UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

UNIVERSIDADE DO ALGARVE

DESIGN OF NEURO-FUZZY MODELS BY EVOLUTIONARY AND GRADIENT-BASED ALGORITHMS

Cristiano Lourenço Cabrita

(Mestre em Engenharia de Sistemas e Computação)

Dissertação para obtenção do

Grau de Doutor em Engenharia Electrónica e Telecomunicações

(Ramo especialização em Sistemas Inteligentes)

Trabalho efetuado sob a orientação de:

António Eduardo Barros Ruano

2013

DESIGN OF NEURO-FUZZY MODELS BY EVOLUTIONARY AND GRADIENT-BASED ALGORITHMS

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Cristiano Lourenço Cabrita

Resumo

Todos os sistemas encontrados na natureza exibem, com maior ou menor grau, um comportamento linear. De modo a emular esse comportamento, as técnicas de identificação clássicas usam, tipicamente e por simplicidade matemática, modelos lineares. Devido à sua propriedade de aproximação universal, modelos inspirados por princípios biológicos (redes neuronais artificiais) e motivados linguisticamente (sistemas difusos) tem sido cada vez mais usados como alternativos aos modelos matemáticos clássicos.

Num contexto de identificação de sistemas, o projeto de modelos como os acima descritos é um processo iterativo, constituído por vários passos. Dentro destes, encontra-se a necessidade de identificar a estrutura do modelo a usar, e a estimação dos seus parâmetros. Esta Tese discutirá a aplicação de algoritmos baseados em derivadas para a fase de estimação de parâmetros, e o uso de algoritmos baseados na teoria da evolução de espécies, algoritmos evolutivos, para a seleção de estrutura do modelo. Isto será realizado no contexto do projeto de modelos neuro-difusos, isto é, modelos que simultaneamente exibem a propriedade de transparência normalmente associada a sistemas difusos mas que utilizam, para o seu projeto algoritmos introduzidos no contexto de redes neuronais. Os modelos utilizados neste trabalho são redes B-Spline, de Função de Base Radial, e sistemas difusos dos tipos Mamdani e Takagi-Sugeno.

Neste trabalho começa-se por explorar, para desenho de redes B-Spline, a introdução de conhecimento à-priori existente sobre um processo. Neste sentido, aplica-se uma nova abordagem na qual a técnica para a estimação dos parâmetros é alterada a fim de assegurar restrições de igualdade da função e das suas derivadas. Mostra-se ainda que estratégias de determinação de estrutura do modelo, baseadas em computação evolutiva ou em heurísticas determinísticas podem ser facilmente adaptadas a este tipo de modelos restringidos.

É proposta uma nova técnica evolutiva, resultante da combinação de algoritmos recentemente introduzidos (algoritmos bacterianos, baseados no fenómeno natural de evolução microbiana) e programação genética. Nesta nova abordagem, designada por programação bacteriana, os operadores genéticos são substituídos pelos operadores bacterianos. Deste modo, enquanto a mutação bacteriana trabalha num indivíduo, e tenta otimizar a bactéria que o codifica, a transferência de gene é aplicada a toda a população de bactérias, evitando-se soluções de mínimos locais. Esta heurística foi aplicada para o desenho

de redes B-Spline. O desempenho desta abordagem é ilustrada e comparada com alternativas existentes.

Para a determinação dos parâmetros de um modelo são normalmente usadas técnicas de otimização locais, baseadas em derivadas. Como o modelo em questão é não-linear, o desempenho deste género de técnicas é influenciado pelos pontos de partida. Para resolver este problema, é proposto um novo método no qual é usado o algoritmo evolutivo referido anteriormente para determinar pontos de partida mais apropriados para o algoritmo baseado em derivadas. Deste modo, é aumentada a possibilidade de se encontrar um mínimo global.

A complexidade dos modelos neuro-difusos (e difusos) aumenta exponencialmente com a dimensão do problema. De modo a minorar este problema, é proposta uma nova abordagem de particionamento do espaço de entrada, que é uma extensão das estratégias de decomposição de entrada normalmente usadas para este tipo de modelos. Simulações mostram que, usando esta abordagem, se pode manter a capacidade de generalização com modelos de menor complexidade.

Os modelos B-Spline são funcionalmente equivalentes a modelos difusos, desde que certas condições sejam satisfeitas. Para os casos em que tal não acontece (modelos difusos Mamdani genéricos), procedeu-se à adaptação das técnicas anteriormente empregues para as redes B-Spline. Por um lado, o algoritmo Levenberg-Marquardt é adaptado e a fim de poder ser aplicado ao particionamento do espaço de entrada de sistema difuso. Por outro lado, os algoritmos evolutivos de base bacteriana são adaptados para sistemas difusos, e combinados com o algoritmo de Levenberg-Marquardt, onde se explora a fusão das características de cada metodologia. Esta hibridização dos dois algoritmos, denominada de algoritmo bacteriano memético, demonstrou, em vários problemas de teste, apresentar melhores resultados que alternativas conhecidas.

Os parâmetros dos modelos neuronais utilizados e dos difusos acima descritos (satisfazendo no entanto alguns critérios) podem ser separados, de acordo com a sua influência na saída, em parâmetros lineares e não-lineares. Utilizando as consequências desta propriedade nos algoritmos de estimação de parâmetros, esta Tese propõe também uma nova metodologia para estimação de parâmetros, baseada na minimização do integral do erro, em alternativa à normalmente utilizada minimização da soma do quadrado dos erros. Esta técnica, além de possibilitar (em certos casos) um projeto totalmente analítico, obtém melhores resultados de generalização, dado usar uma superfície de desempenho mais similar aquela que se obteria se se utilizasse a função geradora dos dados.

Abstract

All systems found in nature exhibit, with different degrees, a nonlinear behavior. To emulate this behavior, classical systems identification techniques use, typically, linear models, for mathematical simplicity. Models inspired by biological principles (artificial neural networks) and linguistically motivated (fuzzy systems), due to their universal approximation property, are becoming alternatives to classical mathematical models.

In systems identification, the design of this type of models is an iterative process, requiring, among other steps, the need to identify the model structure, as well as the estimation of the model parameters. This thesis addresses the applicability of gradient-basis algorithms for the parameter estimation phase, and the use of evolutionary algorithms for model structure selection, for the design of neuro-fuzzy systems, i.e., models that offer the transparency property found in fuzzy systems, but use, for their design, algorithms introduced in the context of neural networks.

A new methodology, based on the minimization of the integral of the error, and exploiting the parameter separability property typically found in neuro-fuzzy systems, is proposed for parameter estimation. A recent evolutionary technique (bacterial algorithms), based on the natural phenomenon of microbial evolution, is combined with genetic programming, and the resulting algorithm, bacterial programming, advocated for structure determination. Different versions of this evolutionary technique are combined with gradient-based algorithms, solving problems found in fuzzy and neuro-fuzzy design, namely incorporation of a-priori knowledge, gradient algorithms initialization and model complexity reduction.

Keywords: systems identification, fuzzy systems, neural networks design, parameter separability, functional training, evolutionary algorithms

Acknowledgements

My sincere thanks go to my head supervisor, Professor Doutor António Eduardo de Barros Ruano, for his ever-willing assistance throughout my studies.

I am also very grateful to János Botzheim for all the support, ideas clarification and for joining me during a significant part of this work, as a colleague and very skillful PhD student.

I cannot forget the helpful suggestions of some of my working colleagues at the Department of Electronics and Informatics from the Faculty of Sciences at the University of the Algarve. A special thanks goes to Prof. João Lima for his patience and suggestions. I also would like to express my appreciation to Prof. José Luis Argand from the department of Physics. I also would like to thank Nelson Martins, Cristiano Soares and many other which I have shared ideas and suggestions on this work.

I also wish to thank my wife Paula, daughter Marta, son Gabriel, and father Floriano for supporting all endeavors and for being particularly patient.

Lastly, I would like to dedicate this to my mother Almerinda, who I miss much since April 2011.

Contents

List of Tables

List of figures

List of algorithms

List of symbols

$N_{i,k}^{j}(x)$	j^{th} spline function of order k in the i^{th} submodel
$N_{i,k,n}^{j}(x_n)$	j^{th} spline function of order k in the n^{th} dimension for the i^{th} submodel
$\|\|\mathbf{w}\|\|$	2-norm of vector **w**
Ψ_a	analytical version of the Ψ criterion
n_{g}	arity in Gene Expression Programming
$\mathbf{c}_i$	center for the i^{th} neuron
T_{i}	coding of the i^{th} terminal in an ET
s_i	*degree of firing* or *fire strength* for the i^{th} rule
$(\boldsymbol{\Gamma})_v$	derivative of the basis functions outputs matrix in respect to the v parameters
Ψ_d	discrete version of the Ψ criterion
Ω_d	discretized version of Ω
e	error function
$\boldsymbol{e}$	error vector
Ψ_f	functional version of the Ψ criterion
Ω_f	functional version of Ω
$\mathbf{g}_{\Psi}$	gradient vector for the Ψ criterion
$\mathbf{g}_{\psi_d}$	gradient vector for the Ψ criterion in the discrete approach
$\mathbf{g}_{\psi_f}$	gradient vector for the Ψ criterion in the functional approach
$\mathbf{g}_{\Omega}$	gradient vector for the Ω criterion
$\mathbf{g}_{\Omega_d}$	gradient vector for the Ω criterion in the discrete approach
$\mathbf{g}_{\Omega_f}$	gradient vector for the Ω criterion in the functional approach
h_{g}	head length for a chromosome in Gene Expression Programming
H	Hessian matrix
Q	input points matrix for derivative equalities
P	input points matrix for function equalities
x	input vector
φ_i	i^{th} basis function

$a_i^{(j)}$	i^{th} consequent parameter in the j^{th} rule for a TS FS
$\mathbf{x}^{(i)}$	i^{th} input pattern
x_i	i^{th} input variable
$S_{k_i,\boldsymbol{\lambda}_i}^{\mathbf{x}_{\underline{i}}}$	i^{th} submodel with the set of input variables $x_{\underline{i}}$ of order k_i, and knot vector $\boldsymbol{\lambda}_i$
$\mathbf{J}$	*Jacobian* matrix
$\mathbf{J}_{\text{GP}}$	*Jacobian* matrix definition by Golub-Pereyra
$\mathbf{J}_{\text{K}}$	*Jacobian* matrix definition by Kaufmann
$\mathbf{J}_{\text{R}}$	*Jacobian* matrix definition by Ruano
$\mathbf{J}_{\Psi}$	*Jacobian* matrix for the Ψ criterion
$\mathbf{j}$	*Jacobian* vector for the functional approach
$\mathbf{j}_{\Psi_f}$	*Jacobian* vector $\mathbf{j}$ for the Ψ criterion
$\mathbf{j}_{\mathbf{u}}$	*Jacobian* vector $\mathbf{j}$ with respect to the $\mathbf{u}$ parameters
$\mathbf{j}_{\mathbf{v}}$	*Jacobian* vector $\mathbf{j}$ with respect to the $\mathbf{v}$ parameters
$\lambda_{i,j}$	j^{th} interior knot in the i^{th} dimension
$\lambda_{SD,i,j}$	j^{th} interior knot in the i^{th} dimension in submodel SD
$I_{\text{i,j}}$	j^{th} interval in the i^{th} dimension
η	learning parameter
$\mathbf{w}$	linear weights vector
$\mathbf{SC}_i$	list of coordinates for the i^{th} submodel in a merged grid
$\mathbf{L}$	lower coordinates for a merge
$\boldsymbol{\Gamma}_{d\_der}$	matrix of basis functions derivatives, corresponding to $\mathbf{w}_d$
$\boldsymbol{\Gamma}'_{d\_der}$	matrix of basis functions derivatives, corresponding to $\mathbf{w}'_d$
$\boldsymbol{\Gamma}_{d\_fun}$	matrix of basis functions outputs, corresponding to $\mathbf{w}_d$
$\boldsymbol{\Gamma}'_{d\_fun}$	matrix of basis functions outputs, corresponding to $\mathbf{w}'_d$
$\mathbf{B}$	matrix of restrictions
$\boldsymbol{\Gamma}$	matrix of the basis functions outputs
$\boldsymbol{\Phi}$	matrix of the integral of basis functions, in the functional approach
$\mathbf{C}$	matrix of the neurons centers values
$x_{i_{MAX}}$	maximum for the i^{th} dimension
$\mu_{A_{i,j}}$	membership function for fuzzy set A in the i^{th} rule and j^{th} dimension

$x_{i_{\min}}$	minimum for the i^{th} dimension
Ψ	modified version of the Ω criterion
N_{i}	number of cells in the i^{th} dimension
N_{clones}	number of clones
N_{gen}	number of generations in a execution of an evolutionary algorithm
N_{ind}	number of individuals in a population
N_{inf}	number of infections
n	number of input dimensions
r_{i}	number of interior knots in the i^{th} dimension
N	number of iterations
h	number of linguistic terms in the antecedent part of a rule base
m_{c}	number of linguistic terms in the consequent part of a rule base
m	number of training samples
$\hat{\Psi}$	optimal value for Ψ
$\hat{\mathbf{u}}$	optimal value for $\mathbf{u}$
$\hat{\mathbf{v}}$	optimal value for $\mathbf{v}$
$\mathbf{t}$	output target vector
$\mathbf{\Gamma}^{r}$	partitioned matrix of the basis functions outputs
$\mathbf{\Gamma}_{d}$	partitioned matrix of $\mathbf{\Gamma}$, corresponding to $\mathbf{w}_{d}$
$\mathbf{\Gamma}'_{d}$	partitioned matrix of $\mathbf{\Gamma}$, corresponding to $\mathbf{w}'_{d}$
$\mathbf{\Gamma}_{i}$	partitioned matrix of $\mathbf{\Gamma}$, corresponding to $\mathbf{w}_{\text{i}}$
$\mathbf{e}^{\text{P}}$	prediction error vector
$d(t)$	real world process output at time instant t.
ρ	regularization parameter for the LM algorithm
$\mathbf{R}$	rotation matrix for gradient-based training algorithms
$\mathbf{j}_{\Psi_{f_R}}$	Ruano's version of the $\mathbf{j}_{\Psi_{f}}$ jacobian
k_{i}	spline order in the i^{th} dimension
t_{g}	tail length for a chromosome in Gene Expression Programming
$t(\text{x})$	target function in the input variables x
p	total number of basis functions in the neural models

n_u total number of linear parameters

n_v total number of nonlinear parameters

n_z total number of the model internal parameters

Ω training criterion based on the sum-of-squared-errors

$\mathbf{u}_f$ **u** parameters for the functional approach

$\mathbf{s}[k]$ update vector in the training algorithms, at the k^{th} iteration

U upper coordinates for a merge

υ_i variance for the i^{th} neuron

$\boldsymbol{\varphi}$ vector of basis functions

$\mathbf{ind}_{\mathbf{P}}^{i}$ vector of indices of the activated basis functions for the **P** points in the i^{th} submodel

$\mathbf{i}(t)$ vector of input signals to a dynamic system at time instant t.

u vector of linear parameters

$\mathbf{w}_d$ vector of linear weights, dependent on the restrictions

$\mathbf{w}'_d$ vector of linear weights, independent but activated by the restrictions

$\mathbf{w}_i$ vector of linear weights, independent of the restrictions

v vector of nonlinear parameters

b vector of restrictions target values

y vector of the model output

z vector of the model's internal parameters

x_{MAX} vector specifying the maxima for the input domain

x_{min} vector specifying the minima for the input domain

X universe of discourse

$w_{SD,i,j}$ weight for the i^{th} univariate function (from the 1^{st} dimension) and the j^{th} univariate function (from the 2^{nd} dimension) of the j^{th} submodel

List of acronyms

1PR	1 Point Recombination
2PR	2 Point Recombination
ABBMOD	Adaptive B-Spline Basis Function Modeling of Observational Data
AIC	Akaike Information Criterion
AIS	Artificial Immune Systems
AMN	Associative Memory Network
ANN	Artificial Neural Networks
ARMAX	AutoRegressive Moving Average with eXogenous input
ARX	AutoRegressive with eXogenous input
ASMOD	Adaptive Splines Modelling of Observable Data
BAR	Balanced Aspect Ratio tree
BBD	Balanced Box Decomposition tree
BEA	Bacterial Evolutionary Algorithm
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BIC	Bayesian Information Criterion
BMA	Bacterial Memetic Algorithm
BP	Error BackPropagation algorithm
BPA	Bacterial Programming
BPana	Analytical version of error backpropagation
BPd	Discrete version of error backpropagation
BPf	Functional version of error backpropagation
BPLM	Bacterial Programming for Levenberg-Marquardt
BSNN	B-Spline Neural Network
BSP	Binary Space Partition
CART	Classification Adaptive Regression Trees
CI	Computational Intelligence
DE	Differential Evolution
EA	Evolutionary Algorithm
ET	Expression Tree
FPE	Final Prediction Error

FS	Fuzzy Systems
FUREGA	Fuzzy Rule Extraction by Genetic Algorithms
GA	Genetic Algorithm
GEP	Gene Expression Programming
GFS	Genetic Fuzzy System
GMDH	Group Method of Data Handling
GP	Genetic Programming
ICT	Inverse Coordinates Transformation
IS	Insertion Sequence
LLM	Linear Local Models
LM	Levenberg-Marquardt Algorithm
LMana	Analytical version of LM
LMd	Discrete version of LM
LMf	Functional version of LM
LOLIMOT	Local Linear Model Tree
MARS	Multivariate Adaptive Regression Linear Splines
MC	Monte Carlo numerical integration
MF	Membership function
MGFS	MultiGrid Fuzzy Systems
MLP	MultiLayer Perceptron neural network
MSE	Mean Square of the absolute Error
MSRE	Mean Square of the Relative Error
NARMAX	Nonlinear AutoRegressive Moving Average with eXogenous input
NF	Neuro-Fuzzy
OLS	Ordinary Least Squares
ORF	Open Reading Frame
PBGA	Pseudo Bacterial Genetic Algorithm
pH	pH problem
PMRE	Percentage of Mean Relative Error
RBF	Radial Basis Function neural network
RIS	Reverse Insertion Sequence
RMSE	Root Mean Square of absolute Error
SCR	Simpson’s Composite Rule

SOGP	Single Objective Genetic Programming
SONN	Self-Organizing Neural Network
SQP	Sequential Quadratic Programming
SSE	Sum of the Squared Errors
TS	Takagi-Sugeno

Chapter 1

INTRODUCTION

Among the most important concepts used nowadays by the scientific community is the concept of modeling. The set of tools and methodologies used to design models from experimental data is usually called systems identification. These models can afterwards be employed for different objectives, such as prediction, simulation, optimization, analysis, control, fault detection, etc. Traditionally, the models used are described by mathematical expressions. Examples of these models are Volterra, Wiener series and ARMAX-NARMAX modeling.

More recently, tools and methodologies coming from the Computational Intelligence (CI) area have been applied in Systems Identification. These tools are biologically and linguistically motivated computational paradigms. Examples are Artificial Neural Networks (ANNs) and Fuzzy Systems (FS) which, from the point of view of Systems Identification, are nonlinear models.

Fuzzy systems offer an important advantage over neural networks, which is model transparency. On the other hand, there is an important body of work devoted to training algorithms for neural networks, which are typically gradient-based algorithms. This thesis is focused on Neuro-Fuzzy (NF) models, i.e., models that offer the transparency property, and that can use, for estimation of their parameters, algorithms which were initially proposed for neural networks. One such model is the B-Spline Neural Network (BSNN) and therefore is the model most employed in this work.

Whenever there is a-priori knowledge, it is important to use this knowledge in the model design procedure. This thesis will show how it can be integrated, considering a BSNN model type. One disadvantage of neuro-fuzzy and fuzzy models is that, typically, to obtain a good performance, high-complex models must be used. This work will also present a technique, based on input-domain decomposition, which can result in models with good accuracy, and with reduced complexity.

Evolutionary Algorithms (EAs) are also now recognized tools, in the Systems Identification community, for determining the model structure, which typically can be seen as a combinatorial problem. One recent class of EA algorithms is the Bacterial Evolutionary Algorithm (BEA), which is discussed in this thesis. Combining BEAs and gradient-based algorithms, the Bacterial Memetic Algorithm (BMA) is proposed, which is shown to be a viable tool to design neuro-fuzzy and fuzzy models, both in terms of rule extraction and as a technique to avoid local minima in the training procedure.

Training of ANN and NF models is highly dependent on the data available. This thesis will also propose a methodology whose aim is to decrease the dependence of the training algorithms on the data, focusing on the (typically unknown) mathematical function behind the data.

This chapter starts with brief historical background on the three paradigms from the CI used in this work. Then the basic concepts of system identification are presented, focusing on two of the steps needed for model design that will be covered in this work, namely parameter estimation and model structure selection. Section 1.3 gives an overview of the remaining chapters of this thesis and, the final section of this chapter outlines the contributions of this work.

1.1 Computational intelligence techniques

Computational Intelligence (CI) deals with the theory, design, application, and development of biologically and linguistically motivated computational paradigms emphasizing neural networks, connectionist systems, genetic algorithms, evolutionary programming, fuzzy systems, and hybrid intelligent systems in which these paradigms are contained [http://cis.ieee.org/scope.html]. Further, CI is a collection of computing tools and techniques, shared by closely related disciplines that include fuzzy logic, artificial neural networks, genetic algorithms, belief calculus, and some aspects of machine learning like inductive logic programming [1]. Depending on the type of domain of application these tools are used independently or jointly. The current trend is to develop hybrids of paradigms since no paradigm is superior to the others in every situation.

The following subsections give an overview and historical background on artificial neural networks, fuzzy systems and evolutionary algorithms.

1.1.1 Artificial neural networks

Artificial Neural networks (ANNs) can be regarded as a (very) simplified mathematical model of the human brain in the form of a parallel distributed computing system. To perform a task most ANNs require some form of learning through a process which is similar to training. Typically, they are provided with a training data set which contains information about the behavior of some system, consisting of the inputs of the system and corresponding desired output values. Using adequate training techniques, ANNs can adapt to the environment and learn new associations and new functional associations. Thus, they are seen as promising new generation information processing networks. The key feature of ANNs is that they have the ability to map similar input patterns to similar output patterns [2], characteristic that allows them to have a reasonable generalization capability as well as performing exceptionally well on patterns never previously presented.

In biological Neural Networks, the basic component of brain circuitry is a specialised cell called the neuron. Circuits can be formed by a number of neurons. Any particular neuron has many inputs (some receive nerve terminals from hundreds or thousands of other neurons), each one with different strengths. The neuron integrates the strengths and fires action potentials accordingly.

The input strengths are not fixed, but vary with use. The mechanisms behind this modification are now beginning to be understood. These changes in the input strengths are thought to be particularly relevant to learning and memory.

Donald Hebb (Canadian psychologist) published *The Organization of Behavior* in 1949 [3]. One of his most important conclusions is drawn from his statement:

"…when an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased", which basically means that:

- neurons that fire together are wired,
- some mechanism of memory is present,
- there is a basic form of learning.

He then introduced a learning rule which tried to explain this associative learning mechanism in biological neural networks. It is probably the oldest learning rule in the artificial neural network field and is denoted as Hebb's rule.

However, the first neural network that would come to form as a neurocomputer was called the *Perceptron* and was introduced by Frank Rosenblatt [4] in 1960. He proposed a learning rule for this first basic artificial neural network and proved that, given linearly separable classes, a perceptron would, in a finite number of training trials, develop a weight vector that would separate the classes.

The *Perceptron* output is a threshold version of the linear combination of its inputs x_i with an additional offset *b*, often called the bias or offset. Each input is weighted with a corresponding weighted value. The basic rule with this learning model is to change the value of the weights only on active lines and only when an error exists between the network output and the desired output.

At about the same time, Bemard Widrow [5] introduced learning from the point of view of minimizing the mean-square error between the output of a different type of ANN processing element, the *ADALINE* [6], and the desired output vector over the set of patterns. This work led to modern adaptive filters. Adalines and the Widrow-Hoff learning rule were applied to a large number of problems, probably the best known being the control of an inverted pendulum.

Despite the progress in this area, the biggest setback came when Minsky and Papert began promoting the field of artificial intelligence (what is known nowadays as expert systems) at the expense of neural networks research. They wrote “Perceptrons” [7], a book where it was mathematically proved that these neural networks were not able to compute certain essential computer predicates like the *EXCLUSIVE OR* Boolean function. It was not until the middle of 1980’s that interest in artificial neural networks re-started to rise substantially, making ANN one of the most active current areas of research, partially to work by Hopfield, Rumelhart [8] and McLelland [9].

Several different architectures of ANN have been proposed over the years. All of them try, with different degrees, to exploit the available knowledge of the mechanisms of the human brain. Generally, an ANN consists of an input layer, hidden layers and an output layer. At each layer, a pre-defined number of artificial neurons are connected to others in the next layer. A typical ANN structure is shown in Fig. 1.1. These ANN have had numerous applications, including diagnosis, speech recognition, image processing, forecasting, robotics, classification, and many others.

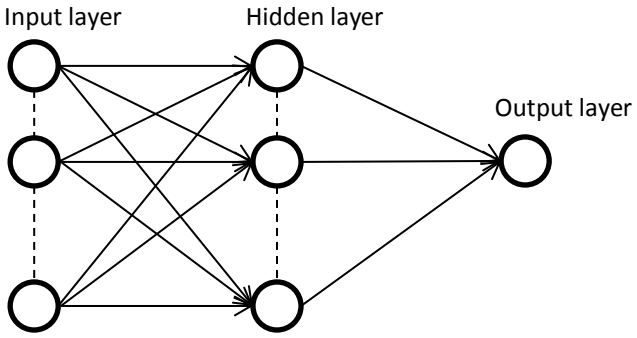


Fig. 1.1. Illustration of an artificial neural network

Among the most studied structures of ANNs are certainly Multi-Layer-Perceptron Networks (MLPs), Radial Basis Function Networks (RBFs) and B-Spline Neural networks, which may be considered neuro-fuzzy systems.

1.1.2 Fuzzy systems

Computing systems use binary encoding to represent the information or knowledge about a problem. Associated with Boolean logic is the traditional two-valued set theory where the element belongs to a class or not, a form of defining precise class membership.

However, a solution to many real-world problems cannot be achieved by mapping the problem domain to two-valued variables. In contrast, they require a representation language that is able to process incomplete, imprecise and vague information. Fuzzy theory provides the formal tools for dealing with such vague information. With fuzzy logic, domains are characterized by linguistic terms instead of numbers. Fuzzy logic provides the tools for dealing with statements such as "*it is slightly cold*" and "*this person is very old*". Thereby, it defines appropriate linguistic terms for "*slightly*" and "*very*", describing the magnitude of the fuzzy variables "*cold*" and "*old*", respectively. The human brain is capable of understanding these statements and infer that probably it is a cloudy day or that a very old person is not a child.

Combining fuzzy logic and fuzzy set theory provides a means to model human reasoning. Furthermore, in fuzzy sets, an element belongs to a set given a degree, indicating the percentage of membership.

The foundation of two-valued logic traces back to 400 BC and is due to research developed by Aristotle and other philosophers of that time, when the first version of *Laws of Thought* [10], the *Law of the Excluded Middle* was proposed. This first version stated that every proposition must have only one of two possible outcomes, either *true* or *false*.

The foundation to what today is referred as fuzzy logic is rewarded to another philosopher, Plato. It was not until the 1900's that, an alternative to Aristotle's two-valued logic was proposed by Lucasiewicz [11] and mathematical background on fuzzy theory (infinite valued logic) was produced in 1965 by Lofti Zadeh [12]. His main idea was that most of the phenomena of the real world could not be described by two values, so he defined a function which would assign fuzzy truth degrees between zero and one to elements of a universal set. This way he introduced the mathematical way of representing vagueness in everyday life.

Everyday language is one example of ways of applying vagueness. If one says "the tomato is ripe" or "the tomato is red", one knows that these are similar concepts. So, a fuzzy concept of uncertainty is associated with the decisions one makes, though from imprecise information.

As abovementioned, the main motivation for developing fuzzy systems was the need to represent human knowledge and corresponding decision processes. In the specific case of systems identification, fuzzy rule-based systems are applied where the relations between variables are expressed by *if-then* rules. Fuzzy sets are then used to represent the ambiguity in the definition of the linguistic terms that form these rules. They are defined by membership functions which map the elements of the considered universe to the interval [0,1]. A value between 0 and 1 defines a partial membership, whereas the extreme values 0 and 1 denote null and complete membership. This partial membership allows one particular element to belong to different fuzzy sets and facilitates a smooth outcome of the reasoning when using fuzzy *if-then* rules.

1.1.3 Evolutionary algorithms

Evolutionary algorithms (EAs) are inspired by Darwin's theory of evolution instead of mathematically mimicking a biological process as is the case of ANNs. They are driven by the quest for a solution given a goal. With EAs, the search space S must represent the set of all possible DNA strings in nature. Likewise in living organisms, the search space consists of elements $s \in S$ which play the role of the natural genotypes. Thus, it is common practice to refer to S as the genome (or chromosome) and to the elements s, as the genotypes. In the problem space X, the solution candidates (termed phenotypes) $x \in X$ are instances of genotypes and are obtained from genotype-phenotype conversion process, $s \rightarrow x$. This is similar to nature where an organism is an instance of its genotype formed by embryogenesis. Their fitness is then rated according to objective functions which are subject to optimization and drive the evolution into specific directions.

Living organisms have some characteristics which drive their ability to survive and reproduce. In EAs this is represented by the *reproduction* process. Sexual reproduction will produce offspring combining features from each of the parents. Efficiency of the evolution strategy will depend on the propagation of the best characteristics from the parents into the offspring and on the natural selection in choosing the better individuals to mate most of the time. In EAs there is no such thing as "gender". Each individual from the mating pool can potentially be recombined with every other one.

Occasionally, the chromosomes will experience little changes (mutation) modifying some characteristics of the individuals, thus affecting the individuals' ability to survive or reproduce. Moreover, *mutation* is important to the evolutionary process as it can avoid premature convergence of the population where individuals vary only slightly from each other. Thus, natural evolution results from the interplay between the creation of new genetic information and its evaluation and selection. As this creative process is based on the Darwinian evolution, organisms adapt themselves to the environment through cumulative processes of natural selection accompanied by genetic operations such as *mutation* and *crossover* of their genes.

Performing the *reproduction* process iteratively over and over again enhances the probability of final solutions found being close to the optimum.

Thus, as a form of emulating this process of natural selection, evolutionary algorithms (EA) consist of techniques seeking for an optimal solution to a given problem through a stochastic search.

There are two basal aspects which guide the search process in EAs: EAs start from a set of initial points allowing a parallel search of a large area of the search space, and only the fitness values of individuals is used to drive the search, therefore not requiring any derivative information.

Although early developments with EAs may go as back as 1950's it was the introduction of Genetic Algorithms (GAs) by Holland [13] that strongly motivated research in this area. Other important contributions are credited to Koza [14] through genetic programming, Fogel [15], and Rechenberg by introducing evolution strategies [16].

The range of application of EAs is extensive and includes planning of routing optimization, design of filters, neural network architectures, controllers, classification and clustering, function approximation, among others. For systems identification, evolutionary algorithms are considered most suitable [17][18][19][20][21].

1.2 Systems identification

To understand the concept of system identification, consider a multiple-input, single-output (*d(t)*) nonlinear time-invariant dynamic system:

$$d(t) = f(\mathbf{i}(t)), \mathbf{i}(t) \in \Re^n, \tag{1.1}$$

where **i** is the vector of input signals.

The mathematical description of the system itself may be unknown but it is assumed that input-output data are available, drawn from a constant rate sampling. This way, an approximation to the continuous system in (1.1) can be carried out using a discrete-time model:

$$y(k+1) = g(\mathbf{x}(k)), \tag{1.2}$$

where y is the estimated output of the model and $\mathbf{x}(k)$ is the regression vector at sampling time *k*.

In classical systems identification, many are the structures of models which can be employed. They are distinguished by the past signals considered in $\mathbf{x}(k)$. Examples of such models are NARX, ARMAX, NARMAX, Wiener models, etc.

Thus, the problem of nonlinear system identification is to infer the unknown function *f* from function *g*, using the sampled data sequences in $\mathbf{x}(k)$, and its main goal is to determine models that can afterwards be employed for different objectives, such as prediction, simulation, optimization, control, etc.

In the context of systems identification, neural networks and fuzzy systems can be considered as black-box models, or gray-box models, if a priori knowledge is available and using in the design. So, they simply act as models that perform a nonlinear transformation between a n^{th} dimensional input space and a one-dimensional output space. Considering the nonlinear system of (1.1), ANNs and FS may be regarded as an input-output mapping mechanism characterized by a set of nonlinear parameters ***z*** and a structure which requires determination:

$$y = h(\mathbf{x}[k], \mathbf{z}) \tag{1.3}$$

where **z** is the set of parameters for a particular model structure.

Systems identification is an iterative procedure, and a sequence of steps must be carried out so that a satisfactory model can be obtained in the end. First, the relevant input variables must be identified. This involves determining the physical input variables ***i***. Next, data must be acquired and pre-processed. Subsequently, a model architecture must be chosen. If the

system has dynamics, a correct representation must be chosen, which means that the dynamic regressors **x** in (1.2) must be selected. Then, the model structure must be identified and their parameters estimated. At the end of each iteration, results are evaluated using unseen data and are used as prior data for subsequent iterations.

Both the set of parameters and structure will vary upon the model architecture considered (please refer to chapter 2, section 2.2 for a detailed description of ANN models).

Irrespectively of the model chosen, there are two basic steps in system identification: structure identification and parameters estimation.

This work will focus on two of the steps mentioned above, parameter estimation and model structure selection.

Parameters estimation is an iterative process when the output of a model is nonlinear on its parameters. Using an appropriate criterion (typically the *sum-of-squared-errors*) and since all models are differentiable in their parameters, it is possible to apply gradient based algorithms. If the parameter update is performed after the presentation of a data set, which will be used in every iteration of the learning algorithm, the process is denominated offline learning, batch learning, or simply training. On the other hand, if the update is performed in a pattern-by-pattern basis, or after the presentation of a data set that changes from iteration to iteration, the learning process is called online learning or adaptation. A description on the most common algorithms for parameter estimation is given in Chapter 2, Section 2.3.

