitly obtained from $\partial Q/\partial \mathbf{a}'_k = 0$, because the “softmax” function is highly nonlinear in \mathbf{a}'_k . The updating of \mathbf{a}'_k has to be iterative and improper learning gain rates may result in the divergence of the EM algorithm. To resolve these problems, Gan and Harris [74] use B-splines instead of the “softmax” functions to design $p(k|\mathbf{x}, \mathbf{a}'_k)$, since B-splines have high degree of flexibility, good interpolation properties, and desirable numerical properties such as transparency and a partition of unity which makes $p(k|\mathbf{x}, \mathbf{a}'_k)$ satisfy $p(k|\mathbf{x}, \mathbf{a}'_k) \geq 0$ and $\sum_{k=1}^K p(k|\mathbf{x}, \mathbf{a}'_k) = 1$. Moreover, the MASMOD algorithm is available not only for identifying the B-splines but also for determining automatically the hyperparameter K , while keeping the EM equations (8.25) and (8.26) for identifying the local models unchanged.

The hybrid learning scheme which combines the MASMOD and EM algorithms can be summarised as follows [74].

The Hybrid Learning Modelling Algorithm

1. Begin with an empty model or a model of minimum size. Initialise all the free model parameters.
2. Use the MASMOD algorithm to enlarge or prune the model by fuzzy input space partitioning and to determine the knot positions and orders of the corresponding B-splines.
3. Compute parameters in the local probabilistic models, i.e. \mathbf{a}_k and Σ_k , by the EM algorithm.
4. Compute performance indexes. If the pre-set values are satisfied, exit. Otherwise, go to 2.

This learning scheme uses a growing-pruning scheme to design the parameters in membership functions (or gating functions), instead of using gradient-descent algorithms. The EM algorithm is only used to design the parameters in local models, in which there is basically no problem of divergence. The above hybrid learning scheme not only results in an NLL model, but also provides covariance information about the model mismatch which is helpful for the consequent estimator or controller design.

8.4 Neurofuzzy feedback linearisation (NFFL)

Feedback linearisation using neural networks has attracted much attention in recent years. Sastry and Isidori [173] used a linear-in-parameters network model to approximate the uncertain nonlinear functions in original state coordinates and developed a normalised gradient-type parameter update law, which was able to yield bounded tracking if some constraints on the nonlinear system were satisfied. French and Rogers [72] generalised the result

of Sastry and Isidori to the system with bounded output disturbances. If the nonlinear functions are uncertain or unknown, high-order Lie derivatives have to be calculated based upon the approximate models of the nonlinear functions. The calculation produces very heavy computational load and, like the calculation of ordinary derivatives based upon approximate function models, it may be not accurate enough [77] to guarantee the convergence of the parameter update and the adaptive control. To some extent, the results of feedback linearisation based on the neural network approximation of the nonlinear functions in the system's original coordinates can be viewed as just existence theorems, and further work is required to make these results more realistic [72]. Alternatively, a lot of effort has been put into approximating directly the nonlinear feedback term in the feedback linearised canonical form. Both multilayer neural networks [43, 44, 119, 120, 130, 136, 209] and linear-in-parameters networks [127, 131, 143, 144, 145, 146, 172] have been applied to approximate the nonlinear functions in the canonical form. Neural network weight tuning algorithms have been developed and their convergence and stability analysed. However, this approach assumes that the state vector in the canonical form is observable, or the original nonlinear system can be described by a nonlinear ARMA model which can be converted into a canonical state-space form with delayed outputs as the states, i.e. the underlying nonlinear systems should have the following formulation:

$$\begin{aligned} x_1(t+1) &= x_2(t) \\ &\dots \\ x_{n-1}(t+1) &= x_n(t) \\ x_n(t+1) &= f[\mathbf{x}(t), \mathbf{w}_f] + \sum_j g_j[\mathbf{x}(t), \mathbf{w}_{g_j}] u_j(t) + d(t) \\ y(t) &= x_1(t), \end{aligned} \tag{8.27}$$

where $\mathbf{x}(t)$ is an observable state vector or a delayed output vector, \mathbf{w}_f and \mathbf{w}_{g_j} are free parameters, and $d(t)$ represents a bounded disturbance. The reality is that the state vector in the feedback linearised canonical form is often in a transformed form and is usually unobservable even though the state vector in the original form is observable. This is because the state coordinate transformation is not known *a priori* or cannot be correctly constructed if the original system is unknown or uncertain.

There exist well-developed theories on the transformation of continuous-time nonlinear affine systems into the following affine normal form by state-feedback linearisation:

$$\dot{\mathbf{z}} = A_c \mathbf{z} + B_c \mathbf{r}(\mathbf{z}, \mathbf{u}), \tag{8.28}$$

$$\mathbf{y} = C_c \mathbf{z}, \tag{8.29}$$

with

$$\mathbf{z} = T(\mathbf{x}), \tag{8.30}$$

$$\mathbf{r}(\mathbf{z}, \mathbf{u}) = \mathbf{q}(\mathbf{z}) + \mathbf{S}(\mathbf{z})\mathbf{u}, \quad (8.31)$$

where matrices A_c , B_c and C_c are in a canonical form, $T(\cdot)$ and $\mathbf{r}(\cdot)$ represent the state coordinate transformation and the feedback control law respectively, $\mathbf{q}(\cdot)$ and $\mathbf{S}(\cdot)$ are smooth functions and $\mathbf{S}(\mathbf{z}) \neq \mathbf{0}$. Most of the research work [43, 44, 71, 72, 77, 119, 120, 130, 136, 172, 173, 179, 193, 209] on feedback linearisation using neural networks has been focused on systems in this normal form or its discrete-time form (8.27). However, there is some work which attempts to extend the results to more complex systems (continuous-time systems only). The strict feedback system or parametric-pure-feedback form [71, 127] is one of them, in which the following continuous-time SISO system is considered:

$$\begin{aligned} \dot{z}_1 &= z_2 + r_1(z_1, z_2, \mathbf{w}_1) \\ \dot{z}_2 &= z_3 + r_1(z_1, z_2, z_3, \mathbf{w}_2) \\ &\dots \\ \dot{z}_{n-1} &= z_n + r_{n-1}(z_1, \dots, z_n, \mathbf{w}_{n-1})u \\ y &= z_1, \end{aligned} \quad (8.32)$$

where $r_i(\cdot)$ represent feedback terms and \mathbf{w}_i are free parameters. In general, $\mathbf{z} = T(\mathbf{x})$ represents a state vector in the transformed coordinates. Some physical systems, such as induction motors and batch bioreactors, can be represented directly in this form, i.e. with $T(\cdot)$ as an identity.

It is also noteworthy that in recent years output-feedback linearisation using adaptive algorithms or linear-in-the-parameters neural networks has made considerable progress [43, 44, 71, 72, 119, 120, 127, 130, 131, 136, 143, 144, 145, 146, 172, 209]. In output-feedback linearisation the following continuous-time SISO systems are often considered:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}) + \mathbf{q}_0(\mathbf{x}, u) + \sum_{i=1}^p w_i \mathbf{q}_i(\mathbf{x}, u) \\ y &= h(\mathbf{x}). \end{aligned} \quad (8.33)$$

All the functions in (8.33) are smooth and known *a priori*, and there is only parametric uncertainty in $\{w_i\}$. For the purpose of adaptive control, the functions in (8.33) are further restricted to be linear in the control signal u . Similarly, if some conditions imposed on $\mathbf{f}(\cdot)$, $\mathbf{q}_i(\cdot)$ and $h(\cdot)$ are satisfied, by using a state coordinate transformation and a feedback control law, (8.33) can be converted into the following canonical form [143, 146]:

$$\begin{aligned} \dot{\mathbf{z}} &= A_c \mathbf{z} + \psi_0(y, u) + \sum_{i=1}^p w_i \psi_i(y, u) \\ y &= C_c \mathbf{z}, \end{aligned} \quad (8.34)$$

where, $\mathbf{z} = T(\mathbf{x})$ represent the state vector in the transformed coordinates, A_c and C_c are in canonical form and $\psi_i(\cdot)$ are smooth function vectors. Compared

with the state-feedback linearisation, the output-feedback term $\psi_0(y, u) + \sum_{i=1}^p w_i \psi_i(y, u)$ has some special properties. It is a function of observed input and output $\{u, y\}$, and there only exists uncertainty in the parameters $\{w_i\}$.

So far the application of neural networks in feedback linearisation is limited to the identification of uncertain parameters or weights, with fixed basis functions or preselected neural network structures. Currently all the results of feedback linearisation using neural networks are based on methods which assume the following:

- The state vector in the canonical form is observable or the state coordinate transformation can be correctly constructed.
- The basis functions in the linear-in-the-parameters network are known *a priori* or the structure of the multi-layered network is known or preselected, that is, there exists only parametric uncertainty.

These assumptions restrict the practical application of feedback linearisation techniques using neural networks from important industrial applications. In the reported applications of feedback linearisation using neural networks, the systems studied so far are restricted to those that can be converted into canonical forms without state coordinate transformation or just by using intuitive linear transformations.

Gan and Harris [77] developed a recurrent neurofuzzy network structure and its learning scheme based on the MASMOD algorithm developed previously for neurofuzzy local linearisation. It can be viewed as a method for approximate feedback linearisation, or neurofuzzy feedback linearisation, with the following assumptions:

- The original output equation or measurement process is linear (it is reasonable to assume the sensors have been designed or can be calibrated to be linear).
- The state coordinate transformation for obtaining the feedback linearised canonical form is unknown but linear.
- The nonlinear functions are totally unknown, i.e. both parameters (weights) and basis functions are to be identified.

These assumptions are reasonable because, first, there exist a number of realistic processes whose measurement equations are linear; second, although the state coordinate transformation constructed theoretically using Lie derivatives is usually nonlinear, it is only one of the methods for constructing state coordinate transformations and it cannot exclude the existence of linear state coordinate transformations; and third, recurrent neural networks have a good capability of approximating unknown nonlinear dynamic processes [123, 182]. Obviously, this new method will be applicable to a larger class of nonlinear dynamic systems.

Many practical nonlinear systems have linear and *a priori* known measurement processes, for instance, some vehicle tracking systems. Such systems are

often transformable into a feedback linearised canonical form by a linear state coordinate transformation, see for example Marino and Tomei [144, 145]. Although the linear state coordinate transformation is usually unknown *a priori*, it is possible to identify the transformation by neural learning algorithms. Assume the state coordinate transformation is linear, i.e. $\mathbf{z} = P\mathbf{x}$, with P representing a nonsingular matrix. Then the feedback linearised canonical form described by (8.28) and (8.29) can be reformed and represented in a discrete-time form as follows, with the state vector in original coordinates:

$$\mathbf{x}(t+1) = A\mathbf{x}(t) + B\mathbf{r}[P\mathbf{x}(t), \mathbf{u}(t), \alpha], \quad (8.35)$$

$$\mathbf{y}(t) = C\mathbf{x}(t), \quad (8.36)$$

with

$$\mathbf{r}[P\mathbf{x}(t), \mathbf{u}(t), \alpha] = [r_1(\mathbf{x}(t), \mathbf{u}(t), \alpha_1), \dots, r_L(\mathbf{x}(t), \mathbf{u}(t), \alpha_L)]^T, \quad (8.37)$$

where, A and B include information about the state coordinate transformation and are generally *a priori* unknown, $r_l(\cdot)$ represent the feedback terms which include both unknown basis functions and unknown parameter vectors α_l . (8.35) can be viewed as a general recurrent neural network (RNN) structure, which is capable of approximating a large class of nonlinear dynamic systems and can be trained using various learning algorithms (see Chapter 3), or such as back-propagation through time and real-time-recurrent-learning [94, 123, 160, 162, 182, 201, 203, 204, 205]. Note that in (8.35) the nonlinear system is approximated by a feedforward linear model plus a feedback nonlinear error model. If the nonlinearity in the system is not too strong, a deliberately designed learning algorithm can keep $\partial\mathbf{r}/\partial\mathbf{x}$ small. This makes some linear techniques for state estimation and control applicable to the feedback linearisation form (8.35) and (8.36).

Actually, all of the feedback linearisation canonical forms can be viewed as the special cases of recurrent neural networks [130]. In the previous investigations on the neural network approximation of nonlinear functions in the canonical form, the state coordinate transformation has not been considered, or it has been assumed that the original system can be directly represented in a canonical form without state coordinate transformation. Considering this fact, (8.35) and (8.36) can be viewed as an extension of the previous work on feedback linearisation using neural networks. The extension includes:

- Matrices A, B, C are not necessarily in canonical form and they are identified through learning.
- Both the neurofuzzy network structure and free parameters are also adaptively identified by learning algorithms.

This means that the feedback linearisation form (8.35) and (8.36), with its state vector in the original coordinates, can represent a larger class of nonlinear systems than the previously studied feedback linearised canonical form. Additionally, because the state vector in (8.35) and (8.36) is in its

original coordinates, it is easy to obtain training data for neural network learning.

Assuming the measurement equation is *a priori* known, we focus on the identification of (8.35), which can be reformulated as follows:

$$x_i(t+1) = \sum_{j=1}^n a_{ij} x_j(t) + \sum_{l=1}^L b_{il} \cdot r_l[\mathbf{x}(t), \mathbf{u}(t), \alpha_l], \quad i = 1, 2, \dots, n, \quad (8.38)$$

with the feedback terms represented in a local basis function expansion:

$$r_l[\mathbf{x}(t), \mathbf{u}(t), \alpha_l] = \alpha_{l0} + \sum_{k=1}^K \alpha_{lk} \cdot N_{lk}[\mathbf{x}(t), \mathbf{u}(t)], \quad (8.39)$$

where $N_{lk}[\mathbf{x}(t), \mathbf{u}(t)]$ are B-spline basis functions and α_{lk} are free parameters. For the purpose of adaptive construction of the B-spline basis functions and in order to combat the curse of dimensionality problem which is associated with any lattice based approximation expansion, (8.39) is decomposed into the following ANOVA form:

$$r_l[\mathbf{x}(t), \mathbf{u}(t), \alpha_l] = \alpha_{l0} + \sum_{s=1}^S \sum_{k=1}^{K_s} \alpha_{lk}^{(s)} \cdot N_{lk}^{(s)}[\mathbf{x}_s(t), \mathbf{u}_s(t)], \quad (8.40)$$

where, s is an index of submodels, \mathbf{x}_s and \mathbf{u}_s represent subspaces of \mathbf{x} and \mathbf{u} respectively. From (8.40) it can be seen that a model with high-dimensional input space has been decomposed into additive submodels with low-dimensional input subspaces. In this way, the number of free parameters used in the whole model are dramatically reduced, since usually the following inequality holds: $K >> \sum_{s=1}^S K_s$.

The partitioning of the input space by submodels and the further partitioning of each submodel by B-spline knots can be automatically carried out by the MASMOD algorithm, which gives a special preference to univariate or bivariate submodels in (8.40). By combining (8.38)–(8.40), we obtain

$$\begin{aligned} x_i(t+1) &= \sum_{q=1}^{M_q} w_{iq} \varphi_q[\mathbf{x}(t), \mathbf{u}(t)] = \mathbf{w}_i^T \varphi[\mathbf{x}(t), \mathbf{u}(t)] \\ &= \mathbf{w}_i^{(0)T} \varphi^{(0)}[\mathbf{x}(t)] + \mathbf{w}_i^{(1)T} \varphi^{(1)}[\mathbf{x}(t), \mathbf{u}(t)], \end{aligned} \quad (8.41)$$

where $\varphi[\mathbf{x}(t), \mathbf{u}(t)] = \varphi(t) = [\varphi^{(0)}(t); \varphi^{(1)}(t)] = [1, x_1(t), \dots, x_n(t); N_1[\mathbf{x}(t), \mathbf{u}(t)], \dots, N_{M_q-n}[\mathbf{x}(t), \mathbf{u}(t)]]^T$ can be regarded as a regressor vector, with $N_q[\mathbf{x}(t), \mathbf{u}(t)]$ corresponding to a particular B-spline function $N_{lk}^{(s)}[\mathbf{x}_s(t), \mathbf{u}_s(t)]$ in the s th submodel, and $\mathbf{w}_i = [\mathbf{w}_i^{(0)}; \mathbf{w}_i^{(1)}] = [w_{i0}, w_{i1}, \dots, w_{in}; w_{i(n+1)}, \dots, w_{iM_q}]^T$ is a free parameter vector. Therefore, the identification of (8.35) turns out to be a generalised linear regression problem. Note that the first $n+1$ regressors are linear and fixed, and the remaining regressors are B-spline functions which can be automatically constructed using the MASMOD algorithm, i.e. the number of submodels, the number of B-splines required in each

submodel and their positions and shapes (B-spline order) are automatically determined by the algorithm. The construction process begins with the basic regressor vector $\varphi^{(0)}[\mathbf{x}(t)] = [1, x_1(t), \dots, x_n(t)]^T$, and the values of parameter vectors $\mathbf{w}_i^{(0)} = [w_{i0}, w_{i1}, \dots, w_{in}]^T$ are obtained by a least squares algorithm. Using the model mismatch data at this stage, the MASMOD algorithm then carries out the following model refinement based on a criterion of structural risk minimisation: adding new univariate B-splines, inserting new knots in the existing B-splines, forming tensor product of the existing univariate B-splines, or pruning knots and B-splines if they are redundant. To illustrate the difference in the applications of the MASMOD algorithm in neurofuzzy local linearisation (NFFL) models, neurofuzzy feedback linearisation (NFLF) models and local basis function expansion (LBFE) models, Figure 8.7 gives details of the kernel block in Figure 8.1 about how to calculate the B-spline matrix, the coefficient vector, the function estimation vector and the performance index.

In general, the free parameters in (8.35) or (8.41), including the components in matrix A , are time-varying. Therefore, the model adaptation is usually divided into two sessions. In the training session, the free parameter vectors \mathbf{w}_i are off-line identified by the least squares algorithm simultaneously with the B-spline structure construction. After the training session, with the constructed B-splines fixed, the parameters should be continuously adapted on-line by the errors from state estimation or output tracking using the following normalised least mean squares algorithm (see Section 3.4):

$$\begin{aligned} \Delta \hat{\mathbf{w}}_i(t+1) &= \hat{\mathbf{w}}_i(t+1) - \hat{\mathbf{w}}_i(t) = -\beta \frac{\varepsilon(t+1)}{c + \varphi^T(t)\varphi(t)} \frac{\partial^+ \varepsilon}{\partial \hat{\mathbf{w}}_i(t)} \\ \varepsilon(t+1) &= x_i(t+1) - \hat{\mathbf{w}}_i^T(t)\varphi(t), \end{aligned} \quad (8.42)$$

where $\beta \in (0, 2)$, c is an arbitrarily small positive constant. Due to the recurrence in the network connections, the gradient needs to be calculated in the “real-time-recurrent-learning” fashion [203, 204], i.e. $\frac{\partial^+ \varepsilon}{\partial \hat{\mathbf{w}}_i(t)}$ should be ordered partial derivatives.

8.5 Formulation of neurofuzzy state estimators

The previous sections have presented algorithms for identifying neurofuzzy local and feedback linearisation (NFLF and NFFL) models of unknown nonlinear processes based on observational data. This section expounds methods for NFLF or NFFL model based state estimation of nonlinear processes which are generally described by the following nonlinear discrete-time state-space model:

$$\mathbf{x}(t) = \mathbf{f}[\mathbf{x}(t-1), \mathbf{u}(t)] + \mathbf{v}(t), \quad (8.43)$$

$$\mathbf{y}(t) = \mathbf{h}[\mathbf{x}(t)] + \xi(t), \quad (8.44)$$

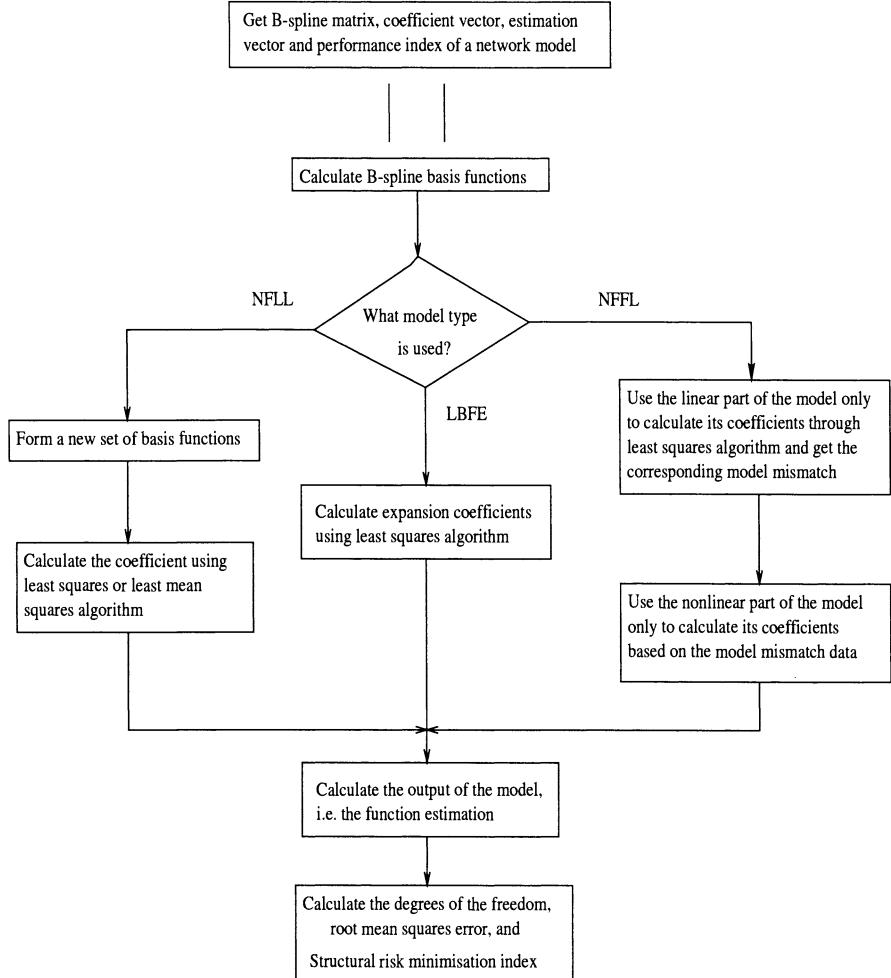


Fig. 8.7. Block diagram 2 of the MASMOD algorithm ©1999 IEEE

where $\mathbf{x}(t) = [x_1(t) \dots x_n(t)]^T$ is a state vector, $\mathbf{u}(t) = [u_1(t) \dots u_r(t)]^T$ is an input vector, $\mathbf{y}(t) = [y_1(t) \dots y_m(t)]^T$ is an observation vector, $\mathbf{f}[\cdot]$ is an unknown smooth nonlinear function vector describing the state transition process, $\mathbf{h}[\cdot]$ is a smooth function vector describing the measurement process, $\mathbf{v}(t)$ and $\xi(t)$ represent additive Gaussian noise, with $E[\mathbf{v}(t)] = 0$, $E[\xi(t)] = \mathbf{0}$, $E[\mathbf{v}(i)\mathbf{v}^T(j)] = \delta_{ij}Q(i)$, $E[\xi(i)\xi^T(j)] = \delta_{ij}R(i)$, and $E[\mathbf{v}(i)\xi^T(j)] = \mathbf{0}$. By using the NFLL modelling methods based on observational data, a nonlinear state-space model, which is *a priori* unknown, can be transformed into a time-varying linear state-space model.

Let $\mathbf{x}'(t) = [\mathbf{x}^T(t-1) \ \mathbf{u}^T(t)]^T$ and $\hat{\mathbf{f}}[\mathbf{x}(t-1), \ \mathbf{u}(t)] = \hat{\mathbf{f}}[\mathbf{x}'(t)] = [\hat{f}_1(\mathbf{x}') \ \hat{f}_2(\mathbf{x}') \ \dots \ \hat{f}_n(\mathbf{x}')]^T$. Based on the NFLL modelling formulation, $\hat{f}_l(\mathbf{x}')$,

$l = 1, 2, \dots, n$, can be modelled as follows:

$$\begin{aligned}\hat{f}_l(\mathbf{x}') &= \sum_{k'=1}^P N_{k'}(\mathbf{x}') [a_{k'l1}x_1(t-1) + \dots + a_{k'ln}x_n(t-1) \\ &\quad + b_{k'l1}u_1(t) + \dots + b_{k'l_r}u_r(t)] \\ &= \alpha_{l1}x_1(t-1) + \dots + \alpha_{ln}x_n(t-1) \\ &\quad + \beta_{l1}u_1(t) + \dots + \beta_{lr}u_r(t),\end{aligned}\tag{8.45}$$

with

$$\alpha_{li}(t) = \sum_{k'=1}^P N_{k'}(\mathbf{x}') a_{k'l_i}, \quad i = 1, 2, \dots, n\tag{8.46}$$

$$\beta_{lj}(t) = \sum_{k'=1}^P N_{k'}(\mathbf{x}') b_{k'l_j}, \quad j = 1, 2, \dots, r\tag{8.47}$$

$$N_{k'}(\mathbf{x}') = \prod_{i=1}^n B_{i,k_i}(x_i(t-1)) \cdot \prod_{j=1}^r B_{j,k_{n+j}}(u_j(t)),\tag{8.48}$$

where k' corresponds to an ordered sequence $k_1, \dots, k_n, k_{n+1}, \dots, k_{n+r}$ and P represents the total number of the sequences. Based on (8.45)–(8.48), (8.43) can be reformed as

$$\mathbf{x}(t) = A(t)\mathbf{x}(t-1) + B(t)\mathbf{u}(t) + \mathbf{v}'(t),\tag{8.49}$$

where, $A(t) = [\alpha_{li}(t)]_{n \times n}$, $B(t) = [\beta_{lj}(t)]_{n \times r}$, and $\mathbf{v}'(t) \sim N(0, Q'(t))$ represents the original state transition noise plus the error noise introduced by model mismatch.

Equation (8.44) can be reformulated in a similar way. By setting $\hat{\mathbf{h}}[\mathbf{x}(t)] = [\hat{h}_1(\mathbf{x}) \ \hat{h}_2(\mathbf{x}) \ \dots \ \hat{h}_m(\mathbf{x})]^T$, we have

$$\begin{aligned}\hat{h}_q(\mathbf{x}) &= \sum_{k=1}^p N_k(\mathbf{x}) [c_{kq1}x_1(t) + \dots + c_{kqn}x_n(t)] \\ &= \gamma_{q1}(t)x_1(t) + \dots + \gamma_{qn}(t)x_n(t),\end{aligned}\tag{8.50}$$

with

$$\gamma_{ql}(t) = \sum_{k=1}^p N_k(\mathbf{x}) c_{kql}, \quad l = 1, 2, \dots, n\tag{8.51}$$

$$N_k(\mathbf{x}) = \prod_{i=1}^n B_{i,k_i}(x_i(t)),\tag{8.52}$$

where k corresponds to an ordered sequence k_1, \dots, k_n and p is the total number of the sequences. Now (8.44) can be reformed as

$$\mathbf{y}(t) = C(t)\mathbf{x}(t) + \xi'(t),\tag{8.53}$$

where $C(t) = [\gamma_{ql}(t)]_{m \times n}$ and $\xi'(t) \sim N(0, R'(t))$ includes the observation noise and the error noise resulting from model mismatch.

Equations (8.45)–(8.48) and (8.50)–(8.52) can be further decomposed into ANOVA forms, and the model structure (number of B-splines and their knot positions, etc.) and free parameters can be identified by the MASMOD algorithm. It is significant to note here that the neurofuzzy linearisation modelling techniques do not require *a priori* process models (as does the extended Kalman filter), but adequate input–output data pairs, D_N , to construct and parameterise the state-space model described by (8.49) and (8.53).

If the underlying nonlinear function vector $f(\mathbf{x}, \mathbf{u})$ is approximately linear over part of an operating regime, then the coefficients in $A(t)$ will change slowly in the operating regime and conventional linear techniques for state estimation and control are directly applicable.

It is possible to convert (8.35) and (8.36) into a canonical form if matrices A, B, C have been identified and satisfy the conditions of system observability [87]. Based on the resultant feedback linearised canonical form, adaptive observers can be designed using a method similar to that used in [143, 146], in which the gain with a fixed value should be deliberately designed to ensure the convergence. However, note that the feedback terms $r_l(\mathbf{x}, \mathbf{u}, \alpha_l)$ will change slowly with respect to \mathbf{x} if the nonlinearity in the original system is not too strong, because the feedback term in (8.35) actually approximates the mismatch of the feedforward linear model $A\mathbf{x}(t)$. In this case, the standard Kalman filter algorithm [50], with a recursively adjusted Kalman gain, can be directly applied to the state estimation based on the feedback linearisation model (8.35) and (8.36).

Based on the neurofuzzy local or feedback linearisation models, the following standard Kalman filter algorithm can be used for state estimation [50, 78, 141]:

Prediction:

$$\hat{\mathbf{x}}(t|t-1) = A(t)\hat{\mathbf{x}}(t-1|t-1) + B(t)\mathbf{u}(t), \quad (8.54)$$

$$P(t|t-1) = A(t)P(t-1|t-1)A^T(t) + Q'(t). \quad (8.55)$$

Estimate (Correction):

$$K(t) = P(t|t-1)C^T(t)[C(t)P(t|t-1)C^T(t) + R'(t)]^{-1}, \quad (8.56)$$

$$\hat{\mathbf{x}}(t|t) = \hat{\mathbf{x}}(t|t-1) + K(t)[\mathbf{y}(t) - C(t)\hat{\mathbf{x}}(t|t-1)], \quad (8.57)$$

$$P(t|t) = [I - K(t)C(t)]P(t|t-1), \quad (8.58)$$

where $\hat{\mathbf{x}}(t|t)$ represents the estimate of the state vector $\mathbf{x}(t)$, $P(t|t)$ is the state estimate covariance matrix and $K(t)$ is the Kalman gain matrix.