Determining the structure of the model is an extremely complex task, especially if dealing with real-world problems, because it involves choosing the complexity of the model. The aim here is to find parsimonious models that have a satisfactory performance for the given data, with the smallest possible complexity. It is known that, as the model complexity increases, for the same amount of training data, the model performance in this data set improves but, above a certain limit in the complexity, the model performance in the test data is deteriorated. There is a compromise between the approximation obtained in the training set and the generalization in data not seen by the model. The final goal of any structure selection method is to find this compromise value.

Depending on the modeling architecture distinct approaches can be taken. As this an extremely hard task, the methods are distinct but can be incorporated into four different categories [22]:

- *Generic*. This involves methods which try to solve this problem as a combinatorial one. These include evolutionary algorithms (see Section 2.5.1), which will be used in this work.
- *Constructive* or *forward selection*. They start from a simple model and which is iteratively refined by adding more parameters. These methods are based on *incremental learning* and have the advantage that unnecessarily complex models need being computed.
- *Backward selection* or *pruning*. The idea is opposite to constructive methods. The iterative process starts from a very complex model and parameters reduction is carried out in each iteration. They are usually more time consuming than constructive type techniques.
- *Stepwise* or *mixed*. These strategies combine the ideas from both constructive and pruning methods. Pruning is applied when some of the model's parameters or sub-structures become redundant. This strategy is typically more effective than using constructive or pruning methods alone. Some typical algorithms which implement this type of selection are *classification and regression tree* (CART) and *multivariate adaptive regression linear splines* (MARS), and ASMOD (a detailed description is given in chapter 2, section 2.5.2.3.1).

1.3 Thesis overview

The current chapter gives the motivation that led to this work, briefly introduces CI methodologies and systems identification, presents an overview of the thesis and highlights the main contributions.

The second chapter gives the theoretical background, needed for the work produced during this PhD. It describes the models used throughout this work, the most important gradient-based algorithms, and the structure optimization techniques. It also points out the equivalence between BSNNs and Fuzzy systems, as well as a summary on numerical integration techniques. A state of the art for each topic is also included.

The next four chapters will focus on new design approaches, involving gradient-based and evolutionary-based algorithms, specifically for the BSNN models:

- Chapter 3 presents a new approach to incorporate a priori knowledge in model design.
- Chapter 4 proposes the bacterial programming algorithm (BPA) as a global optimization methodology.

- Chapter 5 combines the BPA algorithm in a hybrid structure with local optimization techniques to improve the quality of search.
- Chapter 6 introduces a new methodology to cope with the structure complexity of BSNN models, which is based on a new input domain decomposition.

The techniques presented in the last 4 Chapters, although being introduced in the scope of BSNNs, are also applicable to FS, provided certain assumptions are met. If this is not the case, Chapter 7, shows how the same evolutionary algorithms used in the ANN scope can be applied to generic fuzzy systems (FS). It also includes the description of a new memetic algorithm for optimizing the fuzzy rule base of a FS.

Chapter 8 looks at the training problem for neural networks and fuzzy systems in a different way, aiming at focusing the estimation problem to the function underlying the data, and not the data in itself.

Finally, Chapter 9 draws conclusions and gives a perspective on future work.

Some theoretical derivations and complementary results can be consulted in the appendices, which are organized as follows.

In Appendix A all benchmark problems used in this thesis are outlined.

Appendix B gives complementary mathematical background on the calculus of the linear weights in BSNNs, in the context of the new input domain decomposition proposed by Chapter 6.

Appendix C provides the mathematical formulation necessary to apply the functional approach, proposed in Chapter 8. This is provided for the B-Spline neural networks and for the RBF networks.

1.4 Main contributions

As a result of the studies conducing to the PhD degree, some publications were written and presented, describing the major contribution within this work:

- Specifying equality restrictions to the training of B-Spline neural networks:
  - C. L. Cabrita, and A. E. Ruano, B-spline and neuro-fuzzy models design with function and derivative equalities, In *Proceedings World Automation Congress* (WAC2004), June 28-July 1, Sevilha, Spain, 2004.
- Proposal of a new hybrid algorithm combining the Levenberg-Marquardt algorithm and Genetic Programming for training B-Spline neural Networks:

- C. L. Cabrita, J. Botzheim, A. E. Ruano, and L. T. Koczy, A Hybrid Method for B-Spline Neural Networks Training, *International Symposium on Intelligent Signal Processing*, Faro, Portugal, 2005.

- Empirical performance comparison of the Genetic Programming and Bacterial Programming Algorithm when applied to various identification problems:
  - C. L. Cabrita, J. Botzheim, A. E. Ruano, and L. T. Kóczy, Design of B-spline Neural Networks using a Bacterial Programming Approach, *International Joint Conference on Neural Networks (IJCNN 2004) and IEEE International Conference on Fuzzy Systems (Fuzz-IEEE 2004)*, Budapest, Hungary, 25July-29July, 2004.
  - J. Botzheim, C. L. Cabrita, L. T. Koczy, and A. E. Ruano, Genetic and Bacterial Programming for B-Spline Neural Networks Design, *Journal of Advanced Computational Intelligence & Intelligent Informatics*, Vol 11, nº2, 2007, pp. 220-231, 2007.
- Presenting Bacterial Programming as a valid technique for fuzzy rule extraction:
  - C. L. Cabrita, J. Botzheim, T. Gedeon, A. E. Ruano, L. T. Kóczy, and C. Fonseca, Bacterial memetic algorithm for fuzzy rule base optimization, 2006 *World Automation Congress (WAC)- ISSCI Symposium*, Budapest, Hungary, 24-26July, 2006.
- Validating the application of the Levenberg-Marquardt algorithm for the extraction of trapezoidal fuzzy rules:
  - J. Botzheim, C. L. Cabrita, A. E. Ruano, and L. T. Kóczy, Estimating Fuzzy Membership Functions Parameters by the Levenberg-Marquardt Algorithm, *International Joint Conference on Neural Networks (IJCNN 2004) and IEEE International Conference on Fuzzy Systems (Fuzz-IEEE 2004)*, pp 1667–1672, Budapest, Hungary, 25July-29July, 2004.
- Fuzzy rule extraction technique using bacterial memetic algorithms:
  - J. Botzheim, C. L. Cabrita, L. T. Kóczy, and A. E. Ruano, Fuzzy rule extraction by bacterial memetic algorithms, In *Proceedings of the 11th World Congress of International Fuzzy Systems Association*, IFSA 2005, pp. 1563–1568, Beijing, China, July, 2005.

- J. Botzheim, C. L. Cabrita, L. T. Kóczy, and A. E. B. Ruano, Fuzzy rule extraction by bacterial memetic algorithms, *International Journal of Intelligent Systems*, Vol.24, pp.312-339, 2009.

o A New input domain decomposition for B-spline neural networks:

- C. L. Cabrita, A. E. B. Ruano, and L. T. Kóczy, A new domain decomposition for B-spline Neural Networks, *WCCI 2010 IEEE World Congress on Computational Intelligence*, July 18-23, Barcelona, pp. 308-315, 2010.

o Proposal of a new nonlinear local optimization methodology based on the target function and not on the input patterns:

- A. E. Ruano, C. L. Cabrita, and P. M. Ferreira, Towards a More Analytical Training of Neural Networks and Neuro-Fuzzy Systems, *IEEE International Symposium on Intelligent Signal Processing 2011* (WISP11), Floriana, Malta, 19-21 September 2011.
- C. L. Cabrita, A. E. Ruano, and P. M. Ferreira, Exploiting the functional training approach in Radial Basis Function Networks, *IEEE International Symposium on Intelligent Signal Processing 2011*, WISP11, 19-21 September 2011.
- C. L. Cabrita, A. E. Ruano, P. M. Ferreira, and Lázsló T. Kóczy, Extending the functional training approach for B-Splines, *IEEE World Congress on Computational Intelligence*, Brisbane, Australia,10-15 June, pp. 2702-2709, 2012.
- A. E. Ruano, C. L. Cabrita, and P. M. Ferreira, Exploiting the functional training approach in B-Spline, *1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control*, Wurzburg, Germany, 3-5, April 2012.
- C. L Cabrita, A. E. Ruano, P. M. Ferreira, and L. T. Kóczy, Exploiting the Functional Training Approach in Takagi-Sugeno Neuro-fuzzy Systems. In *Soft Computing Applications*, ed. Valentina Emilia Balas, János Fodor, Annamária R. Várkonyi-Kóczy, Joszef Dombi, Lakhmi C. Jain, 543 - 559. ISBN: 978-3-642-33940-0. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

Chapter 2

THEORETICAL BACKGROUND AND STATE OF THE ART

2.1 Introduction

Systems identification is a research area in constant evolution. Classical modeling methods are progressively being replaced by new recent advanced techniques. These new techniques are closely related to Artificial Intelligence (AI). Soft computing encompasses techniques such as evolutionary algorithms, artificial neural networks (ANN) and fuzzy systems (FS). While evolutionary algorithms consist of techniques based on the natural evolution and selection of species, ANNs are modular structures which are inspired on the biological behavior of the human brain, whereas fuzzy systems are conceptual mechanisms where uncertainty and human reasoning is represented by fuzzy logic.

In the context of systems identification, ANNs and FS are the most used CI model architectures, which are described in Section 2.2. Within the ANN arena, the most used are MLPs, RBFs, and B-Spline NNs. Typically, MLP and RBF models are more employed, as they have been introduced earlier, and are less complex than BSNNs. The latter are, however, more adequate for online adaptation, due to their localized properties. Furthermore, they can be seen as neural networks or as fuzzy systems, where the model interpretability is a main issue. As such, they are denoted as neuro-fuzzy systems. As BSNNs and FS are used the models most used in this thesis, they are described in more detail.

Designing the ANN or fuzzy model that represents best the behavior of a process is a two steps procedure. First, the structure has to be identified, with typically nonlinear global optimization methods, and, subsequently, the internal parameters of the model have to be optimized, typically with nonlinear local optimization techniques.

Local optimization gradient-based methods are described in Section 2.3. The most important first and second-order methods are described, as well as their application to models where parameter separability between linear and nonlinear can be exploited. This concept can be also be used to provide a relationship between BSNNs (and RBFs) and FS, which is employed in this thesis and therefore, it is described in Section 2.4.

The structure identification issue is discussed in Section 2.5. As in this work evolutionary techniques are proposed for this task, genetic algorithms, genetic programming, gene expression programming and bacterial evolutionary algorithms are described in some detail. Other techniques, with a special emphasis on functional decomposition and input space decomposition methods are also introduced for model structure identification.

As a new approach of model training is introduced in this thesis, which uses numerical integration techniques, a brief summary of the most used methods is described in Section 2.6.

Conclusions are drawn in Section 2.7.

2.2 Model architectures

Artificial Neural Networks and Fuzzy Systems, due to their universal approximation and adaptation properties, are widely used as (nonlinear) models for function approximation purposes. In the former class of models, Multilayer Perceptrons (MLPs), Radial Basis Function (RBFs) networks and B-Splines Networks (BSNNs) have been used for a large range of applications. They will be described in sections 2.2.1.1, 2.2.1.2 and 2.2.1.3, the latter two with more detail, as they will be used in this work. Fuzzy Systems are addressed in Section 2.2.2, where the most important fuzzy models are introduced: Mamdany (or linguistic FS) in Section 2.2.6.1, and Takagi-Sugeno in Section 2.2.6.2.

2.2.1 Artificial neural networks

2.2.1.1 MultiLayer Perceptron networks

Multilayer perceptrons (MLPs) are the most widely known type of ANNs. The original Perceptron, as introduced by Rosenblatt [4], used a hard nonlinearity as activation function. Due to this fact, its computational capabilities were small and only in 1986, with the replacement of this activation function by a sigmoidal differentiable function and the introduction of the error back-propagation algorithm [8], multilayer perceptrons became widely known.

Regarding the model of the *Perceptron*, a simple way to design a network structure can be employed with *Perceptrons*. However, to design the whole structure globally to implement a more complicated function (or sometimes even easier functions, like the XOR function) is impossible, because of the hard nonlinearity of the *Perceptron*.

Typically, learning algorithms require the calculation of the derivatives of the network with respect to its parameters and this is not practical when using the standard *Perceptron*. An alternative is to replace the sign function of the *Perceptron* with a smoother and differentiable non-linearity, such as a sigmoid or a hyperbolic tangent function. This modified *Perceptron* is then applied as processing units in the Multilayer Perceptron (MLP) which is a feedforward multilayer ANN. Fig. 2.1 presents an illustration of the MLP neural network.

MLPs refers to the kind of feedforward artificial network consisting of a set of sensory units (source nodes or source neurons) that constitute the input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes. Neurons in any layer of the network are connected to all the neurons in the previous layer through parameters commonly called weights. The input signal propagates through the network in a forward direction, from left to right and on a layer-by-layer basis.

Multilayer Perceptrons have been proved to be universal approximators [24][25].

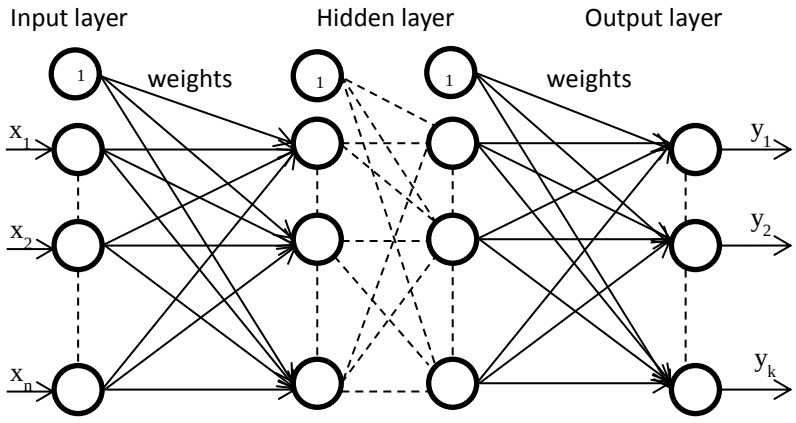


Fig. 2.1. Structure of a Multi-Layer Perceptron Neural Network

2.2.1.2 Radial Basis Function networks

RBFs were firstly introduced in the context of neural networks by Broomhead and Lowe [26].

Like the MLP, the RBF is a feed-forward neural network. RBF's are easier to initialize and train than MLPs [27][28], possess the ability to generalize [29], and can implement localized representations of functions.

The RBF is composed of three fully connected layers as illustrated by Fig. 2.2. The first is the input layer, which connects the source nodes to a set of *p-1* nodes in the hidden layer. The response of the network is given by the output layer, which is a linear combination of the neurons in the hidden layer. The hidden layer consists of the basis functions.

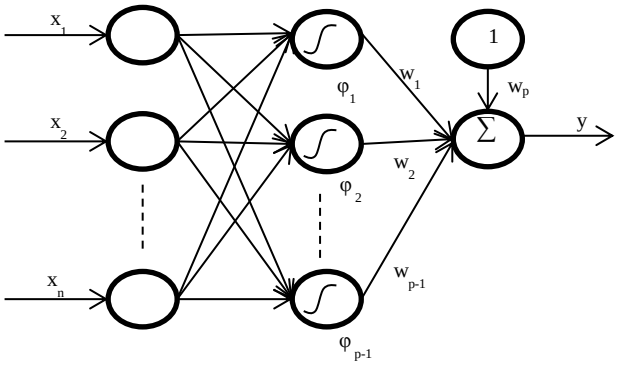


Fig. 2.2. Structure of a Radial Basis function network

The output of a RBF neural network is defined as

$$y(\mathbf{x}) = \sum_{i=1}^{p} w_i \boldsymbol{\varphi}_i(\mathbf{x}, \mathbf{c}_i, \upsilon_i) = \boldsymbol{\varphi}^T(\mathbf{x}, \mathbf{C}, \boldsymbol{\upsilon})\mathbf{w} \tag{2.1}$$

As a bias term is usually employed in RBFs, then:

$$\boldsymbol{\varphi}(\mathbf{x}, \mathbf{v}) = \boldsymbol{\varphi}(\mathbf{x}, \mathbf{C}, \boldsymbol{\upsilon}) = \left[\varphi_1(\mathbf{x}, \mathbf{c}_1, \upsilon_1) \ \ldots \ \varphi_{p-1}(\mathbf{x}, \mathbf{c}_{p-1}, \upsilon_{p-1}) \ \ 1 \ \right]^T, \tag{2.2}$$

where $\varphi_p = 1$.

The most common basis function employed is the Gaussian function:

$$\varphi_i(\mathbf{x}, \mathbf{c}_i, \upsilon_i) = \mathrm{e}^{-\frac{\|\mathbf{x}-\mathbf{c}_i\|_2^2}{2\upsilon_i}} \tag{2.3}$$

Where υ_i is the variance and $\mathbf{c}_i$ the center associated with the basis function of the i^{th} neuron.

With discretized input data, a compact form for eq.(2.1) is

$$\mathbf{y}(\mathbf{X}, \mathbf{v}, \mathbf{u}) = \boldsymbol{\Gamma}(\mathbf{X}, \mathbf{v})\mathbf{w} \tag{2.4}$$

where $\mathbf{v} = \{\mathbf{C}, \boldsymbol{\upsilon}\}$.

With the Gaussian basis function, adaptation of the local representation degree of the RBF depends on the value of the term υ_i. If υ_i is sufficiently small, (2.1) will approximate $\varphi_i(\mathbf{x}^{(k)}, \mathbf{c}_i, \upsilon_i) \times w_i$ for the k^{th} input sample, $\mathbf{x}^{(k)}$. By increasing υ_i more terms will contribute

to the output and so the RBF will become less localized. Fig. 2.3 depicts this situation for c_i=0 and υ_i ranging from 0.25 and 1.

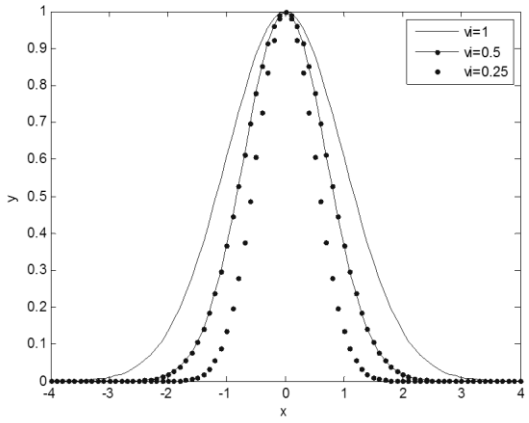


Fig. 2.3. Gaussian Radial Basis Function with variance {0.25, 0.5, 1}.

Also, the Gaussian function can be shown to be optimal in a least squares sense, for fitting data with normally distributed noise in the input [30]. Moreover, it is also infinitely differentiable and all derivatives are continuous.

To obtain the center, random values can be employed. A somewhat better strategy is to randomly select a pre-defined number of input data patterns as centers. A better structure identification can be achieved if clustering techniques such as the k-means, or more complex unsupervised learning methods as Kohonen's self-organizing map is applied [31]. Through clustering, the centers will be determined according to the input data distribution in the input space. Nevertheless, clustering does not take into account the complexity of process. It would be desirable to generate many basis functions in regions where the process possesses complex behavior and seldom ones where the process is very smooth. To accomplish this, information about the output is required. One such approach was proposed in [32].

Other more efficient techniques can be implemented adapting the Orthogonal Least Squares (OLS) to RBFs. The OLS algorithm has been accepted as the state-of-the-art for training RBFs [27]. The basic idea of the OLS is to orthogonalize a set of vectors into a set of orthogonal basis functions. This way, it allows to calculate the individual contribution of the desired output variance from each basis vector. For orthogonalization, a Gram-Schmidt, modified Gram-Schmidt or Givens transformation can used. In RBFs, OLS is used to iteratively select a subset of basis functions from a large set of already determined basis functions. As it is a constructive type method, in each iteration it chooses one basis function

to orthogonalize, which corresponds to the one with the largest-error-reduction rate. It terminates when a pre-defined number of regressors is achieved or when the explained output variance reaches some value. Because of the number of potential regressors can be enormous for large size data sets, a substantial increase on the computational demand becomes impractical. Therefore an alternative is to use clustering in a first phase reducing the number of potential centers to a practical amount [33]. Other variants use OLS incorporating a regularization term [34].

Incremental learning strategies such as the resource allocation network method described in [35] have also been proposed. In this case, a new neuron is added to the network when the error between process and current output or the distance to the nearest basis function center exceeds a certain threshold. Introduction of the extended Kalman filter improved this method's performance [36]. In [37], a pruning strategy was proposed and a comparison of performance between the different alternatives is shown in [38].

For the estimation of the corresponding width values of the Gaussians, some authors express the basis functions in terms of fuzzy membership functions, as determined from cluster analysis [39][40][41]. Alternatively, estimation of both centers and widths of the radial basis function parameters as a direct result of the clustering process is employed in [42].

Applications of Radial Basis Function networks (RBF) include fuzzy regression [43], face recognition [44], control systems [45][46][47] and time series forecasting [48], to name a few.

2.2.1.3 B-spline artificial neural networks

B-Spline neural networks play a central role in the context of the current work; hence they will be described in more detail. Real problems applications of the B-Spline neural networks include PID auto-tuning [49], fault detection [50], and power electronic applications [51], among others.

B-Splines are well known as surface-fitting algorithms within the graphical visualization community. A major landmark in the use of B-Splines is due to Cox and De Boor [52] in 1972, when a stable and efficient recurrence relationship for evaluating the B-Splines was introduced.

B-spline neural networks offer definite advantages over more commonly used neural networks, such as multilayer perceptrons or radial basis function networks. B-spline networks store the information locally, which means that learning in one part of the input space affects the rest only minimally. For this reason, they are suitable for on-line adaptive modeling and control applications [53][54].

A B-spline neural network (BSNN) consists of a set of piecewise polynomials to model an unknown function for which a finite set of input–output samples are available. They possess characteristics that allow local adjusting and simple calculation and for these reasons they have been widely used in graphical processes [55].

BSNNs belong to the class of networks denoted as *grid* or *lattice-based associative memories networks* (AMN). This type of network is composed of three layers: a *normalized input space* layer, a *basis functions* layer and a *linear weight* layer (see Fig. 2.4).

Their grid-based structure makes them transparent, which, in contrast to other networks, means that it is easier to understand the knowledge stored in these networks. This is seen as an advantage that is also assigned to fuzzy rule-based models over conventional neural networks, such as Radial Basis Function Networks or Multi-Layer Perceptrons. In fact, and at a high level the basic information principles of B-spline networks and fuzzy systems are the same. So, under certain conditions, the low-level algorithms are also identical (see section 2.4).

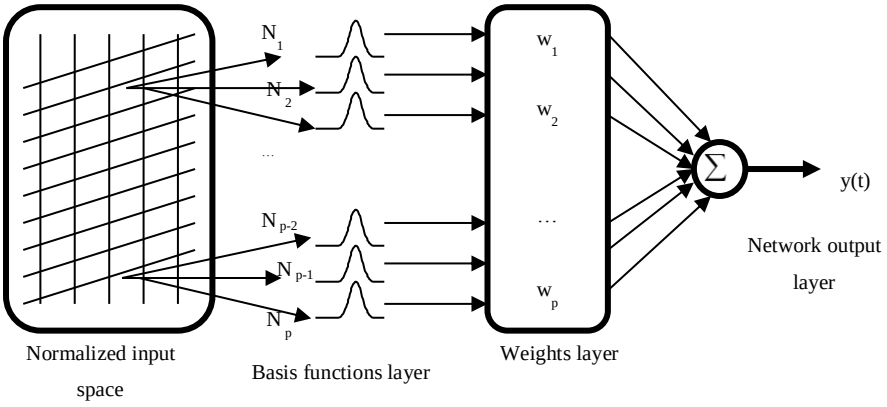


Fig. 2.4. Structure of a lattice-based network

The normalized input layer is usually a grid on which the basis functions are defined. To evaluate the basis functions, vectors of *knots* must be defined, one for each input axis. There are usually a different number of knots for each dimension, and they are generally placed at different positions.

The interior knots, for the i^{th} input, are $\lambda_{i,j}, j=1,\cdots,r_i$, where r_i denotes the number of interior knots in the i^{th} input. They are arranged in such a way that:

$$x_{i_{\min}} < \lambda_{i,1} \le \lambda_{i,2} \le \cdots \le \lambda_{i,r_i} < x_{i_{MAX}} \tag{2.5}$$

The interior knots are considered nonlinear parameters. At each extreme of each axis, a set of k_i *exterior knots* must be given which satisfy:

$$\lambda_{i,-(k_i-1)} \le \cdots \le \lambda_{i,0} = x_{i_{\min}}, \quad x_{i_{MAX}} = \lambda_{i,r_i+1} \le \cdots \le \lambda_{i,r_i+k_i} \tag{2.6}$$

These exterior knots are required to generate the basis functions that are close to the boundaries. These knots are usually coincident with the extreme of the input axes, or are equidistant. The network input space is $\left[x_{1_{\min}}, x_{1_{MAX}}\right] \times \cdots \times \left[x_{n_{\min}}, x_{n_{MAX}}\right]$, and so the exterior knots are only used for defining these basis functions at the extreme of the lattice.

The j^{th} interval of the i^{th} input is denoted as $I_{i,j}$ and is defined as:

$$I_{i,j} = \begin{cases} \left[\lambda_{i,j-1} \quad \lambda_{i,j}\right[\ for \ j = 1, \cdots, r_i \\ \left[\lambda_{i,j-1} \quad \lambda_{i,j}\right] \ if \ j = r_i + 1 \end{cases} \tag{2.7}$$

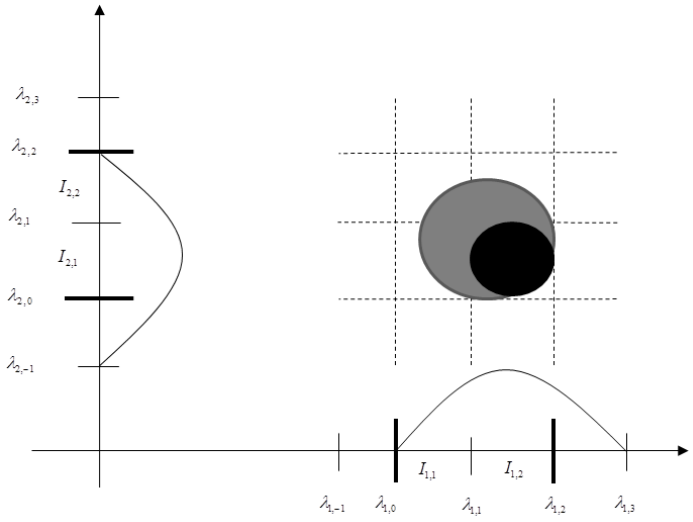


Fig. 2.5. Lattice for a bidimensional B-Spline neural network with one interior knot in each axis. The circles illustrate the degree of activation of the two basis functions sketched.

The previous figure (Fig. 2.5), sketches the input lattice for a bivariate B-spline consisting of one interior knot in each input axis. The input space is $\left[x_{1_{\min}}, x_{1_{MAX}}\right] \times \left[x_{2_{\min}}, x_{2_{MAX}}\right] = \left[\lambda_{1,0}, \lambda_{1,2}\right] \times \left[\lambda_{2,0}, \lambda_{1,2}\right]$. The input lattice consists of four cells resulting from the existence of two intervals in each axis.

Within the range of the i^{th} input, there are r_i+1 intervals (two for each input in Fig. 2.5), which means that there are $p' = \prod_{i=1}^{n} (r_i + 1)$ cells in an n-dimensional lattice.

The output of the hidden layer is determined by a set of p basis functions defined on the n-dimensional lattice. The shape, size and distribution of the basis functions are characteristics of the particular AMN employed, and the support of each basis function is bounded.

In B-Splines neural networks, the *order* of the spline implicitly sets the size of the basis functions support and its shape. The univariate B-Spline basis function of order *k* has a support, which is *k* intervals wide. Hence, each input is assigned to *k* basis functions.

The j^{th} univariate basis function of order *k*, in the i^{th} dimension is denoted by $N_{k_i}^{j}(x_i)$, and it is defined by the following recurrence relationships [52]:

$$N_{k_i}^{j}(x_i)=\left(\frac{x_i-\lambda_{i,j-k}}{\lambda_{i,j-1}-\lambda_{i,j-k}}\right)N_{k-1}^{j-1}(x_i)+\left(\frac{\lambda_{i,j}-x_i}{\lambda_{i,j}-\lambda_{i,j-k+1}}\right)N_{k-1}^{j}(x_i)$$
$$N_1^{j}(x_i)=\begin{cases}1 \;\; if \;\; x_i \in I_{i,j}\\ 0 \;\; otherwise\end{cases} \tag{2.8}$$

Note that these functions are continuous from the right, and that $N_{k_i}^{j}$ form a partition of unity:

$$\sum_j N_{k_i}^{j}(x_i)=1 \tag{2.9}$$

The following figure shows the four cubic polynomials that make up a certain spline of order *k=4* (cubic spline).

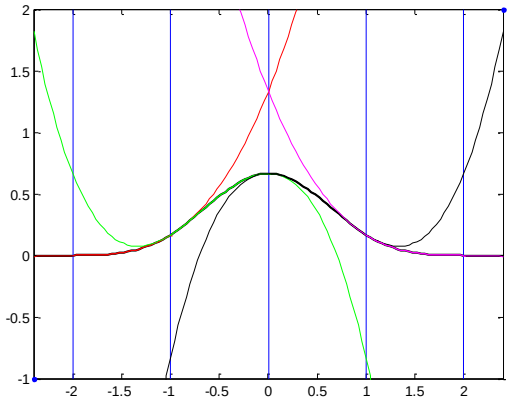


Fig. 2.6. The four cubic splines which make up a univariate cubic spline

2.2.1.3.1 Example of a univariate quadratic B-spline

Suppose a B-Spline Neural Network with a structure where the vector of interior knots is

$$\left\{\lambda_{1,-2},\lambda_{1,-1},\lambda_{1,0},\lambda_{1,1},\lambda_{1,2},\lambda_{1,3},\lambda_{1,4}\right\}=\left\{-2,-3,-1,0,1,2,3\right\} \tag{2.10}$$

And the order k_1=3.

According to (2.6), $x_{i_{\min}}=-1$ and $x_{i_{MAX}}=1$.

As there is one interior knot, the two intervals, $I_{1,1}$ and $I_{1,2}$ are defined as,

$$I_{1,1} = [-1 \quad 0[$$
$$I_{1,2} = [0 \quad 1[\tag{2.11}$$

And the number of basis functions is given by

$$p = r_1 + k_1 = 1 + 3 \tag{2.12}$$

Fig. 2.7 illustrates how the four basis functions are defined across the input domain.

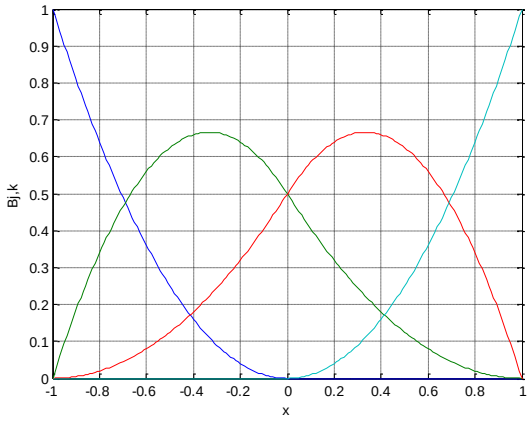


Fig. 2.7. Quadratic spline basis functions for a univariate B-spline NN with one interior knot

2.2.1.3.2 Multivariate B-Splines

Multivariate basis functions are formed by taking the *tensor product* of the univariate basis functions. Therefore, each multivariable basis function is formed from the product of n univariate basis functions, one from each input, and every possible combination of univariate basis function is taken:

$$N_{\mathbf{k}}^{j}(\mathbf{x}) = \prod_{i=1}^{n} N_{k_i}^{j}(x_i) \tag{2.13}$$

The number of basis functions of order k_i defined on an axis with r_i interior knots is r_i+k_i. Therefore, the total number of basis functions for a multivariate B-Spline is:

$$p = \prod_{i=1}^{n} (r_i + k_i) \tag{2.14}$$

Note that there is an exponential increase of the number of basis functions, as the number of inputs grows. This is generally referred to as the "*curse of dimensionality*", which was first introduced by Bellman [56].

2.2.1.3.3 Output evaluation

The output of an AMN is a linear combination of the outputs of the basis functions:

$$y = \sum_{i=1}^{p} \boldsymbol{\varphi}_i w_i \,, \tag{2.15}$$

where $\boldsymbol{\varphi}_i = N_{\mathbf{k}}^{i}(\mathbf{x})$.

In compact form the output is:

$$\mathbf{y} = \boldsymbol{\Gamma}(\boldsymbol{\lambda})\mathbf{w} \tag{2.16}$$

In (2.16)**,** $\boldsymbol{\Gamma}$ is the basis functions matrix of size $m \times p$, where m is the number of input patterns.

And for any combination of the interior knots, the optimal value of the linear parameters (in the least squares sense) is:

$$\hat{\mathbf{w}} = \boldsymbol{\Gamma}^{+}(\boldsymbol{\lambda})\mathbf{t} \tag{2.17}$$

where $\boldsymbol{\Gamma}^{+}$ denotes the pseudo-inverse of the $\boldsymbol{\Gamma}$ matrix.

2.2.1.3.4 Example of a bivariate quadratic B-spline

Consider a B-Spline Neural Network for a two-dimensional input space. Thus, a structure must be defined consisting of two knot vectors.

Suppose that:

- in dimension x_1 the knot vector has two interior knots, i.e.:

$$\left\{\lambda_{1,-2}, \lambda_{1,-1}, \lambda_{1,0}, \lambda_{1,1}, \lambda_{1,2}, \lambda_{1,3}, \lambda_{1,4}\right\} = \left\{-2, -3, -1, 0, 1, 2, 3\right\} \tag{2.18}$$

- in dimension x_2 there is only one interior knot:

$$\left\{\lambda_{2,-2}, \lambda_{2,-1}, \lambda_{2,0}, \lambda_{2,1}, \lambda_{2,2}, \lambda_{2,3}\right\} = \left\{-2, -3, -1, 1, 2, 3\right\} \tag{2.19}$$

Thus, the number of basis functions for this bivariate B-Spline NN is

$$p = \left(r_1 + k_1\right)\left(r_2 + k_2\right) = 4 \times 3 = 12 \tag{2.20}$$

The following figure sketches the B-Spline output from 4 of the basis functions over the input domain [-1,1]*[-1,1].