If we define a so-called information state vector: $\hat{\mathbf{z}}(j|l) \equiv P^{-1}(j|l)\hat{\mathbf{x}}(j|l)$, where $P^{-1}(j|l)$ is called information matrix, the standard Kalman filter can be transformed into the following information form [78, 141]:

Prediction:

$$\begin{aligned}\hat{\mathbf{z}}(t|t-1) &= P^{-1}(t|t-1)A(t)P(t-1|t-1)\hat{\mathbf{z}}(t-1|t-1) \\ &\quad + P^{-1}(t|t-1)B(t)\mathbf{u}(t),\end{aligned}\quad (8.59)$$

$$P(t|t-1) = A(t)P(t-1|t-1)A^T(t) + Q'(t). \quad (8.60)$$

Estimate (Correction):

$$\hat{\mathbf{z}}(t|t) = \hat{\mathbf{z}}(t|t-1) + C^T(t)R'^{-1}(t)\mathbf{y}(t), \quad (8.61)$$

$$P^{-1}(t|t) = P^{-1}(t|t-1) + C^T(t)R'^{-1}(t)C(t). \quad (8.62)$$

The information form of the Kalman filter (or information filter) is functionally equivalent to the standard Kalman filter, but has some computational advantages, especially in multisensor data fusion applications where the innovation covariance matrix $[C(t)P(t|t-1)C^T(t) + R'(t)]$ in the Kalman filter is usually highly dimensional and non-diagonal. The major difference between the standard Kalman filter and the information filter lies in the estimate phase. The estimate phase of the information filter is simpler because the gain matrix $K(t)$ in the standard Kalman filter is more complex than the term $C^T(t)R'^{-1}(t)$ in the information filter, especially in multisensor data fusion applications where the term $[C(t)P(t|t-1)C^T(t) + R'(t)]^{-1}$ in $K(t)$ may become computationally prohibitive.

8.6 An example of nonlinear trajectory estimation

A simulation example of nonlinear trajectory estimation is given in this section to demonstrate the application of the NFLL and NFFL modelling techniques in nonlinear state estimation. The state-space model that is used to generate trajectories is described by the following equations:

$$r(t+1) = \sqrt{[r(t) + Tr(t)]^2 + [Tr(t)\dot{\theta}(t)]^2} + v_1(t), \quad (8.63)$$

$$\theta(t+1) = \theta(t) + \tan^{-1}\left[\frac{Tr(t)\dot{\theta}(t)}{r(t) + Tr(t)}\right] + v_2(t), \quad (8.64)$$

$$\begin{aligned}\dot{r}(t+1) &= \cos[\theta(t+1)]\{\dot{r}(t)\cos[\theta(t)] - r(t)\dot{\theta}(t)\sin[\theta(t)]\} \\ &\quad + \sin[\theta(t+1)]\{\dot{r}(t)\sin[\theta(t)] \\ &\quad + r(t)\dot{\theta}(t)\cos[\theta(t)]\} + v_3(t),\end{aligned}\quad (8.65)$$

$$\begin{aligned}\dot{\theta}(t+1) &= \{\cos[\theta(t+1)]\{\dot{r}(t)\sin[\theta(t)] + r(t)\dot{\theta}(t)\cos[\theta(t)]\} \\ &\quad - \sin[\theta(t+1)]\{\dot{r}(t)\cos[\theta(t)] \\ &\quad - r(t)\dot{\theta}(t)\sin[\theta(t)]\}\}/r(t) + v_4(t)\end{aligned}\quad (8.66)$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} r(t) \\ \theta(t) \\ \dot{r}(t) \\ \dot{\theta}(t) \end{bmatrix} + \begin{bmatrix} \xi_1(t) \\ \xi_2(t) \end{bmatrix} \quad (8.67)$$

where $r(t)$ and $\theta(t)$ represent the range and bearing of a moving target respectively, $\dot{r}(t)$ and $\dot{\theta}(t)$ represent the range velocity and bearing velocity respectively, $y_1(t)$ and $y_2(t)$ are radar observations of the target position, $v_i(t) = N(0, Q_i)$ and $\xi_j(t) = N(0, R_j)$ represent additive Gaussian noise, and T is the sampling period. The model is only used for training data generation and the estimation performance test. The nonlinear process is unknown *a priori* in state estimation. For the sake of easy understanding, Cartesian coordinates (x, y) can be used to describe target positions. With different initial positions, this model can generate trajectories in different directions. In order to make the size of the training data reasonable, the range of the initial state is restricted as follows: $x(0) = -10\text{km}$, $y(0) = 10\text{km}$, $\dot{x}(0) = 0.2 \sim 0.6\text{km/min}$ and $\dot{y}(0) = -0.6 \sim 0.6\text{ km/min}$.

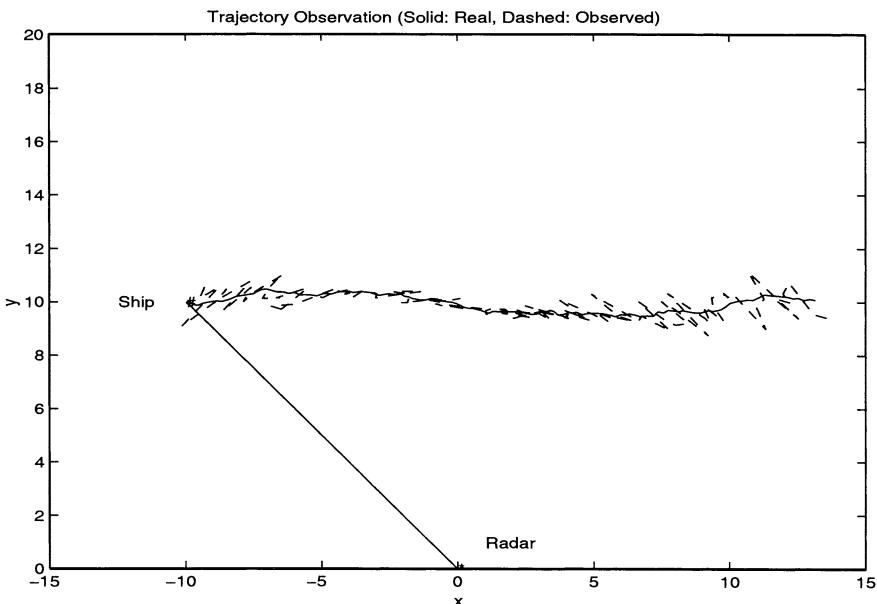


Fig. 8.8. Trajectory 1 and its observation: $x(0) = -10\text{ km}$, $y(0) = 10\text{km}$, $\dot{x}(0) = 0.4\text{ km/min}$, $\dot{y}(0) = 0\text{ km/min}$ ©1999 IEEE

In the simulation, four neurofuzzy feedback linearisation (NFFL) models, each approximating one of the components of the state vector, are first established using the training data. Based on the NFFL models, the standard Kalman filter algorithm is used for state (target trajectory) estima-

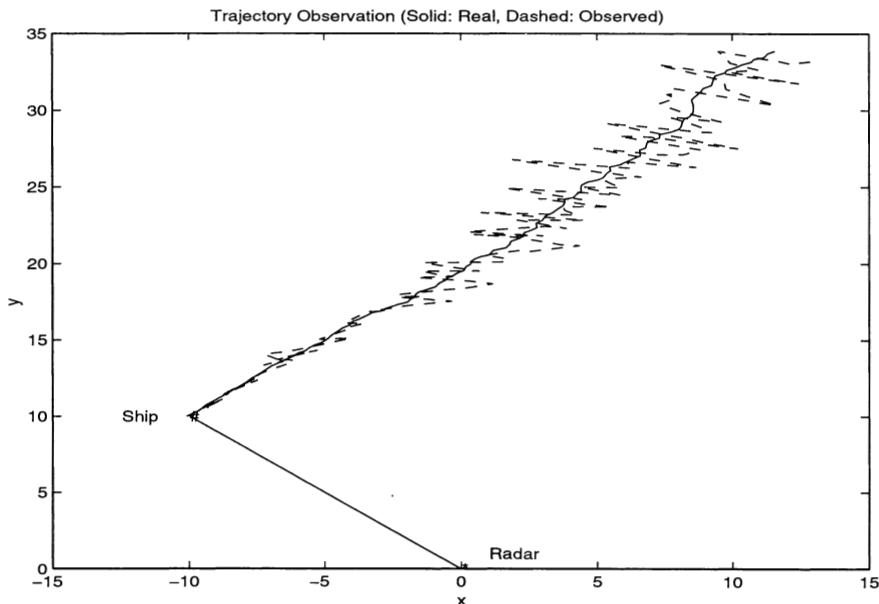


Fig. 8.9. Trajectory 2 and its observation: $x(0) = -10\text{km}$, $y(0) = 10\text{km}$, $\dot{x}(0) = 0.4\text{km/min}$, $\dot{y}(0) = 0.4\text{km/min}$ ©1999 IEEE

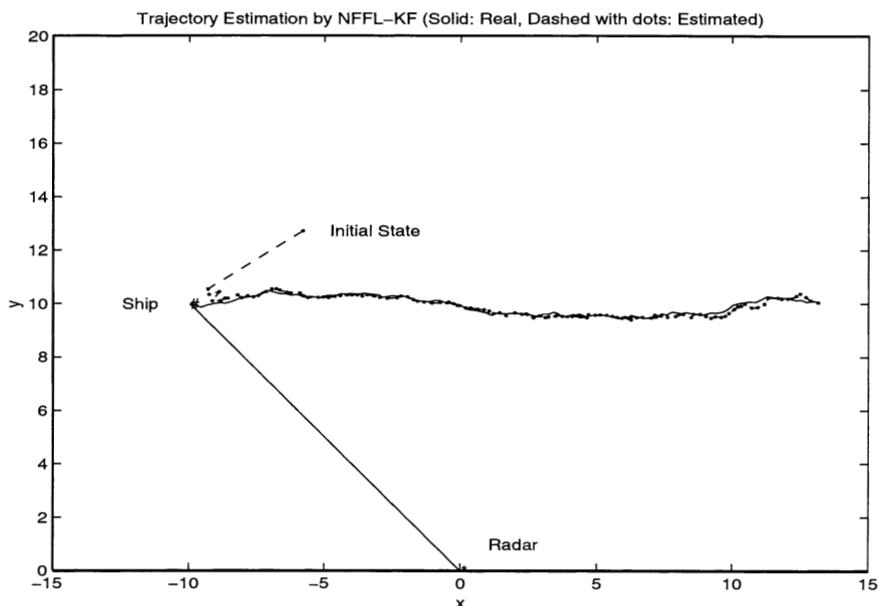


Fig. 8.10. Estimation of trajectory 1 by standard Kalman filter based on NFFL models ©1999 IEEE

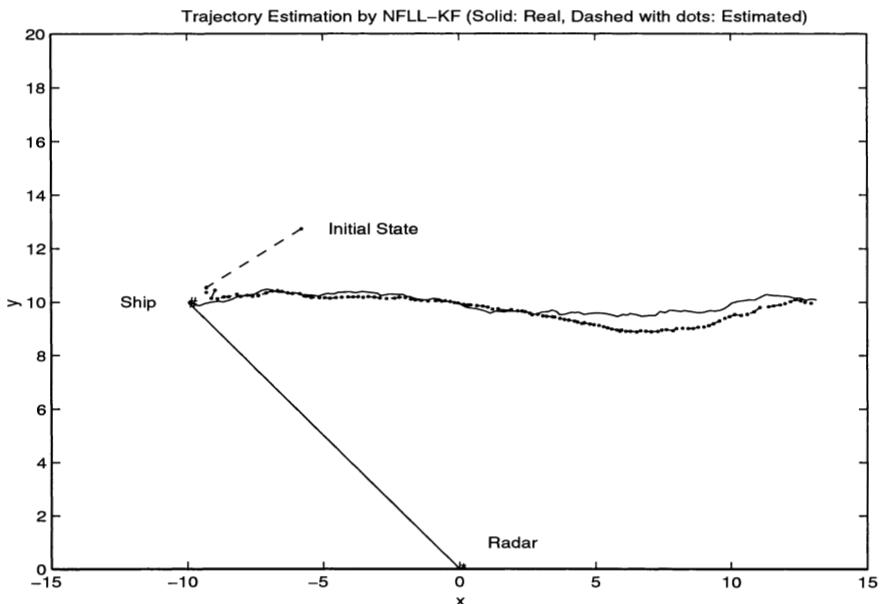


Fig. 8.11. Estimation of trajectory 1 by standard Kalman filter based on NFLL models ©1999 IEEE

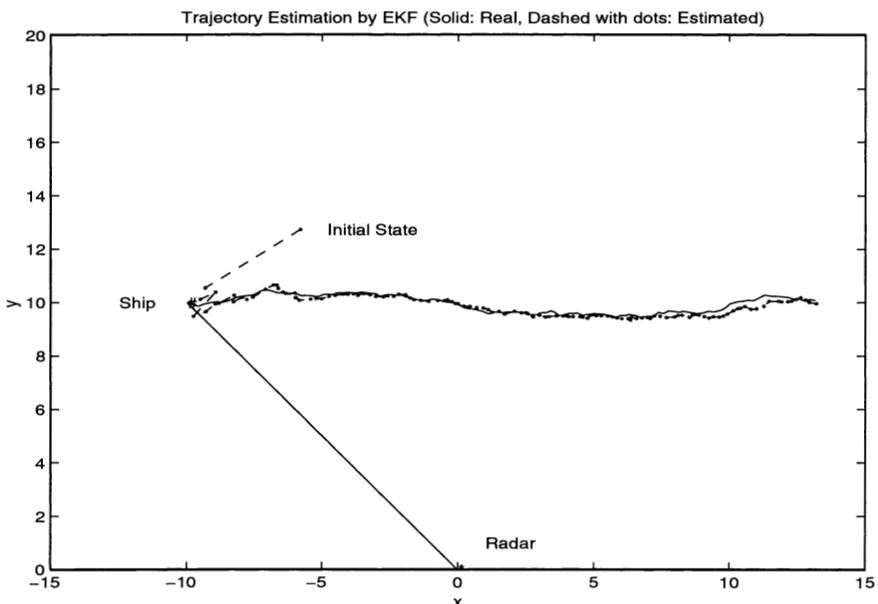


Fig. 8.12. Estimation of trajectory 1 by extended Kalman filter based on *a priori* known nonlinear dynamics ©1999 IEEE

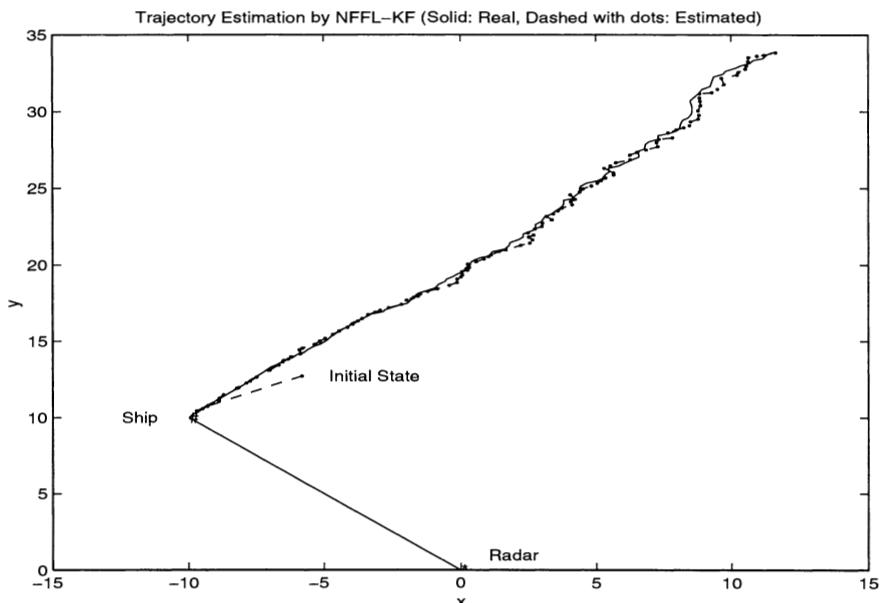


Fig. 8.13. Estimation of trajectory 2 by standard Kalman filter based on NFFL models ©1999 IEEE

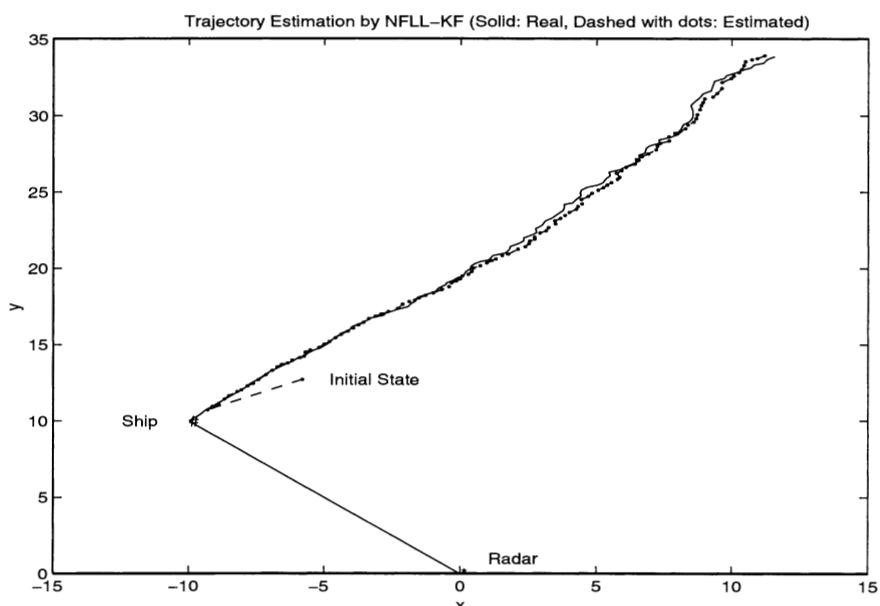


Fig. 8.14. Estimation of trajectory 2 by standard Kalman filter based on NFFL models ©1999 IEEE

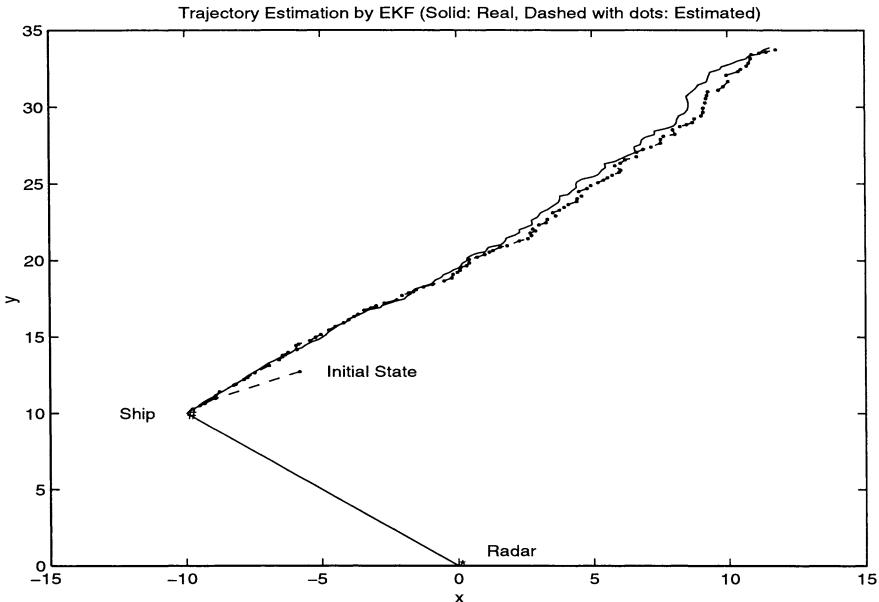


Fig. 8.15. Estimation of trajectory 2 by extended Kalman filter based on *a priori* known nonlinear dynamics ©1999 IEEE

tion, given the noisy observation data. Although the trajectory model is very complex, it is linearisable. The standard Kalman filter based on the NFFL models works very well. For performance comparison, the standard Kalman filter based on NFLL models and the extended Kalman filter based on *a priori* knowledge of the functions in (8.63)–(8.66) are also implemented. Two of the trajectories tested in the simulation, with their noisy observations ($R_1 = 0.01, R_2 = 0.0025$), are shown in Figures 8.8 and 8.9, respectively. The estimation results of trajectory 1 by the standard Kalman filter based on NFFL models, the standard Kalman filter based on the neurofuzzy local linearisation (NFLL) models, and for benchmarking the conventional extended Kalman filter based on *a priori* known model functions are given in Figures 8.10–8.12, respectively. The estimation results of trajectory 2 are given in Figures 8.13, 8.14, and 8.15, respectively. Note that even though the nonlinear process is unknown, the Kalman filters based on NFFL and NFLL models perform as well as or better than the extended Kalman filter which makes use of *a priori* known process functions.

It should be noted that the standard Kalman filter can be directly applicable to NFLL or NFFL models only when the nonlinear process is locally or feedback linearisable. For a process with strong nonlinearities, the standard Kalman filter algorithm may diverge if the model mismatch caused by the linearisation has not been properly taken into account.

9. Multisensor data fusion using Kalman filters based on neurofuzzy linearisation

9.1 Introduction

The previous chapter discussed nonlinear state estimation based on neurofuzzy linearisation models, where available measurements are from a single set of measurements. One of the most important state estimation applications is in target tracking, where nonlinearity and uncertainty are inherently severe. The main purpose of a target tracking system is to obtain a target trajectory, which is more accurate than the direct trajectory observation, and to predict the target behaviour in the near future. In many target tracking problems, in order to obtain more accurate and robust tracking performance, a dynamic target is often detected by multiple disparate sensors, each with different measurement dynamics and noise characteristics. For example, in ship collision avoidance guidance and control, as shown in Figure 9.1, each ship has on-board sensors to measure the positions of itself and other ships or obstacles, and there is a vessel traffic control centre that can also measure ship positions and communicate with each ship. For a particular target (ship or obstacle), there are plenty of associated sensor measurements at different levels and with different accuracy and reliability. Various data fusion problems naturally arise such as how can the multisensor measurements be combined to obtain a joint estimate of the target state vector, which is superior to the individual sensor based estimates and how can the target information be shared robustly from the various data sources (or locations)?

There exist various approaches to multisensor data fusion, of which the Kalman filter or its information form (the information filter, see (8.59)–(8.62)) is one of the most significant and applicable. Methods for Kalman-filter-based data fusion, including measurement fusion and state-vector fusion, have been widely studied over the last decade [11, 12, 13, 14, 41, 42, 58, 75, 78, 88, 141, 165, 166, 167, 168, 169]. As shown in Figure 9.2, measurement fusion methods directly fuse observations or sensor measurements to obtain a weighted or combined measurement and then use a single Kalman filter to obtain the final state estimate based upon the fused measurement. Whereas state-vector fusion methods use a group of local Kalman filters to obtain individual sensor-based state estimates which are then fused to obtain an improved joint state estimate, as shown in Figure 9.3. In practice, measurement fusion methods generally provide better overall estimation performance, whilst state-vector

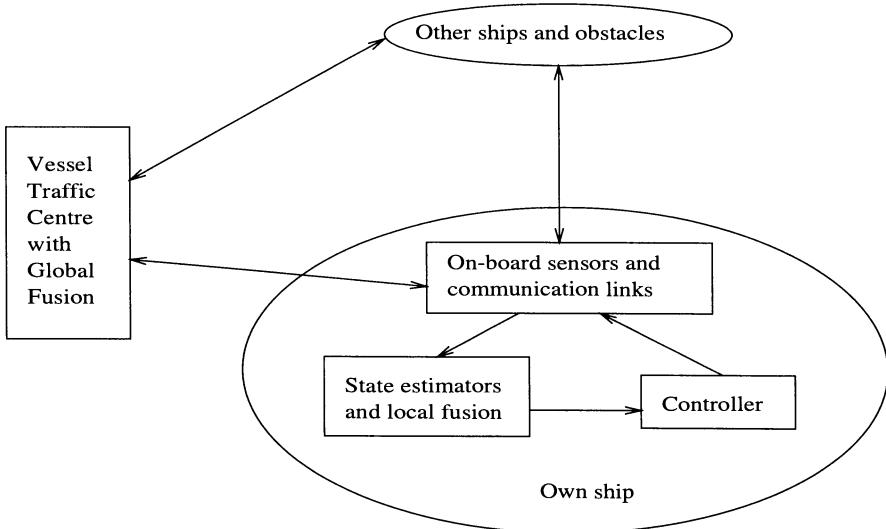


Fig. 9.1. A block diagram of ship collision avoidance guidance and control

fusion methods have a lower computation and communication cost and have the advantages of parallel implementation and fault-tolerance [41, 42, 167]. It is also noteworthy [13] that state-vector fusion methods are effective only when the utilised Kalman filters are consistent. In many practical applications such as target tracking and navigation, the underlying processes are often nonlinear and uncertain, and the consequent Kalman filters are based on approximate linearised process models (by Jacobian linearisation or neurofuzzy linearisation [74, 76, 77, 79] (see also Chapter 8) and will usually be inconsistent due to the model errors introduced by the linearisation process. Therefore, practical applications of state-vector fusion methods are inherently restricted, and for applications of Kalman-filter-based multisensor data fusion techniques, measurement fusion is often preferable to state-vector fusion.

Both of the data fusion architectures shown in Figures 9.2 and 9.3 are centralised. A decentralised data fusion architecture is shown in Figure 9.4, in which the state estimation is carried out locally based on measurement from a single sensor or a subgroup of sensors and data fusion is also conducted locally without any central control. The advantages of the decentralised architecture include robustness, flexibility and scalability. However, the efficiency of the decentralised architecture often depends on its communications mechanism and fusion criteria.

This chapter presents various Kalman-filter-based multisensor data fusion schemes, with focuses on measurement fusion methods and hierarchical data fusion architectures. Two measurement fusion methods are formulated and their performance analytically compared in Section 9.2. In Section 9.3, two

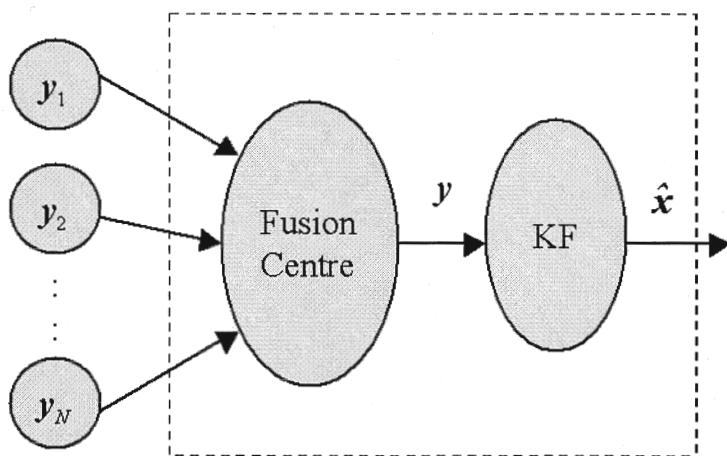


Fig. 9.2. Centralised architecture for measurement fusion ©2001 IEEE

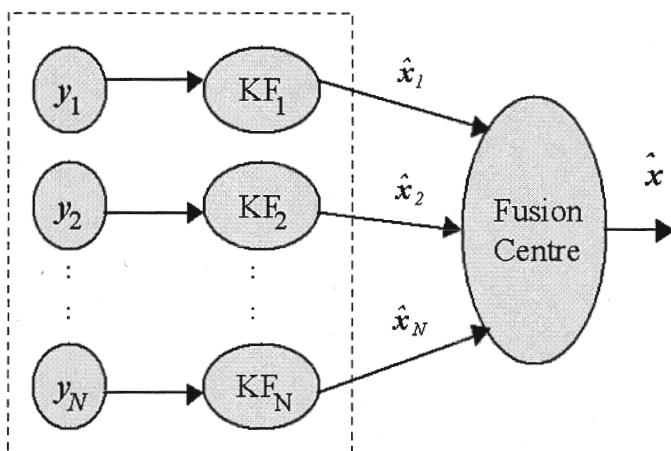


Fig. 9.3. Centralised architecture for state-vector fusion ©2001 IEEE

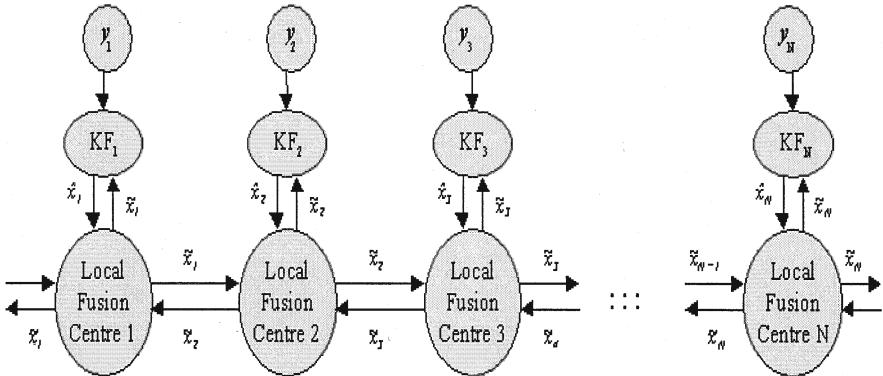


Fig. 9.4. A decentralised data fusion architecture

state-vector fusion methods are formulated in both centralised and decentralised architectures. A hierarchical architecture for multisensor data fusion is proposed in Section 9.4. Some simulation results are given in Section 9.5 to illustrate the various approaches.