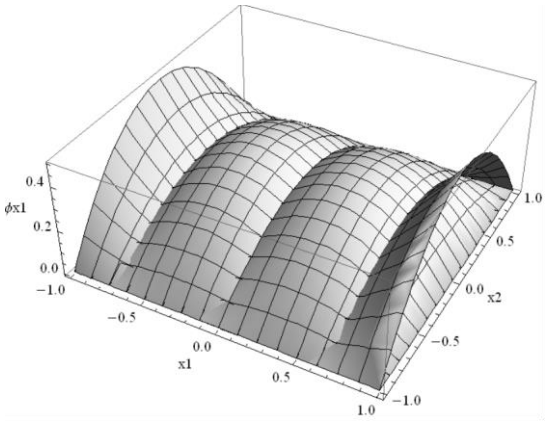


Fig. 2.8. Bivariate Quadratic spline basis functions with one interior knot in dimension x_1 and zero interior knots in dimension x_2

Notice that the basis functions sketched in the previous figure are:

$$\mathbf{N}(x_1, x_2) = \left[N_3^1(x_1) N_3^2(x_2) \quad N_3^2(x_1) N_3^2(x_2) \quad N_3^3(x_1) N_3^2(x_2) \quad N_3^4(x_1) N_3^2(x_2) \right]$$

2.2.1.3.5 Properties

In summary, there are several properties shared by the B-Splines defined in (2.8) [57]:

BSNN models offer interesting properties, such as:

- There is a simple recursion formulae (2.8) to compute the basis functions;
- They are universal approximators;
- In contrast with other neural network models, they are interpretable;
- All derivative algorithms applicable to neural networks are applicable to BSNN models;
- The parameters of the model (knots and weights) can be decomposed into nonlinear (knots) and linear (weights) parameters, which speeds up the training process;
- They are strictly local models, which is an advantage if the models are used for on-line learning.
- A basis function is defined on a *bounded* support and the output of the basis function is positive on its support:

$$\begin{cases} N_k^j(x_i) = 0, & x \notin [\lambda_{i,j-k}, \lambda_{i,j}] \\ N_k^j(x_i) > 0, & x \in (\lambda_{i,j-k}, \lambda_{i,j}) \end{cases} \tag{2.21}$$

- The basis functions form a partition of unity. This way, the sum of the outputs of the basis functions gives always one (2.9).

- The basis functions are member of the continuity class $C^{(k-2)}$, for simple knots. This requires that the basis function $N_{k_i}^{j}(x_i)$ and its derivatives up to the $(k_i\text{-}2)^{th}$ order be continuous on $x_i \in \left[x_{i_{\min}}, x_{i_{MAX}} \right]$.
- The output of the network for an input pattern lying in the j^{th} interval is bounded below and above by the values of the weights which are activated by this pattern:

$$\min(w_j, w_{j+1}, ..., w_{j+k-1}) \leq y(x) \leq \max(w_j, w_{j+1}, ..., w_{j+k-1}) \tag{2.22}$$

2.2.1.3.6 Spline differentiation

The first derivative of a univariate spline function with respect to the i^{th} input using expression (2.15) is simply given by differencing its B-spline coefficients, thus obtaining the B-spline coefficients of its first derivative:

$$\frac{d}{dx_i}\left(\sum_j w_j N_{k_i}^{j}(x_i) \right) = (k_i - 1) \sum_{j=r-k+2}^{s-1} \frac{w_j - w_{j-1}}{\lambda_{i,j+k-1} - \lambda_{i,j}} N_{k_i-1}^{j}(x_i) \tag{2.23}$$

The m^{th} derivative of a univariate spline is obtained from applying (2.23) repeatedly, i.e.:

$$\frac{d}{dx_i^{(m)}}\left(\sum_j w_j N_{k_i}^{j} \right) = \sum_j w_j^{(m+1)} N_{k_i-1}^{j} \tag{2.24}$$

Where,

$$w_i^{(m+1)} = \begin{cases} w_i, & m = 0 \\ (k_j - m)\dfrac{w_i^{(m)} - w_{i-1}^{(m)}}{\lambda_{j,i+k-m} - \lambda_{j,i}}, & m > 0 \end{cases} \tag{2.25}$$

2.2.1.3.7 Learning techniques

As noted in 2.2.1.3.2 these ANN suffer from the *curse of dimensionality*. Solutions to this issue can be solved through heuristics that determine the structure iteratively and change the values of the nonlinear parameters. The most common is the ASMOD algorithm [58] which was derived by Kavli. It consists of identifying an ANOVA model decomposition from the training data. The ASMOD algorithm is considered a mixed type algorithm as it starts from a low dimensional model and, progressively increases the complexity by adding another input dimension, eventually reaching a final model which best represents the input to output relation. Occasionally it applies a strategy of reducing the complexity by eliminating some of the interrelationships between the inputs and the output or by reducing the number of parameters. This algorithm iteratively generates a globally partitioned B-spline model which is directly applicable to neuro-fuzzy models. In alternative, meta-heuristics such as

evolutionary algorithms can be employed. One such example is Genetic Programming [59]. Both algorithms are explained in more detail in section 2.5.

2.2.2 Fuzzy systems

In the following subsections only basic concepts of fuzzy logic are described. For a more extended explanation of fuzzy logic and fuzzy systems the reading of [60][61][62] is suggested.

2.2.2.1 Concept of membership function and fuzzy set

Conventional set theory is based on the premise that one element either belongs to or does not belong to a set. In contrast, fuzzy set theory allows elements to have a degree of membership of a particular set so that an element can be assumed to be "somewhat" in the set.

Consider for example a conventional (crisp) set of numbers T from 25 to ∞, which could be the concept of some person describing the atmospheric temperature as being "hot". From conventional set theory the set of temperature being hot:

$$T = \{x \in \Re \mid x \geq 25\} \tag{2.26}$$

In fuzzy set theory there is no precise representation of imprecise knowledge and so a membership function is defined. The membership function describes a relationship between a variable and the degree of membership to a certain fuzzy set that corresponds to possible values for that variable. The degree of membership is usually defined in terms of a number in the range [0,1]. Zero implies total absence of membership and 1 complete membership whereas any value in between means partial membership.

In this example, the set of temperature being hot T would be described by a crisp membership function $\mu_T : \Re \to \{0,1\}$ defined as

$$\mu_T(x) = \begin{cases} 1, & x \geq 25 \\ 0, & otherwise \end{cases} \tag{2.27}$$

Equation (2.27) defines a membership representing a crisp set, where all real numbers map onto the two points $\{0,1\}$. The interpretation is either it *is* or it is *not* hot.

To define the membership of a fuzzy set T, a function $\mu(x)$ must be defined which assigns a real number in the interval [0,1] to each member of its domain.

This way, a fuzzy membership function $\mu(x)$ would be defined as:

$$\mu(x) \in [0,1] \quad for \quad x \in X\,, \tag{2.28}$$

where *X* refers to the universe of discourse, the possible values over which variable *x* should be defined.

One possible function to represent the membership to the fuzzy set *temperature is hot*, could be (many more can be defined):

$$\mu(x)=\begin{cases}0, & x<20\\ \dfrac{x-20}{25-20}, & 20\le x\le 25\\ 1, & x\ge 25\end{cases} \tag{2.29}$$

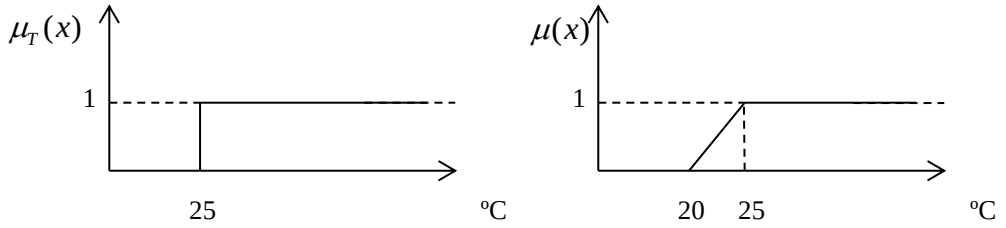


Fig. 2.9. Left: Membership function for a crisp set of $\Re$; right: membership function for fuzzy set *temperature is hot*.

Specific values like 15, 20 and 25 are called *crisp* values. In the fuzzy set theory, linguist terms like *hot* or *cold* are designated by *fuzzy* or *linguistic* values.

Possible fuzzy membership functions are the Trapezoidal, Triangular or Gaussian function.

The next figure shows a fuzzy partition with 5 fuzzy sets associated with the linguistic terms cold, warm, hot, very hot and extremely hot.

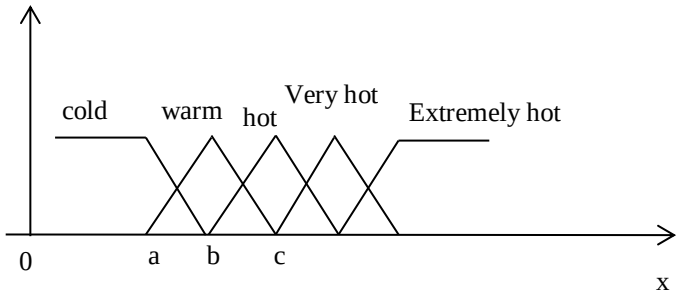


Fig. 2.10. Example of fuzzy partition with 5 fuzzy sets

Using the membership function to represent the fuzzy set is called *vertical representation*, if one regards the graphical representation of the function. See for example fuzzy set *warm* in the previous figure. It can be represented by a triangular membership function with parameters {a,b,c}.

But it can happen that a fuzzy set may be defined from all membership degrees α, those elements of *x* that have at least the membership degree α. This is called the *horizontal representation* of fuzzy sets by using the α-cuts.

This way, the set

$$[\mu]_\alpha = \{x \in X \mid \mu(x) > \alpha\} \tag{2.30}$$

is called the α-cut of μ. As an example please see Fig. 2.11.

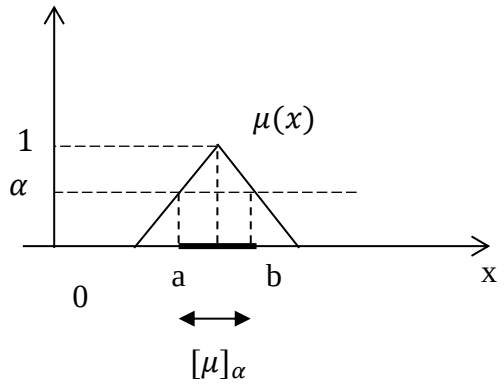


Fig. 2.11. α-cuts on a fuzzy set.

2.2.2.2 Operations on fuzzy sets

To combine several fuzzy sets into statements that can form a rule of a fuzzy system, one has to be able to yield similar unary and binary operations as used in classical logic. The basic set theoretical operations of intersection, union and complement can also be applied to fuzzy sets. However it is desirable that such operations be calculated element by element, i.e., one seeks a function

$$\Upsilon : [0,1]^2 \to [0,1] \tag{2.31}$$

for which $\left(\mu_i \cap \mu_j\right)(x) = \Upsilon\left(\mu_i(x), \mu_j(x)\right)$ holds.

To include Υ as an intersection operator it has to fulfill certain axioms which lead to the definition of the *t-norm*. Dual in the concept is the *t-conorm* that can be used for the definition of generalized union operators.

Examples of *t-norm* (*T*) functions are:

$$\begin{aligned} T_{\min}(a,b) &= \min(a,b) \\ T_{prod}(a,b) &= ab \end{aligned} \tag{2.32}$$

Examples of *t-conorm* (T_C) functions are:

$$T_{C\max}(a,b) = \max(a,b)$$
$$T_{C\min}(a,b) = (1, a+b) \tag{2.33}$$

2.2.2.3 Structure of a fuzzy system

The basic configuration of a fuzzy system is depicted in Fig. 2.12 where the following components can be identified [61]:

- data preprocessing. This can be seen as a mapping of the physical values of the input of the fuzzy system to a proper normalized domain via scaling.

- fuzzification. It is responsible for the mapping of the crisp values of the preprocessed inputs of the model into suitable fuzzy sets, represented by fuzzy membership functions. In other words, a fuzzifier calculates the degree of membership of crisp variables to multiple fuzzy sets by evaluating a membership function.

- rule base and data base. The fuzzy rule base stores a number of *if-then* rules which represent expert knowledge.

- inference engine or fuzzy reasoning. This is the computational method which calculates the degree to which each rule fires for a given fuzzified input pattern by considering the rule and label sets. The *degree of firing* or *firing strength*, s_i, depends on the way used to implement the *Premise* proposition in (2.37).There are different logical operators from classical logic which can be applied to calculate the *degree of firing*. Examples of such operators are the *min*, *max*, and *product* functions.

- defuzzification. Within the defuzzifier, a decision is made out of the information provided by each one of the rules.

Defuzzification can be seen as an operator:

$$D: F(x) \to x \tag{2.34}$$

assigning to each fuzzy set $B \in F(x)$ a crisp value $y = D(B) \in x$.

In the case of linguistic fuzzy models, the most usual method is the *centre-of-gravity* or *centroid* method:

$$D^{COG}(B) = \frac{\int_X xB(x)dx}{\int_X B(x)dx}, \tag{2.35}$$

for a universe of discourse *X*.

Other defuzzification operators include the Centre of Sums- defuzzification (COS) or Basic Defuzzification Distributions (BADD).

- postprocessing. It is the step that provides the output of the fuzzy system based on the crisp signal obtained from defuzzification.

In a very concise way, fuzzy models are considered as models that operate on fuzzy sets, thus presenting a methodology for conveniently represent an efficient processing of linguistic knowledge. If a crisp input (real input x) is presented to the fuzzy system, then the membership functions of the multivariate fuzzy input linguistic variables are computed. The network output is obtained by *defuzzifying* this information.

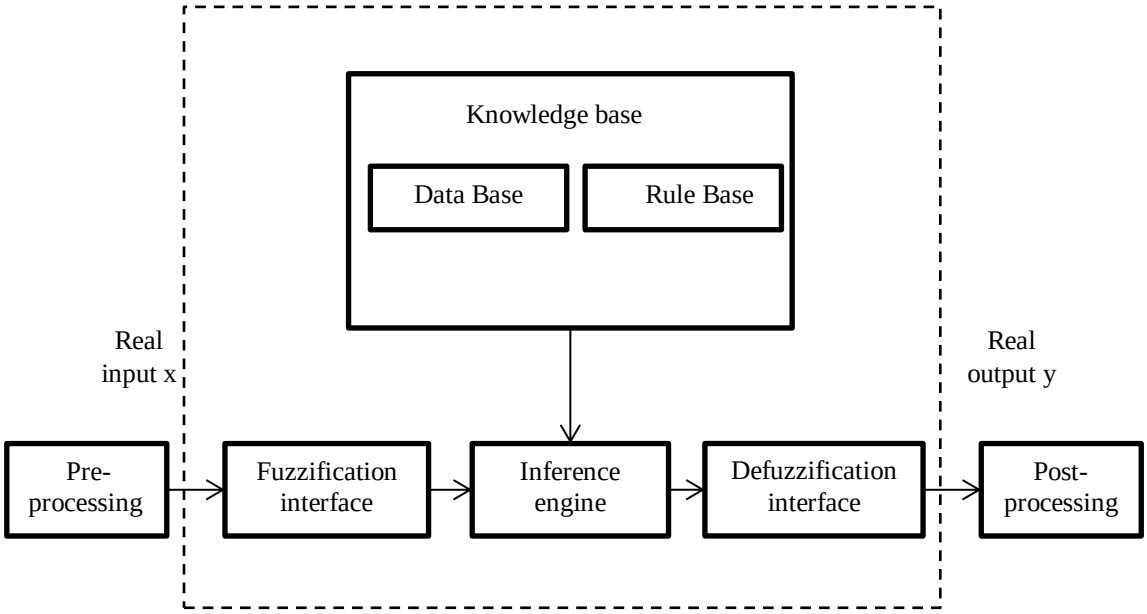


Fig. 2.12. Block diagram of a fuzzy system

2.2.2.4 Fuzzy rule base

In 1973, Zadeh [63] pointed out that the new fuzzy concept could be used for describing very complex problems with a system of fuzzy relations represented by a fuzzy rule base.

A fuzzy rule base contains fuzzy rules that map the multivariate fuzzy input set to the univariate output set.

A rule R_i, is commonly denoted by:

$$R_i\text{: IF } (x_1 \text{ is } A_{i1}) \text{ AND } (x_2 \text{ is } A_{i2}) \text{ AND}$$
$$\ldots \textit{ AND } (x_n \textit{ is } A_{in}) \textit{ THEN } (y \textit{ is } B_i), \tag{2.36}$$

where A_{ij} and B_i are fuzzy sets, x_j and y are the fuzzy inputs and output, respectively.

The general form of a rule is the following:

$$\textit{IF Premise THEN Conclusion}, \tag{2.37}$$

where the *Premise* consists of antecedents linked by the *AND* operator, and *Conclusion* consists of the consequents.

To implement a rule set, each of the univariate fuzzy linguistic statements, like x_n *is* A_{in}, needs to be defined and also the operators used to implement the underlying fuzzy logic, such as *AND*, *THEN*, etc., have to be specified. Because there is no single implementation, many implementation methods have been proposed [60].

2.2.2.5 Inference engine

With the logic operators from section 2.2.2.2 it is possible to combine the degrees of membership of the fuzzy sets within the rule premise.

Take for example Fig. 2.13 where two membership functions, μ_1 and μ_2, represent the membership degree of two fuzzy linguistic terms *A* and *B*, respectively.

Considering that at *A* and *B*, respectively:

$$\begin{cases} x_1 = 15 \\ x_2 = 1.5 \end{cases} \tag{2.38}$$

A degree of membership of:

$$\begin{cases} \mu_1(x_1) = 0.5 \\ \mu_2(x_2) = 0.5 \end{cases} \tag{2.39}$$

would be obtained.

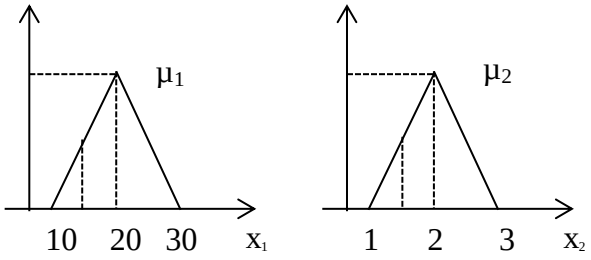


Fig. 2.13. Membership functions for antecedents variables x_1 and x_2

By combining the degrees of membership of all linguistic terms the *degree of firing* for each rule is calculated. This value denotes how well the rule premise matches a specific input value. Obviously the outcome will depend on the specific choice of the logic operators. For example, the *min* operator would result in a degree of firing of 0.5, while the product operator would produce a degree of firing of 0.25.

To generate the output of the fuzzy system (a crisp value), first the *degree of firing* for each rule has to be calculated, secondly, the consequents must also be evaluated and accumulated and, lastly, the fuzzy set has to be defuzzified in order to obtain a crisp value.

The procedure for the last steps described in this last paragraph is dependent on the specific type of the fuzzy rules consequents.

2.2.2.6 Models of fuzzy systems

According to the form of the consequent proposition and to the structure of the rule base, three classes of fuzzy models can be enumerated [61]:

- *fuzzy linguistic models* or *Mamdani-type* models, where both the antecedent and consequent are fuzzy propositions.

- *fuzzy relational models* which are considered as a generalization of the linguistic model, allowing a particular antecedent proposition to be associated with several different consequent propositions via a fuzzy relation.

- *Takagi-Sugeno fuzzy models* where the consequent is a crisp function of the input variables, $f_j(x_i)$:

$$R_j: \textbf{If } x_1 \text{ is } A_{1,j} \textbf{ and … and } x_n \text{ is } A_{n,j} \textbf{ then } y=f_j(\boldsymbol{x}_i) \tag{2.40}$$

The next section gives a description on the concepts and principles associated to two of the classes of fuzzy systems: the *Mamdany-type* model and the *Takagi-Sugeno* type model.

2.2.2.6.1 Mamdany-type models

This is the most popular and widely used type of fuzzy models. They are the most appealing fuzzy models since both inputs and output are described by linguistic variables.

The general rule for a Multiple-Input-Single-Output rule base is

$$R_j: \textbf{If } x_1 \text{ is } A_{j,1} \textbf{ and … and } x_n \text{ is } A_{j,n} \textbf{ then } y \text{ is } B_j \tag{2.41}$$

where R_j denotes the j^{th} rule.

The antecedent variables x_i represent the input of the fuzzy system. $A_{i,j}$ and B_j are fuzzy sets described by membership functions $\mu_{A_{i,j}}(x_i):\rightarrow[0,1]$ and $\mu_{B_j}(y):\rightarrow[0,1]$, x_i is the input for the i^{th} dimension and y is the output.

With a Mamdani-type fuzzy system the following steps must be carried out.

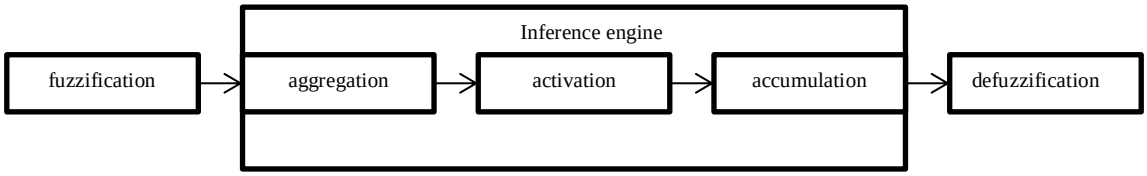


Fig. 2.14. Steps for evaluating a Mamdani-type fuzzy system

Fuzzification maps crisp inputs to degrees of membership. Aggregation combines the linguistic terms and produces the *degree of firing* of the rule. These two steps are identical no

matter what type of fuzzy system is assumed. On the other hand the last three steps require more complex calculations, as is depicted in Fig. 2.15.

If the fuzzy sets are defined by fuzzy trapezoidal membership functions, assume $A_{ij}(a_{ij},b_{ij},c_{ij},d_{ij})$ the membership function belonging to the i^{th} rule and the j^{th} input variable, and $B_i(a_i,b_i,c_i,d_i)$ the output membership function of the i^{th} rule. The relative importance of the j^{th} fuzzy variable in the i^{th} rule is given as

$$\mu_{ij}\left(x_j\right)=\frac{x_j-a_{ij}}{b_{ij}-a_{ij}}\kappa_{i,j,1}(x_j)+\kappa_{i,j,2}(x_j)+\frac{d_{ij}-x_j}{d_{ij}-c_{ij}}\kappa_{i,j,3}(x_j), \tag{2.42}$$

where $a_{ij}\leq b_{ij}\leq c_{ij}\leq d_{ij}$ must hold and,

$$\kappa_{i,j,1}(x_j)=\begin{cases}1, if \quad x_j\in\left[a_{ij},b_{ij}\right]\\ 0, \ if \quad x_j\notin\left[a_{ij},b_{ij}\right]\end{cases}$$

$$\kappa_{i,j,2}(x_j)=\begin{cases}1, \ if \quad x_j\in\left(b_{ij},c_{ij}\right)\\ 0, \ if \quad x_j\notin\left(b_{ij},c_{ij}\right)\end{cases}$$

$$\kappa_{i,j,3}(x_j)=\begin{cases}1, \ if \quad x_j\in\left[c_{ij},d_{ij}\right]\\ 0, \ if \quad x_j\notin\left[c_{ij},d_{ij}\right]\end{cases}. \tag{2.43}$$

Fig. 2.15 describes the inference engine for a system where the *t-norm* is the min operator and the *t-conorm* is the max function.

In the activation step, the output activation ($\mu_{B_i}(y)$) for the i^{th} rule is evaluated. It is obtained by "cutting off" the "output" fuzzy set B_{i} of the i^{th} rule at the membership degree determined by the rule's *degree of firing*. In this example, the *degree of firing* is calculated using the *min* operator. Then, the activation degree of the i^{th} rule is:

$$s_i=\min_{j=1}^{n}\mu_{ij}(x_j) \tag{2.44}$$

where $\mu_{ij}(x_j)$ is the membership function in the j^{th} input variable for the i^{th} rule. The i^{th} output is being cut in the height s_i.

In the accumulation step the output activations for all rules are combined. This is usually done by computing the maximum of all output activations. The result is one fuzzy set (denoted by *B*). If a crisp output is required then defuzzification is applied in the final step.

If the COG defuzzification method from (2.35) is used, the integrals can be easily computed. Therefore $y(\boldsymbol{x})$ will be the following:

$$y(\mathbf{x}) = \frac{1}{3}\frac{\sum_{i=1}^{R}(C_i + D_i + E_i)}{\sum_{i=1}^{R} 2s_i(d_i - a_i) + s_i^2(c_i + a_i - d_i - b_i)}$$

$$C_i = 3s_i(d_i^2 - a_i^2)(1 - s_i)$$
$$D_i = 3s_i^2(c_i d_i - a_i b_i)$$
$$E_i = s_i^3(c_i - d_i + a_i - b_i)(c_i - d_i - a_i + b_i) \quad (2.45)$$

In Fig. 2.15, the output for the crisp values (x_1,x_2) is thus y'.

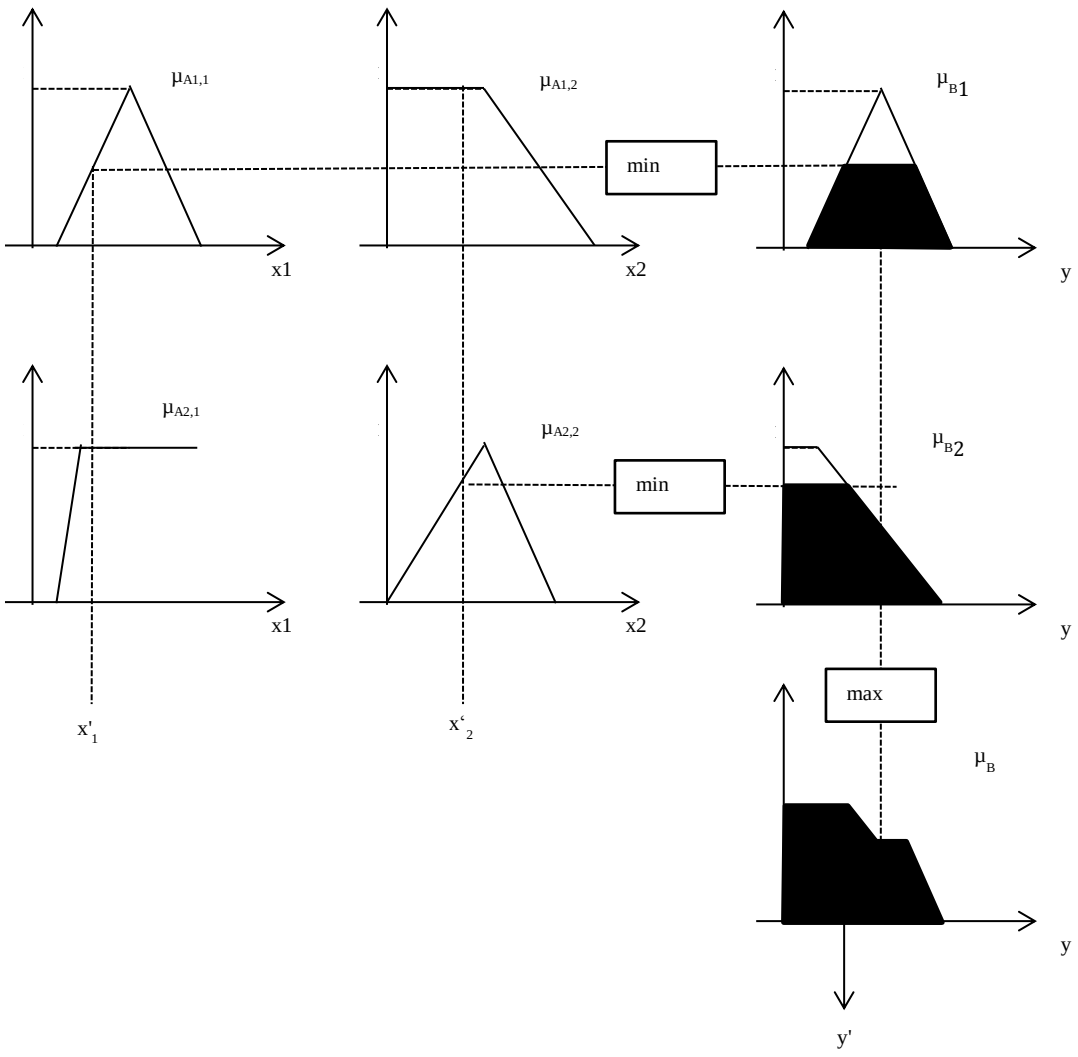


Fig. 2.15. Example of fuzzy inference for a Mamdani-type system with two rules

2.2.2.6.2 Takagi-Sugeno type models

This type of fuzzy system was proposed by Takagi and Sugeno [64].

The inference engine for this type of fuzzy models is less complex than that of the *Mamdani-type*.

In the Takagi-Sugeno type, rules consist of two parts: The fuzzy antecedents and, a mathematical function as consequent part:

$$R_j\text{: } \textbf{If } x_1 \text{ is } A_{j,1} \textbf{ and } \dots \textbf{ and } x_n \text{ is } A_{j,n} \textbf{ then } y=f_j(x_{1,\dots,} x_n) \tag{2.46}$$

where x_i is the antecedent variable in the i^{th} input.

There are two order TS fuzzy systems which depend on the way the rule consequents f_j are chosen:

- zero-th order TS fuzzy system. The consequents are chosen as constants. These are typically known as Sugeno TS fuzzy systems;

- first order TS fuzzy system. The rule consequents are a linear function of the inputs:

$$y = a_0^{(i)} + a_1^{(i)} x_1 + ... + a_n^{(i)} x_n \tag{2.47}$$

2.2.2.6.2.1 Example of a TS fuzzy system

The fuzzy inference mechanism for a TS model with two rules is depicted in Fig. 2.16.

The two rules are defined as

$$R_1\text{: } \textbf{If } x_1 \text{ is } A_{1,1} \textbf{ and } x_2 \text{ is } A_{1,2} \textbf{ then } \; f_1 = a_1^{(1)} x_1 + a_2^{(1)} x_2 + a_0^{(1)}$$

$$R_2\text{: } \textbf{If } x_1 \text{ is } A_{2,1} \textbf{ and } x_2 \text{ is } A_{2,2} \textbf{ then } \; f_2 = a_1^{(2)} x_1 + a_2^{(2)} x_2 + a_0^{(2)} \tag{2.48}$$

Using fuzzy inference based on product-sum-COG, the output is inferred by taking the weighted average of the consequent functions:

$$y = \frac{\sum_{i=1}^{R} s_i f_i}{\sum_{i=1}^{R} s_i} \tag{2.49}$$

Where,

$$s_i = \prod_{j=1}^{n} \mu_{A_{i,j}}(x_j), \tag{2.50}$$

and R is the number of fuzzy *if-then* rules and s_i are the rules' *degree of firing* and $\mu_{A_{i,j}}(x_j)$ is the membership function from the i^{th} rule and j^{th} input variable.

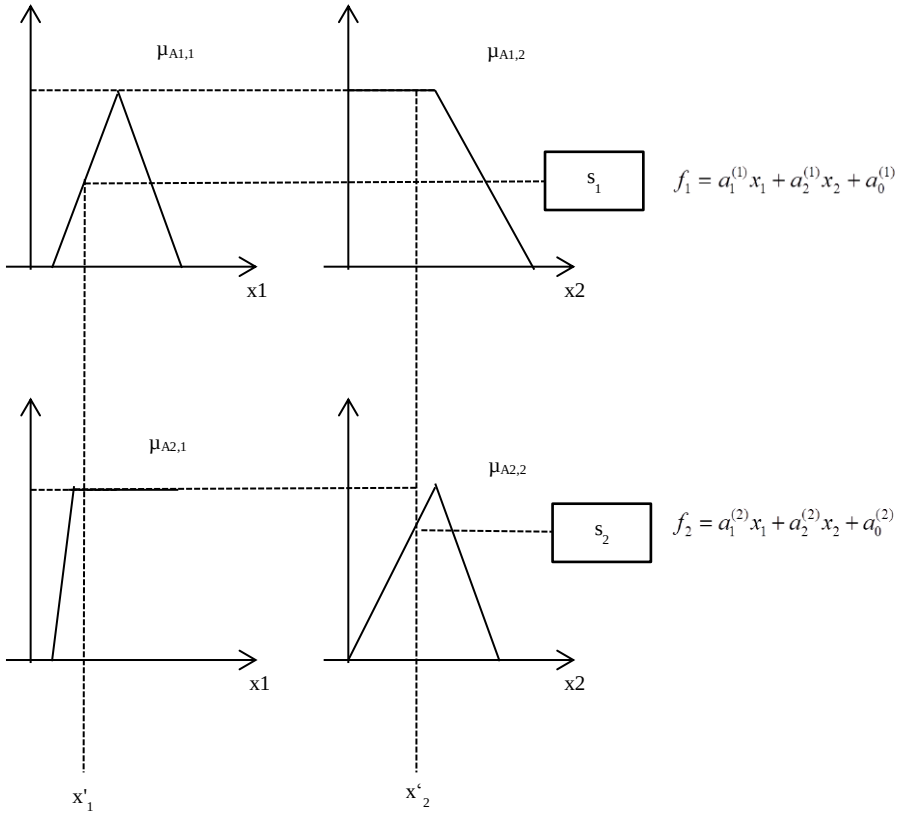


Fig. 2.16. Example of fuzzy inference for a TSK model with two rules and two input variables

2.2.2.7 Learning in fuzzy systems

Several techniques have been developed which can be embraced in the class of methodologies to follow ([22][23]).

With *template-based membership functions*, the domains of the antecedent variables are *a-priori* partitioned by a number of user-defined membership functions. A rule base is formed with the goal of covering all of the combinations of the antecedent terms. This brings several problems, one being the complexity, because the number of rules grows exponentially. Also, the lack of prior knowledge on which variables involve the nonlinearity of the system means that the antecedent variables are partitioned uniformly which can be far from representing the system's behavior.

As an example, to determine an automatic steering system for their model car, Sugeno and Nishida [65], suggested, in 1985, that a "good" approach is one where the operator's experience and knowledge about the control is translated into the fuzzy model. Other examples of this knowledge based methods can be found in [66][67][68].

To overcome the lack of expert knowledge and its drawbacks, construction methods based on *fuzzy clustering* can be used [69]. These methods originate from data mining and pattern recognition where they are used to partition unlabeled data into meaningful groups (clusters), in response to a pre-defined pattern approximation measure. Fuzzy clustering uses the concept of fuzzy membership to represent the degree to which a given data object is similar to a prototypical object. Data vectors are clustered so that data belonging to one cluster as similar as possible, and data from different clusters are dissimilar as possible. To overcome the redundancy problems obtained from fuzzy clustering related to similar membership functions, similarity-based reduction techniques can be employed [70].

Discrete search methods include techniques that successively decompose the antecedent space into hyper-rectangles by axis-orthogonal splits. In each step the quality of the model is evaluated and the region with the worst local error measure is divided into two subsets. As an iterative procedure, it stops when the desired error measure is met or a maximum number of rules is reached. Tree-search algorithms are often used to decompose the input space. Though this approach can be effective for high-dimensional data problems, it can produce quite a few large number of rules. This type of methods is described in more detail in section 2.5.2.5.

Other techniques explore the fact that at a computation level a fuzzy system can be seen as a layered structure similar to ANNs, what is termed as neuro-fuzzy system. So, all training algorithms from the area of neural networks can be employed [71][72].

After selecting the structure, parameter estimation is carried out to fine tune the antecedent and consequent parameters which correspond to the breakpoints used in the definition of the membership functions [73].

2.3 Training Algorithms

Finding the appropriate model for mapping the behavior of a certain system requires determining a vector $\mathbf{z} \in Z$ of parameters of the system under consideration with the objective of minimizing (or maximizing) a certain quality criterion or objective function:

$$f(\mathbf{z}) \rightarrow \min_{\mathbf{z}} \tag{2.51}$$

The solution to the global optimization problem in (2.51) requires finding a vector $\mathbf{z}'$ so that $\forall \mathbf{z} \in Z : f(\mathbf{z}) \geq f(\mathbf{z}') = f'$.

However, multimodality (several local minima), constraints on the *Z* set, large dimensionality, nonlinearities, non-differentiability, etc., make the optimization task very difficult if not unsolvable.

Moreover, model identification requires having a set of historical data. Using computational intelligence methodologies training is in most of the times an *off-line* optimization procedure that modifies the internal parameters of the model with the intention to minimize the error (in some form) between the model output and the desired output value. A single data set is usually processed by the optimization procedure in many iterations before an acceptable fit is obtained between the model output prediction and the actual output measurement.

For the models described in section 2.2 a learning methodology is usually required. Thus, optimization deals with the application of an appropriate approach that allows one to determine both the model parameters and the model structure. The first problem (parameter estimation) is dealt in this section; the latter (model structure) will be discussed in Section 2.5.

Basically, training can be classified according to one of the following three paradigms:

- Supervised learning
- Reinforcement learning
- Unsupervised learning

Supervised learning techniques are based on knowledge about the input and output data of a process. The objective is to minimize some error measure between the process and the model behavior. This fact makes them most suitable for training neuro-fuzzy systems or neural networks for systems identification, pattern recognition or classification purposes.

Reinforcement learning is typically used when only information about the quality of the model is available. Take for instance game applications where the quality of the each move cannot be evaluated. With reinforcement learning, the quality of the applied strategy is only evaluated at the end of the game, which corresponds to the time instance where the success of the strategy is assessed.

Methods that use unsupervised learning use only input data. These methods are mostly applied for preprocessing data. Their objective is grouping or clustering of input data.

The next section gives an overview on the supervised learning techniques used for modeling learning and identification.

2.3.1 Supervised learning techniques

Supervised learning techniques aim at minimizing some error measure, typically obtained from the output of a process and the model behavior.

These are based on knowledge about the input and output data of a process. The objective is to minimize some error measure between the process and the model behavior expressed by (1.3). To employ an optimization algorithm, a training criterion (or simply criterion) has to be defined, which is seen as a mathematical formulation of what has to be minimized.

Typically, this criterion is of the form of a loss function, which is usually computed as the sum of the square of the difference - $e(i)$ - between the measured output $t(i)$ and the model output $y(i)$, over *m* patterns:

$$\Omega(\mathbf{z}, \mathbf{X}) = \sum_{i=1}^{m} \frac{\left(t(\mathbf{x}_i) - y(\mathbf{x}_i, \mathbf{z})\right)^2}{2}, \tag{2.52}$$

A compact form of the former criterion is

$$\Omega(\mathbf{z}, \mathbf{X}) = \frac{\left\|\mathbf{t} - \mathbf{y}(\mathbf{z})\right\|_2^2}{2} = \frac{\left\|\mathbf{t} - \mathbf{y}\right\|_2^2}{2} = \frac{\left\|\mathbf{e}\right\|_2^2}{2} \tag{2.53}$$

where $\| \ \|_2$ is the Euclidean norm.

As the gradient of the error function in (2.53) is not linear in the **z** parameters, it is necessary to adopt a nonlinear optimization technique to search the optimal values of the parameters.

Nonlinear local optimization can be divided into two main classes: direct search and gradient based techniques. Though direct search techniques may be easy to understand and to implement, they are not reasonable to apply if the derivatives of the criterion are easily available. Furthermore, they require smooth functions for higher performance. Examples of direct search techniques are the *Simplex Search method*, *Hooke-Jeeves*, etc. For further understanding on these techniques the reader can refer to [62].

On the other hand, gradient based algorithms are the most common nonlinear local optimization techniques. However, their success depends mainly on the availability of the derivatives of the training criterion whether through analytic calculations or by finite difference techniques.

The concept of all gradient based algorithms is to update the model's parameters **z** proportionally to some step size into the gradient direction rotated and scaled by a direction matrix **R**:

$$\begin{aligned}\mathbf{z}[k+1] &= \mathbf{z}[k] + \mathbf{s}[k] \\ \mathbf{s}[k] &= -\eta.\mathbf{R}[k].\mathbf{g}[k]\end{aligned} \tag{2.54}$$

The existent algorithms are distinguished by different choices of the rotation matrix **R** and step size η.

In the following subsections the update using first order and second order optimization techniques is described. This includes the *steepest descent*, *Newton*, *Quasi-Newton* algorithms, *Gauss-Newton* and the *Levenberg-Marquardt* algorithm.

2.3.2 Steepest Descent

Steepest descent is the simplest version of (2.54) since the rotation matrix **R** is the identity matrix, **I**.

This way, to use this technique one requires only the gradient of criterion (2.53).

The gradient of criterion (2.53) is

$$\mathbf{g}_{\Omega}(\mathbf{x},\mathbf{z}) = \frac{\partial \Omega}{\partial \mathbf{z}^T} = -\mathbf{J}^T(\mathbf{x},\mathbf{z})\mathbf{e}(\mathbf{x},\mathbf{z}) \tag{2.55}$$

where $\mathbf{J}$ is the *Jacobian* matrix and is defined by

$$\mathbf{J}(\mathbf{x},\mathbf{z}) = \frac{\partial \mathbf{y}(\mathbf{x},\mathbf{z})}{\partial \mathbf{z}} \tag{2.56}$$

And so the steepest descent update is

$$\mathbf{s}[k] = -\eta \mathbf{J}^T \mathbf{e}. \tag{2.57}$$

Though this algorithm is extensively used in many model architectures, it has special impact with the MLPs where it is known as the error-backpropagation algorithm (BP) [74]. The gradient is computed using a local application of the derivative chain rule [8].

Several problems have been reported related to the BP algorithm. It is an unreliable algorithm (it can diverge), it is difficult to determine a correct step size, and the convergence rate is usually very slow.

To cope with these problems many strategies have been developed. They can be classified as global and local adaptation strategies [75]. Example of global adaptation techniques include *steepest descent*, and *conjugate gradient* methods. Examples of local adaptation strategies include *sign changes*, *rprop* [76], *quickprop* [77], etc. Algorithms like *dynamic momentum factor* or *dynamic learning rate* are also a part of local adaptation strategies.

2.3.3 Newton's method

Equation (2.53) can be written as

$$\Omega(\mathbf{x},\mathbf{z}) = \frac{1}{2}\mathbf{e}^T\mathbf{e}. \tag{2.58}$$

Consider now the problem of minimizing (2.58):

$$\hat{\mathbf{z}} = \arg\min_{\mathbf{z}}\left(\Omega(\mathbf{x},\mathbf{z})\right). \tag{2.59}$$

Based on Newton's method and assuming the input sample *x** for which $\frac{\partial\Omega(x^*)}{\partial\mathbf{z}^T} = 0$, expansion of (2.79) in a Taylor Series yields:

$$\Omega(x+s) = \Omega(x) + \frac{\partial\Omega(x)}{\partial\mathbf{z}^T}s + O(\|s\|^2) \tag{2.60}$$

And, if $s \to 0$ then,

$$\Omega(x+s) \approx \Omega(x) + \frac{\partial\Omega(x)}{\partial\mathbf{z}^T}s \tag{2.61}$$

The derivative of (2.61) is

$$\frac{\partial\Omega(x+s)}{\partial\mathbf{z}^T} \approx \frac{\partial\Omega(x)}{\partial\mathbf{z}^T} + \frac{\partial\Omega(x)}{\partial^2\mathbf{z}^T}s \tag{2.62}$$

Regarding that $\frac{\partial\Omega(x^*)}{\partial\mathbf{z}^T} = 0$ and given (2.62), then a solution for (2.59) is given as

$$\mathbf{s}_n = -\frac{\partial\Omega(x)}{\partial^2\mathbf{z}^T}\frac{\partial\Omega(x)}{\partial\mathbf{z}^T} = -\left(\mathbf{H}(x)\right)^{-1}\frac{\partial\Omega(x)}{\partial\mathbf{z}^T} = -\mathbf{H}^{-1}\mathbf{J}\mathbf{e}, \tag{2.63}$$

where **H** is the Hessian matrix, defined as

$$\mathbf{H} = \begin{bmatrix} \frac{\partial\Omega(x)}{\partial^2 z_1} & \frac{\partial\Omega(x)}{\partial z_1\partial z_2} & \cdots & \frac{\partial\Omega(x)}{\partial z_1\partial z_{n_z}} \\ \frac{\partial\Omega(x)}{\partial z_2\partial z_1} & \frac{\partial\Omega(x)}{\partial^2 z_2} & \cdots & \frac{\partial\Omega(x)}{\partial z_2\partial z_{n_z}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial\Omega(x)}{\partial z_p\partial z_1} & \frac{\partial\Omega(x)}{\partial z_p\partial z_2} & \cdots & \frac{\partial\Omega(x)}{\partial^2 z_{n_z}} \end{bmatrix}. \tag{2.64}$$

assuming n_z parameters. This way, the rotation matrix **R**[k] when using the Newton method becomes:

$$\mathbf{R}[k] = \mathbf{H}^{-1} \tag{2.65}$$

As it is not always easy to compute the Hessian matrix, alternative methods denoted by quasi-Newton can be adopted. The idea of the quasi-Newton methods is to replace the Hessian or its inverse by an approximation.

Conjugate gradient methods are considered a hard approximation to the quasi-Newton methods. They do not require a direct computation of the Hessian and thus have a computational complexity of order n_z. Despite requiring more iterations for convergence the computational time will be shorter because each iteration is less computationally expensive. The update for this class of methods has the form:

$$\begin{gathered}\mathbf{s}\left[k\right]_{cg} = -\eta\mathbf{p}[k-1] \\ \text{where } \mathbf{p}[k-1] = \mathbf{g}[k-1] - \beta[k-1]\mathbf{p}[k-2]\end{gathered}. \tag{2.66}$$

One very popular approach to finding the value of β is due to *Fletcher-Reeves* [78]:

$$\beta[k-1] = \frac{\mathbf{g}^T[k-1]\mathbf{g}[k-1]}{\mathbf{g}^T[k-2]\mathbf{g}[k-2]}. \tag{2.67}$$

Equation (2.66) shows that the method is similar to steepest descent with a momentum term whose value depends on the direction taken by the search direction. The value of β determines whether the search direction follows the steepest descent methodology or the quasi-Newton methods.

Another approach is *the Broyden-Fletcher-Goldfarb-Shanno* (BFGS) method [78] which updates the initial Hessian matrix:

$$\begin{gathered}\mathbf{H}[k+1] = \mathbf{H}[k] + \left(1 + \frac{\mathbf{q}^T[k]\mathbf{H}[k]\mathbf{q}[k]}{\mathbf{s}^T[k]\mathbf{q}[k]}\right)\frac{\mathbf{s}[k]\mathbf{s}^T[k]}{\mathbf{s}^T[k]\mathbf{q}[k]} - \\ \left(\frac{\mathbf{s}[k]\mathbf{q}^T[k]\mathbf{H}[k] + \mathbf{H}[k]\mathbf{q}[k]\mathbf{s}^T[k]}{\mathbf{s}^T[k]\mathbf{q}[k]}\right)\end{gathered} \tag{2.68}$$

where

$$\begin{aligned}\mathbf{R}[k] &= \mathbf{H}[k] \\ \mathbf{q}[k] &= \mathbf{g}[k] - \mathbf{g}[k-1]\end{aligned} \tag{2.69}$$

The *Gauss-Newton* method approximates the Hessian matrix by $\mathbf{H} \approx \mathbf{J}^T\mathbf{J}$ thus returning the following update:

$$\mathbf{s}[k]_{gn} = -\left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}\mathbf{e} \tag{2.70}.$$

Some advantages can be assigned to the Gauss-Newton method when in comparison to the Newton method. First, it is computationally less demanding because it avoids the calculation

of second derivative terms when estimating the *Hessian* matrix. Second, the *Hessian* estimate is generically positive definite and third, it displays better transformation invariance properties. However, it possess the disadvantage that it has a generic first order rate of convergence and, when applied to neural networks, it is numerically positive semi-definite.

2.3.4 Levenberg-Marquardt algorithm

Because of singularity issues with equation (2.70), Levenberg-Marquardt [79][80] proposed an algorithm where a *regularization term* is added to the Hessian matrix.

At the k^{th} iteration, the update ***s**[k]*, is given from the solution of:

$$\left(\mathbf{J}^T[k]\mathbf{J}[k]+\rho[k]\mathbf{I}\right)\mathbf{s}[k]=-\mathbf{J}[k]\mathbf{e}[k] \tag{2.71}$$

which can be expressed as

$$\mathbf{s}[k]=-\begin{bmatrix}\mathbf{J}[k]\\ \rho\mathbf{I}\end{bmatrix}^{+}\begin{bmatrix}\mathbf{e}[k]\\ \mathbf{0}\end{bmatrix} \tag{2.72}$$

Basically, the Levenberg-Marquardt (LM) algorithm performs an interpolation between the Gauss-Newton and steepest descent gradient method based upon the maximum neighborhood in which the truncated Taylor series provides an adequate representation of the nonlinear model [79]. Thus, the LM method is classified as a trust-region method.

The convergence direction of the method depends on the *regularization parameter* value, ρ. For high values of ρ the LM method employs a direction similar to the steepest descent method. But, for low values of ρ the Gauss-Newton direction is taken. Therefore, the algorithm, to be well implemented, depends on two other parameters:

- the predicted error vector, $\mathbf{e}^p[k]=\mathbf{e}[k]-\mathbf{J}[k]\mathbf{s}[k]$ and,
- the predicted reduction of the training criterion at iteration *k*, $\Delta\Omega^p[k]$ which reflects how well the actual function is approximated by a quadratic function:

$$\Delta\Omega^p[k]=\Omega(\mathbf{z}[k])-\frac{\left(\mathbf{e}^p[k]\right)^T\left(\mathbf{e}^p[k]\right)}{2} \tag{2.73}$$

Noting that the actual reduction is given by

$$\Delta\Omega[k]=\Omega(\mathbf{z}[k])-\Omega(\mathbf{z}[k]+\mathbf{s}[k]), \tag{2.74}$$

the decision on whether the update of the regularization parameter depends on the value of another parameter, *r*[k], which calculates the rate between the actual reduction, $\Delta\Omega[k]$ and its predicted value, $\Delta\Omega^p[k]$, at iteration *k*:

$$r[k] = \Delta\Omega[k] / \Delta\Omega^{p}[k] \tag{2.75}$$

The algorithm's outline is summarized next.

Algorithm 2.1. Levenberg-Marquardt

1. Choose the model's parameters initial values
2. Set $\rho[1] = 1$.
3. While the stopping criterion is not met, repeat:
   a. Compute $\mathbf{e}[k], \mathbf{e}^{p}[k], \mathbf{J}[k]$ and $r[k]$.
   b. Update the regularization parameter: $\rho[k+1] = \begin{cases} 4\rho[k], \text{if } r[k] < \frac{1}{4} \\ \frac{\rho[k]}{2}, \text{if } r[k] > \frac{3}{4} \\ \rho[k], \text{else} \end{cases}$
   c. If r[k]<0, the update is not accepted and ρ is increased.
   d. If r[k]>0,
      i. Compute the new update as given by (2.72).
      ii. Compute the new value for the parameters: $\mathbf{z}[k+1] = \mathbf{z}[k] + \mathbf{s}[k]$
      iii. k=k+1
   e. Test for a stopping criterion

As it can be seen from above, the LM starts with a *regularization term* value of 1.

As the *regularization parameter* influences both the direction and the update step size, there is no need to apply a linear search to find the optimal size of the update step at each iteration.

Still, the LM algorithm requires the computation of the *Jacobian* matrix, **J**, and the computation of a pseudo-inverse (2.72), which have a complexity of at least $O(m*n_z^2)$, where *m and* n_z are the number of rows and columns of **J**, respectively.

2.3.5 Parameters separability

The concept of separability was introduced as a way of eliminating the linear variables or one set of the nonlinear variables, followed by the minimization of the resulting *Variable Projection Functional* that depends only on the remaining variables.

This concept for general nonlinear least-squares problems was introduced by Golub and Pereyra [81]. A good review on the applications of this concept can be found in [82]. In the

context of neural networks it has been proposed in the training of MLPs [83], B-Splines and neuro-fuzzy systems [84] and RBFs [85]. Improvements on this concept have been proposed in [86].

Assuming that the network structure is discovered, and denoting **v** the vector of the nonlinear parameters, and ***u*** the vector of linear parameters, then **z** is an augmented vector of all design parameters (linear and nonlinear):

$$\mathbf{z} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \tag{2.76}$$

In these conditions, the output of the model in (1.3) is given by a function of the input variables ***x***, the nonlinear parameters, ***v*** and linear parameters, ***u***:

$$y(\mathbf{x}_i) = \boldsymbol{\varphi}(\mathbf{x}_i, \mathbf{v})\mathbf{u}, \tag{2.77}$$

which in compact form is:

$$\mathbf{y} = \boldsymbol{\Gamma}(\mathbf{x}, \mathbf{v})\mathbf{u} \tag{2.78}$$

In (2.78), the input data is, as usual, discretized and the model has n_u linear parameters and n_v nonlinear parameters, such that $n_z = n_u + n_v$. $\boldsymbol{\Phi}$ is the matrix of the basis functions and is of size $m \times n_u$.

The general definition of the training criterion in (2.52) using (2.78) is

$$\Omega(\mathbf{X}, \mathbf{v}, \mathbf{u}) = \frac{\left\| \mathbf{t} - \boldsymbol{\Gamma}(\mathbf{X}, \mathbf{v})\mathbf{u} \right\|_2^2}{2} \tag{2.79}$$

The gradient of criterion (2.79) is

$$\begin{aligned} \mathbf{g}_\Omega(\mathbf{X}, \mathbf{u}, \mathbf{v}, \mathbf{t}) &= \frac{\partial \Omega}{\partial \left[\mathbf{u}^T \mid \mathbf{v}^T \right]} \\ &= -\mathbf{J}_\Omega{}^T(\mathbf{X}, \mathbf{u}, \mathbf{v})\mathbf{e}(\mathbf{X}, \mathbf{u}, \mathbf{v}, \mathbf{t}) \end{aligned} \tag{2.80}$$

where $\mathbf{J}_\Omega$ is the *Jacobian* matrix and is defined by:

$$\begin{aligned} \mathbf{J}_\Omega(\mathbf{X}, \mathbf{u}, \mathbf{v},) &= \frac{\partial \mathbf{y}(\mathbf{x}, \mathbf{u}, \mathbf{v})}{\partial \left[\mathbf{u}^T \mid \mathbf{v}^T \right]} \\ &= \left[\boldsymbol{\Gamma}(\mathbf{v}, \mathbf{X}) \quad \mid \quad \frac{\partial \boldsymbol{\Gamma}(\mathbf{v}, \mathbf{X})}{\partial \mathbf{v}^T}\mathbf{u} \right] \\ &= \left[\boldsymbol{\Gamma}(\mathbf{v}, \mathbf{X}) \quad \mid \quad (\boldsymbol{\Gamma})_v(\mathbf{v}, \mathbf{X}) \right] \end{aligned} \tag{2.81}$$

For any value of the nonlinear parameters **v** it is possible to minimize (2.79) with respect to the linear parameters **u**. One approach is by applying the pseudo-inverse:

$$\hat{\mathbf{u}}(\mathbf{v}) = \boldsymbol{\Gamma}^{+}\mathbf{t} = \left(\boldsymbol{\Gamma}^{T}\boldsymbol{\Gamma}\right)^{-1}\boldsymbol{\Gamma}^{T}\mathbf{t}, \tag{2.82}$$

By incorporating (2.82) in (2.79), a new definition for the training criterion (independent of the linear parameters) is:

$$\Psi(\mathbf{X},\mathbf{v}) = \frac{\left\|\mathbf{t}-\boldsymbol{\Gamma}\hat{\mathbf{u}}\right\|_2^2}{2} = \frac{\left\|\mathbf{t}-\boldsymbol{\Gamma}\boldsymbol{\Gamma}^{+}\mathbf{t}\right\|_2^2}{2} = \frac{\left\|\mathbf{P}_{\boldsymbol{\Gamma}_\perp}\mathbf{t}\right\|_2^2}{2} \tag{2.83}$$

where $\mathbf{P}_{\boldsymbol{\Gamma}(\mathbf{X},\mathbf{v})_\perp} = \mathbf{I}-\boldsymbol{\Gamma}\boldsymbol{\Gamma}^{+}$ is the projector on the orthogonal complement of the column space of basis functions matrix, $\boldsymbol{\Gamma}$. Equation (2.83) is called the *Variable Projection Functional*.

The variable projection algorithm consists of minimizing (2.83) and then using the optimal values $\hat{\mathbf{v}}$, solve $\hat{\mathbf{u}}$ in (2.82). This algorithm usually converges in fewer iterations than the minimization of the full functional (2.79), and convergences even when the same minimization algorithm for the full functional diverges [87].

The gradient of the error with the new criterion (2.83) is:

$$\mathbf{g}_{\Psi}(\mathbf{X},\mathbf{v}) = \frac{\partial\Psi}{\partial\mathbf{v}^{T}} = -\mathbf{J}_{\Psi}{}^{T}\mathbf{e}_{\Psi}, \tag{2.84}$$

where:

$$\begin{aligned}\mathbf{J}_{\Psi}{}^{T}(\mathbf{X},\mathbf{v}) &= \mathbf{J}^{T}(\mathbf{X},\mathbf{v},\mathbf{u})\big|_{\mathbf{u}=\hat{\mathbf{u}}} = \frac{\partial\boldsymbol{\Gamma}\boldsymbol{\Gamma}^{+}}{\partial\mathbf{v}^{T}}\mathbf{t}\\ \mathbf{e}_{\Psi}(\mathbf{X},\mathbf{v}) &= \mathbf{e}(\mathbf{X},\mathbf{v},\mathbf{u})\big|_{\mathbf{u}=\hat{\mathbf{u}}} = \mathbf{P}_{\boldsymbol{\Gamma}_\perp}\mathbf{t}\end{aligned} \tag{2.85}$$

Thus, it is necessary to obtain the derivatives for the vector function in equation (2.85). Three solutions to that issue have been addressed:

- Golub and Pereyra [81] have proposed the following Jacobian:

$$\mathbf{J}_{GP} = \mathbf{P}_{\boldsymbol{\Gamma}_\perp}\mathbf{J} + \left[\boldsymbol{\Gamma}^{+}\right]^{T}\left[\frac{\partial\boldsymbol{\Gamma}}{\partial\mathbf{v}^{T}}\right]^{T}\mathbf{P}_{\boldsymbol{\Gamma}_\perp}\mathbf{t} \tag{2.86}$$

- Kaufman [88] has proposed the following Jacobian:

$$\mathbf{J}_{K} = \mathbf{P}_{\boldsymbol{\Gamma}_\perp}\mathbf{J}^{T} \tag{2.87}$$

- and Ruano [89], using Kaufman's conjecture, has proposed:

$$\mathbf{J}_{R} = \mathbf{J}^{T} \tag{2.88}$$

Notice that all three *Jacobian* matrices, if replaced in (2.55), produce the same gradient vector.

2.3.6 Starting and terminating the training

In order to apply the previous methods initial points must be provided. This can be done by specifying random initial values or using heuristics specific for each model.

The procedure of local nonlinear optimization can be summarized by the following steps:

Algorithm 2.2. Local nonlinear optimization steps

Given: a set of initial points, $\mathbf{z}[0]$, or $\mathbf{v}[0]$, if the new criterion is used

Do k=1,2,…, kMax

1. Compute $\mathbf{e}[k], \mathbf{g}[k], \mathbf{J}[k]$
2. Apply update s[k] as given by (2.54) to all the parameters, or only to the nonlinear parameters, if the new criterion is used
3. If $\mathbf{z}_{i+1}[k] < \mathbf{z}_i[k]$ use (2.89)
4. If termination criterion is satisfied end otherwise increment *k* and go to 1.

Step 3 in the algorithm refers to the situation when parameters order relation at iteration *k* must be preserved (i.e. $\mathbf{z}_{i+1}[k] > \mathbf{z}_i[k]$). This occurs in B-Splines neural networks and in fuzzy systems. If the condition in step 3 is met, in order to maintain the same search direction the update vector is reduced by a factor Δ so that the position of the $(i+1)^{th}$ parameter is located half-way between the previous distance of the two corresponding parameters:

$$\Delta = \frac{\mathbf{z}_{i+1}[k-1] - \mathbf{z}_i[k-1]}{2\left(\mathbf{s}_{i+1}[k] - \mathbf{s}_i[k]\right)} \tag{2.89}$$

In unconstrained optimization [78], three conditions are frequently combined with the maximum allowed number of iterations (*kMax*). They are presented next. These termination criteria are used in this work whenever a nonlinear local optimization technique is used. When the input data is divided into different purpose data sets, strategies based on model validation can be employed as described in subsection 2.3.6.2.

2.3.6.1 Termination criteria

A first criterion emphasizes the need for the convergence of the parameters to its optima, i.e.,

$$\left\|\mathbf{z}[k-1] - \mathbf{z}[k]\right\|_2 < \sqrt{\tau}\left(1 + \left\|\mathbf{z}[k]\right\|_2\right) \tag{2.90}$$

A second criterion is meant to assess the proximity of $\psi[k]$ to the expected minimal value, in other words,

$$\Omega[k-1]-\Omega[k]<\beta[k] \tag{2.91}$$

where $\beta=\tau\left(1+\left|\Omega[k]\right|\right)$ and τ is a measure of the desired number of correct digits in the objective function.

The final termination criterion tests the gradient convergence to zero, i.e.,

$$\left\|\mathbf{g}[k]\right\|_2<\sqrt[3]{\tau}\left(1+\left|\Omega[k]\right|\right) \tag{2.92}$$

When criterion (2.83) is employed, Ω and **z** should be replaced by Ψ and **v**, in the above equations.

2.3.6.2 Model validation

A bad choice of τ in the previous termination criterion may drive the algorithm to overtraining. With model validation this can be avoided. This subsection describes some of the most common strategies in this context.

2.3.6.2.1 Training-validation-test data sets

The simplest approach is to divide the data set into two subsets. One of the subsets is used for training the model, thus noted by training data set. The remaining of the input data is used for validating the model. This approach is efficient and straightforward when the amount of input data is large. This becomes unfeasible if there are scarce input data and/or the data does not represent the dynamics of the process for all its working regimes. In these cases the model obtained will hardly perform well for any of the process working regimes.

Assuming a sufficient amount of data, another strategy can be employed which consists of dividing the input set into several subsets. Fig. 2.17 illustrates this strategy.

One subset is used for estimating *M* models. Another subset will be used to validate the performance of the model. This is the validation set. The best model is the one which gives lowest error value with the validation subset. A third subset is used to take a decisive choice on the best model. The test subset tries to avoid *overfitting*. In order to apply this strategy a significant amount of input data cannot be used for training. Thus, it is usual to use only two sets of data. The first, which corresponds to most input patterns, is used for training. The second which uses the remaining patterns, is used for testing the model.

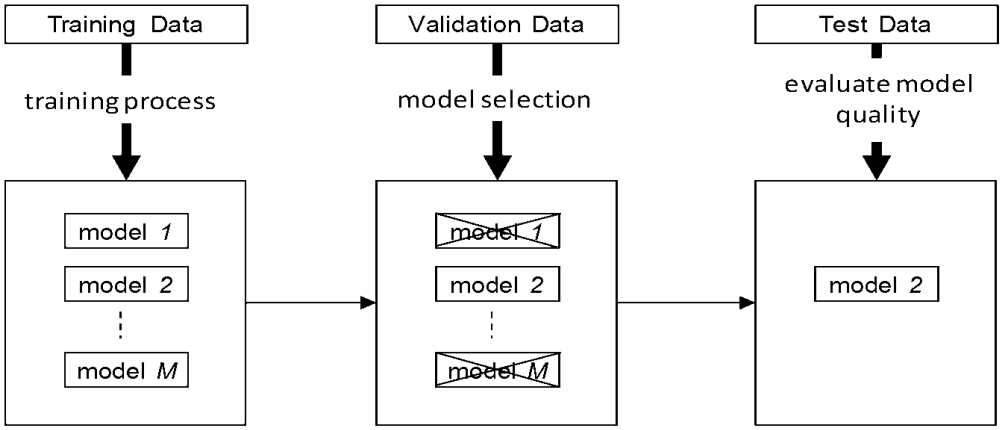


Fig. 2.17.Training, validation and testing of models

2.3.6.2.2 Cross validation

Almost frequently the data set is sparse and not viable to be divided into several subsets. However, if the user is willing to play a more calculation-intensive, an alternative procedure consists in dividing data into *S* equal parts where *S-1* of these parts are used for training and the remaining is used for testing [90]. This process is repeated for all combinations between parts *S*, allowing to exploit a higher fraction of the original data by keeping a small set of test data. But, as training and testing is carried out with the *S* combinations of data, it is possible to evaluate an average error for the test data, allowing the measurement of the performance of the model on new data. In the end, the model can also be trained with the original data set.

2.3.6.2.3 Early stopping

This approach is employed if the training set is divided in two data sets: the training and test data set.

As training progresses, the model tends to approximate the data better. But at some time it starts memorizing the peculiarities of the *training* data, eventually becoming *overtrained* and losing the capability of generalizing to new data samples. By using early stopping, the performance of the model is evaluated on the test data and training stops at a point where the performance in the test set deteriorates.

2.3.6.3 Statistical tests

One of the strengths associated with this approach is the absence of a test data set. The basic strategy of statistical tests is to assess model performance from the information given by the prediction error (or residuals).

So, in order to reduce the level of residuals, statistical tests like mean and variance over the residuals can be used to check if residuals are low to a certain acceptable level.

For the class of linear models validation, tests based on Auto Correlation Function and Cross Correlation Function can be found in [91]. Alternatively, for nonlinear model validation higher order correlation-test-based approaches must be applied to detecting nonlinear correlation between inputs and outputs [92][93][94][95].

Through validity tests [96][97] it is also possible to check the quality of identified neural networks.

2.4 Relations between artificial neural networks and fuzzy systems

While performance of fuzzy techniques is considered to be at least as good as classical techniques, especially in low dimensional systems, the same is arguably true for high dimensional and often more complex systems. By combining fuzzy techniques with other new methodologies which try to model other biological processes such as artificial neural networks, their behavior becomes more interesting. Under these conditions, fuzzy systems are not solely designed from expert knowledge but are partially learned from data, and so they are said to be neuro-fuzzy models. This way, the fuzzy model structure is depicted in a neural network structure. One typical structure is seen in Fig. 2.18. There are four layers. The neurons in the Layer I represent the input linguistic variables. The degree of membership is defined by the nodes in layer II. Each node in Layer III is a fuzzy rule. The output of the system is defined at Layer IV. This representation of a fuzzy system elicits the basics of fuzzy logic as it nears the neural network approaches.

Hence, the same learning techniques established for the neural networks can be applied in the neuro-fuzzy model. Furthermore, there exists a direct relationship between the fuzzy rule base and a neural network structure because of the way fuzzy membership functions and the fuzzy operators are defined. However, only under certain conditions the neuro-fuzzy systems are equivalent to some artificial neural networks (ANN).

An *n*-dimensional neuro-fuzzy system consists of a rule base where the j^{th} rule is described as:

$$R_j: \textbf{If } x_1 \text{ is } A_{j,1} \textbf{ and } \ldots \textbf{ and } x_n \text{ is } A_{j,n} \textbf{ then } y \text{ is } B_j \ (c_{j,i}) \tag{2.93}$$

In (2.93) each rule has a confidence c_j which lies in interval [0,1] and represents the confidence of a particular rule being true.