9.2 Measurement fusion

Consider a dynamical target which is tracked by N sensors. For simplicity, set $\mathbf{u}(t) = 0$. The target dynamics and the sensors are generally nonlinear, but can be modelled as follows by using linearisation techniques such as neurofuzzy linearisation modelling described in Chapter 8:

$$\mathbf{x}(t) = A(t)\mathbf{x}(t-1) + \mathbf{v}(t), \quad (9.1)$$

$$\mathbf{y}_j(t) = C_j(t)\mathbf{x}(t) + \xi_j(t), \quad j = 1, 2, \dots, N, \quad (9.2)$$

where $\mathbf{x}(t)$ is the state vector, $\mathbf{y}_j(t)$ are measurement vectors from the N sensors, $\mathbf{v}(t)$ and $\xi_j(t)$ are zero-mean white Gaussian noise with covariance matrices $Q(t)$ and $R_j(t)$, respectively. Based on the assumption that the measurement noise is independent, measurement fusion methods are formulated in this section. The formulations for state-vector fusion are given in the next section.

In the formulation of measurement fusion methods, we are concerned about how to combine the observational data such that the N measurement equations described by (9.2) can be transformed into the following single measurement equation:

$$\mathbf{y}(t) = C(t)\mathbf{x}(t) + \xi(t). \quad (9.3)$$

There are two commonly used methods for measurement fusion. The first simply augments the multisensor data [58, 88, 141, 165, 166], which increases the dimension of the observation vector of the Kalman filter, and the second

combines the multisensor data based on minimum-mean-square-error estimates, keeping the observation vector dimension unchanged [167].

9.2.1 Outputs augmented fusion (OAF)

In OAF, sensor measurements are augmented to form the observation information to the Kalman filter in the following way:

$$\mathbf{y}(t) = \mathbf{y}^{(I)}(t) = [\mathbf{y}_1^T(t) \dots \mathbf{y}_N^T(t)]^T, \quad (9.4)$$

$$C(t) = C^{(I)}(t) = [C_1^T(t) \dots C_N^T(t)]^T, \quad (9.5)$$

$$R(t) = R^{(I)}(t) = \begin{bmatrix} R_1(t) & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & R_N(t) \end{bmatrix}. \quad (9.6)$$

It is straightforward to apply the standard Kalman filter or information filter as the global estimator in a centralised data fusion architecture by using (9.4)–(9.6) to obtain fused measurement information.

9.2.2 Optimal weighting measurement fusion (OWMF)

In OWMF, sensor measurements are combined based on a minimum-mean-square-error criteria. By utilising the independence of the measurement noise, the observation information to the Kalman filter is given by [13, 60]:

$$\mathbf{y}(t) = \mathbf{y}^{(II)}(t) = [\sum_{j=1}^N R_j^{-1}(t)]^{-1} \sum_{j=1}^N R_j^{-1}(t) \mathbf{y}_j(t), \quad (9.7)$$

$$C(t) = C^{(II)}(t) = [\sum_{j=1}^N R_j^{-1}(t)]^{-1} \sum_{j=1}^N R_j^{-1}(t) C_j(t), \quad (9.8)$$

$$R(t) = R^{(II)}(t) = [\sum_{j=1}^N R_j^{-1}(t)]^{-1}. \quad (9.9)$$

In OWMF the combined measurements, $\mathbf{y}(t)$, and associated measurement matrix $C(t)$, are the normalised weighted sum of the measurements, and associated measurement matrices, weighted by the inverse of the associated covariance matrices of measurements noise.

Similarly, it is easy to use the standard Kalman filter or information filter for the final state estimation in OWMF.

9.2.3 On functional equivalence of OAF and OWMF

From (9.4)–(9.6) and (9.7)–(9.9), we can see that the treatments in the two measurement fusion methods are quite different. However, it has been demonstrated that there exists a certain form of functional equivalence between the two measurement fusion methods, which can be described by the following theorem [75, 78]:

Theorem 9.1: If the sensors used for data fusion, with different and independent noise characteristics, have identical measurement matrices, i.e. $C_1(t) = C_2(t) = \dots = C_N(t)$, then OAF is functionally equivalent to OWMF.

Proof. The following formula in linear algebra will be used to cope with the inversion of matrices:

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}^{-1} = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} \quad (9.10)$$

$$(A + HBH^T)^{-1} = A^{-1} - A^{-1}H(B^{-1} + H^TA^{-1}H)^{-1}H^TA^{-1}, \quad (9.11)$$

where $B_1 = (A_1 - A_2A_4^{-1}A_3)^{-1}$, $B_2 = -B_1A_2A_4^{-1}$, $B_3 = -A_4^{-1}A_3B_1$, and $B_4 = A_4^{-1} + A_4^{-1}A_3B_1A_2A_4^{-1}$.

If the standard Kalman filter is used, in order to testify the functional equivalence of the two measurement fusion methods we only need to check whether the terms $K(t)C(t)$ and $K(t)\mathbf{y}(t)$ in OAF and OWMF are functionally equivalent, where $K(t)$ represents the Kalman gain. Alternatively, if the information filter is used, then we need to check the terms $C^T(t)R^{-1}(t)C(t)$ and $C^T(t)R^{-1}(t)\mathbf{y}(t)$ of both methods. To avoid confusion, the variables in OAF are indexed by $^{(I)}$ and the variables in OWMF indexed by $^{(II)}$. For simplicity, the time argument in $C_j(t)$ and $R_j(t)$ will be omitted in the following derivation.

First, consider the case when the standard Kalman filter is applied. Because of the complexity of the innovation covariance matrix, the derivation would be complex. Let us begin with sensor-to-sensor fusion, i.e. $N = 2$. Letting $C_1 = C_2 = C_0$ and $\Omega^{(I)} = C_0P^{(I)}(t|t-1)C_0^T$, we obtain the Kalman gain in OAF as follows:

$$\begin{aligned} K^{(I)}(t) &= P^{(I)}(t|t-1)C_0^T[I \ I] \begin{bmatrix} R_1 + \Omega^{(I)} & \Omega^{(I)} \\ \Omega^{(I)} & R_1 + \Omega^{(I)} \end{bmatrix}^{-1} \\ &= P^{(I)}(t|t-1)C_0^T[(R_2 + \Omega^{(I)})^{-1}R_2 \cdot \\ &\quad [R_1 + \Omega^{(I)} - \Omega^{(I)}(R_2 + \Omega^{(I)})^{-1}\Omega^{(I)}]^{-1}, \\ &\quad (R_2 + \Omega^{(I)})^{-1} - (R_2 + \Omega^{(I)})^{-1}R_2 \cdot \\ &\quad [R_1 + \Omega^{(I)} - \Omega^{(I)}(R_2 + \Omega^{(I)})^{-1}\Omega^{(I)}]^{-1}\Omega^{(I)}(R_2 + \Omega^{(I)})^{-1}]. \quad (9.12) \end{aligned}$$

The following equalities can be established [75]:

$$\begin{aligned} & (R_2 + \Omega^{(I)})^{-1} R_2 [R_1 + \Omega^{(I)} - \Omega^{(I)}(R_2 + \Omega^{(I)})^{-1} \Omega^{(I)}]^{-1} \\ & = [\Omega^{(I)} + R_1(R_1 + R_2)^{-1} R_2]^{-1} R_2(R_1 + R_2)^{-1} \end{aligned} \quad (9.13)$$

and

$$\begin{aligned} & (R_2 + \Omega^{(I)})^{-1} - (R_2 + \Omega^{(I)})^{-1} R_2 \cdot \\ & [R_1 + \Omega^{(I)} - \Omega^{(I)}(R_2 + \Omega^{(I)})^{-1} \Omega^{(I)}]^{-1} \Omega^{(I)}(R_2 + \Omega^{(I)})^{-1} \\ & = [\Omega^{(I)} + R_1(R_1 + R_2)^{-1} R_2]^{-1} R_1(R_1 + R_2)^{-1}. \end{aligned} \quad (9.14)$$

Based on (9.12)–(9.14), we have

$$\begin{aligned} K^{(I)}(t) &= P^{(I)}(t|t-1) C_0^T [C_0 P^{(I)}(t|t-1) C_0^T + R_1(R_1 + R_2)^{-1} R_2]^{-1} \cdot \\ & [R_2(R_1 + R_2)^{-1}, R_1(R_1 + R_2)^{-1}] \end{aligned} \quad (9.15)$$

$$\begin{aligned} & K^{(I)}(t) C^{(I)}(t) \\ & = P^{(I)}(t|t-1) C_0^T [C_0 P^{(I)}(t|t-1) C_0^T + R_1(R_1 + R_2)^{-1} R_2]^{-1} C_0 \end{aligned} \quad (9.16)$$

$$\begin{aligned} & K^{(I)}(t) \mathbf{y}^{(I)}(t) \\ & = P^{(I)}(t|t-1) C_0^T [C_0 P^{(I)}(t|t-1) C_0^T + R_1(R_1 + R_2)^{-1} R_2]^{-1} \cdot \\ & [R_2(R_1 + R_2)^{-1} \mathbf{y}_1(t) + R_1(R_1 + R_2)^{-1} \mathbf{y}_2(t)]. \end{aligned} \quad (9.17)$$

If $C_1 = C_2 = C_0$, then $C^{(II)} = C_0$, $R^{(II)} = R_1(R_1 + R_2)^{-1} R_2$, and we obtain the Kalman gain in OWMF as follows:

$$\begin{aligned} & K^{(II)}(t) \\ & = P^{(II)}(t|t-1) C_0^T [C_0 P^{(II)}(t|t-1) C_0^T + R_1(R_1 + R_2)^{-1} R_2]^{-1} \end{aligned} \quad (9.18)$$

$$\begin{aligned} & K^{(II)}(t) C^{(II)}(t) \\ & = P^{(II)}(t|t-1) C_0^T [C_0 P^{(II)}(t|t-1) C_0^T + R_1(R_1 + R_2)^{-1} R_2]^{-1} C_0 \\ & \end{aligned} \quad (9.19)$$

$$\begin{aligned} & K^{(II)}(t) \mathbf{y}^{(II)}(t) \\ & = P^{(II)}(t|t-1) C_0^T [C_0 P^{(II)}(t|t-1) C_0^T + R_1(R_1 + R_2)^{-1} R_2]^{-1} \cdot \\ & [R_2(R_1 + R_2)^{-1} \mathbf{y}_1(t) + R_1(R_1 + R_2)^{-1} \mathbf{y}_2(t)]. \end{aligned} \quad (9.20)$$

Note that (9.16) and (9.19) are in the same form and that (9.17) and (9.20) are also in the same form. Therefore, with the same initial conditions, i.e. $P^{(I)}(0|0) = P^{(II)}(0|0)$ and $\hat{\mathbf{x}}^{(I)}(0|0) = \hat{\mathbf{x}}^{(II)}(0|0)$, the Kalman filter algorithm based on observations described by (9.4)–(9.6) and observations by (9.7)–(9.9) will result in the same state estimate $\hat{\mathbf{x}}(t|t)$. This means that the two measurement fusion methods are functionally equivalent in the sensor-to-sensor case. This result can be generalised into the situation of using N sensors. More conveniently, the generalised result can also be obtained based on the following proof with the information filter and the functional equivalence between the standard Kalman filter and information filter.

Now, consider the case when the information filter is applied. From (9.4) to (9.9), with a centralised data fusion architecture, it is easy to prove the following equalities:

$$[C^{(I)}(t)]^T [R^{(I)}(t)]^{-1} C^{(I)}(t) = \sum_{j=1}^N C_j^T R_j^{-1} C_j, \quad (9.21)$$

$$[C^{(I)}(t)]^T [R^{(I)}(t)]^{-1} \mathbf{y}^{(I)}(t) = \sum_{j=1}^N C_j^T R_j^{-1} \mathbf{y}_j, \quad (9.22)$$

$$\begin{aligned} & [C^{(II)}(t)]^T [R^{(II)}(t)]^{-1} C^{(II)}(t) \\ &= [(\sum_{j=1}^N R_j^{-1})^{-1} \sum_{j=1}^N R_j^{-1} C_j]^T \sum_{j=1}^N R_j^{-1} C_j \end{aligned} \quad (9.23)$$

$$\begin{aligned} & [C^{(II)}(t)]^T [R^{(II)}(t)]^{-1} \mathbf{y}^{(II)}(t) \\ &= [(\sum_{j=1}^N R_j^{-1})^{-1} \sum_{j=1}^N R_j^{-1} C_j]^T \sum_{j=1}^N R_j^{-1} \mathbf{y}_j. \end{aligned} \quad (9.24)$$

If $C_j = C_0$, $j = 1, 2, \dots, N$, then we have

$$[C^{(I)}(t)]^T [R^{(I)}(t)]^{-1} C^{(I)}(t) = [C^{(II)}(t)]^T [R^{(II)}(t)]^{-1} C^{(II)}(t), \quad (9.25)$$

$$[C^{(I)}(t)]^T [R^{(I)}(t)]^{-1} \mathbf{y}^{(I)}(t) = [C^{(II)}(t)]^T [R^{(II)}(t)]^{-1} \mathbf{y}^{(II)}(t). \quad (9.26)$$

Based on the formulation of the information filter (8.59)–(8.62), obviously there exists a functional equivalence between the two measurement fusion methods. \square

The functional equivalence of the two measurement fusion methods is limited to the case when $C_1 = C_2 = \dots = C_N$. If $C_1 \neq C_2 \neq \dots \neq C_N$, then the functional equivalence will no longer hold and there is no consistent conclusion about which method provides a better estimate. This can be easily proved by a particular example, such as the one studied in the simulation of Section 9.5. From the above derivation, we note that output augmented fusion (OAF) is simpler than optimal weighting measurements fusion (OWMF) when the information filter is used and that OWMF is simpler than OAF when the standard Kalman filter is applied.

9.2.4 On the decentralised architecture

In a decentralised architecture, there are local estimators and local fusion centres. A local estimator may be based upon a subgroup of sensors, where measurement fusion may be carried out, but it is nothing more than the

centralised way in a strict sense. A local fusion centre processes state estimates from the corresponding local estimators and other local fusion centres. Therefore, local fusion centres basically conduct state-vector fusion, which is presented in the next section. There exist complex communications problems in decentralised architectures, which are beyond the scope of this chapter.

9.3 State-vector fusion

Most data fusion methods make use of statistical properties, such as covariance, of various data sources to obtain more accurate fused data or information. Measurement fusion methods utilise the covariance matrices of measurement noise. In state-vector fusion, state estimate covariance matrices should be used. However, the state estimates of the same target from different estimators are often dependent, even if the measurement noise of different sensors is independent. Sometimes, the state estimates are inconsistent and the estimate covariance matrices can not reflect real estimation errors. These problems make the design of state-vector fusion more difficult than measurement fusion. Nevertheless, state-vector fusion is amiable to the decentralised architecture and has the advantages of robustness, flexibility, and parallel implementation.

This section introduces two state-vector fusion methods in a centralised way, then discusses decentralised architectures. The first is based on the assimilation of local state estimates [88], and the second is based on the minimisation of mean square errors of local state estimates [11, 12, 13, 14, 41, 42, 167, 168, 169]. It is noteworthy that the first method is not an optimal approach and that the second method is only effective when the local state estimators (e.g. Kalman filters) are consistent.

9.3.1 State-vector assimilation fusion (SVAF)

SVAF, without making use of the statistical properties of the data, is not an optimal method for data combination. It is simply based on the assimilation of data sources. Both the standard Kalman filter and information filter can be used in SVAF. The formulation of SVAF given below is based on information filter [88]:

Local estimate:

$$\hat{\mathbf{z}}_j(t|t) = \hat{\mathbf{z}}_j(t|t-1) + C_j^T(t)R_j^{-1}(t)\mathbf{y}_j(t), \quad (9.27)$$

$$P_j^{-1}(t|t) = P_j^{-1}(t|t-1) + C_j^T(t)R_j^{-1}(t)C_j(t). \quad (9.28)$$

Assimilation fusion:

$$\hat{\mathbf{z}}(t|t) = \hat{\mathbf{z}}(t|t-1) + \sum_{j=1}^N [\hat{\mathbf{z}}_j(t|t) - \hat{\mathbf{z}}(t|t-1)], \quad (9.29)$$

$$P^{-1}(t|t) = P^{-1}(t|t-1) + \sum_{j=1}^N [P_j^{-1}(t|t) - P^{-1}(t|t-1)], \quad (9.30)$$

where $\hat{\mathbf{z}}_j(t|t)$ and $P_j^{-1}(t|t)$, $j = 1, 2, \dots, N$ are local estimates of the information state vector and information matrix, $\hat{\mathbf{z}}(t|t)$ and $P^{-1}(t|t)$ are the fused information state vector and information matrix.

SVAF is a simple method with a low computational cost. It is very suitable for decentralised data fusion implementation, as will be discussed later. However, it does not provide optimal fusion results.

9.3.2 Track-to-track fusion (TTF)

It is not easy to optimally combine the local state estimates because they are usually dependent on each other and sometimes it is difficult to obtain the correct state estimate covariance matrices. The track-to-track fusion (i.e. $N = 2$) based on the standard Kalman filter is a widely studied optimal approach, which is formulated as follows [11, 12, 13, 14, 41, 42, 167, 168, 169]:

$$\hat{\mathbf{x}}(t|t) = \hat{\mathbf{x}}_1(t|t) + P_{xy} P_{yy}^{-1} [\hat{\mathbf{x}}_2(t|t) - \hat{\mathbf{x}}_1(t|t)], \quad (9.31)$$

$$P(t|t) = P_1(t|t) - [P_1(t|t) - P_{12}(t|t)] P_{yy}^{-1} [P_1(t|t) - P_{21}(t|t)], \quad (9.32)$$

$$P_{xy} = P_1(t|t) - P_{12}(t|t), \quad (9.33)$$

$$P_{yy} = P_1(t|t) + P_2(t|t) - P_{12}(t|t) - P_{21}(t|t), \quad (9.34)$$

$$\begin{aligned} & P_{12}(t|t) \\ &= [I - K_1(t)C_1(t)]A(t-1)P_{12}(t-1|t-1)A^T(t-1) \cdot \\ & \quad [I - K_2(t)C_2(t)]^T + [I - K_1(t)C_1(t)]Q(t-1)[I - K_2(t)C_2(t)]^T, \end{aligned} \quad (9.35)$$

$$\begin{aligned} & P_{21}(t|t) \\ &= [I - K_2(t)C_2(t)]A(t-1)P_{21}(t-1|t-1)A^T(t-1) \cdot \\ & \quad [I - K_1(t)C_1(t)]^T + [I - K_2(t)C_2(t)]Q(t-1)[I - K_1(t)C_1(t)]^T, \end{aligned} \quad (9.36)$$

where $\hat{\mathbf{x}}_1(t|t)$ and $\hat{\mathbf{x}}_2(t|t)$ are local state estimates from Kalman filters, $P_1(t|t)$ and $P_2(t|t)$ are covariances of $\hat{\mathbf{x}}_1(t|t)$ and $\hat{\mathbf{x}}_2(t|t)$, respectively, $P_{12}(t|t)$ and $P_{21}(t|t)$ are cross-covariance between $\hat{\mathbf{x}}_1(t|t)$ and $\hat{\mathbf{x}}_2(t|t)$, $\hat{\mathbf{x}}(t|t)$ and $P(t|t)$ are the fused state estimate and the corresponding covariance.

The cross-covariance is the critical issue that influences the accuracy of the fused state estimate, but its computation is very complex. There has been significant research into efficiency and performance evaluation of track-to-track fusion algorithms [42, 167, 168, 169]. In spite of its optimality, there still exists difficulty in the realistic application of TTF.

9.3.3 On the decentralised architecture

SVAF is more suitable for decentralised architectures than TTF, because it does not need the covariance information from local estimators and local fusion centres. In a decentralised architecture, there is no global fusion centre. Each local fusion centre provides information with equal importance. This section only presents formulations of local estimators and local fusion centres based on the information filter and SVAF.

Suppose the i th local fusion centre is connected with the i th local state estimator and other local fusion centres in its neighbour N_i . Then the i th local information state estimator can be described by

$$\tilde{\mathbf{z}}_i(t|t) = \tilde{\mathbf{z}}_i(t|t-1) + C_i^T(t)R_i^{-1}\mathbf{y}_i(t), \quad (9.37)$$

$$P_i^{-1}(t|t) = \tilde{P}_i^{-1}(t|t-1) + C_i^T(t)R_i^{-1}C_i(t), \quad (9.38)$$

where $\tilde{\mathbf{z}}_i$ and \tilde{P}_i^{-1} represent the fused information state vector and the corresponding information matrix from the i th local fusion centre, respectively. The i th local fusion centre is formulated as follows:

$$\tilde{\mathbf{z}}_i(t|t) = \hat{\mathbf{z}}_i(t|t) + \sum_{j \in N_i} [\tilde{\mathbf{z}}_j(t|t) - \tilde{\mathbf{z}}_{ji}(t|t-1)], \quad (9.39)$$

$$\tilde{P}_i^{-1}(t|t) = P_i^{-1}(t|t) + \sum_{j \in N_i} [P_j^{-1}(t|t) - \tilde{P}_{ji}^{-1}(t|t-1)], \quad (9.40)$$

where $\tilde{\mathbf{z}}_{ji}$ and \tilde{P}_{ji}^{-1} represent an additional information filter that is needed to compute information common to the i th and j th local estimators. This additional information filter can be designed as follows [88]:

$$\tilde{P}_{ji}^{-1}(t|t) = P_j^{-1}(t|t) + P_i^{-1}(t|t) - \tilde{P}_{ji}^{-1}(t|t-1), \quad (9.41)$$

$$\tilde{\mathbf{z}}_{ji}(t|t) = \tilde{\mathbf{z}}_j(t|t) + \tilde{\mathbf{z}}_i(t|t) - \tilde{\mathbf{z}}_{ji}(t|t-1), \quad (9.42)$$

$$\tilde{P}_{ji}^{-1}(t|t-1) = [A(t)\tilde{P}_{ji}^{-1}(t-1|t-1)A^T(t) + Q(t)]^{-1}, \quad (9.43)$$

$$\tilde{\mathbf{z}}_{ji}(t|t-1) = \tilde{P}_{ji}^{-1}(t|t-1)A(t)\tilde{P}_{ji}^{-1}(t-1|t-1)\tilde{\mathbf{z}}_{ji}(t-1|t-1). \quad (9.44)$$

Decentralised architectures are highly flexible, which is achieved at the price of complex computation and communications of the information that is common to the connected local estimators and local fusion centres. This chapter will not focus on the details about the analysis of decentralised data fusion architectures.

9.4 Hierarchical multisensor data fusion – trade-off between centralised and decentralised Architectures

Centralised and decentralised data fusion architectures have their own advantages and disadvantages. Centralised architectures are less complex, and easy in managing sensor resources, but are inflexible, and vulnerable to system failure, especially the central processor on which severe computational load are imposed. Alternatively, decentralised architectures have some complementary properties. As a compromise, hierarchical architectures are often used in practice. A hierarchical multisensor data fusion architecture, as shown in Figure 9.5, was developed by the authors for target trajectory tracking [97]. There are local fusion nodes, each having a set of sensors and being able to receive globally fused track feedback, and also a global fusion centre which may have its own sensors and can receive local track estimates or local sensor measurements with related information such as covariance matrices. The globally fused track feedback is utilised at local fusion nodes to improve local estimates.

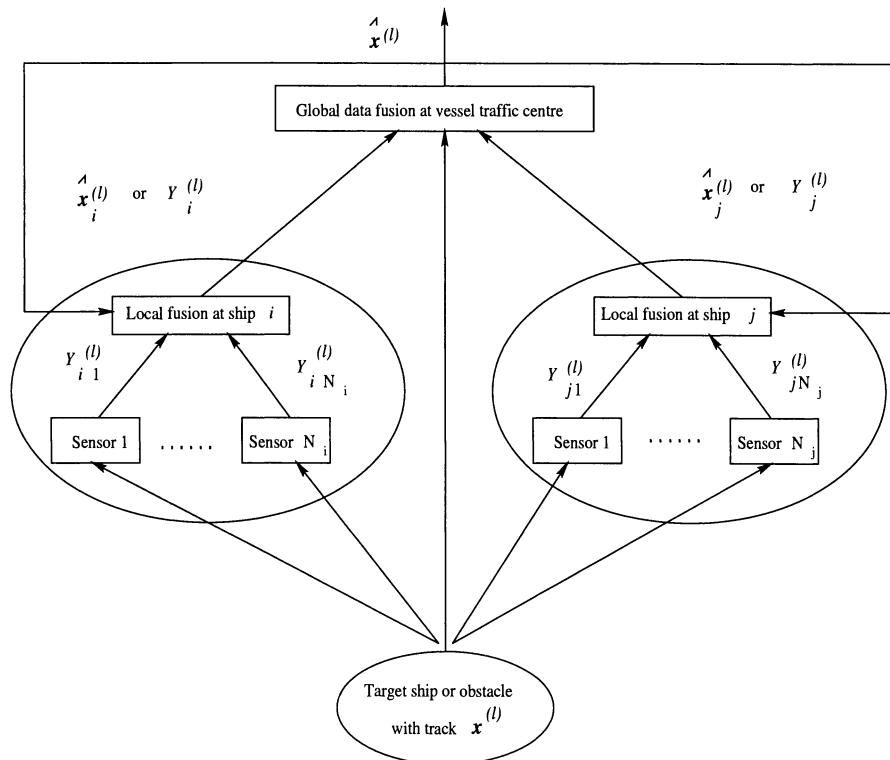


Fig. 9.5. A hierarchical architecture for multisensor data fusion

At the global level, Kalman-filter-based multisensor data fusion algorithms introduced in the previous sections can be applied directly. At local levels, there exist not only local sensor measurements but also globally fused track feedback. As shown in Figure 9.5, for a particular target with track $\mathbf{x}^{(l)}$, at local fusion node j , local sensor measurement vector $\mathbf{Y}_j^{(l)}(t)$, previous local track estimate $\hat{\mathbf{x}}_j^{(l)}(t-1|t-1)$, and globally fused track estimate $\hat{\mathbf{x}}^{(l)}(t-d|t-d)$ (d represents a track feedback delay which depends on feedback rate), and the corresponding covariance matrices are available for updating the local track estimate. If track-to-track fusion methods are directly used at local levels as in [128], the problems such as timing, recounting and cross-correlation have to be carefully considered. To avoid these problems, if $d = 1$, the global track feedback is used to replace the previous local track estimate (including the covariance matrix) for track prediction, which is then corrected by the fused local sensor measurement. If $d > 1$ (this happens when the feedback rate is less than the measurement rate), the previous local track estimate will be used for track prediction until a new global track feedback is received.

9.5 Simulation examples

This section gives two simulation examples. The first is to testify the theoretical result obtained in Section 9.2 on the functional equivalence of the two measurement fusion methods, in which the state estimate covariance matrices from the two measurement fusion methods are compared. The second example, trajectory tracking of a linearisable dynamic process, is designed to demonstrate the performance of the hierarchical multisensor data fusion scheme developed in Section 9.4.

9.5.1 On functional equivalence of two measurement fusion methods

The following target and sensor models have been extensively studied [11, 12, 13, 14, 41, 42, 167, 168, 169] as an example of data fusion:

$$\mathbf{x}(t) = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \mathbf{x}(t-1) + \begin{bmatrix} T^2/2 \\ T \end{bmatrix} v(t), \quad (9.45)$$

$$y_j(t) = C_j \mathbf{x}(t) + \xi_j(t), \quad j = 1, 2, \quad (9.46)$$

where T is the sampling time (it is usually set as 1), $v(t) = N(0, q)$ and $\xi_j(t) = N(0, R_j)$. In [11, 12, 13, 14, 41, 42, 167, 168, 169], simulation studies were carried out for comparing the fused state estimate covariance of a state-vector fusion or measurement fusion method with single-sensor (unfused) state estimate covariance. In the first stage of our simulation studies, some results on the gain of the multisensor data fusion, which are similar to those

reported in [11, 12, 13, 14, 41, 42, 167, 168, 169], are repeated. However, the simulation studies here will focus on the comparison of the fused state estimate covariances of the two measurement fusion methods. Two cases are considered here. In case 1, two sensors have identical measurement matrices, which are set as $C_1 = C_2 = [1 \ 0]$. In case 2, two sensors have different measurement matrices, which are set as $C_1 = [1 \ 0]$ and $C_2 = [1 \ 0.5]$. In both cases, two situations, i.e. $R_1 = R_2$ and $R_1 \neq R_2$, will be considered.

The state-space model described by (9.45)–(9.46) satisfies the conditions under which the fused state estimate covariance $P(t|t)$ will converge to a steady-state value [13], which is denoted here by $\begin{bmatrix} P_{11} & P_{12} \\ P_{12} & P_{22} \end{bmatrix}$. Simulation studies are designed to compare these covariance values from the two measurement fusion methods. Figures 9.6–9.8 illustrates the results in case 1 with $R_1 = R_2 = 1$, where the fused state estimate covariances in steady-state values of the two measurement fusion methods are given with variation of the process noise covariance q from 0.001 to 1000. Curves of P_{11} , P_{12} , and P_{22} versus q are shown in Figures 9.6–9.8, respectively. The results from OAF are in solid lines and those from OWMF are in *dashed lines*. Similar results in case 1, with $R_1 = 1$ and $R_2 = 1.25$, are given in Figures 9.9–9.11 in the same format. From Figures 9.6–9.11, we note that OAF and OWMF obtain the same state estimation results, i.e. they are functionally equivalent as the Theorem 9.1.

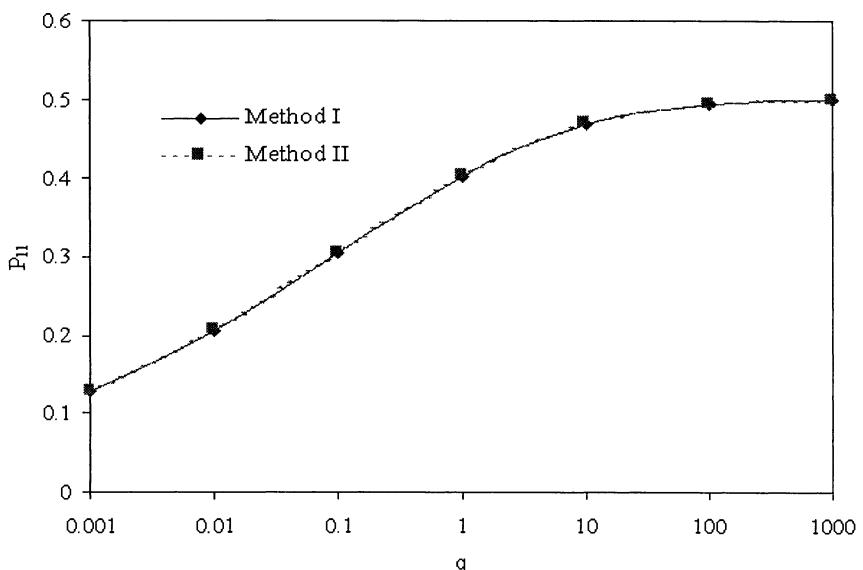


Fig. 9.6. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{11} vs. q , with $C_1 = C_2$, $R_1 = R_2 = 1$ ©2001 IEEE

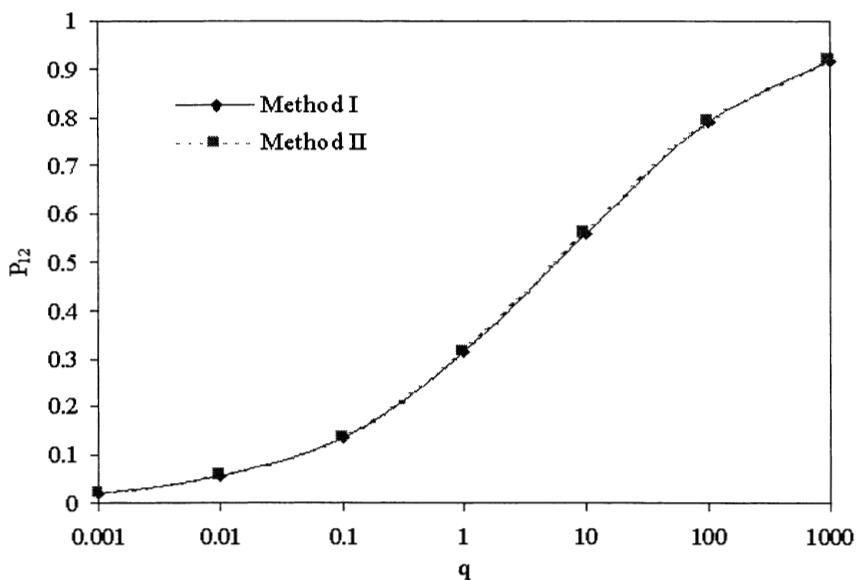


Fig. 9.7. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{12} vs. q , with $C_1 = C_2$, $R_1 = R_2 = 1$ ©2001 IEEE

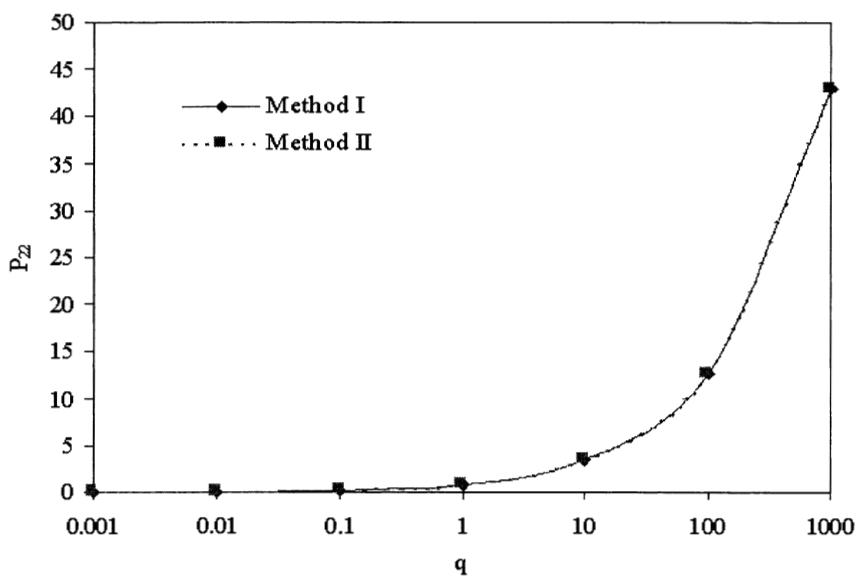


Fig. 9.8. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{22} vs. q , with $C_1 = C_2$, $R_1 = R_2 = 1$ ©2001 IEEE

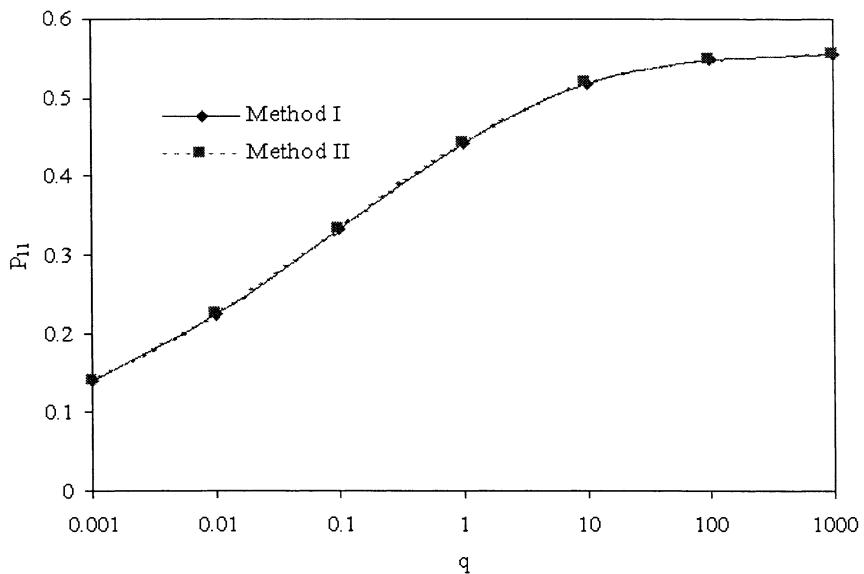


Fig. 9.9. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{11} vs. q , with $C_1 = C_2$, $R_1 = 1$, $R_2 = 1.25$ ©2001 IEEE

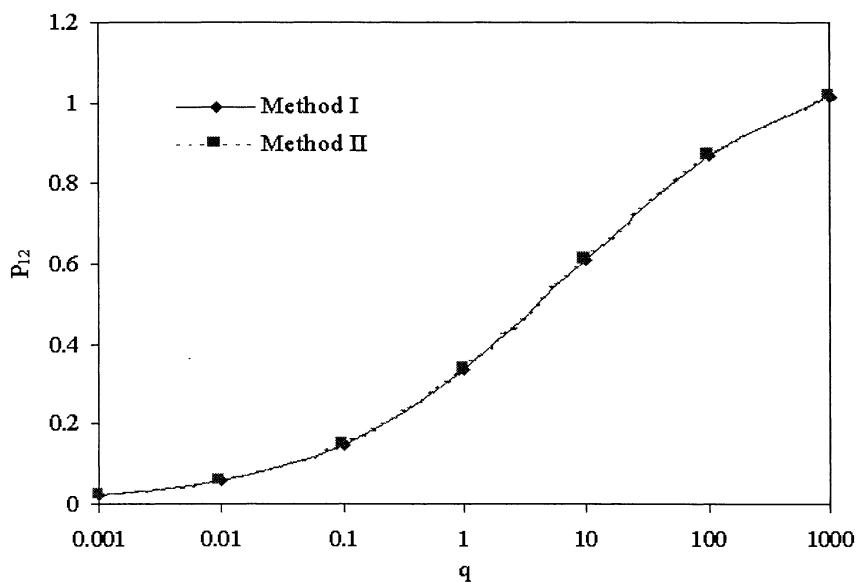


Fig. 9.10. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{12} vs. q , with $C_1 = C_2$, $R_1 = 1$, $R_2 = 1.25$ ©2001 IEEE

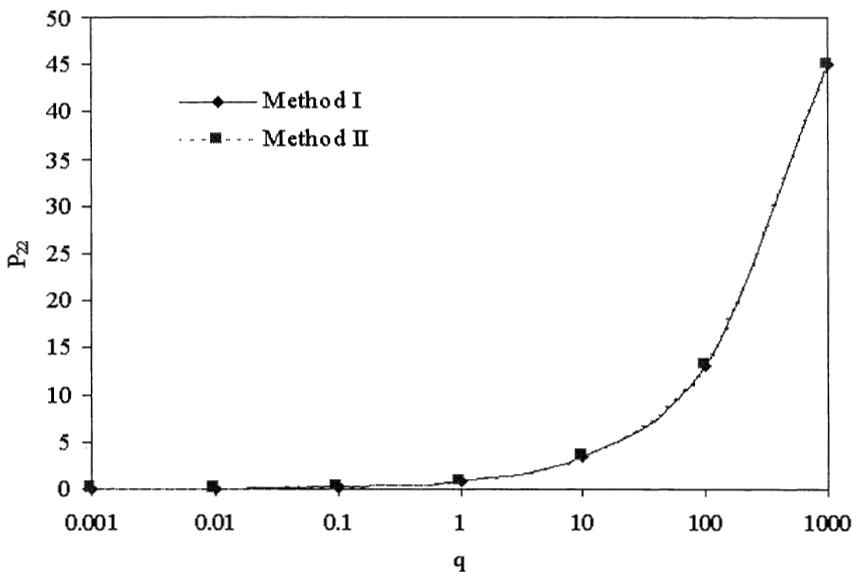


Fig. 9.11. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{22} vs. q , with $C_1 = C_2$, $R_1 = 1$, $R_2 = 1.25$ ©2001 IEEE

In a similar way, the results in case 2 are shown in Figures 9.12–9.14 with $R_1 = R_2 = 1$ and in Figures 9.15–9.17 with $R_1 = 1$ and $R_2 = 1.25$, respectively. Note that in case 2 the results from OAF and OWMF are different, with OAF generally outperforming OWMF. This is because OAF provides complete measurement information to the Kalman filter which has the self-ability of information integration.

9.5.2 On hierarchical multisensor data fusion

The second example uses the same state-space model as described by (8.63)–(8.66) in Section 8.6. For simplicity, we only consider two local nodes, each having one sensor (Sensor 1 and Sensor 2 respectively) to detect the same nonlinear dynamic target and both being able to receive global track feedback. At the global fusion centre, measurements from Sensor 1 and Sensor 2 can be received and fused to obtain a global target track estimate. This is the simplest case described by Figure 9.5.

The sensor models are as follows:

$$\begin{bmatrix} y_{j1}(t) \\ y_{j2}(t) \end{bmatrix} = C_j \begin{bmatrix} r(t) \\ \theta(t) \\ \dot{r}(t) \\ \dot{\theta}(t) \end{bmatrix} + \begin{bmatrix} \xi_{j1}(t) \\ \xi_{j2}(t) \end{bmatrix}, \quad j = 1, 2. \quad (9.47)$$

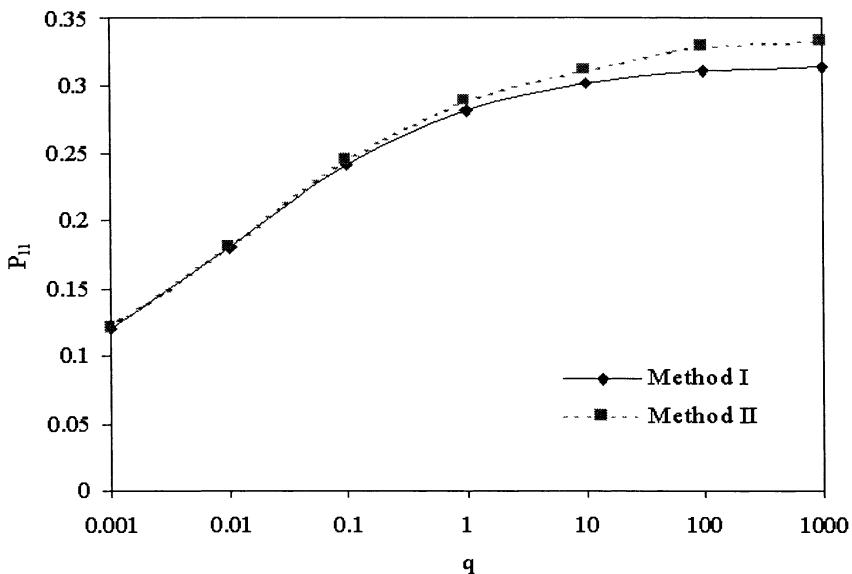


Fig. 9.12. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{11} vs. q , with $C_1 \neq C_2$, $R_1 = R_2 = 1$ ©2001 IEEE

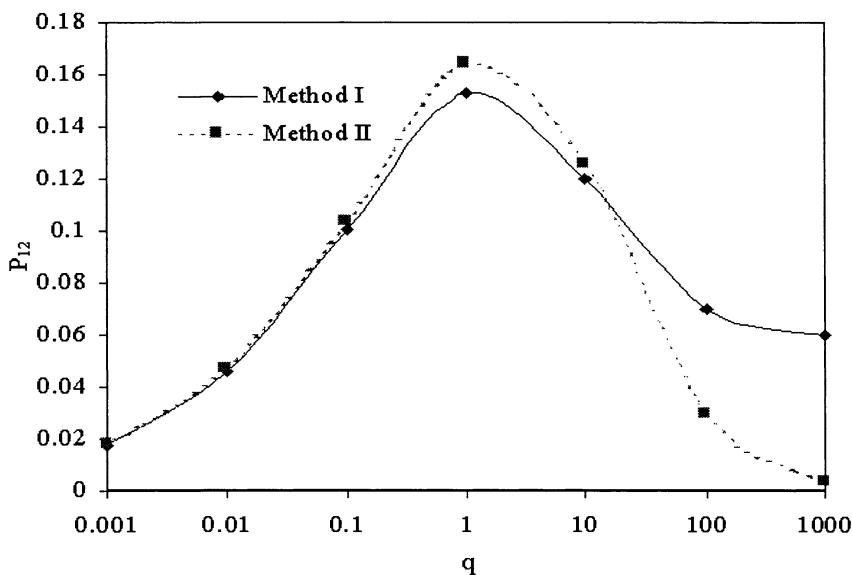


Fig. 9.13. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{12} vs. q , with $C_1 \neq C_2$, $R_1 = R_2 = 1$ ©2001 IEEE

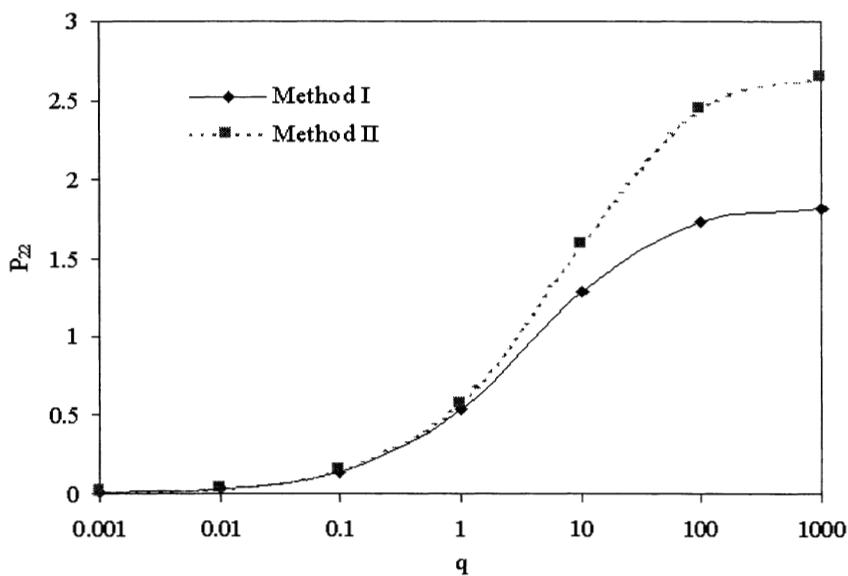


Fig. 9.14. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{22} vs. q , with $C_1 \neq C_2$, $R_1 = R_2 = 1$ ©2001 IEEE

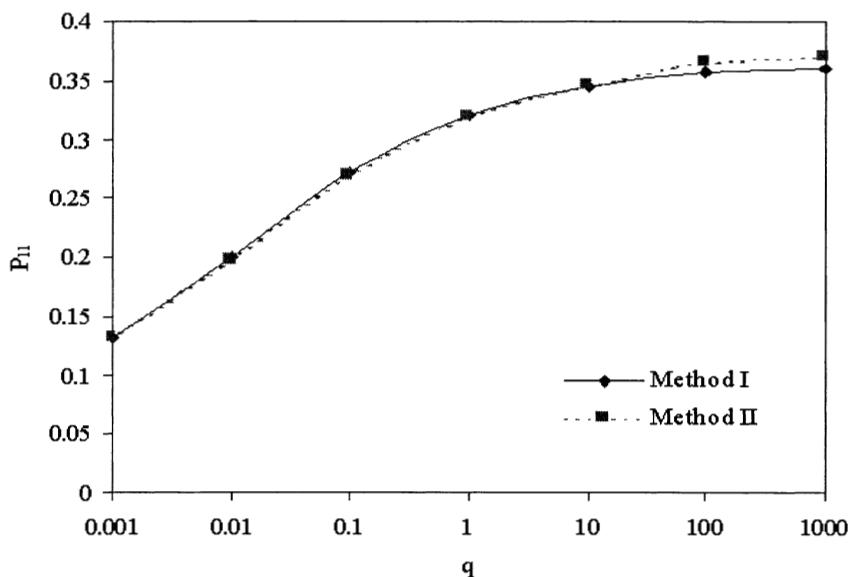


Fig. 9.15. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{11} vs. q , with $C_1 \neq C_2$, $R_1 = 1$, $R_2 = 1.25$ ©2001 IEEE

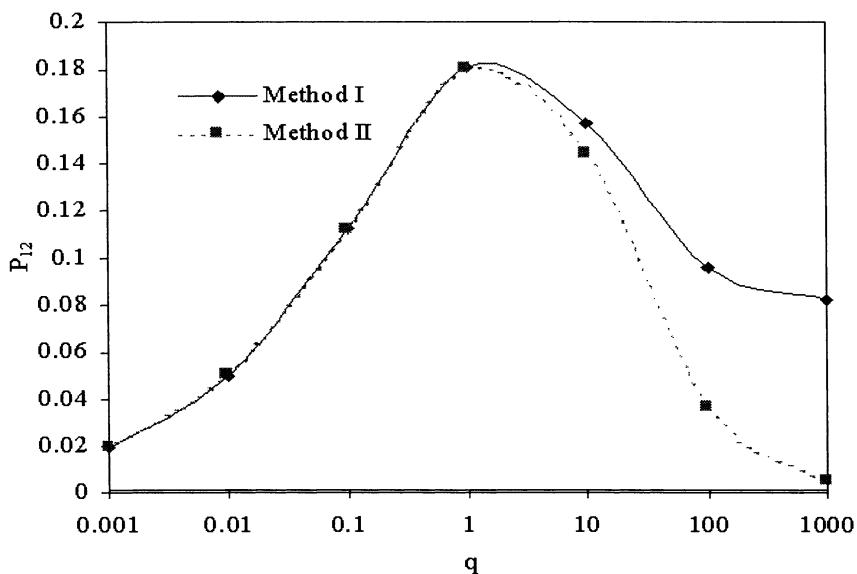


Fig. 9.16. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{12} vs. q , with $C_1 \neq C_2$, $R_1 = 1$, $R_2 = 1.25$ ©2001 IEEE

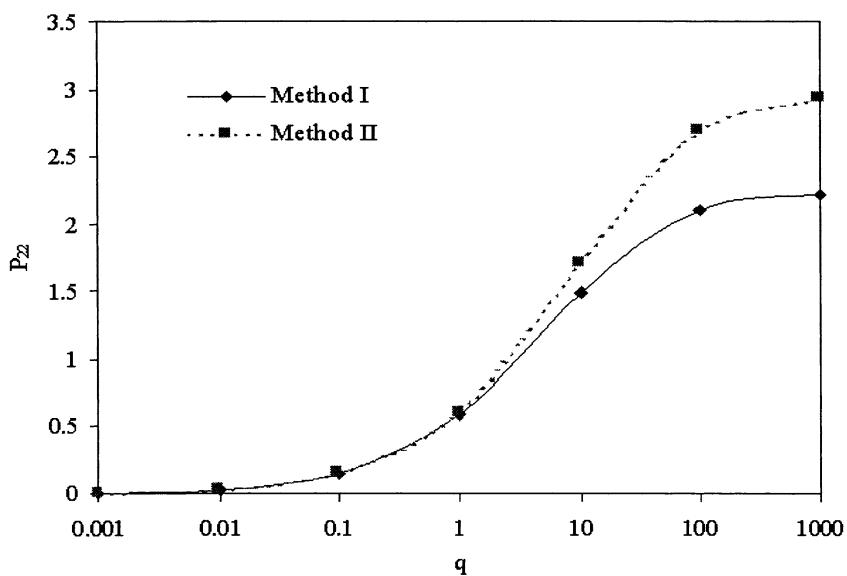


Fig. 9.17. Comparison of the fused state estimate covariances of OAF (Method I) and OWMF (Method II): P_{22} vs. q , with $C_1 \neq C_2$, $R_1 = 1$, $R_2 = 1.25$ ©2001 IEEE

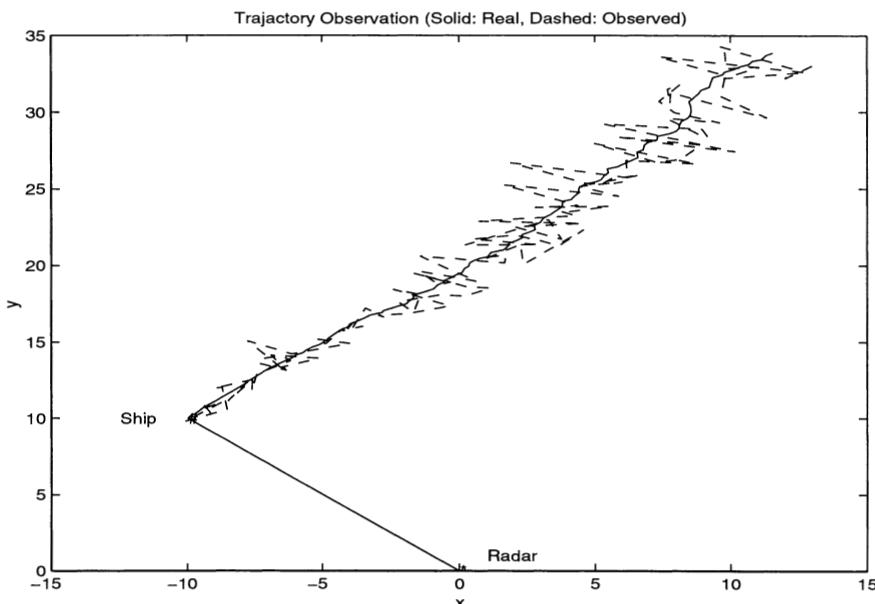


Fig. 9.18. One of the tested trajectories (in *solid line*) and its observation (in *dashed line*) by Sensor 1: $R_{11} = 0.25$, $R_{12} = 0.0025$

Sensor 1 and Sensor 2 only measure the range and bearing of the target. The measurement matrices are C_1 and C_2 , with dimension of 2×4 . The measurement noise covariance matrices are $R_1 = \text{diag}\{R_{11}, R_{12}\}$ and $R_2 = \text{diag}\{R_{21}, R_{22}\}$. One of the trajectories tested in the simulation and its observations with different noise levels are shown in Figure 9.18 (observed by Sensor 1: $R_{11} = 0.25$, $R_{12} = 0.0025$) and Figure 9.19 (observed by Sensor 2: $R_{21} = 0.01$, $R_{22} = 0.0001$), respectively.

The trajectory estimation result using measurement fusion at the global fusion centre is given in Figure 9.20, with the *solid line* and *dashed line* denoting the real trajectory and estimated trajectory, respectively. The trajectory estimation results at the local node with Sensor 1, which are based on the noisy measurement from Sensor 1 and the track feedback, are shown in Figures 9.21–9.23, with variation of the track feedback rate. The ratios of measurement rate to track feedback rate (*m/f ratio*) in Figures 9.21 and 9.22 are 1 and 5, respectively. In Figure 9.23, no track feedback is utilised and the trajectory estimation is based on the measurement from Sensor 1 only, which naturally obtains the worst performance. From Figures 9.20–9.23 we can see that the global measurement fusion achieves the best performance and that the globally fused track feedback is obviously useful in improving the local state estimates. For a numerical comparison, the root mean square errors (RMSEs) of the trajectory estimation results shown in Figures 9.20–9.23 are given in Table 9.1.