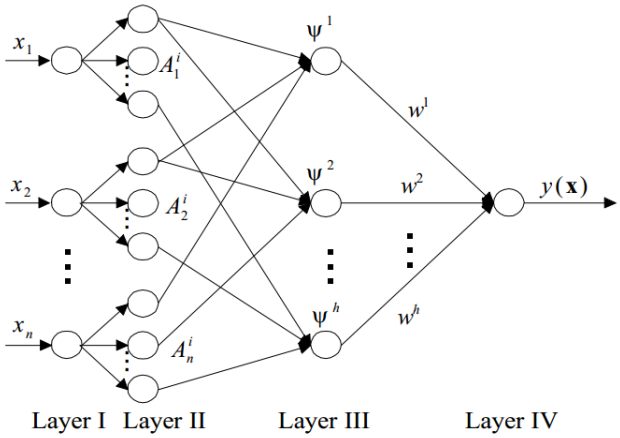


Fig. 2.18. Configuration of a neuro-fuzzy system (after [18])

The following subsections describe the equivalences between neuro-fuzzy and the two ANN architectures investigated during the course of this thesis.

2.4.1.1 B-Spline neural networks

Because the structure of a B-spline neural network (BSNN) is grid-based they begin by benefiting from the same interpretability feature as the fuzzy systems (FS) which is seen as an advantage over the conventional neural networks. So, it is not surprising that, at a high level, the basic information principles of BSNN and FS are the same. Therefore, under certain conditions, the low-level algorithms are also identical.

In the simplest form, fuzzy systems calculate their response by taking a linear combination of the input membership functions in an analogous way as several neural networks. The relationships between fuzzy and *B-spline* networks, have been addressed by several authors (see, for example [62][84]). In order to have a strict equivalence between FS and BNN models, the FS must satisfy some assumptions, which will be reproduced here [84].

Consider, for example, an *n*-dimensional BSNN model.

If there are N_1 cells in x_1, N_2 cells in x_2, …, N_n cells in x_n, the model has $N_1 \times N_2 \times ... \times N_n$ cells. This corresponds, in terms of fuzzy systems, to a system with $N_1 \times N_2 \times ... \times N_n$ rules, each one represented as:

If x_1 is $A_{i,1}$ **and** x_2 is $A_{j,2}$ **and … and** x_n is $A_{z,n}$ **then** y is B (2.94)

where $i = 1...N_1$, $j = 1...N_2$, $z = 1...N_z$, the number of linguistic terms in each input variable, respectively.

By introducing the relationships between B-spline neural networks and certain types of fuzzy models, training algorithms developed initially for neural networks can be adapted to fuzzy systems.

The following subsections present the mathematical formulation required for structure equivalences between the fuzzy inference engine and the BSNN evaluation procedure. This will be done for the *Takagi-Sugeno* type and *Mamdani-type* fuzzy systems.

2.4.1.1.1 Takagi-Sugeno type fuzzy systems

The Takagi-Sugeno type of fuzzy system is a model where the consequents of the rules are real-valued functions (2.47).

For the sake of simplicity consider the Single-Input-Single-Output TS model where,

$$f_i(x) = a_0^{(i)} + a_1^{(i)} x \tag{2.95}$$

Then, for a crisp input value *x'*,

$$y' = \frac{\sum_{i=1}^{R} \mu_{A_i}(x) f_i}{\sum_{i=1}^{R} \mu_{A_i}(x)}. \tag{2.96}$$

If the linguistic terms of the rule antecedents A_i are modeled by B-Splines, $\sum_{i=1}^{R} \mu_{A_i}(x) = 1$ which allows one to rewrite (2.96) as a linear combination of the premise membership functions:

$$y = \boldsymbol{\varphi}^T \mathbf{u}, \tag{2.97}$$

where

$$\boldsymbol{\varphi}^T = \left[\mu_{A_1}(x) \;\; \mu_{A_2}(x) ... \mu_{A_R}(x) \;\; \mu_{A_1}(x)x \;\; \mu_{A_2}(x)x ... \mu_{A_R}(x)x \right], \tag{2.98}$$

and

$$\mathbf{u}^T = \left[a_0^{(1)} \;\; a_0^{(2)} ... a_0^{(R)} \;\; a_1^{(1)} \;\; a_1^{(2)} ... a_1^{(R)} \right]. \tag{2.99}$$

Extension to the multi-input-single-output TS model is straightforward.

$$\boldsymbol{\varphi}^T = \left[\mu_A(\mathbf{x}) \;\; \mu_A(\mathbf{x}) Ix_1 ... \mu_A(\mathbf{x}) Ix_{\prod_{p=1}^{n} N_p} \right], \tag{2.100}$$

and

$$\mathbf{U}^T = \left[\mathbf{u}_0 \;\; \mathbf{u}_1 \;\; \mathbf{u}_2 \;\; \mathbf{u}_n \right], \tag{2.101}$$

where,

$$\mu_A(\mathbf{x}) = \left[\mu_{A_1}(\mathbf{x}) \;\; \mu_{A_2}(\mathbf{x}) ... \mu_{A_{\prod_{p=1}^{n} N_p}}(\mathbf{x}) \right]$$

$$\mathbf{u}_j = \left[a_0^{(j)} \;\; a_1^{(j)} ... a_R^{(j)} \right] \tag{2.102}$$

In (2.102), R is the number of rules.

2.4.1.1.2 Mamdani-type fuzzy systems

With the Mamdani-type fuzzy system the analysis is slightly more complicated.

Nevertheless the equivalences are assumed if:

- B-Splines of the same order are used as membership functions. If a spline order of 2 is assumed then a typical triangular membership function is used for the fuzzy model.
- The B-Spline NN model consists of only one submodel, which can contain multiple input variables.
- The *t-norm* is the product logical operator.
- The membership function of the output fuzzy set B is computed by taking the sum over all rule's output fuzzy sets:

$$\mu_B = \sum_{i=1}^{R} \mu_{B_i}(y) \tag{2.103}$$

- Applying the center-of-gravity method of defuzzification, the crisp output value y' for the Single-Input-Single-Output model with m discretization points in the universe of discourse of B is calculated as

$$y' = \frac{\sum_{k=1}^{m} y_k \left(\sum_{i=1}^{R} \mu_{A_i}(x) \mu_{B_i}(y_k) \right)}{\sum_{k=1}^{m} \sum_{i=1}^{R} \mu_{A_i}(x) \mu_{B_i}(y_k)}. \tag{2.104}$$

Because it is desirable to write equation (2.104) in the form of a linear regression model, the formulation above can be extended to the general case where more than one rule exists for each linguistic variable. Thus, if a confidence value $c_{i,j} \in [0,1] (i=1 \ldots h; j=1 \ldots m_c)$ is attached to each rule:

$$y' = \frac{\sum_{k=1}^{m} y_k \left(\sum_{i=1}^{h} \sum_{j=1}^{m_c} \mu_{A_i}(x) \mu_{B_i}(y_k) c_{i,j} \right)}{\sum_{k=1}^{m} \left(\sum_{i=1}^{h} \sum_{j=1}^{m_c} \mu_{A_i}(x) \mu_{B_i}(y_k) c_{i,j} \right)}. \tag{2.105}$$

In (2.105) h and m_c are the number of linguistic terms in the antecedent and consequent parts of the rule.

Now, considering that after some algebraic manipulations, the denominator in (2.105) is written as

$$\begin{aligned} den &= \sum_{k=1}^{m}\left(\sum_{i=1}^{h}\sum_{j=1}^{m_c}\mu_{A_i}(x)\mu_{B_i}(y_k)c_{i,j}\right) \\ &= c_{1,1}\mu_{A_1}(x)\sum_{k=1}^{m}\mu_{B_1}(y_k)+\ldots+c_{h,m}\mu_{A_k}(x)\sum_{k=1}^{m}\mu_{B_{m_c}}(y_k) \end{aligned} \quad ; \qquad (2.106)$$

- $\sum_{k=1}^{m}\mu_{B_1}(y_k)=\sum_{k=1}^{m}\mu_{B_2}(y_k)=\ldots=\sum_{k=1}^{m}\mu_{B_{m_c}}(y_k)=\alpha$, α a constant;

then, the equation in (2.106) is written as

$$\begin{aligned} den &= c_{1,1}\mu_{A_1}(x)\alpha+c_{1,2}\mu_{A_1}(x)\alpha+\ldots+c_{1,m_c}\mu_{A_1}(x)\alpha+\ldots \\ &+\ldots+c_{h,1}\mu_{A_n}(x)\alpha+c_{h,2}\mu_{A_h}(x)\alpha+c_{h,m_c}\mu_{A_h}(x)\alpha= \\ &\mu_{A_1}(x)\alpha\left(c_{1,1}+c_{1,2}+\ldots+c_{1,m_c}\right)+\mu_{A_2}(x)\alpha\left(c_{2,1}+c_{2,2}+\ldots+c_{2,m_c}\right)+\ldots \\ &+\mu_{A_h}(x)\alpha\left(c_{h,1}+c_{h,2}+\ldots+c_{h,m_c}\right) \end{aligned} \qquad (2.107)$$

Moreover, assuming that $\sum_{j=1}^{m}c_{i,j}=1$ then the denominator will be given solely by a constant:

$$den=\alpha\sum_{i=1}^{h}\mu_{A_i}(x)=\alpha \qquad (2.108)$$

To obtain the last equation two other assumptions were established:

- The membership functions are B-Splines, because only so $\sum_{i=1}^{h}\mu_{A_i}(x)=1$ which is another property from B-Splines.
- All the linguistic terms in the output have the same shape and width allowing to have $\sum_{k=1}^{m}\mu_{B_j}(y_k)=\alpha,\ j=1\ldots m_c$.

If the denominator in (2.106) is a constant it can be incorporated as a scaling factor at the output of the model. Thus,

$$\boldsymbol{\varphi}^T=\left[c_{1,1}\mu_{A_1}(x)\sum_{k=1}^{N}y_k\mu_{B_1}(y_k)\ \ldots\ c_{i,j}\mu_{A_i}(x)\sum_{k=1}^{m}y_k\mu_{B_j}(y_k)\ \ldots\ c_{h,m_c}\mu_{A_h}(x)\sum_{k=1}^{m}y_k\mu_{B_{m_c}}(y_k)\right]/\alpha\,, \qquad (2.109)$$

and

$$\mathbf{u}^T = \left[c_{1,1} \ldots c_{i,j} \ldots c_{h,m_c} \right]. \tag{2.110}$$

By denoting $\sum_{k=1}^{m} y_k \mu_{B_j}(y_k) / \alpha$ as y_j^c:

$$\boldsymbol{\varphi}^T = \left[c_{1,1} \mu_{A_1}(x) y_1^c \ \ldots \ c_{i,j} \mu_{A_i}(x) y_j^c \ \ldots \ c_{h,m_c} \mu_{A_h}(x) y_{m_c}^c \right], \tag{2.111}$$

and

$$\mathbf{u}^T = \left[\gamma_1 \ldots \gamma_i \ldots \gamma_h \right], \tag{2.112}$$

where

$$\gamma_i = \sum_{j=1}^{m_c} c_{i,j} y_j^c \ . \tag{2.113}$$

The extension to the Multi-Input-Single-Output case is straightforward.

Assuming n inputs and n_p linguistic terms for the p^{th} input, equations (2.111) and (2.112) become

$$\boldsymbol{\varphi}^T = \left[\mu_{A_1}(\mathbf{x}) \ \ldots \ \mu_{A_i}(\mathbf{x}) \ \ldots \ \mu_{A_{\prod_{p=1}^{n} n_p}}(\mathbf{x}) \right], \tag{2.114}$$

and

$$\mathbf{u}^T = \left[\gamma_1 \ldots \gamma_i \ldots \gamma_{\prod_{p=1}^{n} N_p} \right]. \tag{2.115}$$

In equation (2.114), $\mu_{A_i}(\mathbf{x}) = \prod_{p=1}^{n} \mu_{A_{i,p}}(x_p)$.

2.4.1.2 Radial basis function networks

This section lists how under some assumptions the same advances and new developments of Fuzzy Systems can be applied to Radial Basis Function (RBF) networks and vice-versa. In other words, one can apply the learning rules of RBF to fuzzy inference systems, and the learning rules of fuzzy inference systems can also be utilized to find the structure and parameters of RBFs.

A TS fuzzy *zero-th* order type and a Normalized RBF are identical under the following assumptions [98]:

- Gaussian membership functions are used and the *t-norm* is the product. Because of the Gaussian function properties, one multivariate Gaussian membership function is

obtained if the product is applied to several univariate Gaussian membership functions:

$$\varphi_i(\mathbf{x},\mathbf{c},\upsilon) = e^{\frac{-(\mathbf{x}-\mathbf{c}_i)^2}{2\upsilon_i}} = \prod_{j=1}^{n} e^{\frac{-(x_j-c_{i,j})^2}{2\upsilon_i}} = e^{\sum_{j=1}^{n}\frac{-(x_j-c_{i,j})^2}{2\upsilon_i}} \tag{2.116}$$

- The number of neurons in the RBF and the number of rules in the TS system must be the same.
- The RBF and the neuro-fuzzy system must use the same method to derive the overall outputs (use either weighted average of weighted sum of the fuzzy rules consequents).

To ensure an acceptable degree of interpretability two other assumptions must be regarded for [62]:

- The RBFs have to be placed on a grid. If, for instance three RBFs do not lie on a grid, then three membership functions for each input in the neuro-fuzzy system are required.
- RBFs have to be axis-orthogonal at the expense of not being able to reproduce exactly the original multidimensional membership function.

2.5 Structure optimization

All techniques presented in section 2.2.2.7 assume a fixed structure and start from an initial point in the parameters space and search in directions obtained from neighborhood information such as first and second derivatives. This approach leads to an optimal solution which is close to the initial starting point and in general is not the global one. With structure identification the aim is to devise a parsimonious model which features low complexity and a very acceptable generalization capability, all of this based on the given data only. Though model performance improves with the increase of the model complexity, this happens up to a certain limit, after which performance will deteriorate. This is seen as a trade-off between the quality of approximation of the training data set and the generalization capability for input data not seen by the model during training. Therefore, model quality is closely related to finding a compromise between model complexity and accuracy over the training data. This is the main goal of a structure optimization method.

When only training data is available, statistical significance measure in the form of information criteria can be used to assess a compromise value. The most used are:

- Bayesian Information Criterion (BIC):

$$BIC = m\ln\left(V_m(\mathbf{z})\right) + n_z \ln\left(m\right) \tag{2.117}$$

- Akaike Information Criterion (AIC):

$$AIC = m\ln\left(V_m(\mathbf{z})\right) + 2n_z \tag{2.118}$$

- Final Prediction Error (FPE):

$$FPE = m\ln\left(V_m(\mathbf{z})\right) + m\ln\left(\left(m+n_z\right)/\left(m-n_z\right)\right) \tag{2.119}$$

In the above equations, m is the number of training patterns, n_z is the number of parameters of the model and $V_m(\mathbf{z})$ denotes a measure of the modeling error which can be any of (others may be used):

- SSE (**S**um of **S**quared **E**rrors),

$$SSE = \sum_{i=1}^{m}(t_i - y_i)^2 \tag{2.120}$$

- MSE (**M**ean **S**quare of the absolute **E**rror),

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(t_i - y_i)^2 \tag{2.121}$$

- MSRE (**M**ean **S**quare of the **R**elative **E**rror)

$$MSRE = \frac{1}{m}\sum_{i=1}^{m}\frac{(t_i - y_i)^2}{y_i^2} \tag{2.122}$$

- PMRE (**P**ercentage of **M**ean **R**elative **E**rror).

$$PMRE = \frac{100}{m}\sum_{i=1}^{m}\left|\frac{t_i - y_i}{y_i}\right|. \tag{2.123}$$

- RMSE (**R**oot **M**ean **S**quare of the absolute **E**rror),

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(t_i - y_i)^2} \tag{2.124}$$

With global optimization, the algorithms search for a global optimum but this can become a highly time consuming task. Rather, one expects the global techniques not to find the global optimum but to find a good local optimum. This is especially desirable for high-dimensional problems.

As these methods examine the whole parameter space it is good practice to use the estimated parameters from the global method as initial values for a subsequent local optimization. Rather, the most efficient approach is to combine different optimization methods putting together the specific advantages of either one.

There are many algorithms which employ strategies of global search, including:

- Learning Classifier Systems [99], which are online learning strategies that assign output values to given input values. This mapping is obtained through rules which are developed by the use of a genetic algorithm.
- Particle swarm optimization [100], a strategy based on the simulation of the social behavior of a bird flock. Solutions are called particles and as changes to a particle are influenced by experience and knowledge of its neighbors, it is regarded as a kind of cooperative competitive coevolution.
- Ant colony optimization [101][102], which tries to model the behavior of ants to solve real-world problems.
- Artificial immune systems (AIS) [103][104], which simulate the human immune system. AIS is regarded as a complex network structure protecting against countless different sickness. In [105], Ülker and Arslan used AIS to automatically determine the number and location of interior knots for B-spline functions. Coelho and Pessoa [106], proposed an AIS based algorithm to tune the knots for a BSNN (called aiNet) using a discrete immune network algorithm based on the concepts of AIS.

The next subsection covers evolutionary algorithms, which model the processes of natural selection evolution. They are described in some detail, as they will be used in this work.

2.5.1 Evolutionary algorithms

Briefly, the evolutionary search process of an evolutionary algorithm is influenced by the following components:

- The encoding of the chromosome
- The evaluation of the fitness
- Initialization of the initial population
- Selection operators
- Reproduction operators

The different ways of employing these components lead to several trends in the evolutionary computation context:

- genetic algorithms, which include all algorithms that use binary strings to represent the search space. They will be addressed in subsection 2.5.1.1.

- evolution strategies [16][107], which explore the space of real vectors. Genetic variation is mostly due to the mutation operator. The individual is represented by a chromosome which codes not only the optimization parameters but also the strategy parameters (parameters that have control over the evolution process). The optimization of the strategy parameters is called self adaptation.
- Genetic programming. It includes all algorithms which can evolve programs and all algorithms that evolve tree-shaped individuals. Subsection 2.5.1.2 addresses this topic.
- Gene Expression Programming, consisting on the combination of the principles of genetic algorithms and genetic programming, is described in subsection 2.5.1.3.
- Evolutionary programming [108], which is an approach that regards the instances of the genome as different species, rather than as individuals.
- Differential evolution [109][110], is also a population-based search strategy where the mutation is replaced by a new arithmetic operator which depends on the differences between selected pairs of individuals, rather than being dependent on some probabilistic distribution.
- Coevolution [111], in which different populations evolve in parallel in a competitive way. Evolution is not just locally within each population but also as a result to a changing physical environment where changes are caused by each population.
- And, bacterial evolutionary algorithms which mimic the biological phenomenon of microbial evolution and is addressed in subsection 2.5.1.4.

2.5.1.1 Genetic Algorithms

With Genetic Algorithms (GA) the variables of interest must be first encoded in a binary representation forming a chromosome, a sequence of bits. Traditionally, the strong preference for using binary representation is derived from schema theory [13]; the fundamental reason lies on the maximum number of schemata (building blocks) for a finite number of search points. The most straightforward and at the same time most common approach involves binary strings of fixed length. This type of representation is best suited for problem domains where solutions can be naturally represented as binary vectors, e.g. in some combinatorial optimization problems. Evolution starts when a set of chromosomes is used to define the individuals of the population each one with its distinct genetical features. Then, three genetic operators are used to generate a new population. These are reproduction, crossover and

mutation. These actions are repeated until a predefined number of generations is reached or the required accuracy is attained. The best chromosome in the final population expresses a solution.

Mutation works by inverting bits. Originally, the of rate mutation was kept small, but progressively it was demonstrated that much larger mutation rates, decreasing over the course of evolution were helpful with respect to the convergence reliability and velocity of a genetic algorithm [112]. First introduced by Holland [13], in 1975, it was the first EA paradigm to be developed and applied. It has been popularized by the work of Goldberg [113].

In the original approach by Holland the GA had three distinct features: a bit representation, a proportional selection mechanism and new individual's creation through crossover.

Changes to the original GA have been introduced by the use of different representation schemes, and genetic operators. In contrary to evolution strategies, the GA uses a binary coding to represent all types of parameters (real, integer, binary).

Generally, genetic algorithms are good choices when problems involve discontinuous, nondifferentiable and multimodal objective functions.

The cycle of evolution of a generic GA is summarized next:

Algorithm 2.3. Genetic Algorithm

1. Initial population creation
2. Repeat while termination criterion is not reached
    a. Evaluate the fitness of individuals in the population
    b. Select parents from population, according to fitness.
    c. Form offspring by recombining selected parents, using an appropriate selection operator
    d. Mutate offspring
    e. Select the new generation population from the previous generation and recently created offspring

In the following a detailed description of the basic components of the GA is given.

2.5.1.1.1 Chromosome representation

The motivation behind the binary coding of genetic algorithms (GAs) derives from the form of information coding in nature. Genetic information is stored in a code of 4 symbols, "A", "G", "C", "T" forming the basis of the DNA molecule (which includes the genetic instructions for an organism). Therefore, by using the binary symbols "0" and "1" is a simplified version of the natural genetic code. While it may seem to be necessary to build very long codes of individuals, in addition to being less rational when the original parameters are real, binary coding is more rational when applied directly to structure optimization problems in which each bit controls whether the determined template is active or not (case of neurons, rule fuzzy system, or term of a polynomial).

The classical encoding scheme for the GAs is binary vectors of fixed length. The following figure illustrates the chromosome of an individual where the length of each parameter Θ_i to optimize is fixed and assumed equal.

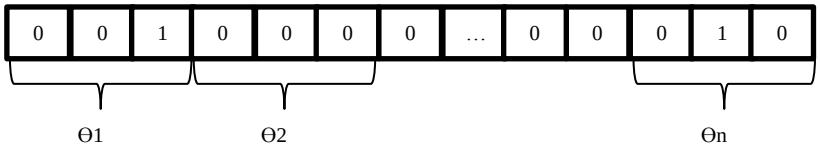


Fig. 2.19. Example of chromosome representation in the GA.

2.5.1.1.2 Mutation

The main purpose of mutation is to prevent the algorithm from getting stuck in some regions of the search space. Therefore, mutation introduces diversity to the genetic characteristics of the population by adding new genetic material into an existing individual.

This operation is characterized by the inversion of one bit in an individual, using the rates of change of each bit of the order of 0.001 to 0.01, reason why the mutation does not play a key role in GAs.

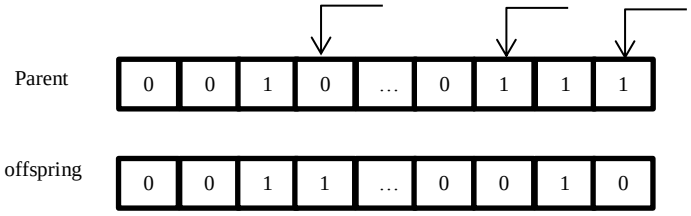


Fig. 2.20. Mutation in a GA inverts randomly a set of bits in the individual

There is obviously one weakness related to one point mutation: while the same rate is applied to all bits, these may not have the same meaning. For example modify "0000" to "1000" results in changing the value to modify 8 while "0001" to "0000" results in a decrease of only -1. That is, the same probability can result in both cases minor modifications but

enormous changes. To correct, one can use a specific mutation rate at each bit according to its significance or use Gray coding [114].

Alternatives can be employed where the mutation rates start with a large value and exponentially decrease as a function of the number of generations. This way, a large search space is covered initially and as the individuals start to converge to the optimum, the mutation rate decreases rapidly improving convergence speed and accuracy.

2.5.1.1.3 Crossover

The aim of crossover is to reproduce offspring from two parents. Crossover happens under a certain probability, reason by which not all groups of parents produce offspring.

It is the main operator in GAs. The bit strings are divided into two parts and these parts are crossed between relatives. Parents result in two descendants so that each offspring contains a descending part from one parent and a descending part from the other parent. The crossover rate is usually of the order of 0.6 to 0.8.

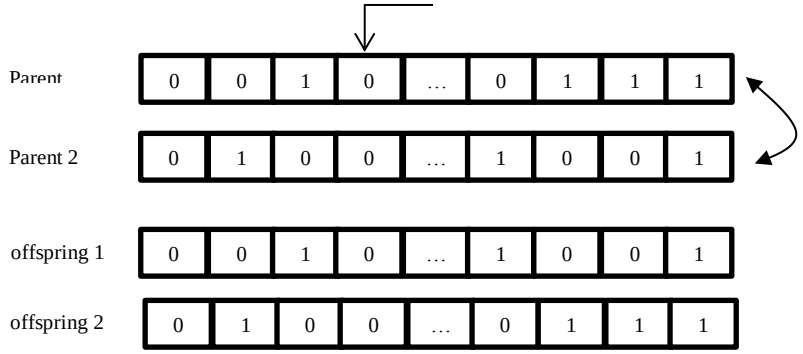


Fig. 2.21. One-point crossover operation between two parents. Parents are cut at one point and information of one side is exchanged.

To specify which bits of the parents should be exchanged, several crossover operators have been developed. The three mostly used are:

- Uniform crossover: a mask of the bits to be exchanged is created at random for each pair of individuals selected for reproduction.
- One-point crossover: one bit position is randomly selected and the subsequent bit string after that point is exchanged between the two parent's chromosomes (see Fig. 2.21).
- Two-point crossover: rather than using one bit position as one-point crossover, two bit positions are selected randomly. Then, the bit string between these points is exchanged.

2.5.1.1.4 Fitness

By *fitness* one means the greater or lesser capacity of the individual to solve the problem at hand. This factor is relevant in the direction taken by the evolution of the population, classifying individuals and determining the most likely to remain in the population or to participate in the process of population renewal. Types of *fitness* include *raw fitness, standardized fitness, adjusted fitness, normalized fitness* and *ranking fitness*.

2.5.1.1.5 Examples of applications

Very successful fuzzy system identification methodologies within the scope of soft computing are genetic fuzzy systems (GFSs) [115][116]. Results have demonstrated that genetic algorithms learn the components of a FS. A GFS is basically a fuzzy system designed through a learning process based on a genetic or an evolutionary algorithm (GA/EA). With the GA the differentiability constraints do not need to be imposed over the membership functions in the FS, making them less restrictive.

Leu [18] used a GA to identify a neuro-fuzzy model where B-Splines are used as the fuzzy membership functions. Each individual in the population is represented by a chromosome coded as an adjustable vector where each component is of floating type. The estimating parameters include the linear parameters or weights and the interior knots positions.

FUREGA [19] combines genetic algorithms and least squares optimization for selecting the fuzzy rule base of a zero-th order Takagi-Sugeno type fuzzy system employing a hybrid methodology. The GA uses a binary coding of the chromosomes, where each gene represents one rule. The combination of all possible rules is coded in a binary string and the GA is used to find the best solutions. FUREGA also integrated elimination of redundant fuzzy *linguistic terms*, based on three strategies and by introducing a penalty in the respective objective function.

The work presented by Oh and Pedrycz in [117] is a follow up of [21], which introduced the Self-Organizing Neural Network (SONN) as an alternative adaptive structure of a neuro-fuzzy system. It uses an improved genetic designed approach, denoted as genetically optimized SONN (gSONN). In gSONN a genetic algorithm with binary representation is used to optimize the structure of SONN. A hybrid method is implemented which combines the GA with a structural phase of GMDH and a least squares estimation technique.

2.5.1.2 Genetic Programming

Growing interest of this technique must be credited to Koza [14] with the publication of his book on Genetic Programming (GP) where he proposed alternative methodologies for automatically develop computer programs. Since then, there has been an intensive dedication to application of GP in many types of applications. Examples for nonlinear systems identification are [118][119].

Similar to genetic algorithms (GAs), in genetic programming (GP) trees are the structures that undergo constant adaptation to dynamic changes in terms of their size and shape. The main objective is to deal with one of the most pertinent issues of computer science: How can computers learn to solve problems without being explicitly programmed? In genetic programming, the search space consists of all computer programs composed of functions and terminals appropriate to the problem domain. So, the logical expressions and mathematical equations have to be realized by trees. The leaves consist of the variables and constants while the other internal nodes implement operators. It uses a modified version of the GAs recombination operators which allow trees to be randomly changed.

While applying PG to any problem, five preparatory steps are necessary to identify:

1. The set of terminals (input values). 2. The set of primitive functions. 3. The measure of fitness. 4. The control parameters of the execution. 5. The method of assigning a result and termination criteria.

Before initiating the algorithm, the user needs to specify some primary control parameters of evolution, such as population size and maximum number of generations. On the other hand, the secondary parameters are specified in terms of variables used to control the execution. It is necessary to specify a termination criterion to decide when to end the run of genetic programming. Generally, a maximum number of generations for the evolution is used.

The initial population consists of a set of random structures composed of functions and terminals appropriate to the problem, representing a "blind search" in the search space. In crossover, the classification of the structures is done by an operator which in most cases is given by a measure of the sum of the absolute error between the output produced by the program and the correct answer to the problem. In the initial population many programs have a poor *fitness* value, however some will prove to be the best and these differences are exploited.

This principle is performed by *reproduction* and survival of the fittest of the population, and the creation of offspring through genetic *crossover* operation. These new structures will

then be inserted in the population. These structures are selected according to their *fitness*, so the best will have a better chance of staying in future generations. After applying the genetic operators in GP for the current population, the new population will take over, and this process is repeated for as many generations as initially defined.

The outline of GP is given next.

Algorithm 2.4. Genetic Programming

1. Initial population creation
2. Crossover
    a. Selection of individuals
    b. Gene Crossover
    c. Copy offspring to population
3. Reproduction
    a. Selection of individuals
    b. Copy to population
4. Mutation
    a. Function mutation or Terminal mutation
5. If termination criterion is reached, terminate otherwise go back to 2

2.5.1.2.1 Initial population

Every individual in the population is represented by an expression tree. Example of an expression tree is shown in the next figure. As every individual represents a computer program, regarding the ET in Fig. 2.22, one would read as A*B+C.

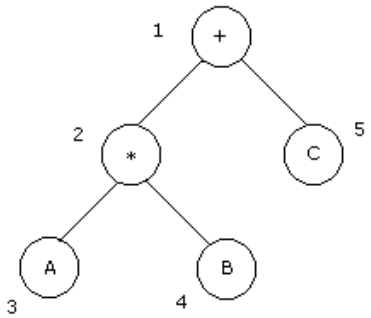


Fig. 2.22. Example of an expression tree

The process of generating these structures can take various forms resulting in initial trees of different sizes and shapes. Two of the most commonly used method is the “full” and the

“grow” methods. The “full” method generates an initial population where, for all individuals, the length between the root and the terminal point is the same and equal to the specified maximum length. Alternatively, the “grow” involves the creation of trees whose length and shape are variable. The length between the point and the outer root point, though, cannot be greater than the maximum length specified. Another method, which combines a mixture of both, is called "ramped half-and-half." This method is produces a variety of trees of various sizes and shapes. It consists of the creation of a number of trees with length between 2 and the maximum value specified. That is, assuming a maximum length of 6, then 20% of the trees would have length 2, 20% would length 3 and so on. For each length value, 50% of the trees would be established by the method "grow" and the remaining 50% by the method “full”.

2.5.1.2.2 Genetic Operators

2.5.1.2.2.1 Reproduction

Reproduction ensures that the fittest individuals will survive and be incorporated in the new population. The individual remains unchanged.

2.5.1.2.2.2 Crossover

This operation involves a pair of individuals which will be used to produce a pair of offspring consisting of parts of both parents. The selection of parents is based on fitness.

The next figure illustrates the crossover operation.

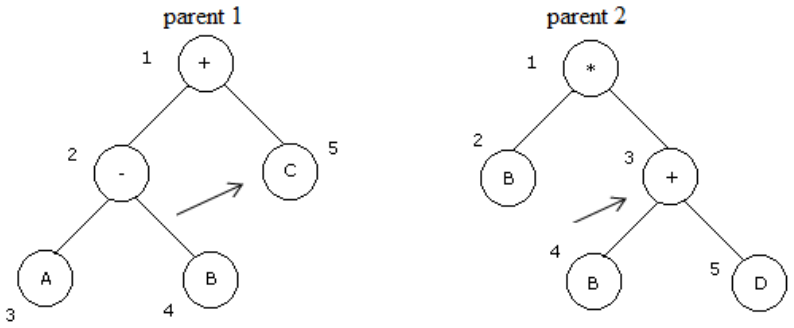


Fig. 2.23. Parts selected in the parents to participate in the crossover operation

In each parent a node in the ET is randomly selected and exchanged between parents.

In the case, part 5 of parent 1 is exchanged with part 3 of parent 2, resulting in the offsprings below.

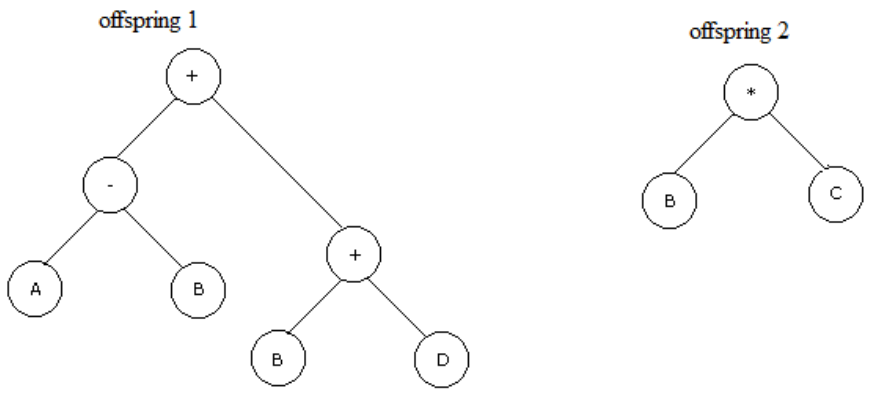


Fig. 2.24. The resulting ET structures after crossover

After crossover the computer programs read as follows.

The parents: (A-B) + C and B (B + D). The offspring: D - A and CB.

Though not so common, crossover operation can generate only one offspring in each mate operation. In this case, only one of the parents has its node replaced with the subtree from the other parent.

2.5.1.2.2.3 Mutation

Mutation is an operation only applied to an individual.

Several mutation operators have been developed: function node mutation, terminal node mutation, swap mutation, grow mutation and Gaussian mutation:

- Function node mutation refers to the case when a randomly selected non-terminal node is replaced with a node with the same number of arguments from the function set.
- Terminal node mutation illustrates the case when a randomly selected terminal node is replaced with a node with another terminal node, selected from the terminal set.
- Swap mutation is the simplest case in which a randomly chosen function node has its arguments swapped.
- Grow mutation. The example shown in the figure below illustrates the grow mutation type. A point in the tree is selected for mutation and replaced by another subtree which consists of random information. It can either be at an interior point or at a point outside (terminal) of the tree. If an interior point is selected then the maximum size of the new subtree to insert must be specified. Generally, the value

of this parameter is equal to the parameter used in the creation of programs of initial population.