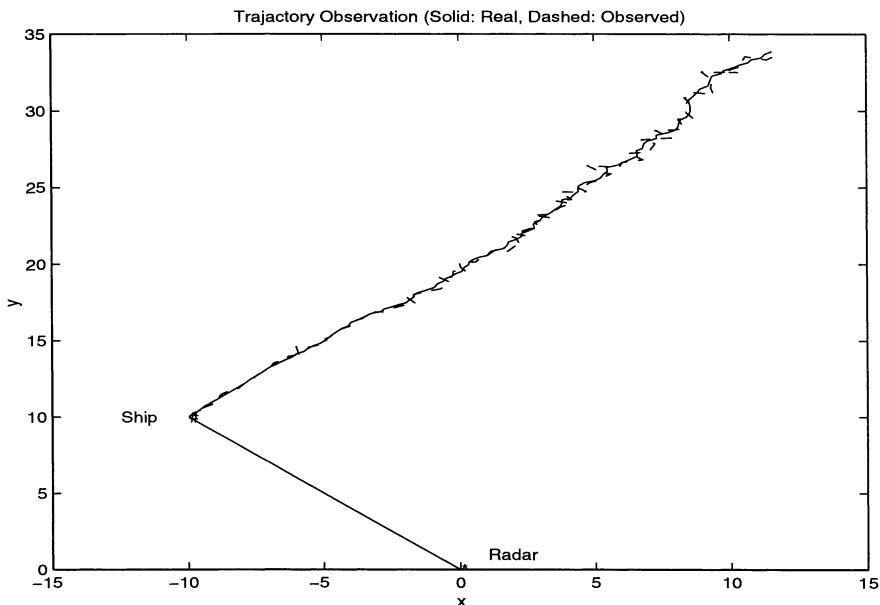


Fig. 9.19. One of the tested trajectories (in solid line) and its observation (in dashed line) by Sensor 2: $R_{21} = 0.01$, $R_{22} = 0.0001$

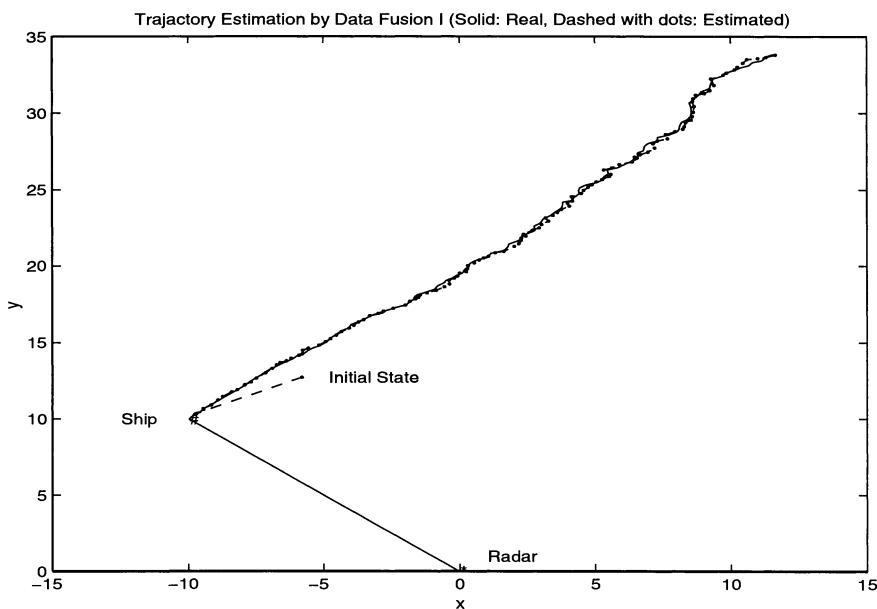


Fig. 9.20. Global trajectory estimation by fusion of data from the two sensors

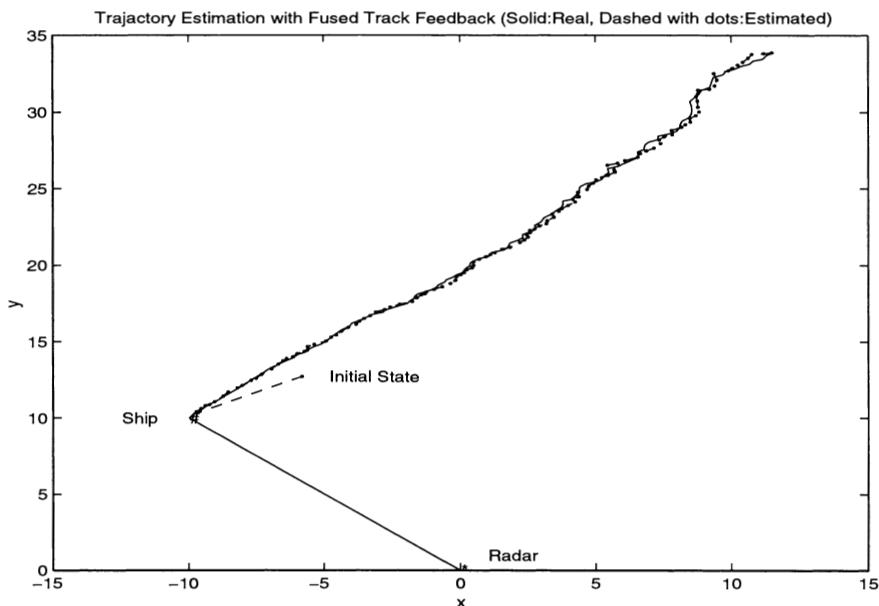


Fig. 9.21. Local trajectory estimation at Sensor 1 with track feedback, m/f ratio=1

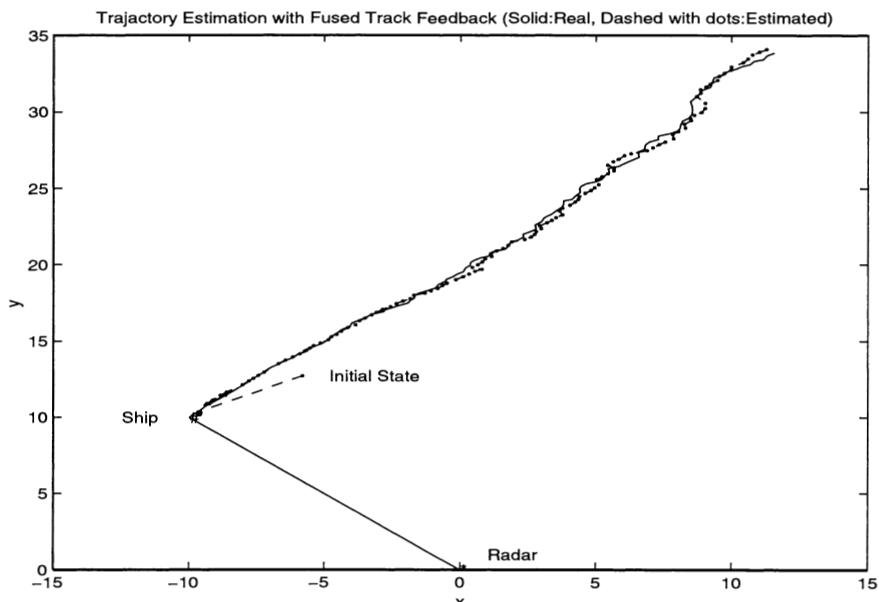


Fig. 9.22. Local trajectory estimation at Sensor 1 with track feedback, m/f ratio=5

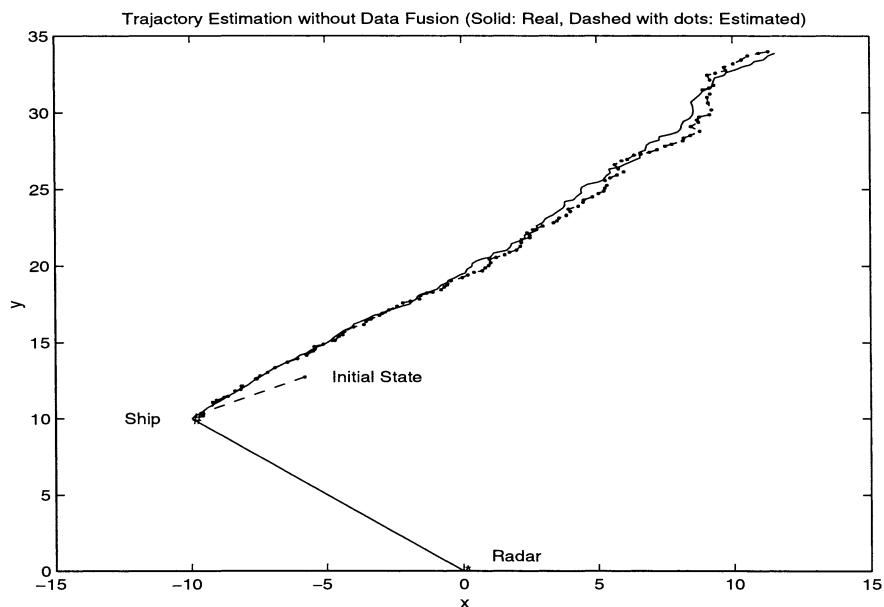


Fig. 9.23. Local trajectory estimation at Sensor 1 without track feedback

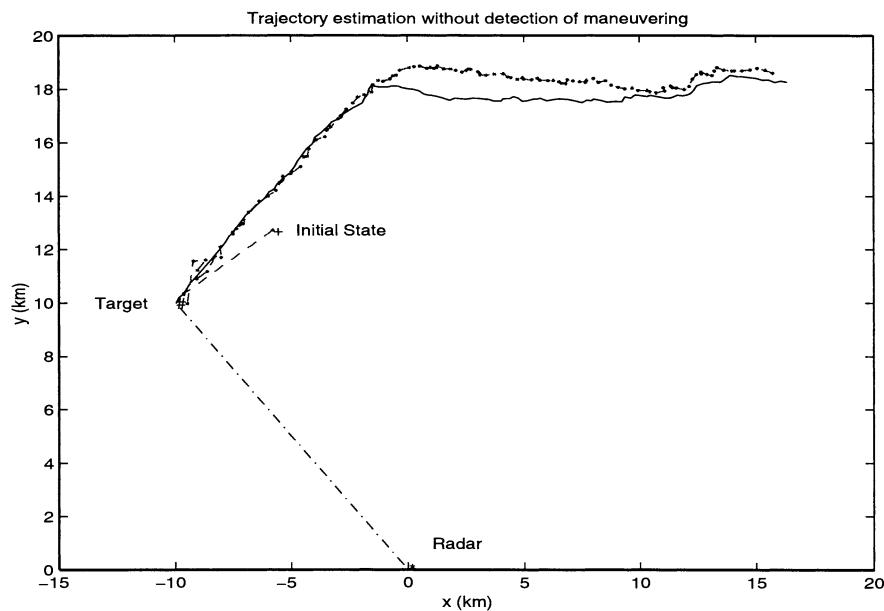


Fig. 9.24. Manoeuvring target trajectory estimation without manoeuvre detection

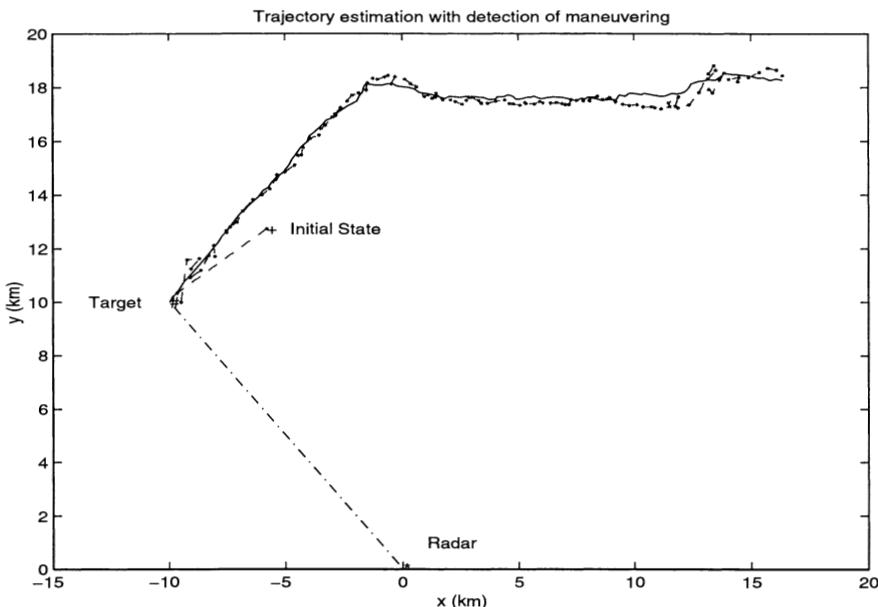


Fig. 9.25. Manoeuvring target trajectory estimation with manoeuvre detection

Table 9.1. Comparison of RMSEs of trajectory estimation results

Methods	RMSEs
Global measurement fusion	0.1233
Local estimation, m/f ratio=1	0.1732
Local estimation, m/f ratio=5	0.2662
Local estimation without track feedback	0.3588

At the local node with Sensor 2 that has smaller noise, the estimation performance is also improved by sharing the measurement information from Sensor 1 (with larger noise) via global track feedback, although it is not so obvious as at the local node with Sensor 1. However, the major benefit from the data fusion at node Sensor 2 is its robustness in the trajectory estimation.

Generally multisensor data fusion methods are ineffectual in improving the performance of manoeuvring target tracking. However, the neurofuzzy linearisation modelling techniques and the associated state estimation and data fusion methods developed in this book can cope with nonlinear manoeuvring target tracking by utilising manoeuvring detection based adaptive techniques. A manoeuvring target trajectory and tracking results by the neurofuzzy local linearisation model based state estimation and fusion are shown in Figure 9.24 and Figure 9.25. The result without manoeuvre detection is shown in Figure 9.24, which indicates large tracking bias following the ma-

noeuvre, and the tracking result with manoeuvre detection is shown in Figure 9.25 where the manoeuvre is detected by checking if the normalised squared innovation is larger than a threshold and the model noise level is adaptively adjusted accordingly.

In the field of multisensor data fusion, we are only now entering a period in which a serious examination of the scientific basis for multisensor data fusion is being conducted. There does not exist, in print, any basic examination of the organising principles governing the fusion process. Virtually all technical reports related to multisensor data fusion focused on the empirical process without regard to the underlying physical and mathematical principles of reasoning, sensing, association, detection, or combination and correlation. Therefore, multisensor data fusion is a field of great potential in terms of both academic and industrial values, and there is much to be achieved in this vital area.

10. Support vector neurofuzzy models

10.1 Introduction

The class of models considered so far are generalised linear models that construct nonlinear models by linear combinations of nonlinear basis functions such as B-splines, or Gaussian radial basis in the input or observed variables \mathbf{x} . The power of these models is their ability to incorporate prior knowledge by structurally designing the network through choice of the type, number and position of the basis functions. This form of structural regularisation is the basis of many of the construction algorithms introduced in this book. All of these generalised linear model are examples of parametric models in which weights or parameters are identified by using linear optimisation techniques (see Chapter 3). A further class of models can be defined which do not explicitly depend on a set of parameters, the so-called nonparametric models, are applicable to sparse data sets in relatively high dimensional spaces. In nonparametric models, the parameters are not predetermined, but are determined by the training data so that the model capacity reflects complexity contained in the data. Of particular importance in this context is the class of nonparametric models whose output is a linear combination of functions of the observations \mathbf{x} , where the linear weighting functions are determined by the characteristics of the kernel functions. The approximation of the approximant is taken over a *functional* including some measure of the data fit and a functional penalising certain characteristics of the approximant, this ensures a well posed solution. Common to nonparametric approaches to function approximation is the idea of reproducing kernel Hilbert spaces (RKHS) – see Section 2.6. These provide a vigorous mathematical treatment of prior knowledge in terms of projections of data onto approximating subspaces. The approximations then inherit properties derived from these subspaces. Interestingly these subspaces are uniquely determined by the so-called reproducing kernel which corresponds to a Gaussian covariance, support vector kernel or regularisation function in the Gaussian process or support vector machine approaches to modelling [139, 191]. RKHS provide a unifying framework to treat prior knowledge. RKHS were first introduced by Aronszajin [7] and have found increasing use in functional approximation. Perhaps most important is the work of Wahba [194] on splines which are derived using RKHS. Parzen [159] was the first to use RKHS methods in dynamic time series mod-

elling. He showed that the RKHS method provided a unified framework for problems in modelling linear Gaussian time series, including least squares estimation of random variables and the detection of signals in Gaussian noise. More recently ideas from RKHS are now commonly applied in the framework of support vector machines [180], Gaussian processes [139] and wavelets [52].

In this final concluding chapter we briefly introduce support vector machines (SVMs) and their links to input space based modelling algorithms such as radial basis function networks and neurofuzzy algorithms developed earlier in the book. SVMs are in some sense no different to any other linear-in-the-parameters or basis function dependent network described in this book. In SVMs learning is based upon quadratic programming which potentially leads to a parsimonious model that matches model capacity with data complexity. Nevertheless further controls are necessary in order to deal with the curse of dimensionality and perhaps not surprising in view of the earlier development of ASMOD, an SVM based on the analysis of variance (so-called SUPANOVA) is discussed in Section 10.5. No formal proofs are given of the basic theory of SVMs as they can be found in the excellent definitive books on SVMs by Vapnik [191], and Cherkassky and Mulier [49]. Proofs are however derived to establish the support vector neurofuzzy modelling algorithm since this is new material.

10.2 Support vector machines

The support vector machine (SVM) is a generalised or universal learning construction algorithm based on the theory of statistical learning which embodies the structural risk minimisation principle (see Section 1.3), and has the ability to learn a variety of model representations such as radial basis functions, polynomials, ridge functions and B-splines. The latter is particularly useful in generating a fuzzy interpretation to SVMs. SVMs provide an elegant means of solving nonlinear modelling problems by transforming them to linear ones in some *feature space* \mathcal{Z} which is related to the input data space \mathcal{X} via some *a priori* selected nonlinear mapping $g(\mathbf{x}) \rightarrow \mathbf{z} \in \mathcal{Z}$ (see Figure 10.1) [174], the inclusion of the constant term b , depends whether the bias is regularised or unregularised.

A linear approximation in the feature space with parameter \mathbf{w} is used to determine the output y . Unlike conventional statistical and neuro(fuzzy) methods the SVM approach does not attempt to manage model complexity by keeping the number of extracted features small. Instead the SVM method overcomes two fundamental problems in its design: the so-called *conceptual problem* of complexity and the *computational problem* of numerical optimisation. The conceptual problem of how to manage or control the complexity of the set of approximating functions in a high dimensional space (in order to achieve good generalisation ability) can be solved by penalising a linear estimator with a large number of basis functions or as in the case of SVMs

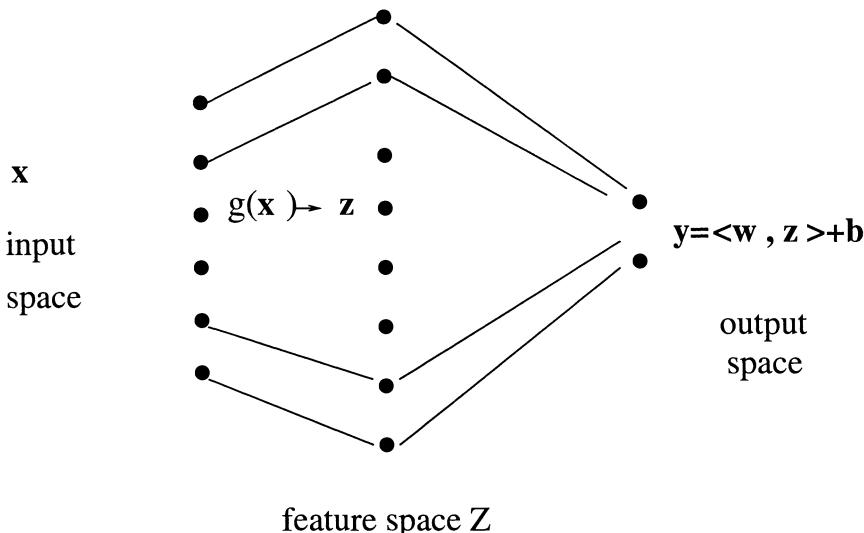


Fig. 10.1. Mappings implemented in SVM

by penalising the norm of the weighting coefficients. The resulting SVM approach results in a controlled quadratic optimisation problem formulation of the learning problem (see Chapter 2), since like other generalised linear additive models the resulting optimisation surface is convex. The computational problem of how to perform this numerical optimisation (i.e. solve the quadratic optimisation problem) in a high dimensional space is resolved by taking advantage of the dual kernel representation of linear functions, since this representation leads to computations in the form of inner products in the input space, which are of a much lower dimension than in the comparable feature space (see Section 10.2.2). In SVM, capacity or complexity control can be effectively controlled through the regularisation function used (see Section 2.5).

In summary the SVM approach is characterised by the following:

- New implementation of the SRM principle;
- Input data samples are mapped into a high dimensional (feature) space by using a set of nonlinear basis functions, which are *a priori* defined;
- Linear functions with constraints on the model complexity are used to approximate data in high dimensional spaces;
- Duality theory of optimisation is used to make estimation of model parameters in a high dimensional space computationally tractable;
- Characterisation of complexity in SVMs is independent of the input dimensionality;
- SVMs inherently employ nonlinear feature selection;
- Sparsity in coefficients

10.2.1 Loss functions

Originally SVMs were applied to classification problems. However modifying the loss function to include a distance metric they are readily extended to nonlinear regression problems. Under conditions of normal additive noise the quadratic loss or cost function

$$V_{\text{quad}}(y) = (y - f(\mathbf{x}, \mathbf{w}))^2 \quad (10.1)$$

provides an efficient (best, unbiased) estimator of the regression function $f(\cdot)$. However, least squares estimators are sensitive to the presence of outlier data samples and may perform badly when the underlying distribution has a long tail. To overcome this limitation Vapnik proposed the ε -insensitive cost or loss functional of Figure 10.2, defined by

$$V_\varepsilon(y) = \begin{cases} 0 & \text{for } |f(\mathbf{x}, \mathbf{w}) - y| < \varepsilon, \\ |f(\mathbf{x}, \mathbf{w}) - y| - \varepsilon & \text{otherwise,} \end{cases} \quad (10.2)$$

where the loss is 0 if the discrepancy between predicted and observed values is less than ε . Here ε represents, in some manner, the modelling desired error approximation or resolution. Hence a robust estimator follows, which is insensitive to small changes in the model. The main reason for choosing this loss function is that it makes the solution sparse in the adjustable coefficients, or equivalently the parameters ε acts as a model complexity parameter.

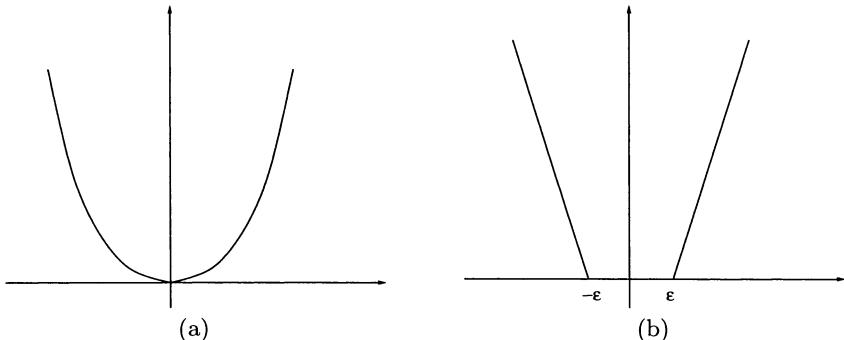


Fig. 10.2. (a) Quadratic cost function and (b) ε -insensitive cost function

The V_ε loss function is an attractive choice in the implementation of the SVM since selecting the value of ε controls the number of *support vectors*, which introduces sparseness to the final model solution, unlike the quadratic cost function where all data points are support vectors.

10.2.2 Feature space and kernel functions

The mapping into the high dimensional feature space is achieved in SVM by the use of reproducing kernels. The prime purpose of the kernel function

is to enable operations to be performed in the input data space \mathcal{X} rather than the high dimensional feature space \mathcal{Z} . Avoiding inner products being performed in the feature space, since via the theory of reproducing kernel Hilbert spaces RKHS [170], they have an equivalent kernel in the input space through $K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x})K(\mathbf{x}')$, provided certain conditions hold. If K is a symmetric positive definite function, which satisfies *Mercers condition* (see [174, 191]), then

$$K(\mathbf{x}, \mathbf{x}') = \sum_{m=1}^{\infty} \alpha_m \Psi(\mathbf{x})\Psi(\mathbf{x}'), \quad \alpha_m > 0$$

$$I = \int \int K(\mathbf{x}, \mathbf{x}') g(\mathbf{x})g(\mathbf{x}') d\mathbf{x} d\mathbf{x}' > 0, \quad \int g^2(\mathbf{x}) d\mathbf{x} < \infty. \quad (10.3)$$

Functions which satisfy *Mercers conditions* as kernel functions for all real \mathbf{x} and \mathbf{x}' include:

(i) *Polynomials* of degree d of

$$K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^d$$

$$\text{or } K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + 1)^d, \quad d = 1, 2, \dots, \quad (10.4)$$

where $\langle \cdot, \cdot \rangle$ denotes scalar product, are popular for nonlinear modelling. The second kernel is preferred as it avoids problems with the Hessian becoming zero.

(ii) *Gaussian radial basis functions*

A support vector neural network [40] can be readily derived based on the Gaussian radial basis kernel

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{(\|\mathbf{x} - \mathbf{x}'\|)^2}{2\sigma^2}\right). \quad (10.5)$$

(iii) *B-splines*

The kernel is defined on the interval $[-1, 1]$, with closed form (see Sections 4.2 and 8.2)

$$K(\mathbf{x}, \mathbf{x}') = B(\mathbf{x} - \mathbf{x}'). \quad (10.6)$$

As the kernels must be compact and symmetrical about its centre only even order B-splines can be chosen. In practice this means selecting fourth order B-splines to provide smooth basis functions. A fourth ordered B-spline with equi-space knots at $[-2\lambda, -\lambda, 0, \lambda, 2\lambda]$ for $\lambda > 0$ is given by

$$B_4(x) = \begin{cases} (2\lambda + x)^3 / 6\lambda^3 & -2\lambda \leq x < -\lambda \\ (4\lambda^3 - 6\lambda x^2 - 3x^3) / 6\lambda^3 & -\lambda \leq x < 0 \\ (4\lambda^3 - 6\lambda x^2 + 3x^3) / 6\lambda^3 & 0 \leq x < \lambda \\ (2\lambda - x)^3 / 6\lambda^3 & \lambda \leq x < 2\lambda \\ 0 & \text{otherwise} \end{cases}, \quad (10.7)$$

which will be used in Section 10.4 for the SVM neurofuzzy network.

(iv) *Multi-dimensional kernels*

Multi-dimensional kernels can be obtained by forming tensor products of kernels through

$$K(\mathbf{x}, \mathbf{x}') = \prod_{j=1}^n K_j(x_j, x'_j). \quad (10.8)$$

This is particularly useful in the construction of multi-dimensional B-spline kernels, since they are simply the product of univariate kernels such as (10.7). To use this, the multivariate B-spline kernel defined by (10.7) and (10.8) can be expressed as

$$K(\mathbf{x}(k), \mathbf{x}'(i)) = \prod_{j=1}^n \frac{B_4(x_j(k) - x_j(i))}{B_4(0)} \quad (10.9)$$

$$= \int_{\mathbf{w} \in \mathbb{R}^n} \Psi(\mathbf{w}) \exp(i\mathbf{w}^T \mathbf{x}) \exp(-i\mathbf{w}^T \mathbf{x}') d\mathbf{w}, \quad (10.10)$$

where $\mathbf{w} = [w_1, \dots, w_n]^T$, and $\Psi(\mathbf{w})$ is a non-negative function given by

$$\Psi(\mathbf{w}) = \prod_{j=1}^n \frac{3\lambda}{4\pi} \text{sinc}^4\left[\frac{\lambda w_j}{2\pi}\right], \quad (10.11)$$

where $\text{sinc}(w) = (\sin(\pi w))/(\pi w)$. Substituting (10.10) into *Mercers* condition (10.3) gives

$$I = \int_{\mathbf{w} \in \mathbb{R}^n} \Psi(\mathbf{w}) \left| \int_{\mathbf{x} \in \mathbb{R}^n} g(\mathbf{x}) \exp(i\mathbf{w}^T \mathbf{x}) d\mathbf{x} \right|^2 d\mathbf{w}. \quad (10.12)$$

Clearly $I > 0$, since $\Psi(\mathbf{w}) > 0$, hence the multivariate B-spline kernel given by (10.9) satisfies *Mercers* conditions and is appropriate for a SVM.

10.3 Support vector regression

In *linear* support vector regression, estimation of an optimal linear function of the form

$$f(\mathbf{x}, \mathbf{w}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \quad (10.13)$$

with precision ε , may be determined by minimising the functional

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N |y(i) - f(\mathbf{x}(i))|_\varepsilon, \quad (10.14)$$

written as a constrained optimisation problem; this is equivalent to the problem finding the pair (\mathbf{w}, b) that minimises the quantity defined by the *slack*

variables $\xi(i)$, $\xi^*(i)$, ($i = 1, \dots, N$). (The introduction of these slack variables into the constraint terms assumes an extra cost for errors.) The optimal regression is then given by the minimum of the cost functional

$$\Phi(\mathbf{w}, \xi, \xi^*) = \frac{1}{2} \|\mathbf{w}\|^2 + C \left(\sum_{i=1}^N \xi(i) + \sum_{i=1}^N \xi^*(i) \right), \quad (10.15)$$

subject to the inequality constraints:

$$\begin{aligned} <\mathbf{w}, \mathbf{x}(i)> + b - y(i) &\leq \varepsilon + \xi(i) \\ y(i) - <\mathbf{w}, \mathbf{x}(i)> - b &\leq \varepsilon + \xi^*(i) \\ \xi(i), \xi^*(i) &\geq 0 \end{aligned}$$

for all $i = 1, \dots, N$.

In the above C is a prespecified smoothing parameter which can be determined by resampling methods (e.g. cross-validation), and $\{\xi(i), \xi^*(i)\}$ are slack variables representing upper and lower errors on the output of the model. The second term in (10.15) represents a bound on the training error and that includes an implicit form of regularisation. It has been shown for certain kernels that the parameter C can be directly related to a regularisation parameter [180]. The quadratic loss function produces a solution which is equivalent to zero-order regularisation, for a regularisation parameter of $\lambda = \frac{1}{2C}$.

Using an ε -insensitivity loss function (see Figure 10.2), the solution to the above constrained optimisation problem is given by the Lagrangian *dual* problem solution given by [90]

$$\begin{aligned} \max_{\alpha, \alpha^*} W(\alpha, \alpha^*) &= \max_{\alpha, \alpha^*} \left\{ -\frac{1}{2} \sum_{i,j=1}^N (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) <\mathbf{x}(i), \mathbf{x}(j)> \right. \\ &\quad \left. + \sum_{i=1}^N (\alpha_i(y(i) - \varepsilon) - \alpha_i^*(y(i) + \varepsilon)) \right\}, \end{aligned} \quad (10.16)$$

or equivalently

$$\begin{aligned} \bar{\alpha}, \bar{\alpha}^* &= \arg \min_{\alpha, \alpha^*} \left\{ \frac{1}{2} \sum_{i,j=1}^N (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) <\mathbf{x}(i), \mathbf{x}(j)> \right. \\ &\quad \left. - \sum_{i=1}^N (\alpha_i - \alpha_i^*)y(i) + \sum_{i=1}^N (\alpha_i + \alpha_i^*)\varepsilon \right\}, \end{aligned} \quad (10.17)$$

with constraints

$$\begin{aligned} 0 \leq \alpha_i &\leq C, \quad i = 1, 2, \dots, N \\ 0 \leq \alpha_i^* &\leq C, \quad i = 1, 2, \dots, N \\ \sum_{i=1}^N (\alpha_i - \alpha_i^*) &= 0. \end{aligned} \quad (10.18)$$

Solving (10.16) with constraints (10.18) determines the Lagrange multipliers α_i, α'_i , and the regression function (10.13) is given by

$$\begin{aligned}\bar{\mathbf{w}} &= \sum_{i=1}^N (\alpha_i - \alpha'_i) \mathbf{x}(i), \\ \bar{b} &= -\frac{1}{2} \bar{\mathbf{w}}[\mathbf{x}(r) + \mathbf{x}(s)].\end{aligned}\quad (10.19)$$

The Karush–Kuhn–Tucker (KKT) conditions that are satisfied by the solution are

$$\alpha_i \alpha'_i = 0, \quad i = 1, 2, \dots, N.$$

Therefore the support vectors are points where exactly one of the Lagrange multipliers is greater than zero.

For *nonlinear* regression approximation we replace the dot product $\langle \mathbf{x}(i), \mathbf{x}(j) \rangle$ above with $K(\mathbf{x}(i), \mathbf{x}(j))$. So for the ε -insensitive loss function, the solution to the constrained optimisation problem is obtained by maximising $W(\alpha, \alpha^*)$ with respect to the Lagrangian multipliers (α, α^*) , i.e.

$$\begin{aligned}\max_{\alpha, \alpha^*} W(\alpha, \alpha^*) &= \max \left\{ -\frac{1}{2} (\alpha_i - \alpha'_i)(\alpha_j - \alpha'_j) K(\mathbf{x}(i), \mathbf{x}(j)) \right. \\ &\quad \left. + \sum_{i=1}^N (\alpha'_i(y(i) - \varepsilon) - \alpha_i(y(i) + \varepsilon)) \right\},\end{aligned}\quad (10.20)$$

subject to

$$\sum_{i=1}^N (\alpha'_i - \alpha_i) = 0, \quad (10.21)$$

$0 < \alpha'_i, \alpha_i \leq C/N$, $i = 1, 2, \dots, N$, where the regularisation constant C and the approximation error bound $\varepsilon > 0$ is selected *a priori* by the user. Solving (10.20) subject to the constraints (10.21), determines the Lagrange multipliers $\{\alpha, \alpha^*\}$ from which the regression estimate takes the form

$$f(\mathbf{x}) = \sum_{SV_s}^p (\alpha_i - \alpha'_i) K(\mathbf{x}(i), \mathbf{x}(j)) + b, \quad (10.22)$$

where p is the number of the non-zero Lagrange multipliers. The subset of the training data $S = \{\mathbf{x}'(1), \dots, \mathbf{x}'(p)\}$ is known as the *support vectors* denoted by

$$S = \{\mathbf{x}(i) : |y(i) - f(\mathbf{x}(i))| \geq \varepsilon\}. \quad (10.23)$$

The constant bias \hat{b} in (10.22) is computed from data that satisfies the equality

$$|y(i) - f(\mathbf{x}(i))| = \varepsilon. \quad (10.24)$$

As the bias b is computed from only part of the N data pairs, then the average modelling error from the SV regression models (10.22) is not necessarily

zero. The number of kernels used in this model (so-called support vector regression (SVR) [174]) is the same as the number of support vectors, which are in turn the centres of the kernels (10.9) or (10.10). Both the number p of the basis functions and the support vectors are determined simultaneously by the constrained optimisation for a given precision ε and regularisation constant C .

Note: As the support vector machine problem formulation and the extended structural risk minimisation principle (see Section 1.3) can be identically formulated [62] for an optimal value of C , then SVMs generate excellent data-based model.

10.4 Support vector neurofuzzy networks

By selecting normalised B-spline kernels the SVR network (10.22) can be represented as a support vector neurofuzzy network (SVNFN – see [39]), i.e. the resultant parsimonious SVM can be represented as a set of fuzzy rules defined by the B-spline membership functions due to the 1:1 equivalence between B-splines and fuzzy membership functions. The kernels of this network are determined by the support vectors of the SVR. Also as its weights are determined by least squares (see below), then its output is unbiased with the additional attribute of the modelling error variance being bounded by the error bound used in determining the support vectors.

Defining $[w'_1, \dots, w'_{p+1}]^T = [\hat{\alpha}_1 - \hat{\alpha}_1^*, \dots, \hat{\alpha}_{p+1} - \hat{\alpha}_{p+1}^*]^T$ as an unknown parameter vector and $K_i(\mathbf{x}(j)) = K(\mathbf{x}(j), \mathbf{x}(i))$, then the output of the SVR (10.22) can be written as

$$\hat{y}(j) = f(\mathbf{x}(j)) = \sum_{i=1}^p w'_i K_i(\mathbf{x}(j)) + w'_{p+1}. \quad (10.25)$$

Define $\lambda(\mathbf{x}(j)) = \sum_{i=1}^p K_i(\mathbf{x}(j))$

$$\Psi = \begin{bmatrix} K_1(\mathbf{x}(1)) & \dots & K_p(\mathbf{x}(1)) \\ \vdots & & \vdots \\ K_1(\mathbf{x}(N)) & \dots & K_p(\mathbf{x}(N)) \end{bmatrix}$$

and

$$\mathbf{L} = \begin{bmatrix} \lambda(\mathbf{x}(1)) & 0 & \dots & 0 \\ 0 & \vdots & \vdots & 0 \\ 0 & 0 & \dots & \lambda(\mathbf{x}(N)) \end{bmatrix}.$$

Rewriting (10.25) in matrix form over all input-output data for $\hat{\mathbf{y}} = [\hat{y}(1), \dots, \hat{y}(N)]^T$, as

$$\hat{\mathbf{y}} = \Psi \begin{bmatrix} w'_1 \\ \vdots \\ w'_p \end{bmatrix} + \begin{bmatrix} w'_{p+1} \\ \vdots \\ w'_{p+1} \end{bmatrix}. \quad (10.26)$$

But since $\sum_{j=1}^p \frac{K_j(\mathbf{x}(j))}{\lambda(\mathbf{x}(j))} = 1$, then $\mathbf{L}^{-1}\Psi[1, \dots, 1]^T = [1, \dots, 1]^T$, then (10.26) becomes

$$\hat{\mathbf{y}} = \Psi \begin{bmatrix} w'_1 \\ \vdots \\ w'_p \end{bmatrix} + \mathbf{L}^{-1}\Psi \begin{bmatrix} w'_{p+1} \\ \vdots \\ w'_{p+1} \end{bmatrix} = \mathbf{L}^{-1}\Psi \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}, \quad (10.27)$$

where the $\{w_1, \dots, w_p\}$ weights are given by

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix} = (\Psi^T \Psi)^{-1} \Psi^T \mathbf{L} \Psi \begin{bmatrix} w'_1 \\ \vdots \\ w'_p \end{bmatrix} + \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} w'_{p+1}. \quad (10.28)$$

From (10.27) the SVR (10.22) becomes

$$\hat{y}(j) = \frac{\sum_{i=1}^p w_i K_i(\mathbf{x}(j))}{\sum_{i=1}^p K_i(\mathbf{x}(j))} = \sum_{i=1}^p w_i N_i(\mathbf{x}(j)), \quad (10.29)$$

where $N_i(\mathbf{x}(j)) = K_i(\mathbf{x}(j))/\lambda(\mathbf{x}(j))$ is a normalised fuzzy membership function and (10.29) is structured as a conventional neurofuzzy model, albeit its parameters are initially generated by SVM methods with their inherent parsimony through the p support vector constrained optimisation selection for a fixed approximation bound ε , followed by least squares to determine the network parameter vector \mathbf{w} . Rewriting (10.29) in matrix form gives

$$\hat{y}(j) = \mathbf{B}^T(\mathbf{x}(j))\mathbf{w}, \quad (10.30)$$

where $\mathbf{B}(\mathbf{x}(j)) = [N_1(\mathbf{x}(j)), \dots, N_p(\mathbf{x}(j))]^T$. For output observations $\{\mathbf{x}(t), y(t)\}_{t=1}^N$, the linear least squares estimate for \mathbf{w} is given by minimising the usual quadratic cost

$$V(\mathbf{w}) = \frac{1}{N} \sum_{t=1}^N (y(t) - \hat{y}(t))^2, \quad (10.31)$$

where $\hat{y}(t)$ is the network estimate for $y(t)$. The least squares estimate for the weights $\hat{\mathbf{w}}$ is given (see Section 3.3) as

$$\hat{\mathbf{w}} = [\mathbf{A}^T \mathbf{A}]^{-1} \mathbf{A}^T \mathbf{y}, \quad (10.32)$$

where

$$\mathbf{A} = \begin{bmatrix} N_1(\mathbf{x}(1)) & \dots & \dots & N_p(\mathbf{x}(1)) \\ \vdots & & & \vdots \\ N_1(\mathbf{x}(N)) & \dots & \dots & N_p(\mathbf{x}(N)) \end{bmatrix}.$$

Theorem 10.1 [39]: For a general nonlinear regression modelling problem represented by the SVNFN of (10.29), the support vectors and estimated weights of the equivalent support vector regression (SVR), $\hat{\mathbf{w}}_L$, are obtained by the constrained maximisation of $W(\alpha, \alpha^*)$ of (10.20) for a given error bound ε , and the estimated weights, $\hat{\mathbf{w}}$, of the SVNFN are obtained by minimising $V(\hat{\mathbf{w}})$ given by (10.31). Then $V(\hat{\mathbf{w}}) < V(\hat{\mathbf{w}}_L) < \varepsilon^2$ holds.

Proof. Let $\hat{\mathbf{w}}_L = [\hat{\alpha}_1, \dots, \hat{\alpha}_n, \hat{b}]^T$ be the estimated weights of the SVR that maximises $W(\alpha, \alpha^*)$. The variance of the modelling errors $V(\hat{\mathbf{w}}_L)$ is given by

$$V(\hat{\mathbf{w}}_L) = \frac{1}{N} \sum_{t=1}^N (y(t) - f(\mathbf{x}(t)))^2$$

where $f(\mathbf{x}(t))$ is the estimate output of the SVR (10.22). Rearrange the input data pairs $\{\mathbf{x}(t), y(t)\}_{t=1}^N$, such that the first p elements are the support vectors $\{\mathbf{x}(1), \dots, \mathbf{x}(p)\}$. For a sufficiently large regularisation constant C , maximising $W(\cdot)$ with the constraints yields $|y(k) - f(\mathbf{x}(k))| \leq \varepsilon$ for $k = 1, \dots, N$, where k denotes the new data index. Therefore for $1 \leq k \leq p$, $|y(k) - f(\mathbf{x}(k))| = \varepsilon$ means that

$$\sum_{k=1}^p |y(k) - f(\mathbf{x}(k))|^2 = p\varepsilon^2. \quad (10.33)$$

But since $|y(k) - f(\mathbf{x}(k))| < \varepsilon$ for $p+1 < k < N$, then

$$\sum_{k=p+1}^N |y(k) - f(\mathbf{x}(k))|^2 < (N-p)\varepsilon^2. \quad (10.34)$$

From (10.33) and (10.34), $V(\hat{\mathbf{w}}_L)$ becomes

$$V(\hat{\mathbf{w}}_L) < \frac{1}{N} (p\varepsilon^2 + (N-p)\varepsilon^2) = \varepsilon^2. \quad (10.35)$$

The estimated weights of the SVNFN $\hat{\mathbf{w}}$ are obtained by minimising $V(\hat{\mathbf{w}})$ given by (10.31) as

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{N} \sum_{k=1}^N (y(k) - \mathbf{B}^T(\mathbf{x}(k))\mathbf{w})^2.$$

But since $\hat{\mathbf{w}}_L$ is obtained by constrained maximisation of the dual cost $W(\alpha, \alpha^*)$, it is generally not identical to $\hat{\mathbf{w}}$, the optimal solution, and hence

$$V(\hat{\mathbf{w}}) < V(\hat{\mathbf{w}}_L),$$

which taken with (10.35) gives the model variance bound

$$V(\hat{\mathbf{w}}) < V(\hat{\mathbf{w}}_L) < \varepsilon^2. \quad (10.36)$$

□

Example 10.1: Consider again the benchmark nonlinear dynamic modelling problem of Section 4.5.2 for the 2-D limit cycle system

$$\begin{aligned} y(t) = & [0.8 - 0.5 \exp(-y^2(t-1))]y(t-1) - [0.3 + 0.9 \exp(-y^2(t-1))] \\ & \times y(t-2) + 0.1 \sin(\pi y(t-1)) + N(0, 0.01). \end{aligned} \quad (10.37)$$

The noise free regression surface for initial condition $\mathbf{x}(1) = [0.1, 0.1]^T$ is shown in Figure 10.3. The phase diagram or iteration map of the noise free system is shown in Figure 10.4 as an attracting limit cycle with an unstable equilibrium at its origin. The noisy 300 training data set of (10.38) for initial condition $\mathbf{x}(1) = [0, 0]^T$ is shown in Figure 10.5.

A fourth B-spline kernel (see (10.9), (10.11) and (10.7)) has been chosen with $\lambda = 1.25$, whilst the network approximation bound ε and constraint parameter C has been set as 0.2 and 300 respectively. As the dynamics are periodic the support vectors can be selected from the first 50 data pairs, from which 10 support vectors are selected from the constrained maximisation procedure of Section 10.3 which are shown circled in Figure 10.6. The resulting response surfaces for the consequent SVR model and SVNFN model for these support vectors are shown in Figures 10.7 and 10.8 respectively. The modelling error variance of the SVR and the SVNFN are 0.1236^2 and 0.0996^2 respectively. The phase diagram for the test data for the SVNFN is shown in Figure 10.9 illustrating the excellent recovery of the underlying dynamical process.

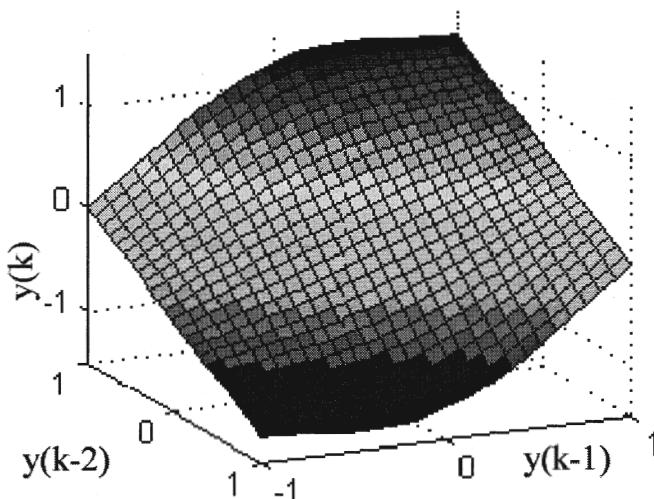


Fig. 10.3. Regression surface of the nonlinear system (Example 10.1)

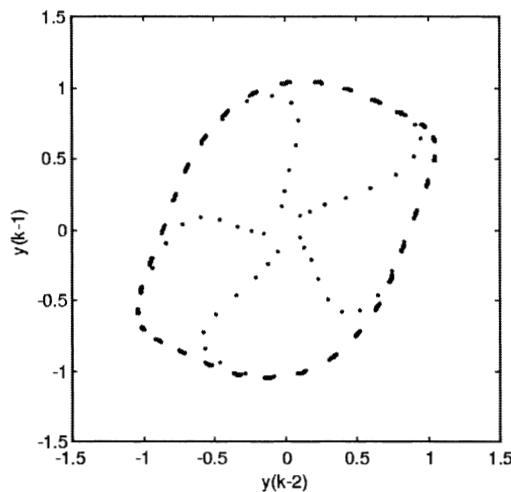


Fig. 10.4. Iterative map of the system output for $\varepsilon = 0$ (Example 10.1)

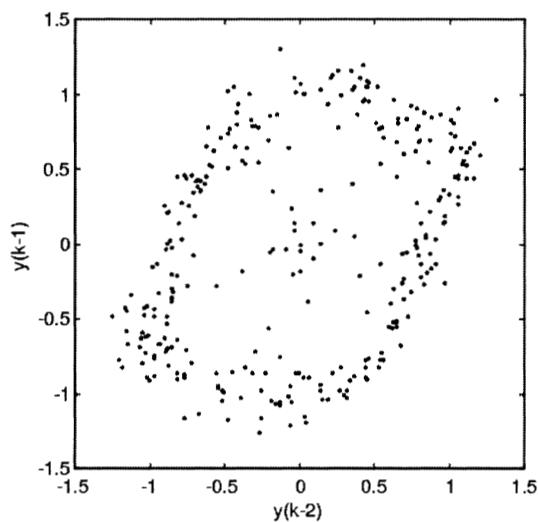


Fig. 10.5. Iterative map of the system output for $\varepsilon \sim N(0, 0.1^2)$ (Example 10.1)

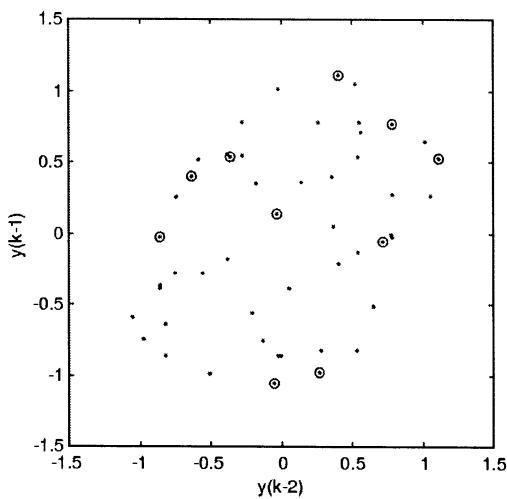


Fig. 10.6. Support vectors selected using 50 data (Example 10.1)

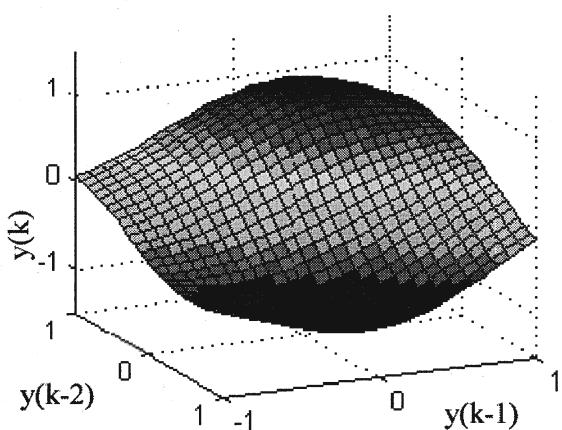


Fig. 10.7. Regression surface for the SVR (Example 10.1)

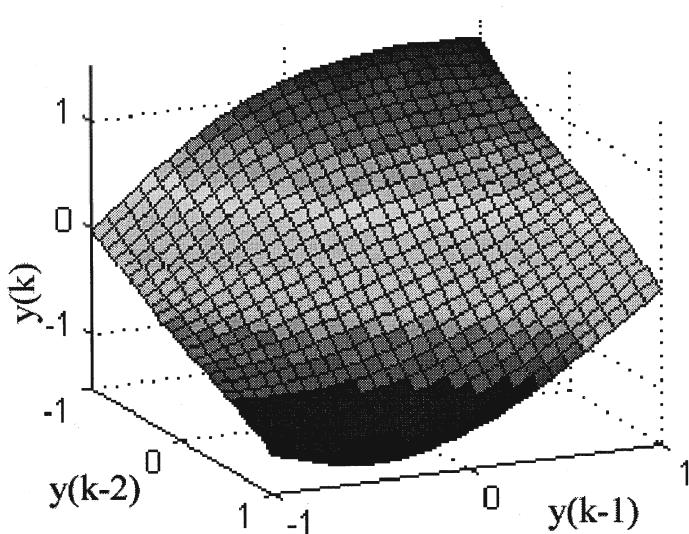


Fig. 10.8. Regression surface for the SVNFN (Example 10.1)

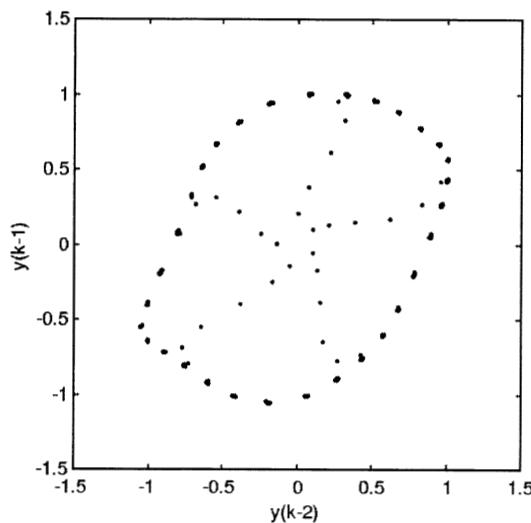


Fig. 10.9. Iterative map of the system output of the SVNFN (Example 10.1)

To further compare the performance of the SVR and the SVNFN, the simulation is repeated with varying ε to determine the number of support vectors required to achieve a modelling error less than the *a priori* selected ε , based upon fourth order B-splines ($\lambda = 1.25$) and Gaussian kernels (with $\sigma = 0.707$), are shown in Table 10.1.

Table 10.1. Number of support vectors p for given ε for Gaussian and B-spline kernels for the SVR and SVNFN models

ε (Gaussian)	0.23	0.225	0.21	0.2	0.185
ε (B-spline)	0.25	0.23	0.22	0.2	0.175
p	7	8	9	10	11
ε (Gaussian)	0.181	0.1807	0.1805	0.16	0.15
ε (B-spline)	0.174	0.162	0.15	0.14	0.13
p	12	13	14	15	16

The variance of the modelling errors relative to the number of support vectors is shown in Figure 10.10. As by Theorem 10.1, the modelling variance of the SVR and the SVNFN is less than ε^2 , with the variance of the SVNFN always less than that of the SVR, while approaches the variance of the noise model (10.37) as the number of support vectors increases.

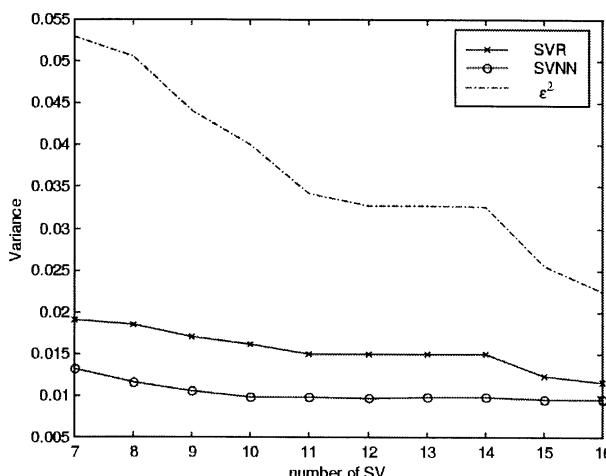


Fig. 10.10. Variance of the modelling error versus the number of support vectors (Example 10.1)

10.5 SUPANOVA

Whilst a predictive model with good generalisation performance, subject to sparse data is the ultimate goal of modelling, this book has also focused on the interpretability of transparency of the final model structure via the principle of parsimony. For a model to be transparent it needs to be simple but not so simple as to suffer from model mismatch. Model transparency not only allows ease of understanding of what inherent relationships are contained in the data but also enables the model to be readily validated. Approaches considered to aid transparency have included input space dimension reduction through pre-processing, feature extraction, orthogonalisation, as well as schemas (such as ANOVA) for decomposing the model into lower dimensional submodels which are easily visualised as 2-D or 3-D surfaces. To address the issue of transparency within a support vector machine, Gunn, et al. [90, 91, 92] have introduced the modified kernel model

$$f(\mathbf{x}) = \sum_{i=1}^N \alpha_i \sum_{j=1}^p c_j K_j(\mathbf{x}(i), \mathbf{x}), \quad c_j \geq 0, \quad (10.38)$$

where the general kernel in (10.22) is replaced by a weighted linear sum of the kernels $K_j(\cdot, \cdot)$. This additive structure is akin to the analysis of variance (ANOVA) representation of (5.4) used in ASMOD (see Section 5.3). Expression (10.38) achieves increased transparency making the coefficients c_j sparse. The kernels $K_j(\cdot, \cdot)$ are positive definite, and the coefficients c_j ensure that the complete kernel in (10.38) is also positive definite. To ensure sparseness in the kernel coefficients c_j it is necessary to minimise their number by selecting the appropriate ρ -norm on the kernel coefficients. As ρ decreases, the solution becomes more sparse, but the computational complexity of the consequent optimisation problem increases. To enforce sparsity in the kernel expansion, we consider an amended regularised cost function

$$V_N(\alpha, \mathbf{c}) = L(y, \mathbf{K}(\mathbf{c})\alpha) + \lambda_\alpha \|\alpha\|_{\mathbf{K}(\mathbf{c})} + \lambda_c \|\mathbf{c}\|_\rho, \quad (10.39)$$

for $\mathbf{c} = [c_1, \dots, c_p]^T \in \Re^p$, $c_i \geq 0$, $(\lambda_\alpha, \lambda_c) > 0$, $\alpha = [\alpha_1, \dots, \alpha_N]^T \in \Re^N$, where $L(\cdot, \cdot) = \|\mathbf{y} - \sum_i c_i K_i \alpha\|_{1,\epsilon}$ and $\|\alpha\|_{\mathbf{K}(\mathbf{c})} = \sum_i c_i \alpha^T \mathbf{K}_i \alpha$ are generalised loss functions, where

$$\mathbf{K}_i = \{K_i(\mathbf{x}(j), \mathbf{x}(k))\} \in \Re^{N \times N}, \quad j, k = 1, \dots, N$$

and $(\lambda_\alpha, \lambda_c)$ are regularising coefficients controlling the model output smoothness (see Section 2.5) and sparsity of the kernel expansion respectively. Selecting $\rho = 0$ gives an ideal solution but leads to a hard optimisation problem; whereas $\rho = 2$ produces straightforward optimisation solution but loses sparsity, $\rho = 1$ is a good compromise between sparsity and computational ease. Even for $\rho = 1$ the direct solution of optimising (10.39) with respect to (α, \mathbf{c}) simultaneously is nontrivial. Gunn and Kandola [90] have resolved this problem by introducing a *dual iterative* optimisation procedure whereby

$\min_{\alpha} V_N(\alpha, \mathbf{c})$ for fixed \mathbf{c} is solved followed by $\min_{\alpha} V_N(\alpha, \mathbf{c})$ for fixed α , then repeated until convergence is achieved. The attraction of this iterative optimisation is that the two subproblems can be treated as simple convex optimisation problems. For the ε -insensitivity loss function (see (10.2)), the solution for α^* is given by a box constrained quadratic program with linear constraints; both readily solved by a standard quadratic programming optimiser via (see [90] for details)

$$\begin{aligned} V_N(\alpha, \mathbf{c}) = & \| \mathbf{y} - \sum_i c_i \mathbf{K}_i \alpha \|_{1,\varepsilon} + \lambda_\alpha \sum_i c_i \alpha^T \mathbf{K}_i \alpha \\ & + \lambda_c \sum_i c_i, \quad \forall i, c_i \geq 0, \end{aligned} \quad (10.40)$$

$$\begin{aligned} \alpha^* = & \arg \min_{\alpha=\alpha^+-\alpha^-} (\alpha^+ - \alpha^-)^T (\lambda_\alpha \sum_k c_k \mathbf{K}_k) (\alpha^+ - \alpha^-) \\ & - \sum_i (\alpha^+ - \alpha^-) y(i) + \sum_i (\alpha^+ + \alpha^-) \varepsilon, \\ \forall i, \quad 0 \leq & \alpha_i^+, \alpha_i^- \leq \frac{1}{2\lambda_\alpha} \end{aligned} \quad (10.41)$$

$$\begin{aligned} \mathbf{c}^* = & \arg \min_{\mathbf{c}, \xi^+, \xi^-} \sum_i (\xi_i^+ + \xi_i^-) + \sum_j c_j (\lambda_\alpha \alpha^T \mathbf{K}_j \alpha + \lambda_c), \quad (10.42) \\ \forall i, j, \quad c_j \geq 0, \quad 0 \leq & \xi_i^+, \xi_i^- \leq 0, \quad -\xi^- \varepsilon \leq \sum_k c_k \mathbf{K}_k \alpha_k \leq \xi^+ + \varepsilon, \end{aligned}$$

where ξ^+, ξ^- are slack variable vectors. If the regularising coefficients are known then the solution can be obtained by the iteration

$$\begin{aligned} \text{initialise : } \alpha_0^* &= \arg \min_{\alpha} V_N(\alpha, \mathbf{c}_0^*); \quad \mathbf{c}_0^* = 1, \\ \text{iteration : } (i) \quad \mathbf{c}_{i+1}^* &= \arg \min_{\mathbf{c}} V_N(\alpha_i^*, \mathbf{c}), \\ (ii) \quad \alpha_{i+1}^* &= \arg \min_{\alpha} V_N(\alpha, \mathbf{c}_{i+1}^*). \end{aligned} \quad (10.43)$$

In practice $(\lambda_c, \lambda_\alpha)$ are *a priori* unknown and are initially set low then gradually increased. If λ_α is too high, it weights the smoother regularisation too much preventing a sparse solution being found, equally if λ_c is too high an overly sparse model will be determined.

The conventional ANOVA decomposition of a multivariate function $f(\mathbf{x})$ is

$$f(\mathbf{x}) = f_0 + \sum_{i=1}^n f_i(x_i) + \sum_{i,j} f_{ij}(x_i, x_j) + \dots$$

The SUPANOVA algorithm [66, 65, 91, 90] expands *kernels* as tensor products of univariate kernels (plus a bias) (see also (10.8) as

$$\begin{aligned}
K_{ANOVA}(\mathbf{x}, \mathbf{x}') &= \prod_i^n (1 + K(x'_i, x_i)) \\
&= 1 + \sum_i^n K(x'_i, x_i) + \sum_{i < j}^n K(x'_i, x_i)K(x'_j, x_j) \\
&\quad + \dots + \prod_{i=1}^n K(x'_i, x_i).
\end{aligned} \tag{10.44}$$

Rather than use the complete ANOVA expansion increased transparency can be achieved by utilising the more parsimonious form

$$\begin{aligned}
K_{ANOVA}(\mathbf{x}, \mathbf{x}') &= c_0 + \sum_i^n c_i K(x'_i, x_i) + \sum_{i < j}^n c_{ij} K(x'_i, x_i)K(x'_j, x_j) \\
&\quad + \dots + c_{1,2,\dots,n} \prod_{i=1}^n K(x'_i, x_i),
\end{aligned} \tag{10.45}$$

with an additional set of positive coefficients c_i ; resulting in a kernel which is a weighted sum of basis kernels $K(\mathbf{x}, \mathbf{x}')$. Each of the additive terms in both (10.44) and (10.45) are by definition positive definite, and as such are valid kernels by *Mercers conditions* (10.3); also a limited number of terms in (10.44) or (10.45) are also valid kernels. As with the conventional SVM, the kernel $K(\mathbf{x}', \mathbf{x})$ controls the final model; for simplicity in SUPANOVA the same kernel is used for each dimension. The normal range of kernel functions can be used in (10.45) including Gaussian radial basis functions, B-splines, ridge functions and polynomials (see Sections 2.3 and 2.5), however first-order infinite splines

$$\begin{aligned}
K_s(u, v) &= \int_0^\infty (u - \tau)_+ (v - \tau)_+ d\tau \\
&= uv + \frac{1}{2}(u + v) \min(u, v) - \frac{1}{6}(\min(u, v))^3,
\end{aligned} \tag{10.46}$$

which are piecewise cubic polynomials that pass through the origin. These splines have advantages over B-splines since they offer the general advantages of splines but also have no scaling parameter, avoiding a multitude of such parameters being computed within the expansion (10.45). Also since the univariate term in the ANOVA expansion (10.45) is constrained to pass through the origin; univariate and higher order terms in (10.45) will be constrained to be zero along their axis, so that the resultant ANOVA model favours low order terms in preference to higher order terms thus contributing to improved model transparency.