- Gaussian mutation, when a terminal node representing a constant is randomly selected and mutated by adding a Gaussian random value to that constant.
- Trunk mutation, if a function node is replaced by a random terminal node. This way, a pruning is performed on the tree.

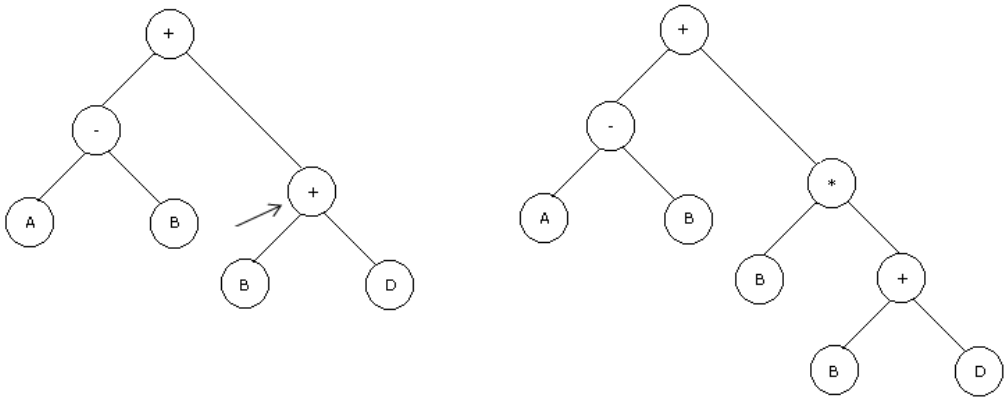


Fig. 2.25. Mutation operation. On the left the original ET structure; on the right, the resulting ET structure after mutation

2.5.1.2.3 Genetic programming for B-spline neural networks

One way of determining a BSNN model is through genetic programming (GP). This meta-heuristic was introduced in [59]. With this approach the design strategy is based on adding submodels from evolving a tree structure. Through an adaptation of the genetic operators, higher dimensionality submodels are *created* from smaller sub-modules (*), and submodels of higher dimensionality are *split* into lower dimensional submodels (/).These are the set of primitive functions that were implemented. Extra information was added to the node terminals. A note terminal consists of the *input variable identification*, the *spline order*, the *number of interior knots* and *their location*.

Instead of coding the network parameters in bit strings, it uses a tree structure, composed of *function* and *terminal* nodes. One individual in the population is represented by one such expression tree (see Fig. 2.26). This tree structure, as well as the characteristics of the nodes, evolves from generation to generation.

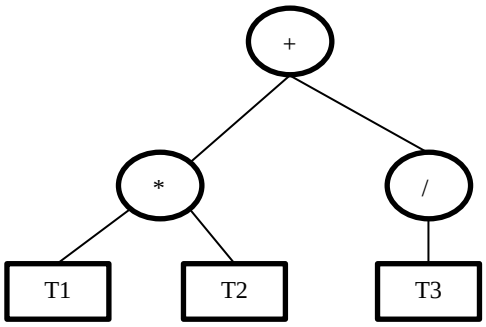


Fig. 2.26. Example of an expression tree in genetic programming for B-Splines

This approach introduced a new way of representing a BSNN model through computer programs. Furthermore, using GP's recombination operators allowed evolving the computer programs (BSNN models) to finding an adequate structure and estimating the interior parameters simultaneously.

2.5.1.2.3.1 Terminal nodes

A terminal consists of:

- Identification of the variable: a parameter defined by the number of the input variables. Assume the set of *n* input variables $\mathbf{x} = \{x_1, x_2, ..., x_n\}$: the terminal node will refer to one input, x_i for example.
- Order of splines associated with the variable: a parameter chosen from the set {1,2,3}. The splines order is limited to 3. The terminal node will specify the order in *k*.
- Number of interior nodes: a parameter whose maximum value is specified by the user. The number of knots in a terminal is defined randomly but limited to *nk*.
- Vector of knots which is created after setting the above parameters. The position of interior nodes is random. The exterior knots are evenly distributed at an interval defined by the specified spline order. $\boldsymbol{\lambda}$ is the vector of knots.

Thus, the i^{th} terminal node will consist of $T = \{x_i, k_i, n\lambda_i, \boldsymbol{\lambda}_i\}$. The following figure illustrates two end nodes. The terminal node on the right represents the B-Spline basic description for input variable x_2: the input variable identification, the order of the splines in this dimension (k_2), the number of interior knots ($n\lambda_2$), and the knot vector ($\boldsymbol{\lambda}_2$).

2.5.1.2.3.2 Function nodes

For the case of the BSNN the function primitives are restricted to the set $F = \{+, *, /\}$.

Considering the specificity of the terminal information in a BSNN network the application of these primitive functions is not so simple. For this reason, the next subsections explain the procedures used when each one of the functions is applied when an individual is evaluated, or when it is undergoing mutation of crossover.

Addition (+)

The first primitive is the add function.

It can return a sum of two submodels, each one spanning two different input subspaces:

$$M_{out} = \sum_{u=1}^{sm_1} S_{k_u,\boldsymbol{\lambda}_u}^{\mathbf{x}_{\underline{u}}} + \sum_{g=1}^{sm_2} S_{k_g,\boldsymbol{\lambda}_g}^{\mathbf{x}_{\underline{g}}} \tag{2.125}$$

where $S_{k_i,\boldsymbol{\lambda}_i}^{\mathbf{x}_{\underline{i}}}$ is the i^{th} submodel with the set of input variables $x_{\underline{i}}$ of order k_i, and knot vector $\boldsymbol{\lambda}_i$.

The required adaptations are considered if any of the input variables in the two submodels overlap.

Tensor product (*)

The purpose of this operation is to augment the model complexity, producing submodels of two or more variables. To accomplish this, it applies the tensor product operator between the submodels.

There are special cases which need to be highlighted.

1. When the tensor product involves the same variable, the resulting submodel is merely a sum of the interior knots in that variable.
2. If the tensor product is applied to submodels with the same variables, the resulting submodel performs the sum between the submodels.

Summarizing, after applying the tensor product the resulting submodel is obtained as

$$M_{out} = \sum_{i=1}^{sm_1} \sum_{u=1}^{sm_2} S_{k_i,\boldsymbol{\lambda}_i}^{\mathbf{x}_{\underline{i}}} \otimes S_{k_u,\boldsymbol{\lambda}_u}^{\mathbf{x}_{\underline{u}}} \tag{2.126}$$

Submodel split (*)

This function returns a model composed of univariate submodels whenever possible. It returns the original model otherwise.

Assuming a model composed of two multivariate submodels, *sm1* and *sm2*:

$$M = \sum_{u=1}^{sm1} S_{k_u,\boldsymbol{\lambda}_u}^{\mathbf{x}_{\underline{u}}} + \sum_{i=1}^{sm2} S_{k_i,\boldsymbol{\lambda}_i}^{\mathbf{x}_{\underline{i}}} \tag{2.127}$$

The returned model is,

$$M_{out} = \sum_{u=1}^{sm3} \hat{S}_{k_u,\boldsymbol{\lambda}_u}^{\mathbf{x}_{\underline{u}}} + \sum_{m=1}^{sm4} S_{k_m,\boldsymbol{\lambda}_m}^{\mathbf{x}_{\underline{m}}} \tag{2.128}$$

where *sm3* represents the number of modified univariate submodels, and $sm4 = \sum_{i=1}^{sm2} \#\left(\mathbf{X}_{\underline{i}}\right)\Big|_{\#\left(\mathbf{X}_{\underline{i}}\right)>2}$, the number of modified multivariate submodels.

The following figure illustrates an expression tree defining a model of a B-spline network.

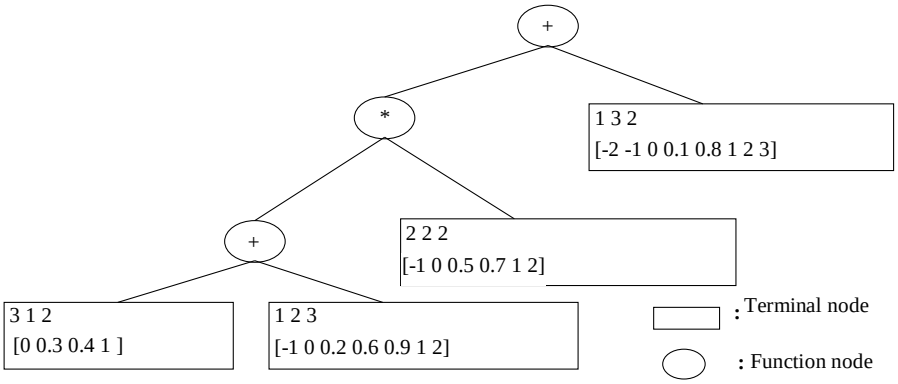


Fig. 2.27 A sample expression tree for representing a B-spline network in GP

For the tree given in Fig. 2.27, for instance the left-most leaf of the tree means that this terminal node contains the input variable 3, its spline order is 1, the number of its interior knots is 2, and they are located in 0.3 and 0.4 positions. The input space range is [0, 1].

The model's output is then given as a function of the addition of 3 submodels, two of each would be two-dimensional, so that:

$$y(x) = f_{2,3}\left(x_3 \times x_2\right) + f_{1,2}\left(x_1 \times x_2\right) + f_1(x_1) \tag{2.129}$$

where x_i denotes the input variable from the i^{th} dimension.

Estimating of the internal parameters was accomplished by defining mutation on a terminal of 6 different types: 1. Full replacement of the terminal; 2. Variable identification replacement; 3. Splines order replacement; 4. Random displacement of an interior knot; Addition of m_i interior knots placed randomly; Removal of m_i interior knots.

2.5.1.3 Gene Expression Programming

This subsection presents the basic concepts of Gene Expression Programming (GEP) [120][121]. Gene Expression Programming (GEP) was introduced by Ferreira [122] in 2001. It was presented as a new technique for the creation of computer programs. GEP is an evolutionary algorithm which combines the principles of genetic programming (GP) and

genetic algorithms (GAs). It also requires a population of individuals which are selected according to fitness. Genetic variation is introduced by several genetic operators. The main difference to GAs and GP resides in the nature of the individuals.

Next, the outline of GEP is shown.

Algorithm 2.5. Gene Expression Programming

1. Create chromosomes of initial population
2. Express chromosomes
3. Execute each program
4. Evaluate fitness
5. If stopping criterion is reached, then stop else continue from step 6.
6. Keep best program
7. Select programs
8. Reproduction
    a. Replication
    b. Mutation
    c. Insertion Sequence transposition
    d. Reverse Insertion Sequence transposition
    e. Gene transposition
    f. 1 point recombination
    g. 2 point recombination
    h. Gene recombination
9. Prepare new programs for next generation. Go back to 2.

With GEP the individuals are encoded as linear strings of fixed length (chromosomes or genome) which are afterwards expressed as nonlinear entities of different sizes and shapes, most commonly through expression trees (ET). With GEP, the chromosomes are easy to manipulate genetically through several genetic operators. Expression trees represent exclusively the respective chromosomes. According to chromosome fitness, selection and reproduction acts upon the chromosomes and not the ET as it happens with the GP.

The evolutionary process is rather simple. It begins by creating a population where the chromosomes are randomly generated. Then, the chromosomes are expressed and their fitness computed. If the maximum number of generations is reached or the solution is obtained then

it terminates. Otherwise, a group of selected programs move on to the *reproduction* with modification phase. Reproduction includes replication and also other genetic operators that introduce genetic variety in the population. In reproduction, a chromosome is randomly selected for modification by each of the operators. It may happen that the same chromosome be modified by several reproduction operators at the same time, or not modified at all.

2.5.1.3.1 Open reading frames, the chromosome and the phenotype

Consider the expression:

$$a*b+c/d\,, \tag{2.130}$$

which can be represented by the expression tree:

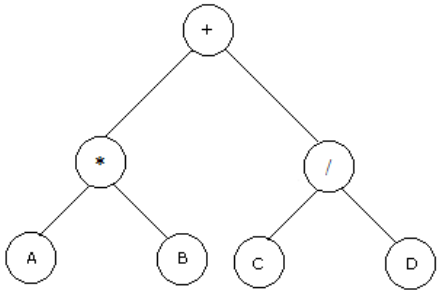


Fig. 2.28. Representation of expression (2.130)

The expression tree is in fact the phenotype of a GEP individual.

Reading the ET from left to right and from top to bottom (following a width-depth first procedure) the chromosome (or genotype) can be obtained:

$$\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ + & * & / & A & B & C & D \end{array} \tag{2.131}$$

The expression in (2.131) is in fact an *Open Reading Frame* (ORF) which in biology is a coding sequence of a gene, which begins with a "start" codon, continues with the amino acid codons and, ends with the termination codon. These ORFs are denoted by K-expressions (from the Karva language). In [123] prefix GEP is introduced, in which the process of translation from genotype to phenotype is done alternatively. The ET expresses the chromosome and is used to evaluate the individuals. In contrast to GP, variation in the population is carried out with GEP genetic operators on the chromosome and not on the ET.

2.5.1.3.2 GEP genes

A GEP gene consists of a head and a tail. The head contains both function and terminal elements. The tail contains terminals only. The head has a fixed length h_g whereas the tail

length t_g, is a function of h_g and the number of arguments of the function with the largest number of arguments n_g:

$$t_g = h_g(n_g - 1) + 1 \tag{2.132}$$

Consider for example the following gene where h_g=5 (the tail is shown in bold):

0	1	2	3	4	5	6	7	8	9	0
+	/	-	A	B	**C**	**D**	**E**	**F**	**G**	**H**

(2.133)

Translating the chromosome to an ET:

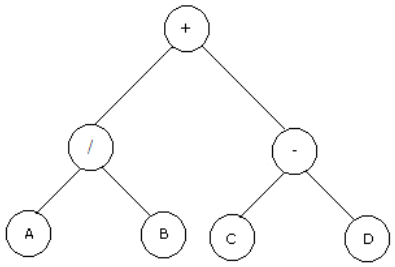


Fig. 2.29. Representation of expression (2.133)

It shows that the ORF is of length 7 whereas the gene is of length 11. So, the ORF may be of equal or less length of the gene, leaving some noncoding regions unused. This feature represents the essence of GEP because it this allows modification of the chromosome without any operator restrictions always producing syntactically correct programs [122].

2.5.1.3.3 Multigenic chromosomes

In her work, Ferreira also introduced a new coding where the GEP chromosome contains several genes, each of the same size.

Take for example the following 2-genic chromosome where each gene is of size 9 (t_g=5,h_g=4, n_g=2):

0	1	2	3	4	5	6	7	8	0	1	2	3	4	5	6	7	8
+	/	-	A	**B**	**C**	**D**	**E**	**F**	-	*	A	-	**A**	**B**	**B**	**C**	**G**

(2.134)

This approach is desired in evolving solutions for complex problems because they allow the modular construction of hierarchical structures. Each gene is a small building block separated from their counterparts, thus evolving independently. With multigenic chromosomes, there are as many sub-expression trees as genes. The translation between ETs and chromosomes remains the same. The ET for a multigenic chromosome results from linking all sub-trees with an appropriate linking function. A choice of the linking function can

be the addition operator "+". The figure below shows the ET resulting from the 2-genic chromosome in (2.134).

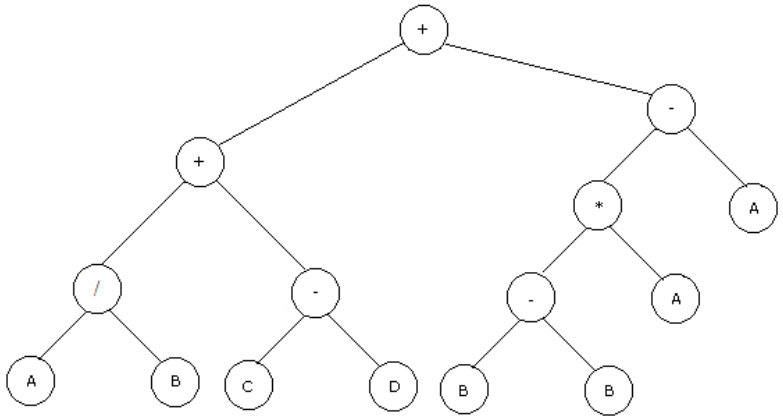


Fig. 2.30. Representation of expression (2.134)

The following subsection will describe the genetic operations used in GEP.

2.5.1.3.4 Reproduction with modification

According to fitness and some selection scheme (roulette wheel selection is the most used), individuals are selected for reproduction. There are the following operators in reproduction:

- Replication: chromosomes are faithfully copied into the next generation according to their fitness values and the selection scheme. One individual will be copied as many times as the outcome of the roulette wheel selection. The fitter individuals have higher probability of selection and so more chances to leave offspring. Individual's selection is carried out for as many times as the number of individuals in the population.
- Mutation: mutations occur anywhere in the chromosome as long as the structural organization of the chromosome remains intact. In the gene head any symbol can be changed into another symbol. In the tail only terminals can replace terminals:

  A mutation rate p_m equivalent to two point mutations per chromosome is typically used.

Consider the chromosome:

0	1	2	3	4	5	6	7	8	9	0
+	/	-	A	B	C	**D**	**E**	**F**	**G**	**H**

(2.135)

where a mutation at element in position 4 changes "B" to "+":

0	1	2	3	4	5	6	7	8	9	0
+	/	-	A	+	C	**D**	**E**	**F**	**G**	**H**

(2.136)

Notice that this simple mutation operation modifies drastically the ET.

After mutation, (2.136) will read as A/(E+F) +(C-D).

- Transposition of insertion sequence elements.

  In GEP there are fragments of the genome that can be activated and moved to another place in the chromosome. There are three kinds of such fragments.

  1. Short fragments that transpose to the head of the gene except the root - *Insertion Sequence elements* (IS).
  2. Short fragments that transpose to the root of the gene - *Root Insertion elements* (RIS).
  3. Entire genes that transpose to the beginning of the chromosome (Gene transposition).

- Recombination consists of exchanging genetic material between randomly chosen pair of parent chromosomes. There are three kinds:
  - 1 point recombination. A bond point at the gene is randomly chosen at the crossover point. The sequence downstream of this point is then cut at this bond and exchanged by the parents, resulting in two offspring chromosomes.

  Consider the two parent chromosomes:

0	1	2	3	4	5	6	7	8	9	0
+	A	-	B	+	C	**A**	**B**	**F**	**G**	**H**
+	/	+	A	+	C	**D**	**E**	**F**	**G**	**H**

(2.137)

Suppose that the bond at position 2 was selected for the crossover point. The offspring chromosome will be given as:

0	1	2	3	4	5	6	7	8	9	0
+	A	-	A	+	C	**D**	**E**	**F**	**G**	**H**
+	/	+	B	+	C	**A**	**B**	**F**	**G**	**H**

(2.138)

- 2 point recombination. Two points of recombination are randomly chosen in the two parents and the genetic material between the recombination points is exchanged, returning the offspring chromosomes.

  Regarding the two parents in (2.137), and supposing as crossover points bond 3 in parent 1 and bond 5 in parent 2, the offspring chromosomes are given as:

0	1	2	3	4	5	6	7	8	9	0
+	A	-	A	+	C	**A**	**B**	**F**	**G**	**H**
+	/	+	B	+	C	**D**	**E**	**F**	**G**	**H**

(2.139)

- Gene recombination. With gene recombination the entire gene is exchanged during crossover. A gene is randomly chosen and is exchanged between the two parents. This is useful if applied for multigenic chromosomes.

For more detailed outlook the reader is suggested to see [121][122][124].

2.5.1.4 Bacterial Evolutionary Algorithm

Bacterial Evolutionary Algorithm (BEA) is another approach in the scope of nature inspired algorithms. The basic concepts of bacterial mutation were introduced by Nawa and Furuhashi in their work on nonlinear system identification where a bacterial mutation operator replaced the usual genetic mutation operator [125]. It was called the pseudo-bacterial genetic algorithm. In the following year they proposed the bacterial evolutionary algorithm, which held the same features of the PBGA, but replaced the usual crossover with the gene transfer operation. The purpose of the gene transfer operator was to allow the chromosomes to directly transfer information to other counterparts in the population. By means of this mechanism, one bacterium can rapidly spread its genetic information to other cells [126].

Similar to the usual evolutionary algorithms BEA uses a chromosome to represent an individual. This representation can be through a sequence of bits or, in alternative, with real values. In either case, this sequence of bits is formed by the genes (or segment) which are smaller information units. Fig. 2.31 shows one chromosome consisting of N_s segments.

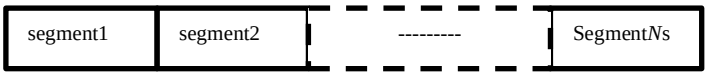


Fig. 2.31. Example of a chromosome in BEA

In the BEA, one individual is represented by the chromosome and is denoted by the *bacterium*. The population of individuals constitutes the group of solutions to the problem or the *bacteria*.

As was said previously, the BEA algorithm uses other operators than those from the genetic algorithm: the bacterial mutation and the gene transfer operation. With the bacterial mutation operation, one optimizes the chromosome of one bacterium. Transfer of information between bacteria in the population is done with the help of the gene transfer operation. Bacteria share chunks of their genes rather than perform neat crossover in chromosomes.

The algorithm consists of three steps (1,3,4) as illustrated below. First, a random initial population with N_{ind} individuals has to be created. Then, bacterial mutation and gene transfer are applied until a stopping criterion is fulfilled, which is usually the number of generations (N_{gen}).

Algorithm 2.6. Bacterial evolutionary algorithm

1. Creation of initial population
    a. Chromosome encoding
2. Set gen=1
3. Bacterial mutation
    a. Bacteria (individuals) fitness evaluation
    b. Bacteria cloning
    c. Gene mutation
4. Gene transfer
    a. Bacteria (individuals) fitness evaluation
    b. Bacteria ranking
    c. Bacteria infection
5. gen=gen+1
6. If stopping criterion is true end, else go back to 2.

When applying evolutionary type algorithms the first step is to define the encoding method. The evaluation of the individuals (bacteria) has to be discussed, too. The encoding method and the fitness evaluation of the individuals depend on the given problem. The operations of the algorithm have to be adapted to the given problem as well.

2.5.1.4.1 Bacterial mutation

The bacterial mutation operation is applied to each one of the bacterium in the population. Fig. 2.32 presents an illustration of the bacterial mutation for the first bacterium. As it can be seen, N_{clones} copies (or clones) of the bacterium are created. Then, a segment is randomly chosen in the bacterium and is mutated in each clone except one clone which is left unmutated (leaving this one out implements an elitist strategy). After mutating the same segment in the clones, each clone is evaluated. This evaluation criterion allows ranking the clones and the one with the best evaluation result transfers the mutated segment to the other clones.

These three steps (mutation of the clones, selection of the best clone, and transfer of the mutated segment) are repeated until all the segments of the bacterium have been mutated once. In the end, the best clone is kept as the new bacterium and the other clones are discarded.

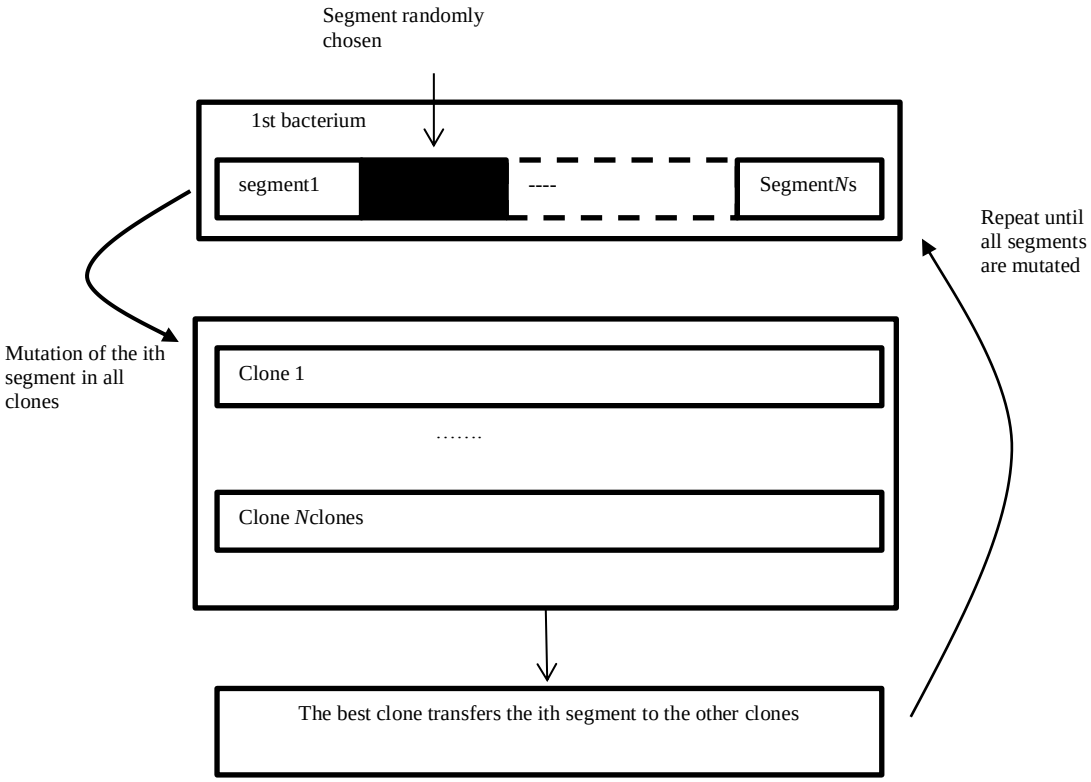


Fig. 2.32. Bacterial mutation operation

The former operations are repeated for each one of the bacteria.

2.5.1.4.2 Gene Transfer

After the bacterial mutation operation, there is a new ranking of the bacteria. With Gene Transfer there are two major phases. In the first, the population has to be sorted and divided into two halves according to the bacterium fitness. The best ranked bacteria will be considered the superior half of the population, whereas the bacteria with worse evaluation are referred to as inferior half (see Fig. 2.33).

The second phase consists of selecting randomly one bacterium from the superior half and another from the inferior half. The one on the superior half is called the source bacterium, and the one in the inferior half, the destination bacterium. Then, a segment of is chosen randomly from the source bacterium and this segment is used to replace the corresponding segment of the destination bacterium. These two phases, which include sorting the population, selection

of the source and destination bacteria and transferring the segment, are repeated N_{inf} times. N_{inf} is related to the number of "infections" per generation.

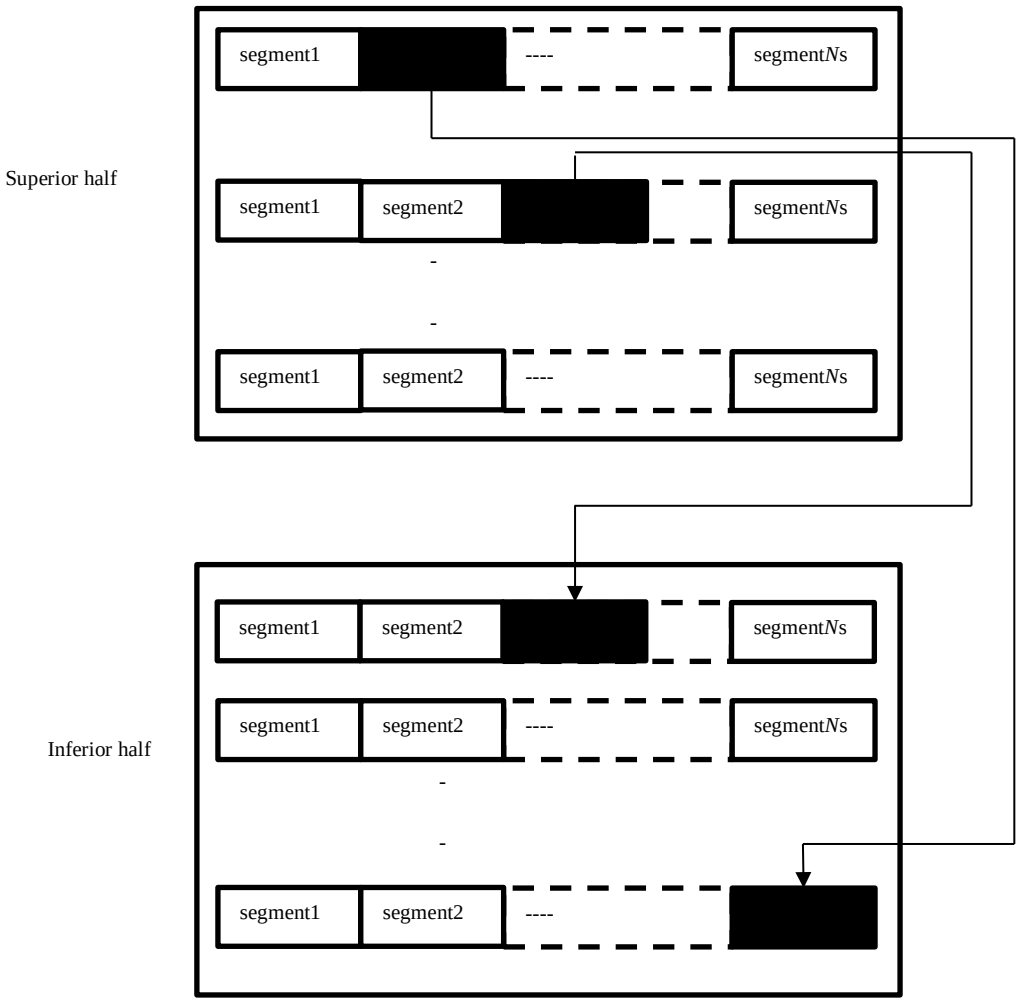


Fig. 2.33. Bacterial Gene Transfer operation

One advantage of the BEA algorithm is related to the number of parameters used to tune the evolutionary process which is substantially inferior than, for example the number required by the genetic algorithms.

2.5.1.4.3 Examples of applications

BEA has been applied to fuzzy systems [70][127] and Cerebellar Model Arithmetic Computer [128], for example. In [70], the Mamdany type fuzzy system with trapezoidal membership functions is optimized using the BEA algorithm. The encoding of the chromosomes is real-valued and each gene is a rule. The parameters of the membership functions parameters from the i^{th} rule are encoded in the i^{th} gene. Hence, the BEA algorithm

optimizes the parameters of the MFs. To help determine the optimal number of fuzzy rules three adaptive fuzzy operators are evaluated after bacterial mutation.

Recently, the BEA algorithm has also been applied for data clustering [129]. In this approach an entire partitioning of the input data is encoded in a single bacterium (chromosome) where each gene encodes one cluster. Its performance was compared to two recent state-of-the-art clustering techniques, where it showed better performance over majority of the benchmark datasets investigated.

Other examples of its application to fuzzy systems modeling are, for instance, given in [126] [130][131].

2.5.1.5 Selection Operators

In EAs, all modification operations rely on algorithms for the selection of individuals participating in these processes. As the genetic operations need to choose a percentage of individuals whose structure will be copied to the next generation, it will be important to choose those individuals who can best represent the nature of the problem. For instance, mutation should be applied to the chromosomes of the individuals who have little diversity, and not to the most fit ones. It is desirable, in most cases, to make a reasonable selection able to distinguish the best candidates worse. To this end, it is important to set an appropriate evolution of the population and is one of the base points related to the selected individuals. For the selection of individuals, it is first necessary to define the amount of individuals to select.

A summary on the most frequently used selection operators is given in the following subsections. For a deeper description please refer to [132].

2.5.1.5.1 Random selection

Individuals are selected with no reference to their fitness. Thus, equal chances of selection are given to good and bad individuals.

2.5.1.5.2 Proportional selection

Selection of individuals is proportional to their fitness values [13]. The probability distribution is created and the individuals are selected through sampling of the distribution.

The most common proportional selection scheme is *roulette wheel selection*. The fitness values are normalized and the probability distribution can be regarded as a roulette wheel

where each slice has a width corresponding to the selection probability of an individual. The selected individual is that which slice ends up at the top after the spinning of the wheel.

2.5.1.5.3 Tournament selection

With this selection scheme [133], a group of individuals is formed (between 2 and 7 individuals) chosen according to a random uniform distribution and the one from the group who presents the best position is selected. With tournament selection the worse of the individuals will not be selected, which can be regarded as an advantage in comparison to roulette wheel selection.

2.5.1.5.4 Rank-Based selection

With this methodology [134], candidates are ranked according to their rank and not according to their fitness, enabling greater distinction between individuals with similar high skills. Thus, this approach avoids domination of the selection procedure by the fittest individuals.

Examples of rank-based selection are linear ranking and exponential ranking.

2.5.2 Other techniques

The previous section described some of the most common evolution based methodologies, which are used in the realm of structure optimization; they present solutions to problems using models of natural selection.

The main drawback of all global approaches is related to the high computational demand. The most straightforward approach to cover the search space may be using a grid. As such, if there are n_z parameters and each one is discretized in Δ intervals, a number of Δ^{n_z} evaluations of the training criterion is necessary. If a convenient search of the space is required and the number of parameters grows, then computing effort will reach an unbearable level. As this exponential increase of dimensionality is problem dependent, alternatives are to examine closer the regions of the search space that are more likely to contain good local optima.

One of such methodologies is *simulated annealing* [135] which is a stochastic method. The name is related to the physical phenomenon of dropping a particle in a potential field. If the particle has non-zero temperature then it will move around in a random path, occasionally jumping to higher potential energy. The idea is that the particle can escape from local minima and may possibly fall in the global optimum. During the process, the particle is annealed, i.e., loses energy and the probability of moving towards higher potential decreases slowly. This

annealing effect may assure convergence to the global optima. In his work, Yiu *et al* [136] used Simulated Annealing as the technique to design the appropriate model for a BSNN to approximate the behavior of three nonlinear systems. His approach was based on the idea of adding the interior knot points in an intelligent way, so he treated the knots as independent decision variables and optimized them together with the weights. By using simulated annealing it was possible to avoid the local minima complications associated with the classical optimization algorithms.