This ANOVA expansion of the SVMs approach to sparse data modelling aims to find a *global* model which selects the most significant subsystems, rather than the local approaches of ASMOD, which may get caught in local minima during that construction phase and additionally may not be strictly well posed.

Example 10.2: Automobile miles per gallon (AMPG) data modelling

The AMPG data set developed by the University of California contains the miles travelled, per gallon of fuel consumed, for various manufacturers' cars. The input variables measure six car characteristics: the number of cylinders (discrete), engine displacement, horsepower, car weight, acceleration and model year of manufacture (discrete). The modelling goal is to discover a transparent relationship between AMPG and the cars' various input characteristics. After removing a small number of entries with aberrant values, there remain 392 entries. A pairwise variable analysis of the data [34] shows that the distributions of the engine displacement, horse power and weight are not surprisingly similar, indicating high correlation between these attributes. The acceleration data distribution suggests that acceleration is the only attribute which has no direct relationship with the AMPG data.

The resulting AMSOD model algorithm of Section 5.3 is shown in Figure 10.11. The power of ASMOD is clearly seen in the rejection of three of the input characteristics, number of cylinders, displacement and acceleration. Furthermore the resulting model is easily interpreted due to additive struc-

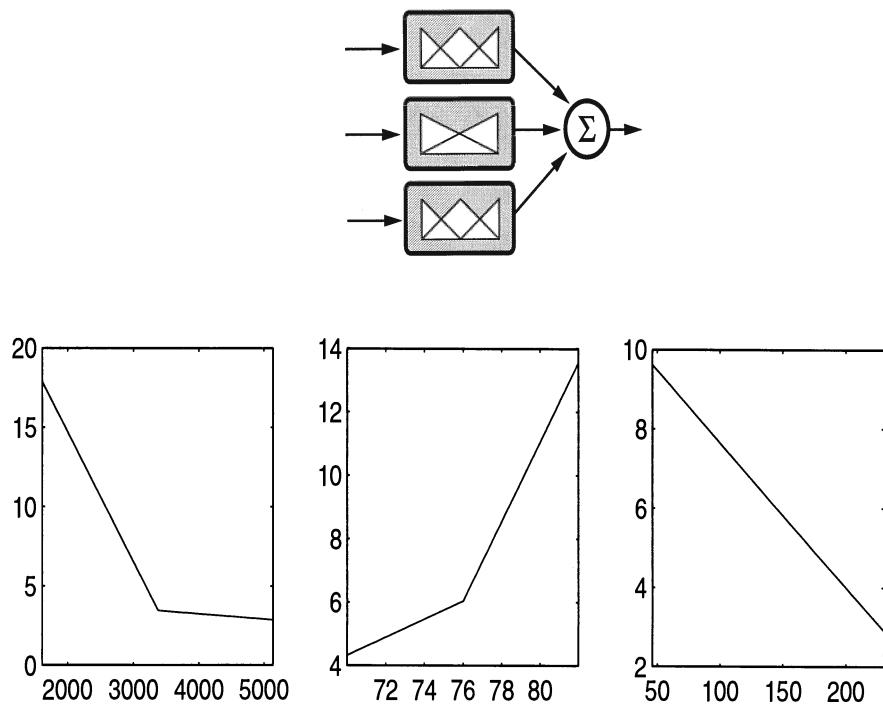


Fig. 10.11. Resulting ASMOD model for the MPG modelling problem

ture of the final model. The ASMOD model forms three additive univariate sub-networks which are plotted in Figure 10.11. From there it can be seen how the MPG increases linearly with the year of manufacture, and decreases with increasing weight and horsepower. All results of ASMOD make common sense, in that the MPG decreases with increasing engine horsepower and vehicle weight. Also the MPG change around 1976 is supported from prior knowledge such as speed limitations imposed in the USA following the international oil crisis of 1973 which in turn influences car manufacture.

The ANFIS and LOLIMOT local models (see Section 6.2) have also been applied to this data set less successfully [34]. Figure 10.12 illustrates one of the 50 potential models produced by the SUPANOVA algorithm, it can be seen that it has selected almost the same three univariate sub-models as ASMOD together with a bias. these three bivariate models and one trivariate

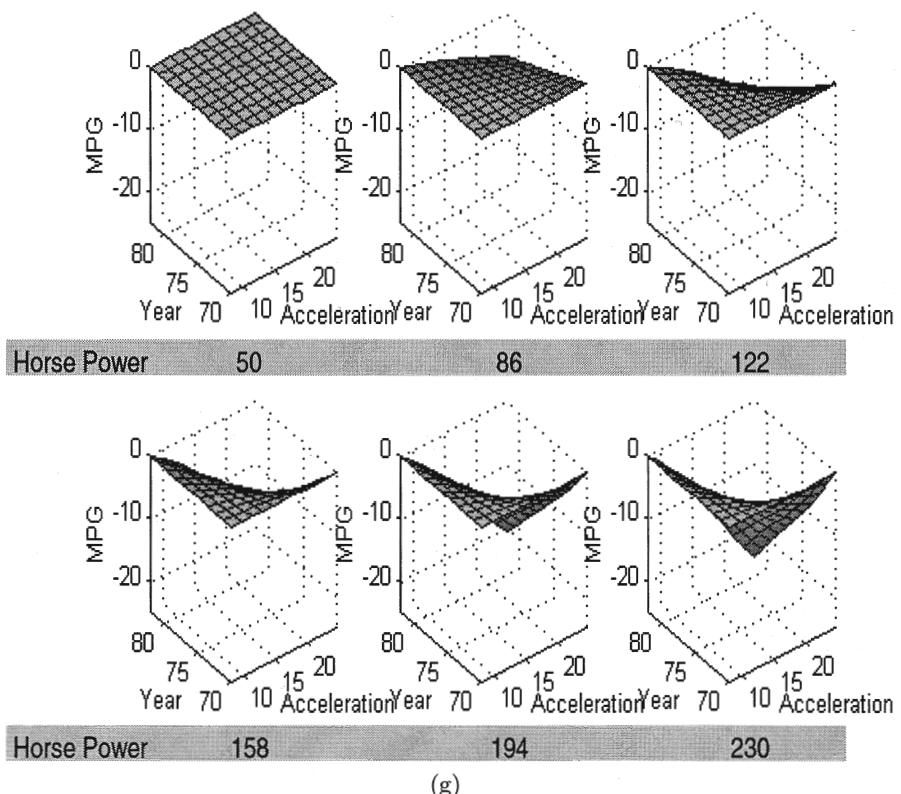
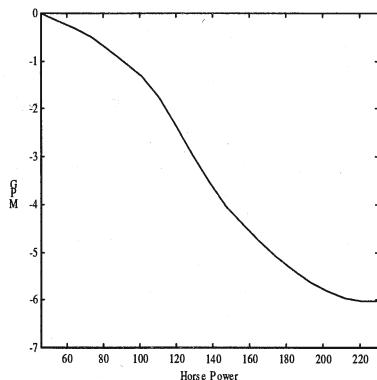
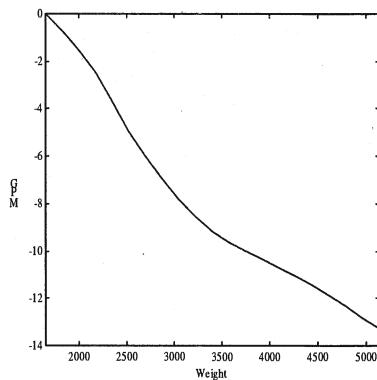


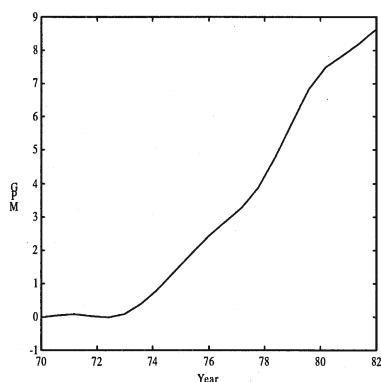
Fig. 10.12. Visualisation of the ANOVA terms from an ϵ -insensitive AMPG model (1 of 50); (a) AMPG versus horse power; (b) AMPG versus weight; (c) AMPG versus year; (d) AMPG versus displacement and weight; (e) AMPG versus displacement and acceleration; (f) AMPG versus year and acceleration; and (g) AMPG versus year and acceleration by varying horse powers



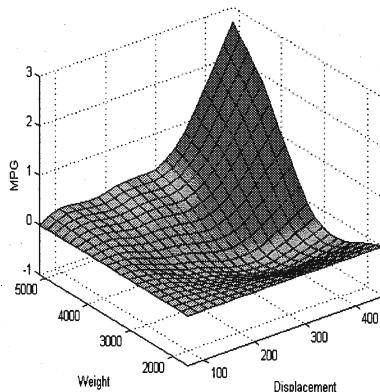
(a)



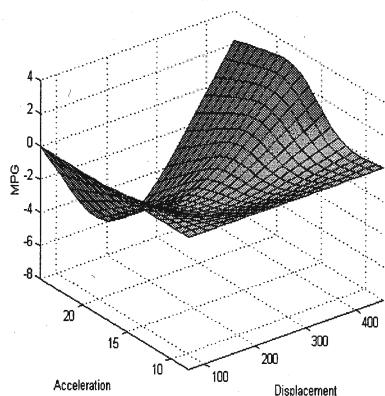
(b)



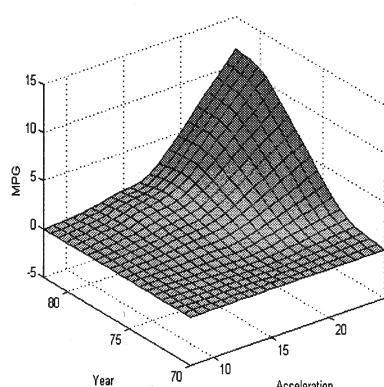
(c)



(d)



(e)



(f)

Fig. 10.12 — continued

from a possible 64 terms. Whilst the trivariate submodel is harder to interpret, the bivariate models clearly illustrate the trend between the interaction terms.

10.6 A comparison among neural network models

Neural networks have been widely studied in this book and have been shown to have strong functional approximation capabilities. The comparative aspects of the neural network models are very complex and case dependent. In the model selection amongst a range of different models, the modeller should take into account different modelling aspects, such as modelling objectives (prediction or controller design), model predictive and generalisation performance, model adaptation capability (simplicity of learning algorithms), model parsimony and model transparency (unique to neurofuzzy system). In practice, the model selection is often a tradeoff of these overall criteria. As we have demonstrated in Chapter 4, the B-spline neurofuzzy system has a unique transparency property that is defined as the knowledge (rule) extraction capability of the system, due to the lattice structure and locality of fuzzy membership functions. For a neurofuzzy system with good transparency property, the model bases can be directly related to linguistic fuzzy rules, and the derived information related to the model bases can be directly understood as the information associated with the fuzzy rules. A model with a good transparency property helps people to understand the system behaviours, oversee critical system operating regions and design local modelling controllers, and/or extract physical law underlying the system. However, it is also noted that this property is often at the expense of the model parsimony. The curse of dimensionality has been a main disadvantage for latticed model, and the main modelling challenge to the authors.

The comparative aspects of the neural network models have been studied based on the Example 10.1 [6] by using the cerebellar model articulation controller (CMAC), B-splines, radial basis functions (RBF) and multilayered perceptron (MLP) network. The results of the piecewise linear model (PL), Bézier–Bernstein (BB) model in Section 7.4, and support vector neurofuzzy networks in Section 10.4 are incorporated with results of An et al. [6] for an overall comparison. Whilst this is a specific example it does manifest generic difficult modelling problems of sparse data areas, high noise levels and potential for over parameterisation, and therefore serves as a useful benchmark example for the various modelling schemes. In this study, all of the derived models have parameter convergence and can capture the underlying dynamics of this problem with comparable predictive performances and surface construction capabilities. They all include some kind of prior knowledge such as predetermined system lag to constrain an over flexible model base, system input regions to select knots, RBF centres, etc., as well as system

smoothness information to constrain the model from overfitting and overparameterisation. Table 10.2 further provides some modelling aspects that need to be taken into account. Note that (i) the learning adaptation level only applies to the cases when the operating regions of the underlying processes drift slowly compared with the model adaptation, such that no significant model structure determination is frequently needed; and (ii) the curse of dimensionality indicates its level that is inherent in the conventional methods, if they are applied to problems with high dimensional input without using counter measures such as those developed in the book.

Table 10.2. Network comparisons for Example 10.1

Network	Size	Learning (adaptation)	Training cycles	Trans- parency	Curse of di- mensionality
CMAC	65	LMS (easy)	20	No	High
B-splines	16 (ASMOD)	LMS (easy)	20	Yes	High ^a
RBF	40	RLS (average)	4	No	Medium
MLP	64	BP (hard)	4	No	Medium
PL	24	VFSR+LMS (easy)	20	Yes	Medium
BB	15	LS (average)	1	No	Low
SVNFN	16	LS (average)	1	Yes	Low

^a Low for ASMOD.

10.7 Conclusions

Extensive comparison [65] of neurofuzzy algorithms such as ASMOD with SVM algorithms such as SUPANOVA on modelling the plupical and tensil properties of AL–Mg–Li powder metallurgy and two wrought AL–Zn–Mg–Cu alloys show that although they exhibit fundamental differences in supporting theory and implementation, yet both produce very similar submodels selections with comparable generalisation capabilities.

However, support vector machines can be interpreted as a maximum *a posteriori* prediction with a Gaussian prior, i.e. as a Gaussian process under

the Bayesian framework so that statistical quantities such as error bars on the resulting predictions can be determined from this probabilistic interpretation [81]. Also current research [80] into the use of frames in function Hilbert space offers new classes of kernel function determination that will improve model generalisation, complementary to this is research into the choice of kernels that reflect signals that exhibit many scales so that wavelet based kernels could be exploited. Whilst Table 10.2 suggests that SVMs with B-spline kernels offer the optimal model schema when the requirement is for transparency, speed of learning and accuracy, recent research on hybrid learning schemas such as the A-optimality criterion and Gram–Schmidt orthogonalisation offer still further improvement. Equally conventional parametric modelling schemas such as OLS combined with regularisation terms based upon functions of the error reduction ratio (see Section 6.6.2) are potentially as good as SVMs producing equally parsimonious models with minimum prediction errors. Despite the claims of SVM researchers' conventional parametric modelling research has plenty to offer as alternatives in data-based modelling in the future.

References

1. Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In: *The 2nd International Symposium on Information Theory*. Akademiai Kiado, Budapest. pp. 267–281.
2. Alford, A. and C. J. Harris (1998). Using B-splines to represent optimal actuator demands for systems featuring position/rate limited actuators. In: *Proc. of the 5th IFAC Workshop, AIARTC'98*. Mexico. pp. 271–276.
3. An, P. E. and C. J. Harris (1995). An intelligent driver warning system for vehicle collision avoidance. *IEEE Transactions on Systems, Man and Cybernetics* **26**(2), 254–261.
4. An, P. E., M. Brown and C. J. Harris (1995). A global gradient noise covariance expression for stationary real gaussian inputs. *IEEE Transactions on Neural Networks* **6**(6), 1549–1551.
5. An, P. E., M. Brown and C. J. Harris (1997). On the convergence rate of the normalised least mean square adaption. *IEEE Transactions on Neural Networks* **8**(5), 1211–1215.
6. An, P. E., M. Brown, C. J. Harris, A. J. Lawrence and C. G. Moore (1993). Comparative aspects of associative memory networks for modelling. In: *Proc. 2nd European Control Conference*, pp. 454–459.
7. Aronszajin, N. (1950). Theory of reproducing kernels. *Transactions of American Mathematics Society* **68**, 337–404.
8. Aström, K. J. and B. Wittenmark (1989). *Adaptive Control*. Addison Wesley, MS.
9. Atkinson, A. C. and A. N. Donev (1992). *Optimum Experimental Designs*. Clarendon Press, Oxford.
10. Avedyan, E. D., M. Brown and C. J. Harris (1995). A deterministic and statistical analysis of the modified NLMS rules. *Beltzer Journals* **30**, 14–43.
11. Bar-Shalom, Y. (1981). On the track-to-track correlation problem. *IEEE Transactions on Automatic Control* **26**(2), 571–572.
12. Bar-Shalom, Y. and L. Campo (1986). The effect of the common process noise on the two-sensor fused-track covariance. *IEEE Transactions on Aerospace and Electronic Systems* **22**(6), 803–805.
13. Bar-Shalom, Y. and T. E. Fortmann (1988). *Tracking and Data Association*. Academic Press, New York.
14. Bar-Shalom, Y. and X. R. Li (1995). *Multitarget–Multisensor Tracking: Principles and Techniques*. YBS, Storrs, Conn.
15. Barto, A. G., R. S. Sutton and C. H. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics* **13**(5), 834–846.
16. Belsley, D. (1991). *Conditioning Diagonistics: Collinearity and Weak Data in Regression*. Wiley, Chichester.

17. Berger, C. S. (1994). Linear splines with adaptive mesh sizes for modelling nonlinear dynamical systems. *Proc. IEE Control Theory and Applications* **141**(5), 77–284.
18. Billings, S. A. and Q. H. Tao (1991). Model validity tests for nonlinear signal processing applications. *International Journal of Control* **54**(1), 157–194.
19. Billings, S. A. and Q. M. Zhu (1994). Nonlinear model validation using correlation tests. *International Journal of Control* **60**, 1107–1120.
20. Billings, S. A. and Q. M. Zhu (1995). Model validation tests for multivariable nonlinear models including neural networks. *International Journal of Control* **62**, 749–766.
21. Billings, S. A. and W. S. F. Voon (1986). Correlation based model validity tests for nonlinear models. *International Journal of Control* **44**(1), 235–244.
22. Billings, S. A. and W. S. F. Voon (1987). Piecewise linear identification of nonlinear systems. *International Journal of Control* **46**, 215–235.
23. Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford.
24. Bossley, K. M. (1997). Neurofuzzy modelling approaches in system identification. PhD thesis. Dept of ECS, University of Southampton.
25. Bossley, K. M., M. Brown and C. J. Harris (1995). Neurofuzzy model construction for the modelling of nonlinear processes. In: *Proc. of the 3rd European Control Conference*. Vol. 3. Rome, Italy. pp. 2438–1443.
26. Bossley, K. M., M. Brown and C. J. Harris (1999). Neurofuzzy identification of an autonomous underwater vehicle. *International Journal of Systems Science* **30**(9), 901–913.
27. Box, G. E. P. and G. M. Jenkins (1976). *Time Series Analysis, Forecasting and Control*. Holden-Day, London.
28. Breiman, L. (1991). The \prod method for estimating multivariate functions from noisy data. *Technometrics* **33**(2), 125–153.
29. Bridgett, N. A. and C. J. Harris (1994). Neurofuzzy identification and control of a gas turbine jet engine. In: *Proc. International Symp. on Signal Processing, Robotics And Neural Networks*. Lille, France. pp. 462–467.
30. Bridgett, N. A., J. Brandt and C. J. Harris (1995). A neural network for use in the diagnosis and treatment of breast cancer. In: *Proc. of the 4th International Conference on ANN*. Cambridge, UK. pp. 448–453.
31. Broomhead, D. S. and D. Lowe (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems* **2**, 321–355.
32. Brown, M. and C. J. Harris (1994). *Neurofuzzy Adaptive Modelling and Control*. Prentice Hall, Hemel Hempstead.
33. Brown, M. and T. Ullrich (1997). Comparison of node insertion algorithms for Delaunay networks. In: *IMACS 2nd Mathmod*. Vol. 2. Vienna, Austria, pp. 775–780.
34. Brown, M., S. R. Gunn, C. N. Ng and C. J. Harris (1997). Neurofuzzy modelling: a transparent approach. In: *Dealing with Complexity: A Neural Network Approach* (K. Warwick, Ed.). pp. 110–124, Springer-Verlag, Berlin.
35. Buja, A., T. Hastie and R. Tibshirani (1989). Linear smoothers and additive models. *Annals of Statistics* **17**(2), 453–535.
36. Buntine, W. L. and A. S. Weigend (1991). Bayesian back-propagation. *Complex Systems* **5**, 603–643.
37. Casdagli, M. C. (1989). Nonlinear prediction of chaotic time series. *Physica D* **20**, 335–356.
38. Casdagli, M. C., S. Eubank, J. D. Farmer and J. Gibson (1991). State space reconstruction in the presence of noise. *Physica D* **51**, 52–98.

39. Chan, C. W., W. C. Chan, K. C. Cheung and C. J. Harris (2000). Support vector neurofuzzy networks for modelling nonlinear dynamic systems. Submitted to *International Journal of Systems Science*.
40. Chan, W. C., C. W. Chan, K. C. Cheung and C. J. Harris (2001). On the modelling of nonlinear dynamic systems using support vector machines. *Engineering and Applications of AI* **14**(2), 105–114.
41. Chang, K. C., R. K. Saha and Y. Bar-Shalom (1997). On optimal track-to-track fusion. *IEEE Transactions on Aerospace and Electronic Systems* **33**(4), 1271–1276.
42. Chang, K. C., Z. Tian and R. K. Saha (1998). Performance evaluation of track fusion with information filter. In: *Proc. of International Conference on Multisource-Multisensor Information Fusion*. USA. pp. 648–655.
43. Chen, F. C. and C. C. Liu (1994). Adaptively controlling nonlinear continuous time systems using multilayer neural networks. *IEEE Transactions on Automatic Control* **39**(6), 1306–1310.
44. Chen, F. C. and H. K. Khalil (1995). Adaptive control of a class of nonlinear discrete-time systems using neural networks. *IEEE Transactions on Automatic Control* **40**(5), 791–801.
45. Chen, S. and S. A. Billings (1992). Neural networks for nonlinear dynamic system modelling and identification. *International Journal of Control* **56**, 319–346.
46. Chen, S., C. F. Cowan and P. M. Grant (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks* **2**, 302–309.
47. Chen, S., S. A. Billings and W. Luo (1989). Orthogonal least squares methods and their applications to nonlinear system identification. *International Journal of Control* **50**, 1873–1896.
48. Chen, S., Y. Wu and B. L. Luk (1999). Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks. *IEEE Transactions on Neural Networks* **10**, 1239–1243.
49. Cherkassky, V. S. and F. M. Mulier (1999). *Learning from Data: Concepts, Theory, and Methods*. Wiley.
50. Chui, C. K. and G. Chen (1987). *Kalman Filtering: with Real-Time and Applications*. Springer-Verlag, Berlin.
51. Cox, E. (1994). *The Fuzzy Systems Handbook*. AP Professional.
52. Daubechies, I. (1992). Ten lectures on wavelets. In: *CBMS-NSF Regional Conference Series in Applied Mathematics*. Vol. 61. SIAM, Princeton, RI.
53. Dempster, A. P., N. M. Laird and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussions). *Journal of the Royal Statistical Society: B* **39**, 1–39.
54. Dixon, L. C. W. and R. Price (1986). Numerical experience with the truncated Newton method. Technical report. Numerical Optimisation Centre, Univ. Hertfordshire, Hatfield, UK.
55. Dodd, T. J. (2000). Prior knowledge for time series modelling. PhD thesis. Dept. of ECS, University of Southampton.
56. Dowdy, S. M. and S. Wearden (1991). *Statistics for Research*. Wiley, New York.
57. Doyle, R. S. (1996). Neurofuzzy multi-sensor data fusion for helicopter obstacle avoidance. PhD thesis. Dept of ECS, University of Southampton.
58. Doyle, R. S. and C. J. Harris (1996). Multi-sensor data fusion for helicopter guidance using neurofuzzy estimation algorithms. *The Royal Aeronautical Society Journal* **98**, 241–251.
59. Driankov, D., H. Hellendoorn and M. Reinfrank (1993). *An Introduction to Fuzzy Control*. Springer-Verlag, Berlin.

60. Durrant-Whyte, H. F. (1988). *Integration, Coordination and Control of Multi-Sensor Robot Systems*. Kluwer Academic, New York.
61. Ellacott, S. W. (1994). Aspects of the numerical analysis of neural networks. *Acta Numerica* **18**, 145–202.
62. Ergec, T., M. Pontil and T. Poggio (1999). A unified framework for regularisation networks and support vector machines. Technical report. MIT AI Lab Memo (BCL paper no. 171).
63. Farin, G. (1994). *Curves and Surfaces for Computer-aided Geometric Design: a Practical Guide*. Academic Press, Boston.
64. Femminella, O. P., M. J. Starink, M. Brown, I. Sinclair, C. J. Harris and P. A. S. Reed (1999). Data preprocessing/model initialisation in neurofuzzy modelling of structure property relationships in Al-Zn-Mg-Cu alloys. *ISIJ* **39**(10), 1027–1037.
65. Femminella, O. P. (2000). Neurofuzzy and SUPANOVA modelling of the processing-property relationships of aerospace Al-alloys. PhD thesis. Faculty of Engineering, University of Southampton.
66. Femminella, O. P., M. J. Starink, S. R. Gunn, C. J. Harris and P. A. S. Reed (2000). Neurofuzzy and SUPANOVA modelling of structural property relationships in Al-Zn-Mg-Cu alloys. In: *Proc. of 7th International Conference Aluminium Alloys, Material Science Forum*. Vol. 331–337. Charlottesville, Va. pp. 1255–1260.
67. Feng, M. and C. J. Harris (1998). Adaptive neurofuzzy control for a class of state-dependent nonlinear processes. *International Journal of Systems Science* **29**(7), 759–771.
68. Feng, M. and C. J. Harris (2001). Piecewise lyapunov stability conditions of fuzzy systems. *IEEE Transactions on Systems, Man and Cybernetics* **31:Part B**(1), 259–262.
69. Feraday, S. (2001). Intelligent approaches to modelling and interpreting disc brake squeal data. PhD thesis. Dept of ECS, University of Southampton.
70. Fletcher, R. (1987). *Practical Methods for Optimisation*. 2nd edn., Wiley, Chichester.
71. French, M. (1998). Adaptive control of functionally uncertain systems. PhD thesis, University of Southampton.
72. French, M. and E. Rogers (1997). Approximate models for adaptive feedback linearisation. *International Journal of Control* **68**(6), 1305–1323.
73. Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics* **19**(1), 1–141.
74. Gan, Q. and C. J. Harris (2001). A hybrid learning scheme combining EM and MASMOD algorithms for nonlinear system modelling. *IEEE Transactions on Neural Networks* **12**(1), 43–53.
75. Gan, Q. and C. J. Harris (2001). Comparison of two measurement fusion methods for Kalman-filter-based multisensor data fusion. *IEEE Transactions on Aerospace and Electronic Systems* **37**(1), 273–280.
76. Gan, Q. and C. J. Harris (1999). Fuzzy local linearisation and local basis function expansion in nonlinear system modelling. *IEEE Transactions on Systems, Man and Cybernetics* **29: Part B**(4), 559–565.
77. Gan, Q. and C. J. Harris (1999). Linearisation and state estimation of unknown discrete-time nonlinear dynamic systems using recurrent neurofuzzy networks. *IEEE Transactions on Systems, Man and Cybernetics* **29: Part B**(6), 802–817.
78. Gan, Q. and C. J. Harris (1999). Multisensor data fusion using Kalman filters based on linearised process models. In: *Proc. of International Conference on Data Fusion, EuroFusion 99*. UK. pp. 105–112.

79. Gan, Q. and C. J. Harris (1999). Neurofuzzy state estimators using a modified ASMOD and Kalman filter algorithm. In: *Proc. of the International Conference on Computational Intelligence for Modelling Control and Automation*. Vol. 1. Vienna, Austria. pp. 214–219.
80. Gao, J. B., C. J. Harris and S. R. Gunn (2001). On a class of support vector kernels based on frames in function Hilbert spaces. *Neural Computation* **13**, 1–20.
81. Gao, J. B., S. R. Gunn and C. J. Harris (2001). Mean field theory for the SVM regression. ISIS Technical Report, Department of Electronics and Computer Science, University of Southampton. Submitted to *IEEE Transactions on Neural Networks*.
82. Geman, S., E. Bienenstock and R. Doursat (1992). Neural networks and the bias/variance dilemma. *Neural Computation* **4**, 1–58.
83. Gershenfeld, N., B. Schoner and E. Metois (1999). Cluster-weighted modeling for time-series analysis. *Nature* **397**, 329–332.
84. Gill, P. E., W. Murray and M. H. Wright (1993). *Practical Optimisation*. Academic Press, San Diego.
85. Girosi, F., M. Jones and T. Poggio (1995). Regularisation theory for neural networks. *Neural Computation* **7**, 219–269.
86. Golub, G. H., M. Heath and G. Wahba (1979). Generalized cross-validation as a method for choosing good ridge parameter. *Technometrics* **21**(2), 215–223.
87. Goodwin, G. C. and K. S. Sin (1984). *Adaptive Filtering Prediction and Control*. Prentice Hall, Englecliff, NJ.
88. Grime, S. and H. Durrant-Whyte (1994). Data fusion in decentralised sensor networks. *Control Engineering Practice* **2**(5), 849–863.
89. Gull, S. F. (1989). Developments in maximum entropy data analysis. In: *Maximum Entropy and Bayesian Methods* (J. Skilling, Ed.). pp. 53–71, Kluwer Academic.
90. Gunn, S. R. and J. S. Kandola (2001). Structural modelling with sparse kernels. *Machine Learning*, Accepted.
91. Gunn, S. R. and M. Brown (1999). SUPANOV – a sparse transparent modelling approach. In: *Proc. of IEEE International Workshop on Neural Networks for Signal processing*. Madison, Wis., USA.
92. Gunn, S. R., M. Brown and K. Bossley (1997). Network performance assessment for neurofuzzy data modelling. In: *Intelligent Data Analysis*, Lecture Notes in Computer Science, pp. 313–323, Springer, Berlin.
93. Hall, M. and C. J. Harris (1998). Neurofuzzy systems for command and control. In: *Proc. of the C² Research and Technology Symposium*. Monterey, Naval PGS. pp. 461–471.
94. Hardier, G. (1997). Recurrent neural networks for ship modelling and control. In: *Proc. of the 7th Ship Control Systems Symposium*. Vol. 1. pp. 39–61.
95. Harris, C. J., A. Bailey and T. J. Dodd (1998). Multi-sensor data fusion in defence and aerospace. *The Royal Aeronautical Journal* **102**(1015), 229–244.
96. Harris, C. J. and P. E. An (1998). An intelligent driver warning system. In: *Knowledge Based Intelligent Techniques in Industry*. pp. 1–52, CRC Press, Boca Raton.
97. Harris, C. J. and Q. Gan (2001). State estimation and multisensor data fusion using data-based neurofuzzy local linearisation process models. *Information Fusion* **2**(1), 17–29.
98. Harris, C. J. and X. Hong (2000). Neurofuzzy network model construction using Bézier–Bernstein polynomial functions. *Proc. IEE Control Theory and Applications* **147**, 337–343.