Other technique is *Tabu search* [137]. It is especially dedicated to combinatorial optimization problems. It uses the local search techniques in the global search strategy. Typically it performs a multi-start approach in which local search is applied. The simplest version is to assign random values to the initial points but this approach is not efficient. Therefore, it requires memory to recall the points which it has investigated, avoiding repeating previously investigated points. Thus, the name stems from the fact that previously investigated points are tabu.

Another important issue is how the complexity of the model increases with the input space dimensionality. Models such as *fuzzy models* and *associative memory networks* (AMN) are lattice or grid based approaches where the input space is covered with a regular grid. Thus, their complexity scales up with the increase of input dimensionality.

The following classes of strategies present distinct ways of reducing the complexity of the problem [62].

2.5.2.1 Hybrid structures

The idea of a hybrid structure is to combine two different types of submodels to form the overall model. If there is a solid mathematical background on the process a submodel can be derived based on first principles, driven from laws of physics, chemical background, etc. This prior submodel can also be one state-of-the-art model obtained from the application of modeling techniques. As this model does not fully reflect the process, it can be combined with another submodel generated from data. The main advantage related to this approach when compared to the solely data driven submodel is to avoid from throwing away all the knowledge that is already acquired.

The authors in [138] developed a Hybrid algorithm where a feedforward artificial neural network is integrated with the MARS approach to perform breast cancer diagnostics. This approach starts by using the MARS algorithm in modeling *breast* cancer. This first phase returns the most significant predictor variables which, in a second phase are used as input

variables for the ANN model training. For example, the Adaptive Neuro-Fuzzy Inference System (ANFIS) proposed by Jang [139] is a five-layer feed-forward neural network and uses a hybrid learning algorithm that combines the least-squares estimator and the gradient descent method. The resulting model is the outcome of combining ANNs and Fuzzy Logic. The back-propagation training method is employed to finding the optimum value for the parameters of the membership functions and a least squares procedure is used for estimating the linear parameters on the fuzzy rules, in such a way as to minimize the error between the input and the output pairs. Jang and Mizutani [140], also discussed the pros and cons of using the Levenberg-Marquardt algorithm to train the ANFIS system instead of the gradient descent algorithm. One practical application of such methodology is described in [141] where it is used as a predictor to power factor in wind turbines. With ANFIS, the architecture has to be given in advance (human expertise decides the number of fuzzy rules) and only Takagi-Sugeno-type (TSK) [64] fuzzy models are implemented, which is seen as more difficult to interpret in comparison to Mamdani-type models [142].

The algorithm proposed in [143] offers a method for structure learning, where the ANFIS model does not. In this approach, the learning algorithm is able to determine the structure and the parameters of a fuzzy system, in an incremental way. If a-priori knowledge is given, some suitable rules are known beforehand, which provide an initial set of rules for the neuro-fuzzy system. The remaining rules are found by learning.

2.5.2.2 Projection-based structures

In this class, the input space is projected onto pre-defined axes (represented by weights) which represent most information on the data. This is based on input data pre-processing mechanism where a set of weights must be determined appropriately. Unsupervised techniques like Principal Component Analysis [144] are commonly applied. The model will be driven by a reduced set of input variables after removing redundant or, not correlated inputs, from the system. Thus, a high-dimensional problem can be approximated by lower-dimensional ones.

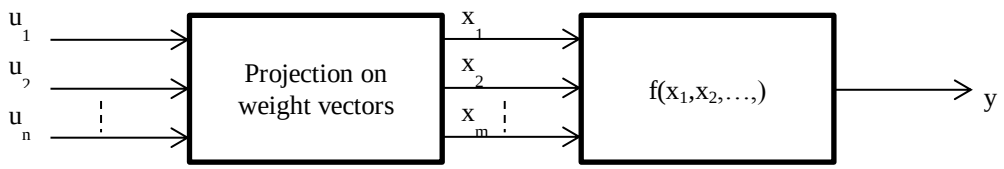


Fig. 2.34. Example of a projection based structure. The inputs u_i are projected on to the weight vectors yielding the reduced input set **x**. The model can be obtained as a function of $m < n$ input variables.

2.5.2.3 Additive structures

With additive structures, the high-dimensional model is decomposed into a sum of lower dimensional models. An example of an additive structure is Taylor series expansion of the process:

$$y = c_0 + c_1 x_1 + ... + c_n x_n + c_{11} x_1^2 + c_{12} x_1 x_2 + ... \tag{2.140}$$

There are some advantages related to this approaches. They can be easily constructed from data. Simple model structures are chosen that are linear in the parameters, so that the overall model will also be linear in all the submodels' parameters. To select the most relevant terms a subset selection technique such as OLS [27] can be used, however the most known technique is illustrated by the *Adaptive Splines Modeling of Observable Data* Algorithm (ASMOD) algorithm, which will be described in the next subsection. One application of the ASMOD for determining a BSNN was that of forecasting concentrations of the cyanobacterium Anabaena in the River Murray at Morgan, South Australia [145]. Comparison to the Multi-Layer-Perceptron (MLP) model showed that BSNN model performed slightly better in terms of accuracy, but it was even more considered in terms of model transparency, because of the clear way that information about the relationship between inputs and outputs was provided, in the form of fuzzy rules.

The multigrid-based fuzzy system (MGFS) model [146] is another additive structure which was proposed for high-dimensional function approximation using the lattice-based models, such as fuzzy systems. With MGFS the output of the model is a weighted average of the outputs of the rules of the components subgrids:

$$y = F[\mathbf{X}] = \sum_{i=1}^{P} F^i[\mathbf{X}^i], \tag{2.141}$$

where, $F^i[\mathbf{X}^i]$ denotes the functional decomposition of the i^{th} subgrid.

The learning mechanism employed in [146] determines the groups of variables in each subgrid (architecture selection) through a bottom-up approach in which the least complex possible structure (one subgrid per input variable) is first tried, and progressively increases groups complexity. All subgrids are regular (or complete) grids.

Group Method of Data Handling (GMDH) is another alternative approach to systematic design of nonlinear relationships between system's inputs and outputs. GMDH was introduced by Ivakhnenko [147] in 1971 as a means for identifying nonlinear relations between input and output variables. This way, it uses nonlinear regression polynomials and

cycles in similar ways to evolutionary algorithms, as it is based on the natural law of the survival of the fittest.

2.5.2.3.1 Training a B-Spline: The Asmod algorithm

The Asmod [148] algorithm employs a mixed structure identification strategy that attempts to adapt the structure of the model to data dependencies. It is based on a functional decomposition using the concept of ANalysis Of VAriance (ANOVA). It considers that a more accurate estimate can be obtained from a smaller amount of parameters when comparing to a general model. So, is applies the ANOVA decomposition of a general high-dimensional model into a set of additive lower dimensional substructures (submodels) which can be interpreted easier, making the model more transparent to the user and substantially reducing the complexity of the model. With this algorithm, the output variable is modeled as the sum of several submodels of lower dimension; each submodel depends upon a subset of input variables:

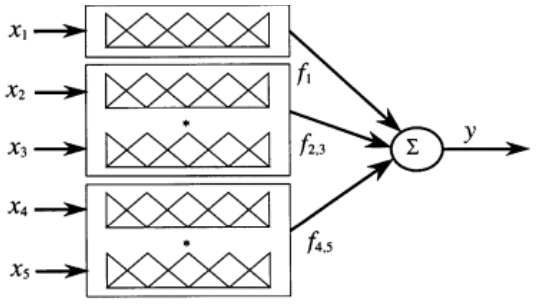


Fig. 2.35. ANOVA decomposition employed by the ASMOD algorithm (after Brown and Harris, [57]).

In Fig. 2.35, a five dimensional problem is decomposed into a sum of two bi-dimensional submodels and one one-dimensional submodel:

$$y(x_1, x_2, x_3, x_4, x_5) = f_1(x_1) + f_{2,3}(x_2, x_3) + f_{4,5}(x_4, x_5) \tag{2.142}$$

The outline of the algorithm is summarized as follows.

Algorithm 2.7. Asmod

1. Initial model definition
2. Repeat until some termination criterion is satisfied
    a. Incremental or refinement stage.
    b. Pruning stage.

The Asmod algorithm cycles through two main phases: refinement and pruning. The initial model corresponds to a very simple model whose structure consists of one set of univariate submodels. The set can include all possible input variables though the alternative to include only one input variable is desirable.

Before going into detail on the two stages of the algorithm's cycle, there are some considerations that follow.

At each step, each candidate is evaluated in accordance with the evaluation criteria adopted. At each stage of refinement, a candidate is chosen and compared with the corresponding best candidate from the pruning phase. The execution of the pruning phase occurs if one of two conditions is satisfied: given a user defined parameter which determines the number of stages of refinement before pruning is conducted; whenever the stopping criterion is reached, so that redundant substructures may be eliminated. According to this operating principle, the structural complexity of candidate models is increased, and a structure reduction is only performed after the creation of the more complex set of candidates. This process proceeds until the stopping criterion of the algorithm is achieved, i.e., as long as all the candidates generated by the refinement phase obtain an evaluation criterion worse than that obtained in the previous iteration.

The evaluation criterion most used is the BIC (2.117), where the measure of modeling used is the MSE (2.121), m is the number of input training patterns and n_z is the complexity of the B-spline model.

The following subsections will provide a description on the refinement and pruning phases.

It will be assumed that

$$L_i = S_i\left(\mathbf{x}_{\underline{i}}\right) \equiv S_{k_i,\boldsymbol{\lambda}_i}^{\mathbf{x}_{\underline{i}}}, \tag{2.143}$$

where $S_i\left(\mathbf{x}_{\underline{i}}\right)$ denotes the i^{th} submodel (substructure), $\mathbf{x}_{\underline{i}}$ is the set of input variables ($\underline{i}$) which composes the submodel and, k_i and $\boldsymbol{\lambda}_i$, are the order of splines and the knot vector for the corresponding i^{th} input variable, respectively.

2.5.2.3.1.1 Refinement phase

The process of refinement results in an ever increasing complexity of the structure of the model and is divided into three steps, with no specific order.

Assume the current model consists of U submodels:

$$L = \sum_{i=1}^{U} L_i \tag{2.144}$$

A brief description on each of the steps is given next.

Addition of univariate models. A new submodel representing a single input variable is added to a candidate model. Typically, this new submodel is created with zero interior nodes. In one iteration of ASMOD, as many candidates as the number of input variables still not present in the current model will be generated by this step.

The new model will include an additional submodel representing the new input variable x_j:

$$L_{U+1} = S^{x_j}_{k_j,\boldsymbol{\lambda}_j},\ x_j \notin \{\mathbf{x}\} \setminus (U^U_{i=1}\{\mathbf{x}_i\}) \tag{2.145}$$

Space dimensionality increase. Two existing submodels are combined and replaced by their tensor product, allowing the modeling of interdependencies between variables combined:

$$L = L_j \otimes L_k + \sum_{i=1, i\neq j,k}^{U} L_i \tag{2.146}$$

In one iteration as many candidates as the number of possible combinations to implement the current model are created.

Interior knot addition. A new interior knot is inserted into the input variables introducing better modeling capabilities to that variable. The position of the interior knot is restricted to the midpoint of two adjacent nodes.

Assuming the i^{th} submodel:

$$L_i = S^{x_{i1}}_{k_{i1},\boldsymbol{\lambda}_{i1}} \otimes S^{x_{i2}}_{k_{i2},\boldsymbol{\lambda}_{i2}} \otimes \ldots \otimes S^{x_{in}}_{k_{in},\boldsymbol{\lambda}_{in}} \tag{2.147}$$

The new knot vector after inserting one interior knot between the p^{th} and $(p+1)^{\text{th}}$ knots is:

$$\boldsymbol{\lambda}'_{ij} = (\lambda_{ij,1}, \lambda_{ij,2}, \ldots \lambda_{ij,k_i}, \ldots, \lambda_{ij,p}, \frac{\lambda_{ij,p} + \lambda_{ij,p+1}}{2}, \lambda_{ij,p+1}, \ldots, \lambda_{ij,n}) \tag{2.148}$$

The new substructure will then become described as:

$$L_i = S^{x_{i1}}_{k_{i1},\boldsymbol{\lambda}_{i1}} \otimes S^{x_{i2}}_{k_{i2},\boldsymbol{\lambda}_{i2}} \otimes \ldots \otimes S^{x_{ij}}_{k_{ij},\boldsymbol{\lambda}'_{ij}} \ldots \otimes S^{x_{in}}_{k_{in},\boldsymbol{\lambda}_{in}} \tag{2.149}$$

2.5.2.3.1.2 Pruning phase

This phase generates new candidate models, with the goal to inhibit irrelevant modifications made by the refining phase. Therefore it attempts to: eliminate the redundancy of the new variables, remove an interior knot which became superfluous, reduce the order of the basis functions as it becomes more convenient.

This stage also consists of three steps with no specific order, which are:

Removal of univariate submodel. A submodel with univariate splines of order k with zero interior nodes is replaced by one with order (*k*-1), without interior nodes. If the spline is linear, then the spline becomes constant and should be removed from the model.

Assuming as L_p the p^{th} univariate substructure for the i^{th} input variable and the j^{th} knot vector as $\boldsymbol{\lambda}_j = (\lambda_{j,1},...,\lambda_{j,nj})$ the reduction of spline order will produce the new knot vector $\boldsymbol{\lambda}_j = (\lambda_{j,2},...,\lambda_{j,nj-1})$ to a new univariate substructure:

$$L_k = S^{x_j}_{k_j-1,\boldsymbol{\lambda}^{'}_j} \tag{2.150}$$

Submodel's split. A combined substructure of n_s variables is split into n_s submodels of (n_s -1) combined variables.

If p is the chosen multivariate substructure where $L_p = L_{p1} \otimes L_{p2}$, after pruning, the resulting model structure will be given as:

$$L = L_{p1} + L_{p2} + \sum_{i=1, i\neq p}^{U} L_i \tag{2.151}$$

This process is applied to all possible decompositions of the current multivariate model.

Removal of an interior knot within the input domain of a spline function. Given the i^{th} substructure $L_i = S^{x_{i1}}_{k_{i1},\lambda_{i1}} \otimes S^{x_{i2}}_{k_{i2},\lambda_{i2}} \otimes ... \otimes S^{x_{in}}_{k_{in},\lambda_{in}}$, the new knot vector shorter by one interior knot is $\boldsymbol{\lambda}^{'}_{ij} = (\lambda_{ij,1}, \lambda_{ij,2}, ...\lambda_{ij,k_i}, ..., \lambda_{ij,p}, \lambda_{ij,p+2}, ..., \lambda_{ij,nj})$. The modified i^{th} substructure will be given by:

$$L_i = S^{x_{i1}}_{k_{i1},\boldsymbol{\lambda}_{i1}} \otimes S^{x_{i2}}_{k_{i2},\boldsymbol{\lambda}_{i2}} \otimes ... \otimes S^{x_{ij}}_{k_{ij},\boldsymbol{\lambda}^{'}_{ij}} ... \otimes S^{x_{in}}_{k_{in},\boldsymbol{\lambda}_{in}} \tag{2.152}$$

This process is applied to all interior nodes of all the input variables of all submodels in the current model, generating, for each application, a candidate.

After a run session of the Asmod algorithm, the output of a B-spline neural network is given by the combination of all its submodels:

$$y(\mathbf{x}) = \sum_{u=1}^{n_u} S_u\left(\mathbf{x}_{\underline{u}}\right) \tag{2.153}$$

The set of input variables satisfy:

$$\#\left(\bigcup_{u=1}^{n_u} \underline{u}\right) \leq n \tag{2.154}$$

If the weight vector and the basis function vector are partitioned accordingly, $\mathbf{w} = \#\left(\bigcup_{u=1}^{n_u} \mathbf{w}_{\underline{u}}\right)$ and $\boldsymbol{\varphi} = \#\left(\bigcup_{u=1}^{n_u} \boldsymbol{\varphi}_{\underline{u}}\right)$, then the output can be transformed in:

$$\mathbf{y} = \sum_{u=1}^{n_u} \sum_{j=1}^{p_n} \boldsymbol{\varphi}_{\underline{j}} \mathbf{w}_{\underline{j}} \tag{2.155}$$

Thus, it becomes obvious that the model output is a linear combination of the weights and the basis functions.

2.5.2.4 Hierarchical structures

These structures try to respond to situations where in a real process there is some kind of hierarchical organization between subsystems. One such example is the subsystem outputs corresponding to the states of the process. Hierarchical structures can be envisaged through a block diagram consisting of a cascaded lineup of subsystems, where one's output serves as one input to another [149]. An example of a hierarchical system is illustrated in Fig. 2.36.

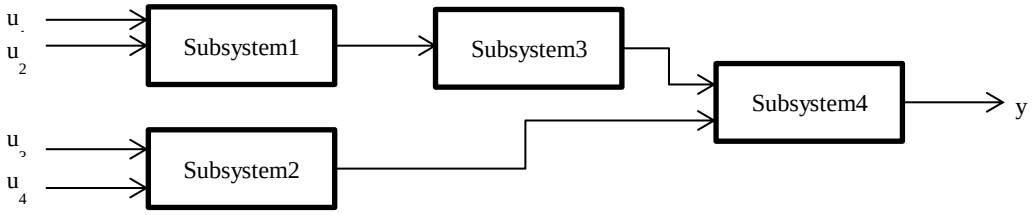


Fig. 2.36. A hierarchical structure where the model is represented by four subsystems in a cascaded architecture

With a hierarchical structure the subsystem's outputs can correspond to the internal dynamics of the system. Because the model is a hierarchy of the lower-dimensional structure, the interpretability is highlighted. Nevertheless they pose some drawbacks. There are no state-of-the-art algorithms for constructing these models. The parameters of the submodels influence the output of the model in a nonlinear way, undermining the optimization task.

To optimize these structures, genetic algorithms have been applied to fuzzy systems [150] [151][152].

An extensive analysis of the universal approximation and sensitivity properties of hierarchical fuzzy systems is given in [153], where the steepest descent algorithm was applied to design the hierarchical fuzzy system.

Chen *et al* [154] also utilized hierarchical based cascaded architecture to employ the selection of lower dimensional BSNN models. With a B-Spline hierarchical architecture the final model consists of multiple B-Spline networks assembled in different level or cascade B-Spline architecture. The hierarchical structure is used for selecting the most important input variables. The estimation of the parameters encoded in the structure is accomplished through Differential Evolution (DE). The encoding of the hierarchical BSNN is through the hierarchical B-Spline neural tree. Later, they extended this approach to real world problems.

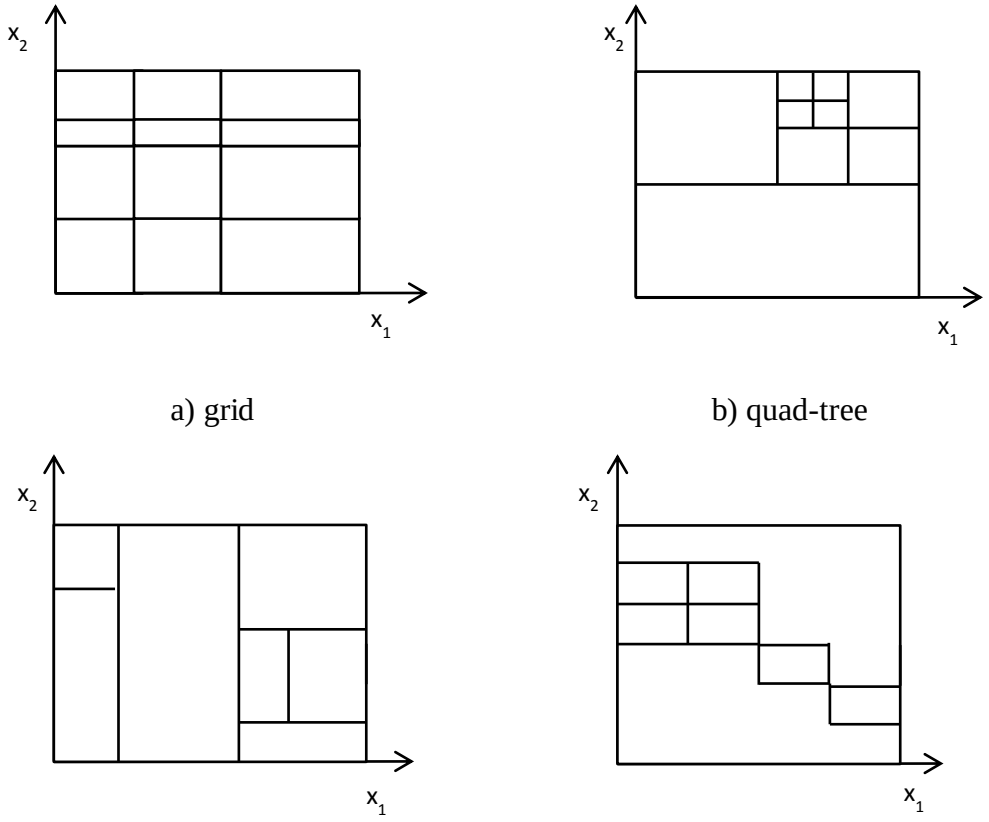
In [155], the same architecture was evolved for Breast Cancer Detection. This is a typical case of classification application. Therein, the hierarchical structure is evolved using another approach: the *Extended Compact Genetic Programming* (ECGP) and the parameters encoded in the structure are tuned using *Particle Swarm Optimization*. ECGP is seen as a direct extension of *Extended Compact Genetic Algorithm* (ECGA), which is based on the PIPE [156] prototype tree. In their work the basis functions order estimation is not considered, as only quadratic splines were used.

2.5.2.5 Input space decomposition

As in projection-based structures, the idea underlying input space decomposition strategies is to reduce a high-dimensional structure by exploiting the complexity of the process, without generating low-dimensional models. These strategies assume that some regions of the input space are less smooth than others, so they split the problem into smaller subproblems using a suitable input space decomposition [157]; in operating regions of the process where there is a strongly nonlinear behaviour, partitioning of the space should be more dense, whereas in smoother areas fewer partitions is acceptable.

The typical space partitioning is known as grid partitioning and is illustrated by Fig. 2.37-a). Tree-based partitioning divides the input domain space into smaller regions so that one can fit simple models to them. A general class of structures known as binary space partition (BSP) trees can be used to define a tree partitioning where axis-orthogonal splits are taken in each dimension. In a BSP, each node represents a convex region in space defining which points lye inside this region. Every node has an associated hyperplane cut partitioning of the upper region into two subregions, each corresponding to a child node. The root of the tree contains all input points.

One class of BSP is the quadtrees [158][159], where axis-orthogonal hyperplanes partitioning divides region volumes equally. Hence, the *n*-dimensional space is successively subdivided in four quadrants in a recursive manner. Variants include the adaptive quadtree partitioning where the partitioning borders are adjusted during model training, yielding rectangular regions and not square regions. The quad-tree of Fig. 2.37-b) is an example when fixed partitioning is used. Another class of BSP is *k-d* tree [160] partitioning illustrated by Fig. 2.37-c). Octree tree partitioning can also be used, where the space is subdivided in eight regions. In fuzzy rule-based system when the antecedent rule base covers only partially the universe of discourse, the partitioning of the input space resembles that of Fig. 2.37-d).



a) grid b) quad-tree

c) k-d tree d) scattered

Fig. 2.37. Example of typical grid partitioning

Another well-known data structure based on hierarchical subdivision of space into rectilinear regions is the Balanced Box-Decomposition (BBD) tree which is fully described in [161]. In this tree, each node is associated to a region of the input space, defined as a cell which can be either a box or the set theoretical difference of two boxes. The tree grows through the use of two recursive operations, *shrink* and *split*. While split refers to subdividing the cell into smaller divisions by axi-orthogonal hyperplanes, a shrink operation partitions a cell with a box inside the cell. Splitting is considered a variant of quadtree splitting rule. This approach partitions the input space using nonhyperplanar cuts with ‘‘holes,’’ sacrificing the convexity property for the regions associated with the tree nodes making it inappropriate for several applications where convexity is prime. The Balanced Aspect Ratio (BAR) tree described in [160] uses convex, but not necessarily axis-parallel, bounding volumes whose facets have a bounded number of different orientations, i.e., the construction method employs axis-orthogonal hyperplane cuts or cuts that form a 45º angles with the coordinate axis, denoted as corner cuts.

Another type of tree partitioning is the one used by Box trees [162]. They consist of binary trees whose leaves store boxes in the input space and where the internal node *k* stores the bounding box of all boxes stored under the subtree at *k*. The box tree is used as bounding volume hierarchy storage for a collision checking of objects in a 3D scene. A comparison study of common partitioning methods can be given in [163].

A quite famous *constructive* learning algorithm which partitions the input space recursively is the *Multivariate Adaptive Regression Splines* (MARS), proposed by Friedman [164]. As it proceeds it generates an output polynomial from a discrete set of truncated polynomial basis functions to represent piecewise mappings. Multivariate basis functions are generated by the product of truncated polynomial basis functions (functions set to zero when the argument is negative, otherwise define a mapping polynomial). MARS builds models in two stages. During the refinement stage, the algorithm uses only truncated polynomial basis functions to form a discrete tree in which each node is considered a *parent* of two *offspring* nodes, each representing truncated basis functions of a univariate polynomial. New basis functions are formed by the product between the respective branch nodes. New nodes are also inserted along the axis of a univariate function, and the tensor product terms are formed when the parent nodes are linear truncated basis functions. The pruning stage consists of reducing the model complexity by removing nodes. At each iteration, models are assessed through cross-validation. Despite the success of this algorithm it cannot be directly applied to a neuro-fuzzy system because it produces models with some undesirable properties. Training is ill-conditioned, representation of input data is non-sparse and it is, not logically consistent with fuzzy interpretation.

Another application of a tree based partitioning is the fuzzy linear regression tree [165] which is used as a mechanism for evolving a fuzzy system with a tree topology. Based on linear regression trees, each node is associated with a linear model of the input variables, in a similar fashion as a TS-type fuzzy system. Starting from the root node, decision tests are taken and a final leaf is reached. Then, the output of the model is obtained from the linear model at the leaf. Each leaf of the tree represents a cell of the partition, and has attached to it a simple model which applies in that cell only The splitting decision tests in the internal nodes depend on the evaluation of the concepts *greater than* and *less than* using the spread parameter of the sigmoid functions used as membership functions. A *constructive* learning algorithm is used to grow a tree. It starts with a single leaf tree and its linear model, and evolves the tree by replacing leaves with subtrees.

Another algorithm (*constructive* type) widely known for being related to input space decomposition is *Local Linear Model Tree* (LOLIMOT) [166]. The basic idea is to approximate a nonlinear map with piece-wise Linear Local Models (LLMs), partitioning the input space by axis-orthogonal splits. It applies input partitioning to the worse performing LLM, by splitting its hyper-rectangle into two halves with an orthogonal split, this being executed in every dimension. The parameters of the LLMs are estimated by the local weighted least squares algorithm and to avoid computational burden no local nonlinear technique is employed. The model output is given by summing up the contributions of the LLMs weighted with their basis functions values:

$$y(x_1, x_2) = \Phi_1(\Phi_{11} y_{11} + \Phi_{12} y_{12}) + \Phi_2(...) \tag{2.156}$$

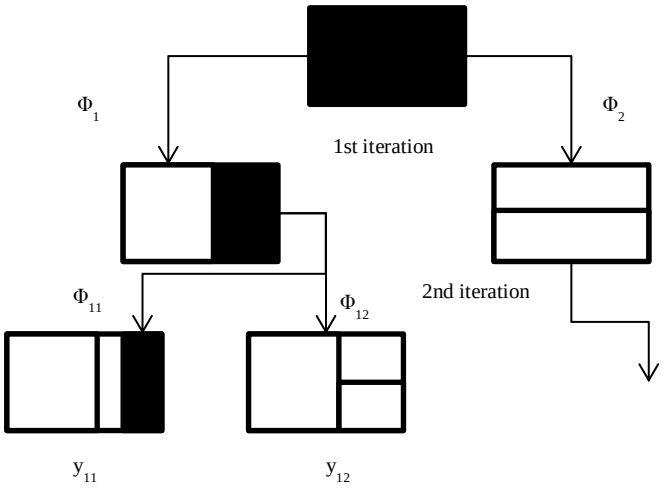


Fig. 2.38. Example of a structure search procedure for the LOLIMOT algorithm for a two-dimensional input space

For the stopping criterion, the number of estimated parameters is considered to balance between complexity and accuracy. As so, either statistical information criteria or the complexity must be used as training criteria. A drawback with LOLIMOT is that the tree building procedure is greedy and so the number of rules obtained can be extremely large.

Employing the same principle as LOLIMOT, hinging hyperplane trees as proposed by Ernst [167] employ input partitioning in similar ways, though basis functions may be defined in each level of a binary tree. The output of the model reflects the contribution that passes from the basis functions from the lower node towards the root.

Another algorithm is the *Classificatio*n *and Regression Trees* (CART) algorithm [168], which is typically used for classification purposes. It includes a pruning strategy typically

applied at the end of the growing phase. However, this is seen as a disadvantage because unnecessarily complex model are first created, and this is quite time consuming.

In order to allow local model flexibility, Kavli suggested a similar algorithm for the construction of BSNN models that use both global and local partitioning. It is denoted as Adaptive B-Spline Basis Function Modeling of Observational Data (ABBMOD) and it results in a *k-d* tree partitioning of the input space. This algorithm features the same two phases of evolution: refinement and pruning. In contrast to ASMOD, the corresponding operations are modified so that local partitioning is exploited. Hence, it performs completely local refinements. For instance, instead of forming the tensor product of univariate subnetworks it multiplies *single* multivariate basis functions. The structure of the ABBMOD algorithm becomes more complex and more general than that of the ASMOD. These modifications impose many restrictions related to the set of possible candidates that can be employed. A similar approach to ABBMOD which implements the same functional decomposition as the ASMOD algorithm but implements a new input domain decomposition is proposed in chapter 6.

2.5.2.6 Additional techniques

Jacobs and Jordan [169] introduced the concept of learning through partitioning a task into two or more functionally independent tasks each represented by a different neural network. As a result, different networks learn different training patterns and learn to compute different functions. Their idea was to approximate the process dynamics using local models that are identified with the process's operating point.

Recently, in [20], Coelho and Guerra proposed Differential Evolution (DE) and an improved DE version using chaotic sequences (DEC) to train a BSNN model for the nonlinear identification of an experimental nonlinear yo–yo motion control system. They compared the performance of the two approaches using different complexity models, by ranging the number of interior knots for each one of the three inputs considered for the training. Hence, the structure of the model was not searched for, but given a-priori.

2.6 Numerical integration techniques

There are many situations in which integrals without analytical solutions have to be evaluated. Examples are econometric models and Limited Dependant Variable (LDV) models. Even many relatively simple integrals cannot be expressed in finite terms of

elementary functions, and thus must be evaluated by numerical methods [170]. For the one dimensional problems an efficient quadrature technique is the Gaussian quadrature which is known to work efficiently for a large class of problems. Direct extension of quadrature techniques to multiple dimensions is straightforward and is processed through tensor product of one-dimensional quadrature rules. This has a direct effect on the computational cost which rises exponentially. But the main problem associated with these product strategy methods is the fact that exact solutions are not restricted to the class of polynomial of a given total order. Therefore it is very difficult to directly derive such techniques. Alternatives are methods of integration on sparse grids such as described in [171] or Monte Carlo.

The integration problem of a *n*-dimensional function is described as

$$\int_{\Omega} g(\mathbf{x})d\mathbf{x}, \tag{2.157}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and Ω is the support.

2.6.1.1 Univariate numerical integration

If one considers the one-dimensional case, the problem can be stated as of finding the area *I* below an arbitrary curve in some interval [a,b].

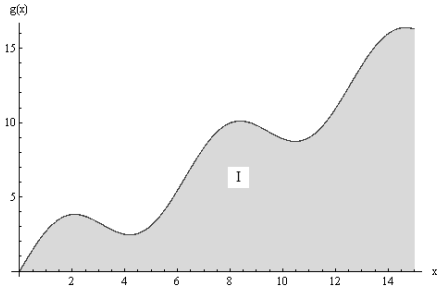


Fig. 2.39. Illustration of function integration in one dimension.

In the simplest possible approach the value of area *I* can be approximated by a direct summation over *m* points drawn from a regular interval Δx:

$$\begin{aligned} I &= \sum_{i=1}^{m} g(x_i).\Delta x \\ &= \frac{b-a}{m}\sum_{i=1}^{m} g(x_i) \end{aligned} \tag{2.158}$$

However, better accuracy is achieved when other strategies are taken.

In general, a solution for the integral of a one-dimensional problem is given from

$$I = \int_{\Omega} g(x)w(x)dx \tag{2.159}$$

In (2.159), *g(x)* is a function depending on random variable *x*, and *w(x)* is the *pdf* of *x* with support Ω. The integral value *I* is the expectation value of *g(x)*.

In the following some of the most common integration rules for one-dimensional problems are described.

2.6.1.1.1 Monte Carlo

For nonlinear functions *g(x)*, a numerical approach to *I* is required, and the one straightforward approach is using the Monte Carlo (MC) simulation.

The *crude* Monte Carlo algorithm estimates the integral *I* by averaging the function values $g(x_i)$ over uniformly random input points within its support.

Given a set of *m* random numbers or "nodes" drawn from a distribution characterized by *w(x)*, the simulated integral of *g(x)* is then:

$$I_{MC} = \frac{1}{m}\sum_{i=1}^{m} g(x_i) \tag{2.160}$$

With the increase of the number of random samples, the estimated standard deviation is reduced to satisfy the precision goals. The expectation of I_{MC} is *I* and its standard deviation decreases as $\sigma_{MC} = \frac{1}{\sqrt{m}}$.

When evenly distributed or a deterministic sequences of points are used to estimate the integral, the MC algorithm is called a quasi Monte Carlo algorithm. In this case, the points are constructed to be approximately equidistributed over the region of integration.