99. Harris, C. J. and X. Hong (2001). A parallel learning and construction algorithm. In: *Proc. of SPIE—Applications and Science of Computational Intelligence II*. Orlando, USA.
100. Harris, C. J., C. G. Moore and M. Brown (1993). *Intelligent Control: Some Aspects of Fuzzy Logic and Neural Networks*. World Scientific Press, Singapore.
101. Harris, C. J., X. Hong and M. Feng (1999). Optimal piecewise locally linear modelling. In: *Proc. of SPIE – Applications and Science of Computational Intelligence II*. Vol. 3722. Orlando, USA. pp. 486–493.
102. Harris, C. J., X. Hong and P. A. Wilson (1999). An intelligent guidance and control system for ship obstacle avoidance. *Proc. IMechE Part I, Journal of Systems and Control Engineering* **213**, 311–320.
103. Hastie, T. J. and R. J. Tibshirani (1996). *Generalised Additive Models*. Chapman and Hall, London.
104. Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall.
105. Hong, X. and C. J Harris (2000). Generalised neurofuzzy network modelling algorithms using Bézier–Bernstein polynomial functions and additive decomposition. *IEEE Transactions on Neural Networks* **11**(1), 889–902.
106. Hong, X. and C. J Harris (2001). A mixture of experts network structure construction algorithm for modelling and control. *Applied Intelligence*, In Press.
107. Hong, X. and C. J Harris (2001). Nonlinear model structure detection using optimum experimental design and orthogonal least squares. *IEEE Transactions on Neural Networks* **12**(2), 425–439.
108. Hong, X. and C. J Harris (2001). Neurofuzzy design and model construction of nonlinear dynamical processes from data. *Proc. IEE Control Theory and Applications*, to appear.
109. Hong, X. and C. J Harris (2001). Neurofuzzy networks knowledge extraction and extended Gram–Schmidt algorithm for subspace decomposition. Technical Report, ISIS Group, Dept of Electronics and Computer Science, University of Southampton.
110. Hong, X. and C. J Harris (2001). Variable selection algorithm for the construction of MIMO operating point dependent neurofuzzy network. *IEEE Transactions on Fuzzy Systems* **9**, 88–101.
111. Hong, X., C. J. Harris and P. A. Wilson (1999). Neurofuzzy state identification using prefiltering. *Proc. IEE Control Theory and Applications* **146**, 234–240.
112. Horn, R. A. and Johnson J. R (1985). *Matrix Analysis*. Cambridge University Press, Cambridge.
113. Huber, P. (1985). Projection pursuit. *Annals of Statistics* **13**, 1465–1481.
114. Hunt, K. J., R. Haas and M. Brown (1995). On the functional equivalence of fuzzy inference systems and spline-based networks. *International Journal of Neural Systems* **6**(2), 171–184.
115. Ingber, L. (1992). Genetic algorithms and very fast simulated reannealing: a comparison. *Mathematical and Computer Modelling* **11**, 87–100.
116. Isidori, A. (1989). *Nonlinear Control Systems*. Springer-Verlag, Berlin.
117. Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. *IEEE Transactions on Systems, Man and Cybernetics* **1**(4), 364–378.
118. Jacobs, R. A., M. I. Jordan, S. J. Nowlan and G. E. Hint (1991). Adaptive mixtures of local experts. *Neural Computation* **3**, 79–87.
119. Jagannathan, S. and F. L. Lewis (1996). Multilayer discrete-time neural net controller with guaranteed performance. *IEEE Transactions on Neural Networks* **7**(1), 107–130.

120. Jagannathan, S. and F. L. Lewis (1997). Feedback linearisation in discrete-time using neural networks. In: *Proc. of the 12th IEEE International Symp. on Intelligent Control*. Istanbul, Turkey. pp. 181–186.
121. Jang, J. S. R. (1993). ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Transactions on Systems, Man and Cybernetics* **23**, 665–685.
122. Jang, J. S. R., C.T. Sun and E. Mizutani (1997). *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. Upper Saddle River, Prentice Hall, NJ.
123. Jin, L., P. N. Nikiforuk and M. M. Gupta (1995). Approximation of discrete-time state-space trajectories using dynamic recurrent neural networks. *IEEE Transactions on Automatic Control* **40**(7), 1266–1270.
124. Johansen, T. A. and B. A. Foss (1993). Constructing NARMAX models using ARMAX models. *International Journal of Control* **58**, 1125–1153.
125. Jordan, M. and L. Xu (1995). Convergence results for the EM approach to mixtures of experts' architectures. *Neural Networks* **8**(9), 1409–1431.
126. Jordan, M. and R. A. Jacobs (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation* **6**, 181–214.
127. Kanellakopoulos, I., P. V. Kokotovic and A. S. Morse (1991). Systematic design of adaptive controllers for feedback linearisable systems. *IEEE Transactions on Automatic Control* **36**(11), 1241–1253.
128. Karlsson, M., A. Malmberg, T. Jensen and L. Axelsson (1999). Track fusion algorithms in decentralised tracking systems with feedback in a flight aircraft application. In: *Proc. of FUSION'99*. USA. pp. 733–740.
129. Kavli, T. (1993). ASMOD – an algorithm for adaptive spline modelling of observation data. *International Journal of Control* **58**(4), 947–967.
130. Kim, Y. H., F. L. Lewis and C. T. Abdallah (1997). A dynamic recurrent neural-network-based adaptive observer for a class of nonlinear systems. *Automatica* **33**(8), 1539–1543.
131. Krstic, M. and P. V. Kokotovic (1996). Adaptive nonlinear output-feedback schemes with Marino–Tomei controller. *IEEE Transactions on Automatic Control* **41**(2), 274–280.
132. Lane, S. H. (1992). Theory and development of higher order CMAC neural networks. *IEEE Control Systems Magazine* **12**, 23–30.
133. Leontaritis, I. J. and S. A. Billings (1985). Input–output parametric models for nonlinear systems – part 1: deterministic nonlinear systems; part 2: stochastic nonlinear systems. *International Journal of Control* **41**(1), 303–344.
134. Leontaritis, I. J. and S. A. Billings (1987). Model selection and validation methods for nonlinear systems. *International Journal of Control* **45**(1), 311–341.
135. Lin, C. T. (1996). *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*. Prentice Hall, Upper Saddle River, NJ.
136. Liu, C. C. and F. C. Chen (1993). Adaptive control of nonlinear continuous time systems using neural networks – general relative degree and MIMO cases. *International Journal of Control* **58**, 317–355.
137. Ljung, L. (1987). *System Identification: Theory for the User*, Prentice Hall.
138. Lo, S. H. (1989). Delaunay triangulation of nonconvex plan domains. *International Journal of Numerical Methods in Engineering* **28**, 2695–2707.
139. Mackay, D. J. (1997). Gaussian processes: a replacement for supervised neural networks. In: *Lecture Notes at NIPS 1997*. <http://www.ra.phy.cam.ac.uk/mackay>.
140. MacKay, D. J. C. (1991). Bayesian methods for adaptive models. PhD thesis. California Institute of Technology, USA.

141. Manyika, J. and H. Durrant-Whyte (1994). *Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach*. Ellis Horwood, New York.
142. Marenbach, P. and M. Brown (1997). Evolutionary versus inductive construction of neurofuzzy systems. In: *Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97)*. Glasgow, UK.
143. Marino, R. and P. Tomei (1992). Global adaptive observers for nonlinear systems via filtered transformations. *IEEE Transactions on Automatic Control* **37**(8), 1239–1245.
144. Marino, R. and P. Tomei (1993). Global adaptive output-feedback control of nonlinear systems, part I: linear parameterisation. *IEEE Transactions on Automatic Control* **38**(1), 17–32.
145. Marino, R. and P. Tomei (1993). Global adaptive output-feedback control of nonlinear systems, part II: nonlinear parameterisation. *IEEE Transactions on Automatic Control* **38**(1), 33–48.
146. Marino, R. and P. Tomei (1995). Adaptive observers with arbitrary exponential rate of convergence for nonlinear systems. *IEEE Transactions on Automatic Control* **40**(7), 1300–1304.
147. Michie, D. and R. A. Chambers (1968). Boxes: an experiment in adaptive control. In: *Machine Intelligence 2* (E. Dale and D. Michie, Eds.). pp. 137–152. Oliver and Boyd, Edinburgh.
148. Miller, W. T., R.S. Sutton and P. J. Werbos (Eds.) (1990). *Neural Networks for Control*. MIT Press, Cambridge, MA.
149. Moody, J. (1994). Prediction risk and architecture selection for neural networks. In: *From Statistics to Neural Networks: Theory and Pattern Recognition and Applications* (V. Cherkassky, Ed.). Springer-Verlag, Berlin.
150. Moore, C. G. and C. J. Harris (1994). Aspects of fuzzy control and estimation. In: *Advances in Intelligent Control* (C. J. Harris, Ed.). pp. 201–242. Taylor & Francis, London.
151. Murray-Smith, R. (1994). A local model network approach to nonlinear modelling. PhD thesis. Dept. of Computer Science, University of Strathclyde.
152. Murray-Smith, R. and T. A. Johansen (1997). *Multiple Model Approaches to Modelling and Control*. Taylor & Francis, London.
153. Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics **118**, Springer, New York.
154. Nelles, O., S. Sinsel and R. Isermann (1996). Local basis function networks for identification of a turbocharger. In: *IEE UKACC Control'96*. Exeter.
155. Netter, J., M. H. Kutner, C. J. Nathtschein and W. Wasserman (1985). *Applied Linear Statistical Models*. Irwin, London.
156. Oinohundro, S. M. (1989). The Delaunay triangulation and function learning. Technical report. Berkeley Tech. Report, 90-001.
157. Orr, M. J. L. (1995). Regularisation in the selection of radial basis function centers. *Neural Computation* **7**(3), 954–975.
158. Papoulis, A. (1965). *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York.
159. Parzen, E. (1967). *Time Series Papers*. Holden Day, San Francisco.
160. Pearlmutter, B. A. (1995). Gradient calculations for dynamic recurrent neural networks: a survey. *IEEE Transactions on Neural Networks* **6**(5), 1212–1228.
161. Pedrycz, W. (1993). *Fuzzy Control and Fuzzy Systems*. 2nd edn., Research Studies Press, Wiley, Taunton.
162. Pineda, F. J. (1987). Generalisation of back-propagation to recurrent neural networks. *Physical Review Letters* **59**(19), 2229–2232.

163. Poggio, T., V. Torre and C. Koch (1985). Computational vision and regularisation theory. *Nature* **317**(26), 314–319.
164. Priestley, M. B. (1988). *Nonlinear and Nonstationary Time Series Analysis*. Academic Press, London.
165. Rao, B. and H. F. Durrant-Whyte (1993). A decentralised Bayesian algorithm for identification of tracked targets. *IEEE Transactions on Systems, Man and Cybernetics* **23**(6), 1683–1698.
166. Rao, B., H. F. Durrant-Whyte and A. Sheen (1991). A fully decentralised multi-sensor system for tracking and surveillance. *International Journal of Robotics Research* **12**(1), 20–45.
167. Roecker, J. A. and C. D. McGillem (1988). Comparison of two-sensor tracking methods based on state vector fusion and measurement fusion. *IEEE Transactions on Aerospace and Electronic Systems* **24**(4), 447–449.
168. Saha, R. K. (1996). Track-to-track fusion with dissimilar sensors. *IEEE Transactions on Aerospace and Electronic Systems* **32**(3), 1021–1029.
169. Saha, R. K. and K. C. Chang (1998). An efficient algorithm for multisensor track fusion. *IEEE Transactions on Aerospace and Electronic Systems* **34**(1), 200–210.
170. Saitoh, S. (1988). *Theory of Reproducing Kernels and its Applications*. Longman, Harlow, Essex.
171. Sakamoto, Y., M. Ishguri and G. Kitagawa (1986). *Akaike Information Criterion Statistics*. D. Reidel, Tokyo.
172. Sanner, R. and J. Slotine (1992). Gaussian networks for direct adaptive control. *IEEE Transactions on Neural Networks* **3**(6), 837–863.
173. Sastry, S. and A. Isidori (1989). Adaptive control of linearisable systems. *IEEE Transactions on Automatic Control* **34**, 1123–1131.
174. Scholkopf, B., P. Barlett and A. J. Smola (1999). Shrinking the data model: a new support vector regression algorithm. In: *Advances in Neural Information Processing Systems* (M. S. Kearns and S.A. Solla, Eds.). MIT Press, Cambridge, Mass.
175. Sinha, N. K. and M. M. Gupta (Eds.) (2000). *Soft Computing & Intelligent Systems: Theory & Applications*. Academic Press.
176. Sjoberg, J., Q. Zhang and L. Ljung (1995). Nonlinear black-box modelling in system identification: a unified overview. *Automatica* **31**(12), 1691–1724.
177. Sjoberg, J., Q. Zhang and L. Ljung (1995). Nonlinear black-box modelling in system identification: mathematical foundations. *Automatica* **31**(12), 1725–1752.
178. Shewchuk, J. R. (1994). An introduction to conjugate gradient method without the agonizing pain. Tech. Report CMU-CS-94, Carnegie Mellon University.
179. Slotine, J. and W. Li (1991). *Applied Nonlinear Control*. Prentice Hall, Englewood Cliffs, NJ.
180. Smola, A. J. (1998). Learning with kernels. PhD thesis. Informatik der Technischen Universität Berlin.
181. Soderström, T. and P. Stoica (1989). *System Identification*. Prentice Hall.
182. Sontag, E. (1997). Recurrent neural networks: some systems-theoretic aspects. In: *Dealing with Complexity: a Neural Network Approach* (M. Karny, K. Warwick and V. Kurkova, Eds.). pp. 1–12. Springer-Verlag, London.
183. Stone, C. J., M. H. Hansen, C. Kooperberg and Y. K. Truong (1997). 1994 Wald Memorial Lecture: Polynomial splines and their tensor products in extended linear modeling. *The Annals of Statistics* **25**(4), 1371–1470.
184. Sun, C. (1994). Rule based structure identification in an adaptive network based inference system. *IEEE Transactions on Fuzzy Systems* **2**(1), 64–73.

185. Takagi, T. and M. Sugeno (1985). Fuzzy identification of systems and its applications to modelling and control. *IEEE Transactions on Systems, Man and Cybernetics* **15**, 116–132.
186. Takens, F. (1981). Detecting strange attractors in turbulence. In: *Dynamical Systems and Turbulence* (D. A. Rand and L. S. Young, Eds.). *Lecture Notes in Mathematics* **898**, pp. 336–381, Springer-Verlag, Berlin.
187. Tolle, H. and E. Ersu (1992). *Neurocontrol: Learning Control Systems Inspired by Neuronal Architectures and Human Problem Solving*. Lecture Notes in Control and Information Sciences, **172**, Springer-Verlag, Berlin.
188. Tong, R. M. (1977). A control engineering review of fuzzy systems. *Automatica* **13**, 559–569.
189. Tran, T. and C. J. Harris (1999). Maritime avoidance navigation, totally integrated systems (MANTIS). In: *Proc. IEEE Fusion 99*. Vol. 2. pp. 394–401.
190. Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.
191. Vapnik, V. (1998). *Statistical Learning Theory: Adaptive Learning Systems for Signal Processing, Communication and Control*. Wiley, Chichester.
192. Verbruggen, H. B. and R. Babuska (1999). *Fuzzy Logic Advances in and Applications*. Series in Robotics and Intelligent Systems, Vol.23, World Scientific, Singapore.
193. Vidyasagar, M. (1993). *Nonlinear Systems Analysis*. Prentice Hall, Saddle River, NJ.
194. Wahba, G. (1990). *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics, Princeton, RI.
195. Wang, H., G. P. Liu, C. J. Harris and M. Brown (1995). *Advanced Adaptive Control*. Pergamon Press, Oxford.
196. Wang, H., M. Brown and C. J. Harris (1996). Modelling and control of nonlinear, operating point dependent systems via associative memory networks. *Journal of Dynamics and Control* **6**, 199–218.
197. Wang, L. X. (1997). *A Course in Fuzzy Systems and Control*. Prentice Hall, Englewood Cliffs, NJ.
198. Wang, L.X. (1994). *Adaptive Fuzzy Systems and Control: Design and Analysis*. Prentice Hall, Englewood Cliffs, NJ.
199. Webb, A. (1993). An approach to nonlinear principal components analysis using radial basis functions. *Defence Research Agency Malvern Memorandum*.
200. Webb, A. (1999). *Statistical Pattern Recognition*. Arnold, London.
201. Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proc. of the IEEE* **78**(10), 1550–1560.
202. Werntges, H. W. (1993). Partitions of unity improve neural functional approximation. In: *Proc. IEEE International Conference Neural Networks*. Vol. 2. San Francisco, USA. 914–918.
203. Williams, R. J. and D. Zipser (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* **1**(2), 270–280.
204. Williams, R. J. and D. Zipser (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In: *Back-Propagation: Theory, Architectures and Applications* (Y. Chauvin and D. E. Rumelhart, Eds.), Erlbaum, Hillsdale, N.J.
205. Williams, R. J. and J. Peng (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation* **2**(4), 490–501.
206. Wolkenhauer, O. (1998). *Possibility Theory with Application to Data Analysis*. Research Studies Press, Wiley.

207. Wu, Z.-Q. and C. J. Harris (1997). A neurofuzzy network structure for modelling and state estimation of unknown nonlinear systems. *International Journal of Systems Science* **28**, 335–345.
208. Xu, L. and M. I. Jordan (1996). On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation* **8**, 129–151.
209. Yeslidirek, A. and F. L. Lewis (1995). Feedback linearisation using neural networks. *Automatica* **31**, 1659–1664.
210. Zhang, Q. (1993). Regressor selection and wavelet network construction. Technical Report, IRISA, Campus universitaire de Beaulieu, 35042 Rennes.
211. Zheng, G. L. and S. A. Billings (2001). Effects of overparameterisation in nonlinear system identification and neural networks. *Submitted to International Journal of System Science*.
212. Zimmermann, H. J. (1996). *Fuzzy Set Theory and its Applications*. Kluwer Academic, Boston.

Index

- adaptive
 - fuzzy systems, 72
 - learning rates, 70
 - modelling history, 21
 - networks based fuzzy inference systems (ANFIS), 155
 - spline modelling algorithm (ASMOD), 107, 226
- additive
 - neurofuzzy modelling algorithms, 106
 - splines, 40
- Akaike information criterion (AIC), 144
- algebraic sum operator, 85
- analysis of variance (ANOVA), 18, 106
- a posteriori* probability, 32
- A-optimality
 - design criterion, 145
 - neurofuzzy model construction (NeuDec) algorithm, 143
- approximation error, 9
- autocorrelation matrix, 56
- α -cut, 77
- backpropagation algorithm, 214
- Barycentric coordinates, 204
- basis functions, 14
 - expansion, global, local and radial, 30
- batch learning laws, 58
- Bayesian regularisation, 129
- Bayes theorem, 32, 129
- bias-variance dilemma, 33
- bivariate
 - Bernstein polynomial, 215
 - de Casteljau algorithm, 215
- B-splines, 30, 41
 - membership function, 79
- Bézier–Bernstein
 - polynomial models, 18, 41
 - modelling network, 209
 - model construction algorithm, 219
 - triangle patch, 216
- canonical state space representations, 19
- centre of gravity defuzzification, 90
- compact support sets, 14, 77
- condition number, 15, 36, 57
- correlation tests, 45
- cross validation, 44, 105
- curse of dimensionality, 17, 73, 156
- cyclic modelling, 7
- data
 - acquisition, 2
 - preprocessing, 3
- de Casteljau algorithm, 212
- decentralised data fusion, 256
 - architecture, 262, 265
- defuzzification, 90
 - mean of maxima, 90

- centre of gravity, 90
- defuzzifier, 75
- Delaunay
 - input space partitioning, 204
 - partitioning modelling, 201
 - simplex, 203
 - triangulation, 18, 202
- direct state estimation method, 169
- divide and conquer principle, 18
- Einstein's principle of simplicity, 17
- empirical
 - risk, 5
 - minimisation principle (ERM), 10
 - loss function, 33
- error
 - bar, 132
 - surface, 55
- estimation error, 10
- expectation–maximisation (EM)
 - algorithm, 226, 237, 237
 - extended
 - additive neurofuzzy model, 119, 120
 - barycentric coordinates, 217
- feature space, 284
- feedback linearisation, 227
 - neurofuzzy models, 239
 - parametric feedback form, 241
- feedforward Gram–Schmidt OLS procedure, 191
- final prediction error, 43, 144
- first order infinite splines, 299
- forward constrained regression (FCR), 177
- functional equivalence of outputs
- augmented fusion (OAF) and optimal weighting measurement fusion (OWMF), 260
- fuzzification, 89
- fuzzifier, 75
- fuzzy
 - boxtrees, 161
 - data clustering algorithm, 155
 - implication, 85
 - intersection, 83
 - membership function, 75
 - modelling, 71
 - operators, 83
 - relation surface, 87
 - set, 75
 - support, 77
 - systems, 74
 - union, 84
 - variable, 78
- Gaussian
 - membership functions, 82
 - radial basis function (RBF), 41, 285
- general
 - learning laws, 58
 - recurrent neural network, 243
- generalisation, 1, 13, 72
- generalised linear models, 6
- global basis functions, 30
- gradient
 - descent algorithms, 59, 237
 - noise, 68
- Gram–Schmidt OLS algorithm, 193
- grey box models, 4
- group method of data handling (GMDH), 104
- growing and pruning model construction, 107
- Gustafson–Kessel algorithm, 155
- Hessian matrix, 59, 123
- hierarchical
 - multisensor data fusion, 266
 - neurofuzzy model, 125
 - hinging hyperplanes, 30
 - hybrid
 - hierarchical multisensor data fusion architecture, 20

- learning scheme, 236, 239
- tensor/additive splines, 41
- hypothesis testing, 6, 43

- ill-posed problems, 5, 143
- “incomplete-data” log likelihood, 238
- independent component analysis (ICA), 17
- index vector, 146
- indirect state estimation methods, 169
- inference engine, 75
- inferencing, 88
- information
 - filter, 249
 - measures, 43
- instantaneous learning laws, 54, 61
- intelligent or adaptive modelling, 7
- input-output models, 26
- introduction to modelling, 1
- inverse procedure of the de Casteljau algorithm, 209
- iterative
 - neurofuzzy model construction, 104 (cyclic), 7

- Kalman filter, 20, 169, 248, 255
- Karush–Kuhn–Tucker (KKT) algorithm, 288
- kernel functions, 6, 30, 281, 284
- k-d trees, 17
- knot insertion/removal, 109
- knowledge base, 74

- lattice based associative networks, 15
- learning theory, 9
 - laws, 53, 58
 - least mean squares, 61
- likelihood function, 32, 129
- linear smoother, 36

- linguistic vagueness, 71
- local
 - basis functions, 30
 - basis function expansion (LBFE), 230
 - linear model tree (LOLIMOT) algorithm, 159
 - model, 136
 - neurofuzzy modelling, 153
 - regularised neurofuzzy loss functions, 11, 284

- Mamdani model, 155
- maximum
 - a posteriori* (MAP) estimate, 32
 - likelihood (ML) estimate, 33, 54, 236
 - likelihood (ML) identification, 129
- max-NLMS, 64
- mean
 - of maxima (MOM), 90
 - squared error, 27
- measurement fusion methods, 255, 258
- minimal capture zone, 66, 69
- minimum description length, 44
- mixture of experts
 - algorithm, 18, 227
 - modelling, 173, 204, 236
- model
 - approximation error, 9
 - bias, 34
 - construction algorithm, 143
 - cross validation, 44, 105
 - identification, 3
 - parameter estimation, 31
 - parsimony, 34
 - quality, 33
 - regressor order, 14
 - selection criteria sensitivity, 44
 - selection methods, 42
 - transparency, 3, 73
 - structural regularisation, 34

- validity tests, 46
- variance, 34
- modified adaptive spline modelling (MASMOD) algorithm, 244
- multi-dimensional kernels, 286
- multilayer perceptron (MLP), 55, 41
- multisensor data fusion, 20, 255

- neurofuzzy
 - feedback linearisation, 239
 - Kalman filtering, 173
 - local function basis expansion models, 230
 - local linearisation (NFLL), 225
 - modelling, 71
 - models, 91
 - network, 94
 - conditioning, 15, 36, 57
 - regularised model, 129
 - state estimators, 245
 - systems, 72
- Newton
 - algorithm, 122
 - Raphson algorithm, 59
- nonlinear
 - affine systems, 240
 - autoregressive models (NARX), 28
 - autoregressive moving average models (NARMAX), 28
 - finite impulse response models, 28
 - output error (NOE) model, 28
 - regression approximation, 288
- normal fuzzy set, 77
- normalised
 - condition numbers, 68
 - least mean squares (NLMS)
 - weight convergence, 63
 - learning, 62, 166, 170

- operating point neurofuzzy model, 164
- optimal weighting measurement fusion (OWMF), 259
- orthogonal least squares (OLS) algorithm, 143, 177
- over-determined learning, 13
- Occam's razor, 15
- output
 - augmented fusion (OAF), 259
 - feedback linearisation, 19, 241
 - measurement fusion, 20

- parametric–pure–feedback form, 241
- parsimonious
 - neurofuzzy modelling, 103
 - parallel modelling algorithm, 183
- partition of unity, 78
- polynomial
 - kernels, 41, 286
 - Bézier–Bernstein, 41, 215
 - B-splines, 30, 41, 286
- principal component analysis (PCA), 17, 144
- priors for neurofuzzy model, 133
- problem
 - computational, 282
 - conceptual, 282

- quad-trees models, 17

- radial
 - basis functions (RBF), 30
 - construction, 30
 - recurrent neural network, 242
 - recursive least squares estimation, 67, 166
 - regularisation, 5, 36
 - Bayesian, 129
 - function, 283
 - networks, 39
 - techniques, 106, 143

- regularised neurofuzzy model, 129
- regulariser coefficient, 35
- regression matrix, 27
- reproducing kernels, 39
 - Hilbert spaces (RKHS), 6, 39, 281
- ridge construction, 30
- Robbins–Munro stochastic approximation algorithm, 70
- rule
 - confidence, 86
 - completeness, 78
- sgn-NLMS, 65
- sigmoidal neural networks, 30
- simplexes, 18, 203
 - Delaunay, 203
- singular value decomposition (SVD), 143
- slack variables, 287
- splines, 41, 286
 - tensor/additive, 41
- stagewise construction, 110
- state
 - direct estimation method, 169
 - feedback linearisation, 240
 - indirect state estimation methods, 169
 - space models, 26
 - space representations of neurofuzzy models, 168
 - vector fusion methods, 255, 263
- vector assimilation fusion (SVAF), 263
- vector fusion, 20
- statistical
 - learning theory, 10
 - significance metrics, 105
- steepest descent, 60
- strict feedback system, 241
- structural
 - risk minimisation, 11, 226, 230, 245, 282
 - regularisation, 5
- support vectors, 288
- analysis of variance (SUPANOVA) algorithm, 20, 297, 298
- machine, 6, 282
- models, 281, 286, 289
- neural networks, 39
- neurofuzzy network
- S-norm, 85
- Takagi–Sugeno (T-S) models
 - fuzzy type, 18
 - local neurofuzzy model, 95, 155, 181, 225
- target tracking, 20
- tensor
 - multiplication/tensor
 - submodel splitting, 108
 - product, 30, 286
 - product splines, 40
- track-to-track fusion, 264
- transparency, 3, 73
- Tsukamoto model, 155
- T-norm, 83
- under-determined learning, 13
- unimodal fuzzy sets, 76
- univariate Bézier–Bernstein polynomials, 209
- validation/verification, 3
- Vapnik–Chervonenkis (VC), 11
- very fast simulated reannealing (VFSR) algorithm, 206
- wavelets, 30
- weighting
 - function, 204
 - identification, 122
- Wiener and Hammerstein models, 28