2.6.1.1.2 Gaussian quadrature

This technique belongs to a class of quadratures for which a formula is obtained that produces the exact integral for a class of functions, typically polynomial ones. Gaussian quadrature techniques produce nodes (input points) and weights for which the rule is exact for polynomial of a given order. The estimate of the integral of function *g(x)* is computed as

$$I_{GQ} = \sum_{i=1}^{m} g(x_i).w_i \tag{2.161}$$

Notice that in (2.161), the sets of nodes satisfy $x_1 < x_2 < ... < x_m$, being both the nodes and the weights depend on Ω and not on *g*. The values of the nodes can be computed, see for example [172], or can be retrieved from tables available in the literature.

Interpolatory quadrature rules where the nodes are equally spaced are called Newton-cotes formulas and belong to the group of classical formulas. A few will be briefly described next.

2.6.1.1.3 Trapezoidal integration rule

The trapezoidal integration rule is based on a linear interpolation where the integral equals the area of a trapezoid with base $(x_{i+1}-x_i)$ times the average height $\frac{1}{2}\left[g(x_{i+1})+g(x_i)\right]$.

The integral considering *m* points is approximated as:

$$I_T=\int_\Omega g(x)dx\approx\frac{1}{2}\sum_{i=1}^{m-1}\left(g(x_{i+1})+g(x_i)\right)(x_{i+1}-x_i) \tag{2.162}$$

With the trapezoidal rule, the standard deviation of the estimate decreases $\sigma_T=\frac{1}{m^2}$.

This is a convergence rate significantly higher than that produced by the MC method.

2.6.1.1.4 Forward integration rule

Similar to the trapezoidal rule, except that the height of the trapezoid is the value $g(x_{i+1})$. The integral is approximated as:

$$I_F=\int_\Omega g(x)dx\approx\sum_{i=1}^{m-1}g(x_i)(x_{i+1}-x_i) \tag{2.163}$$

2.6.1.1.5 Backward integration rule

The same as the forward integration rule, where the height of the trapezoid is calculated using $g(x_i)$.

The integral is approximated as:

$$I_B=\int_\Omega g(x)dx\approx\sum_{i=1}^{m-1}g(x_{i+1})(x_{i+1}-x_i) \tag{2.164}$$

2.6.1.1.6 Simpson's composite integration rule

The Simpsons composite rule is a very popular quadrature rule with two orders of accuracy gained when compared to the trapezoidal rule, without using more function evaluations.

After dividing the interval of integration in an even number of steps of length *h*, the integral is approximated as:

$$I_{SCR}=\int_{x\Omega}g(x)dx\approx\frac{h}{3}\left(g(x_0)+4\left[g(x_1)+g(x_3)+\ldots+g(x_{m-1})\right]+2\left[g(x_2)+g(x_4)+\ldots+g(x_{m-2})\right]+g(x_m)\right) \tag{2.165}$$

The last equation can be expressed in the form of (2.159) where the weights w_i take the values {1,4,2} in a pre-defined order:

$$\begin{aligned}I_{SCR}&=\frac{h}{3}\mathbf{g}.\mathbf{w}\\&=\frac{h}{3}\left[g(x_0),g(x_1),g(x_2),g(x_3),\ldots,g(x_m)\right]\left[1,4,2,4,\ldots,1\right]^T\end{aligned} \tag{2.166}$$

2.6.1.1.7 Integration by extrapolation

Alternatives to the Newton-cotes rules are formulas that avoid some drawbacks of the latter rules. Examples are negative weights for higher order rules.

Based on the Euler-MacLaurin formula an extrapolation of the results from the trapezoidal rule allows determining the suitable step size and order automatically. This strategy is called Romberg's method.

In adaptive quadrature methods, the step sizes are automatically adjusted so that the approximation satisfies an error tolerance ε:

$$\left| I - \int_{\Omega} g(x) dx \right| \leq \varepsilon \tag{2.167}$$

This methodology is desirable when the integrand has strongly varying orders of magnitude in the interval of integration.

2.6.1.1.8 Quadrature rules with free nodes

When there are prescribed nodes several methods can be applied. These belong to the class of quadrature rules with free nodes. Examples are: method of undetermined coefficients, Gauss-Christoffel quadrature rules and Gauss quadrature with preassigned nodes. For a more extensive on these quadratures the reader is suggested to refer to [170].

2.6.1.2 Mathematica® Interpolating polynomial

With this approach numerical integration is applied in two steps. First, basic polynomial interpolation is undertaken using the input samples. In this way, Mathematica® Interpolation function constructs an interpolation of the function values $g(x_i)$. Secondly, numerical integration evaluates the integral of the returned polynomial using Mathematica® NIntegrate function [173]. NIntegrate estimates the integral through sampling of the integrand value over the integration domain. The algorithm it uses employs integration rules where the computed integral is estimated using weighted sums. In NIntegrate, a global adaptive strategy improves integral estimate by recursive bisection of the subregion with the largest error estimate into two halves. It works with both Cartesian product rules and fully symmetric multidimensional rules. It stops when the sum of the errors of all regions satisfies the precision goal. The integral estimate is the sum of the integral estimates of all subregions. The integration rule samples the integrand with a set of sampling points. In order to improve the integral estimate, the estimate of the integrand must be estimated for additional points.

The integration rules employed belong to strategies which can be divided into two general groups: deterministic and Monte Carlo. These strategies are further divided into adaptive, nonadaptive, and specialized strategies. Adaptive strategies aim at improving the integral estimate by concentrating their efforts around the problematic areas. When the improvement of the integral estimate is obtained by increasing the number of sampling points in the integration region, a non-adaptive strategy is applied. Specialized strategies are applied for certain types of integrals. Within the adaptive and stochastic strategies, both Monte Carlo and Quasi Monte Carlo strategies can be applied.

Adaptive strategies encompass GlobalAdaptive and LocalAdaptive and are used with one-dimensional and multidimensional integration rules.

A local adaptive strategy recursively partitions the subregion into smaller disjoint subregions and computes integral and error estimates for each of them. This recursion process terminates when the error of each region is small enough when compared to the estimate of the integral.

2.6.1.3 Mutidimensional integration

Numerical integration for multiple dimensions is also referred to as numerical cubature. The number of function values required to obtain a desirable integral estimate precision grows exponentially with the number of dimensions, *n*. If *m* points are required to estimate an integral in one-dimension, then m^n points would be needed for *n* dimensions. This may signify that obtaining adequate accuracy can be an intractable problem.

If the number of dimensions can be shortened by applying analytical techniques to parts of the numerical integration then the integral can be simplified. This may include transformation of a variable in order to reduce the number of dimensions.

Alternatively other approaches can be undertaken.

A brief description on the strategies employed in the multidimensional cases is given next.

2.6.1.3.1 Repeated one-dimension integration

If it is possible to decompose de input domain Ω into the union of simpler input domains, then one-dimensional integral can be evaluated.

2.6.1.3.2 Product rules

Considering a two-dimensional problem, it relates to introducing an equidistant rectangular grid in the (x,y) plane, with grid spacings q_1, and q_2, respectively.

The product trapezoidal rule is

$$I_T = q_1 q_2 \sum_{i=1}^{m_1-1} \sum_{j=1}^{m_2-1} \frac{1}{4}\left[g\left(x_{1,i-1}, x_{2,j-1}\right) + g\left(x_{1,i-1}, x_{2,j}\right) + g\left(x_{1,i}, x_{2,j-1}\right) + g\left(x_{1,i}, x_{2,j}\right)\right] \tag{2.168}$$

The product composite Simpsons rule is

$$I_{SCR} = \frac{q_1 q_2}{9} SCR_{x_1} \otimes SCR_{x_2} \tag{2.169}$$

where SCR_{x_1} denotes the vector with function values multiplied by the corresponding weights, for dimension x_1:

$$SCR_{x_1} = \left[g(x_0), 4g(x_1), 2g(x_2), 4g(x_3), ..., g(x_m) \right] \tag{2.170}$$

The product Gaussian quadrature rule assumes identical procedure as the product composite Simpsons rule:

$$I_{GQ} = GQ_{x_1} \otimes GQ_{x_2} \tag{2.171}$$

This concept of product integration rules can be applied only to shapes which are obtained from the Cartesian product of lower-dimensional regions (rectangles, cubes, etc).

2.6.1.3.3 Irregular triangular grids

For nonrectangular shapes, the rectangular grid may also be bordered by triangles or "triangles" with one curved side. One advantage in comparison to rectangular grids is the facility to adapt the density of points to the behaviour of the function. Therefore, a complicated region can be approximated by a grid of triangles of arbitrary form.

The use of barycentric coordinates for a triangle helps in doing so.

2.6.1.3.4 Monte Carlo

An efficient way of avoiding the curse of dimensionality posed by product rules is the Monte Carlo approach.

In n dimensions the generalization of equation (2.158) for an interval ($[a_1,b_1]$, $[a_2,b_2]$,…, $[a_n,b_n]$) gives an (n+1)-dimensional volume:

$$I = V^{(n+1)} = \frac{b_1 - a_1}{m_1} \frac{b_2 - a_2}{m_2} \cdots \frac{b_n - a_n}{m_n} \sum_{i_1=1}^{m_1} \cdots \sum_{i_{n-1}=1}^{m_{n-1}} \sum_{i_n=1}^{m_n} g(\mathrm{x}), \tag{2.172}$$

where m_i is the number of samples in the i^{th} dimension.

Eq. (2.172) can also be written as

$$I = V^{(n)} \frac{\sum_{i_1=1}^{m_1} \cdots \sum_{i_{n-1}=1}^{m_{n-1}} \sum_{i_n=1}^{m_n} g(\mathrm{x}_i)}{m} \tag{2.173}$$

Another way of interpreting (2.172) is by taking the average over g:

$$I_{MC} = V^{(n)} \frac{\sum_{i=1}^{m} g(\mathrm{x})}{m} \tag{2.174}$$

2.6.1.3.5 Multivariate quadrature on sparse grids

Using sparse grids the objective is to extend the univariate quadrature rules to multiple dimensions with a substantially lower number of function evaluations than the product rule.

2.7 Conclusions

This chapter introduced the theoretical background needed to present the work developed in this PhD thesis, as well as reviewing important work on systems identification with CI techniques, with a special emphasis on training algorithms and structure identification techniques.

In the following chapter a BSNN model architecture is identified considering that a priori knowledge is given. As will be shown, genetic programming and gradient-based algorithms can be easily adapted to accommodate a priori knowledge in the design.

Chapter 3

INCORPORATING A PRIORI KNOWLEDGE IN B-SPLINES DESIGN

3.1 Introduction

As it has been pointed out in 1.2, whenever there is a priori knowledge about the process, this information should be incorporated in the model. The relevance of this information is definitely related with the model purpose, and the easiness or difficulty of incorporating the information in the model will depend on the architecture chosen. In fuzzy or in neuro-fuzzy models, a-priori knowledge is usually incorporated in modeling via rule or weight initialization [22].

Evolutionary algorithms are powerful optimization tools but, in most of the cases, they are not suitable for real-time control applications. Typically they need a significant number of objective function evaluations to converge and it might not be practical to obtain those values, on-line. One alternative is to employ a model to approximate the mapping between the design variables to the objective function and to use the model to supply, on-line, the objective function values to the optimizer. This occurs, frequently, in controller tuning (please see, for instance, [49][174]). In this type of application, if there is a-priori knowledge of the localization of some local minima, this knowledge should be, if possible, incorporated in the model. This chapter describes on how this can be easily obtained if a B-Spline architecture is chosen.

This chapter, which is an extended version of [175] is organized in the following way.

In section 3.2 the mathematical background is presented. It provides the reader with the necessary equations when equality (function and derivative) restrictions are imposed. As is will be shown this implies setting restriction on the linear weights of the BSNN model. Then, section 3.3 gives complementary mathematical formulation as required.

Section 3.4 describes how this methodology can be incorporated into two different structure determination techniques, ASMOD and SOGP. Section 3.5 evaluates the performance of these methodologies, using bivariate function optimization benchmarks. Finally, conclusions are given in section 3.6.

3.2 Incorporating equality restrictions

Assume that, together with the minimization of (2.17), one needs to add some equality restrictions such as:

$$\mathbf{y}(\mathbf{P}) = \underline{\mathbf{y}}, \tag{3.1}$$

and/or

$$\frac{\partial \mathbf{y}(\mathbf{Q})}{\partial i_i} = \underline{\mathbf{y}}_i'(\mathbf{Q}), \tag{3.2}$$

where **P** denotes the m_f input points where the function equalities should hold, **Q** the m_d points where the derivative equalities should hold, and i_i the i^{th} input variable. Equalities of both types can be recast as a linear system of equations:

$$\mathbf{B}\mathbf{w}_d = \mathbf{b} \tag{3.3}$$

where rank(**B**)<p, p being the number of columns of $\mathbf{\Gamma}$, the matrix of basis functions outputs.

Let us assume that $m_r = m_f + m_d$ restrictions should be satisfied. Then, m_r linear weights will be depending ($\mathbf{w}_d$) on the restrictions while the other (p-m_r) linear weights will be independent of the restrictions ($\mathbf{w}_i$). Assuming, without loss of generality, that the dependent ones will be the first weights, then the BSNN model's output will be given as:

$$\mathbf{y} = \begin{bmatrix} \mathbf{\Gamma}_d & \mathbf{\Gamma}_i \end{bmatrix} \begin{bmatrix} \mathbf{w}_d \\ \mathbf{w}_i \end{bmatrix}, \tag{3.4}$$

where $\mathbf{\Gamma}_d$ and $\mathbf{\Gamma}_i$ are the column partition related with the dependent and independent weights, respectively.

Depending on the type of equalities one is dealing with, whether (3.1) or (3.2), the computation of **B** will differ. It is straightforward for the case of function restrictions but slightly more complicated for the case of derivative restrictions.

3.2.1 Function equality restrictions

Considering the first case (3.1),

$$\mathbf{B} = \mathbf{\Gamma}_d(\mathbf{P}) \tag{3.5}$$

3.2.2 Function derivative restrictions

Considering n_u sub-modules, denote by S_{u^i} those sub-modules that depend on the input variable *x*, and by S_u those that do not depend on *x*. Therefore, performing the analysis for each individual equality:

$$y(\mathbf{Q}) = \sum_{u=1}^{n_{u^i}} S_{u^i}(\mathbf{Q}_{\underset{\sim}{u}}) + \sum_{u=1}^{n_u - n_{u^i}} S_u(\mathbf{Q}_{\underset{\sim}{u}}) \tag{3.6}$$

Obviously, the derivative of the 2nd sum is null. Therefore:

$$\frac{\partial y(\mathbf{Q})}{\partial x} = \sum_{u=1}^{n_{u^i}} \frac{\partial S_{u^i}(\mathbf{Q}_{\underset{\sim}{u}})}{\partial x} \tag{3.7}$$

Considering now the derivative of each sub-model and, for the sake of simplicity, that only 2 input variables (*x* and *y*) intervene in this sub-model:

$$\begin{aligned} \frac{\partial S_{u^i}(\mathbf{X}_{\underset{\sim}{u}})}{\partial x} &= \frac{\partial}{\partial x} \sum_{i=1}^{r_x+k_x} \sum_{j=1}^{r_y+k_y} \mathbf{w}_{i,j} N_{k_y}^j(y) N_{k_x}^i(x) = \\ &= \sum_{j=1}^{r_y+k_y} N_{k_y}^j(y) \frac{\partial}{\partial x} \sum_{i=1}^{r_x+k_x} \mathbf{w}_{i,j} N_{k_x}^i(x) \end{aligned} \tag{3.8}$$

Using only the non-zero derivatives, one has:

$$\frac{\partial S_{u^i}(\mathbf{X}_{\underset{\sim}{u}})}{\partial x} = \sum_{j=1}^{r_y+k_y} N_{k_y}^j(y) \sum_{i=1}^{r_x+k_x-1} (k_x - 1) \frac{\mathbf{w}_{i+1,j} - \mathbf{w}_{i,j}}{\lambda_{x_i} - \lambda_{x_i - k_i + 1}} N_{k_x-1}^i(x) \tag{3.9}$$

Therefore, the derivative of a multidimensional submodel is another multidimensional submodel, all with the same order, except the one related with the variable for which the derivative is taken, which has its order decreased by one unit. All the weights are updated, according to:

$$\mathbf{w}_{i,j} = (k_x - 1) \frac{\mathbf{w}_{i+1,j} - \mathbf{w}_{i,j}}{\lambda_{x_i} - \lambda_{x_i - k_i + 1}} \tag{3.10}$$

In order to satisfy the restrictions, eq. (3.9) can be expressed in another form:

$$\frac{\partial S_{u^i}(\mathbf{X}_{\underset{\sim}{u}})}{\partial x} = \sum_{j=1}^{r_y+k_y} N_{k_y}^j(y) \left(\begin{array}{l} \sum_{i=1}^{r_x+k_x-1} \frac{(k_i - 1)}{\lambda_{x_i} - \lambda_{x_{i-k_x+1}}} N_{k_x-1}^i(x) \mathbf{w}_{i+1,j} - \\ - \sum_{i=1}^{r_x+k_x-1} \frac{(k_x - 1)}{\lambda_{x_i} - \lambda_{x_{i-k_x+1}}} N_{k_x-1}^i(x) \mathbf{w}_{i,j} \end{array} \right) =$$

$$= \sum_{j=1}^{r_y+k_y} N_{k_y}^{j}(y)\left(\sum_{i=1}^{r_x+k_x-1} a_i \mathbf{w}_{i+1,j} - \sum_{i=1}^{r_x+k_x-1} a_i \mathbf{w}_{i,j} \right) =$$

$$= \sum_{j=1}^{r_y+k_y} N_{k_y}^{j}(y)\left(-a_1 \mathbf{w}_{1,j} + \sum_{i=2}^{r_x+k_x-1} (a_{i-1} - a_i)\mathbf{w}_{i,j} + a_{r_x+k_x-1}\mathbf{w}_{r_x+k_x,j} \right) \tag{3.11}$$

where:

$$a_i = \frac{(k_i - 1)}{\lambda_{x_i} - \lambda_{x_{i-k_x+1}}} N_{k_x-1}^{i}(x), \quad i = 1 \cdots r_x + k_x - 1 \tag{3.12}$$

Note that the complexity of (3.11) is misleading. As it is known, there are only k_i active splines in the i^{th} dimension. Therefore, for any one point, only $k_x k_y$ weights need to be considered, or in the case of more than 2 input variables, $\prod_{i=1}^{n} k_i$ weights are non-null.

To ensure (3.2), for just one restriction, one needs to solve just one equation. Consider that point $\mathbf{q}$ lies in the intervals I_{x_s} and I_{y_t}. The active basis functions are $i \in \left[i_s, i_{s+k_x} \right[$ and $j \in \left[j_s, j_{s+k_x} \right[$ for the inputs x and y considered.

$$\underline{y}'_x(\mathbf{z}) = \frac{\partial S_{u^i}}{\partial x} = \sum_{j=j_s}^{j_s+k_y-1} N_{k_y}^{j}(y)\left(-a_{i_s}\mathbf{w}_{i_s,j} + \sum_{i=i_s+1}^{i_s+k_x-1} (a_{i-1} - a_i)\mathbf{w}_{i,j} + a_{i_s+k_x-1}\mathbf{w}_{i_s+k_x,j} \right) \tag{3.13}$$

Therefore the dependent weight can be any of the $\mathbf{w}_{i,j}$, within those limits. Assuming, without loss of generality that the dependent weight is the first one, $\mathbf{w}_{i_s,j_s}$, which can be given as:

$$\mathbf{w}_{i_s,j_s} = \frac{\underline{y}'_x(\mathbf{z}) - \sum_{j=j_s}^{j_s+k_y-1} N_{k_y}^{j}(y)\left(-a_{i_s}\mathbf{w}_{i_s,j} + \sum_{i=i_s+1}^{i_s+k_x-1} (a_{i-1} - a_i)\hat{\mathbf{w}}_{i,j} + a_{i_s+k_x-1}\hat{\mathbf{w}}_{i_s+k_x,j} \right)}{N_{k_y}^{j_s}(y)a_1} - \frac{\sum_{i=i_s+1}^{i_s+k_x-2} (a_{i-1} - a_i)\hat{\mathbf{w}}_{i,j_s} + a_{i_s+k_x-1}\hat{\mathbf{w}}_{i_s+k_x,j_s}}{a_1} \tag{3.14}$$

If there are more sub-modules that depend on the variable x, the former equation can be updated to:

$$\mathbf{w}_{i_s,j_s} =$$

$$\frac{\underline{y}'_x(\mathbf{z}) - \sum_{j=j_s}^{j_s+k_y-1} N_{k_y}^{j}(y)\left(\begin{array}{l} -a_{i_s}\mathbf{w}_{i_s,j} + \sum_{i=i_s+1}^{i_s+k_x-1}(a_{i-1}-a_i)\hat{\mathbf{w}}_{i,j} + \\ +a_{i_s+k_x-1}\hat{\mathbf{w}}_{i_s+k_x,j} \end{array}\right)}{N_{k_y}^{j_s}(y)a_1}$$

$$-\frac{\sum_{i=i_s+1}^{i_s+k_x-1}(a_{i-1}-a_i)\hat{\mathbf{w}}_{i,j_s} + a_{i_s+k_x-1}\hat{\mathbf{w}}_{i_s+k_x,j}}{a_{i_s}} - \tag{3.15}$$

$$-\frac{\sum_{z=2}^{n_{u^i}}\sum_{j=j_s}^{j_s+k_y} N_{k_y}^{j}(y)\left(\begin{array}{l} -a_1\hat{\mathbf{w}}_{i_s,j} + \sum_{i=i_s+1}^{i_s+k_x-2}(a_{i-1}-a_i)\hat{\mathbf{w}}_{i,j} + \\ +a_{i_s+k_x-1}\hat{\mathbf{w}}_{i_s+k_x,j} \end{array}\right)}{N_{k_y}^{j_s}(y)a_1}$$

Please note that the last term in the r.h.s. of the last equation is not strictly correct. In fact, a further index should be used for the parameters associated with the other sub-modules. In order not to further complicate the notation, this additional index will not be used.

Usually, the most important restrictions are those that annul the gradient vector (totally or partially), which means that $\underline{y}'_x(\mathbf{q})$ in the r.h.s of (3.14) or (3.15) is null. Considering that there are m inputs, and the aim is to annul m_d out of these m derivatives, (3.13) can be recast in the form (3.3), where the dimension of $\mathbf{B}$ is $m_d * m_d$:

$$\mathbf{B} = \mathbf{\Gamma}_{der}(\mathbf{Q}) \tag{3.16}$$

3.3 Computing the linear weights

In order to incorporate the function and derivative equalities, the linear weights are split into two types: the independent and the dependent weights; the first are updated with no constraints, except for the purpose of target fitting, whereas the latter, must be carefully chosen as they are directly related to the input patterns subject to the restrictions.

The following section shows the required conditions to estimate the linear weights of the BSNN network. This will have a decisive role on the outline of any algorithm in this context.

3.3.1 Independent weights update

As defined in (3.4), in a B-spline Neural Network (BSNN) there is a constant number of linear weights which computation depends on the input patterns subject to restrictions.

Moreover, the output of the network will be given as:

$$y = \begin{bmatrix} \boldsymbol{\Gamma}_d & \boldsymbol{\Gamma}'_d & \boldsymbol{\Gamma}_i \end{bmatrix} . \begin{bmatrix} \mathbf{w}_d \\ \mathbf{w}'_d \\ \mathbf{w}_i \end{bmatrix} \tag{3.17}$$

where:

- $\boldsymbol{\Gamma}_d$ and $\mathbf{w}_d$ are of dimension (m, m_1) and (m_1), respectively;
- $\boldsymbol{\Gamma}'_d$ and $\mathbf{w}'_d$ are of dimension (m, m_2) and (m_2), respectively;
- $\boldsymbol{\Gamma}_i$ and $\mathbf{w}_i$ are of dimension (n, m_3) and (m_3), respectively.
- m: total number of input patterns;
- m_1: number of basis functions activated, and dependent on the restrictions;
- m_2: number of basis functions activated by, but not dependent on the restrictions;
- m_3: number of basis functions not activated by, and not dependent on the restrictions;

$\boldsymbol{\Gamma}_d$, $\boldsymbol{\Gamma}'_d$ *and* $\boldsymbol{\Gamma}_i$ contain the basis functions that are activated and dependent, activated but not dependent, and not activated, by the input patterns associated with the restrictions, respectively.

Regarding (3.17), the number of the adjustable linear weights ($\boldsymbol{w}_d$) must be enough to satisfy the restrictions determined by (3.3).

Putting it all together, (3.1) and (3.2) can be expressed in matrix form as:

$$\begin{bmatrix} \underline{y}(\mathbf{P}) \\ \underline{y}'(\mathbf{Q}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Gamma}_{d\_fun}(\mathbf{P}) & \boldsymbol{\Gamma}'_{d\_fun}(\mathbf{P}) \\ \boldsymbol{\Gamma}_{d\_der}(\mathbf{Q}) & \boldsymbol{\Gamma}'_{d\_der}(\mathbf{Q}) \end{bmatrix} . \begin{bmatrix} \mathbf{w}_d \\ \mathbf{w}'_d \end{bmatrix} \tag{3.18}$$

where:

$\boldsymbol{\Gamma}_{d\_der}$ stands for the matrix of basis functions derivatives for the m_1 dependent weights.

$\boldsymbol{\Gamma}_{d\_fun}$ stands for the matrix of basis functions outputs for the m_1 dependent weights.

$\boldsymbol{\Gamma}_{d\_der}'$ stands for the matrix of basis functions derivatives for the m_2 activated and independent weights.

$\boldsymbol{\Gamma}_{d\_fun}'$ stands for the matrix of basis functions outputs for the m_2 activated and independent weights.

This distinction is absolutely necessary in order to solve (2.17), incorporating function and derivative restrictions.

In this way, the dependent weights will become the solution of (3.3) where:

$$\mathbf{B} = \begin{bmatrix} \boldsymbol{\Gamma}_{d\_fun}(\mathbf{P}) \\ \boldsymbol{\Gamma}_{d\_der}(\mathbf{Q}) \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \underline{\mathbf{y}}(\mathbf{P}) - \boldsymbol{\Gamma}'_{d\_fun}(\mathbf{P}).\hat{\mathbf{w}}'_d \\ \underline{\mathbf{y}}'(\mathbf{Q}) - \boldsymbol{\Gamma}'_{d\_der}(\mathbf{Q}).\hat{\mathbf{w}}'_d \end{bmatrix}, \tag{3.19}$$

while the optimal values for the independent weights, $\hat{\mathbf{w}}_i$, will be obtained from minimizing the loss function, after performing adequate substitution of $\hat{\mathbf{w}}_d$ and $\hat{\mathbf{w}}'_d$:

$$\hat{\mathbf{w}}_i = \arg\left(\min \left\| \mathbf{t}_i - \boldsymbol{\Gamma}_i \mathbf{w}_i \right\|_2^2\right), \tag{3.20}$$

where $\mathbf{t}_i = \mathbf{t} - \boldsymbol{\Gamma}_d \mathbf{w}_d - \boldsymbol{\Gamma}'_d \mathbf{w}'_d$.

Alternatively, a direct way of computing the weights is by employing Lagrange multipliers [176], where the values of the dependent weights can be obtained using an optimal *constrained* estimator. In other words, (2.17) can be rewritten as:

$$\begin{bmatrix} \hat{\mathbf{w}}_{d_{constrained}} \\ \hat{\mathbf{w}}_{i_{unconstrained}} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{w}}_{d_{unconstrained}} - \mathbf{E}^{-1}\mathbf{B}^T \left(\mathbf{B}\mathbf{E}^{-1}\mathbf{B}^T\right)^{-1} \left(\mathbf{B}\hat{\mathbf{w}}_{d_{unconstrained}} - \mathbf{b}\right) \\ \hat{\mathbf{w}}_{i_{unconstrained}} \end{bmatrix} \tag{3.21}$$

In (3.21), $\mathbf{E}^{-1} = \boldsymbol{\Gamma}_d{}^T \boldsymbol{\Gamma}_d$, being $\boldsymbol{\Gamma}_d$ the column partition of $\boldsymbol{\Gamma}$ related to the dependent weights and the unconstrained linear weights are computed using (2.17).

3.3.2 Derivation of the $\boldsymbol{\Gamma}_{d\_fun}$ and $\boldsymbol{\Gamma}'_{d\_fun}$ matrices

Both $\boldsymbol{\Gamma}_{d\_fun}$ and $\boldsymbol{\Gamma}'_{d\_fun}$ are partitioned matrices, obtained from the matrix of basis functions outputs from the overall network, $\boldsymbol{\Gamma}$.

Assuming a model with n_s submodels, $\boldsymbol{\Gamma} = \bigcup_{i=1}^{n_s} \boldsymbol{\Gamma}_i$:

$$\boldsymbol{\Gamma}_{d\_fun} = \bigcup_{i=1}^{n_s} \boldsymbol{\Gamma}_i(\mathbf{P}) \tag{3.22}$$

and

$$\boldsymbol{\Gamma}'_{d\_fun} = \bigcup_{i=1}^{n_s} \boldsymbol{\Gamma}'_i(\mathbf{P}), \tag{3.23}$$

where $\boldsymbol{\Gamma}_i(\mathbf{P})$ and $\boldsymbol{\Gamma}'_i(\mathbf{P})$ stand for the partitioned basis functions output matrices, associated with the restriction input patterns $\mathbf{P}$, which compose submodel *i*.

Consider a point *p* subject to restrictions in the i^{th} submodel. As noted before, in the case of an *n*-dimensional submodel, this pattern will activate $\prod_{i=1}^{n} k_i$ basis splines in that submodel.

In order to determine these partitioned matrices, consider $\mathbf{ind}_{\mathbf{P}}^{i}$ the vector with the indices of the activated basis functions for all **P** points in the i^{th} submodel, in respect to the overall network. $\mathbf{\Gamma}_i(\mathbf{P})$ will consist of the corresponding columns in the $\mathbf{\Gamma}$ matrix for the i^{th} submodel, indicated by $\mathbf{ind}_{\mathbf{P}}^{i}$. Only of the first m_{f} indices will define $\mathbf{\Gamma}_{d\_fun}$. On the other hand, the columns of $\mathbf{\Gamma}'_i(\mathbf{P})$ will consist of the columns of $\mathbf{\Gamma}$, not used in $\mathbf{\Gamma}_i(\mathbf{P})$. Both matrices have m_{f} rows.

3.3.3 Derivation of the $\mathbf{\Gamma}_{d\_der}$ and $\mathbf{\Gamma}'_{d\_der}$ matrices

Assume again n_{s} submodels and m_{v} variables subject to derivative restrictions. In this case, the matrix of derivatives is computed using the following expression:

$$\mathbf{\Gamma}_{der} = \begin{bmatrix} \left.\frac{\partial f(\mathbf{X}_{\underline{1}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{\underline{1}}^{rest}} & \left.\frac{\partial f(\mathbf{X}_{\underline{2}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{\underline{1}}^{rest}} & \cdots & \left.\frac{\partial f(\mathbf{X}_{\underline{n_s}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{\underline{1}}^{rest}} \\ \left.\frac{\partial f(\mathbf{X}_{\underline{1}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{\underline{2}}^{rest}} & \left.\frac{\partial f(\mathbf{X}_{\underline{2}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{\underline{2}}^{rest}} & \cdots & \left.\frac{\partial f(\mathbf{X}_{\underline{n_s}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{\underline{2}}^{rest}} \\ \cdots & \cdots & & \cdots \\ \left.\frac{\partial f(\mathbf{X}_{\underline{1}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{\underline{m}}^{rest}} & \left.\frac{\partial f(\mathbf{X}_{\underline{2}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{\underline{m}}^{rest}} & \cdots & \left.\frac{\partial f(\mathbf{X}_{\underline{n_s}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=\mathbf{X}_{m_v}^{rest}} \end{bmatrix} \tag{3.24}$$

where,

$$\left.\frac{\partial f(\mathbf{X}_{\underline{i}})}{\partial \underline{\underline{x}}}\right|_{\underline{\underline{x}}=x_{\underline{i}}^{rest}} = \left[\sum_{j=1}^{r_{x_i}+k_{x_i}} N_{k_{x_i}}^{j}(\mathbf{X}_{\underline{i}}) \frac{\partial}{\partial x} N_{k_x}^{1}(\underline{\underline{x}}) \quad \sum_{j=1}^{r_{x_i}+k_{x_i}} N_{k_{x_i}}^{j}(\mathbf{X}_{\underline{i}}) \frac{\partial}{\partial x} N_{k_x}^{2}(\underline{\underline{x}}) \cdots \right. \tag{3.25}$$
$$\left. \cdots \sum_{j=1}^{r_{x_i}+k_{x_i}} N_{k_{x_i}}^{j}(\mathbf{X}_{\underline{i}}) \frac{\partial}{\partial x} N_{k_x}^{r_x+k_x-1}(\underline{\underline{x}}) \right]$$

and $x_{\underline{i}}^{rest}$ refers to the set of input patterns subject to restrictions in the i^{th} input dimension, whilst $\mathbf{X}_{\underline{i}}$ denotes the input patterns present in the i^{th} submodel.

Both $\mathbf{\Gamma}_{d\_der}$ and $\mathbf{\Gamma}'_{d\_der}$ are partitioned matrices of $\mathbf{\Gamma}_{der}$, being their columns provided by the indices vector $\mathbf{ind}_{\mathbf{P}}^{i}$, as described in the previous section. Both matrices have m_{d} rows.

3.4 Evolving the structure

In the last section, the structure of the network is assumed fixed. If the structure of the network is to be evolved, the evolving algorithm must ensure that the restrictions are met. It will be shown, afterwards, how the ASMOD [58] algorithm, and a genetic programming evolutionary based algorithm as SOGP [59] can be updated in such a way that restrictions (3.1) and (3.2) are met. They will be denoted as RBMOD and RBGEN, respectively. These will be described in the following sections.

3.4.1 Algorithms Outline

The following figure illustrates how the RBGEN and RBMOD algorithms evolve a B-spline Network. The main difference between RBGEN and SOGP lies on the linear weights estimation, which is not a straightforward procedure. Instead, it requires a two steps weights estimation (or update), due to the input patterns subject to the restrictions. Thus, it starts by determining the independent weights not related to the input restrictions and i.e., used to approximate the networks output to the target patterns. Secondly, a constant number of weights, equal to the number of restrictions imposed, must be computed and then updated in the network.

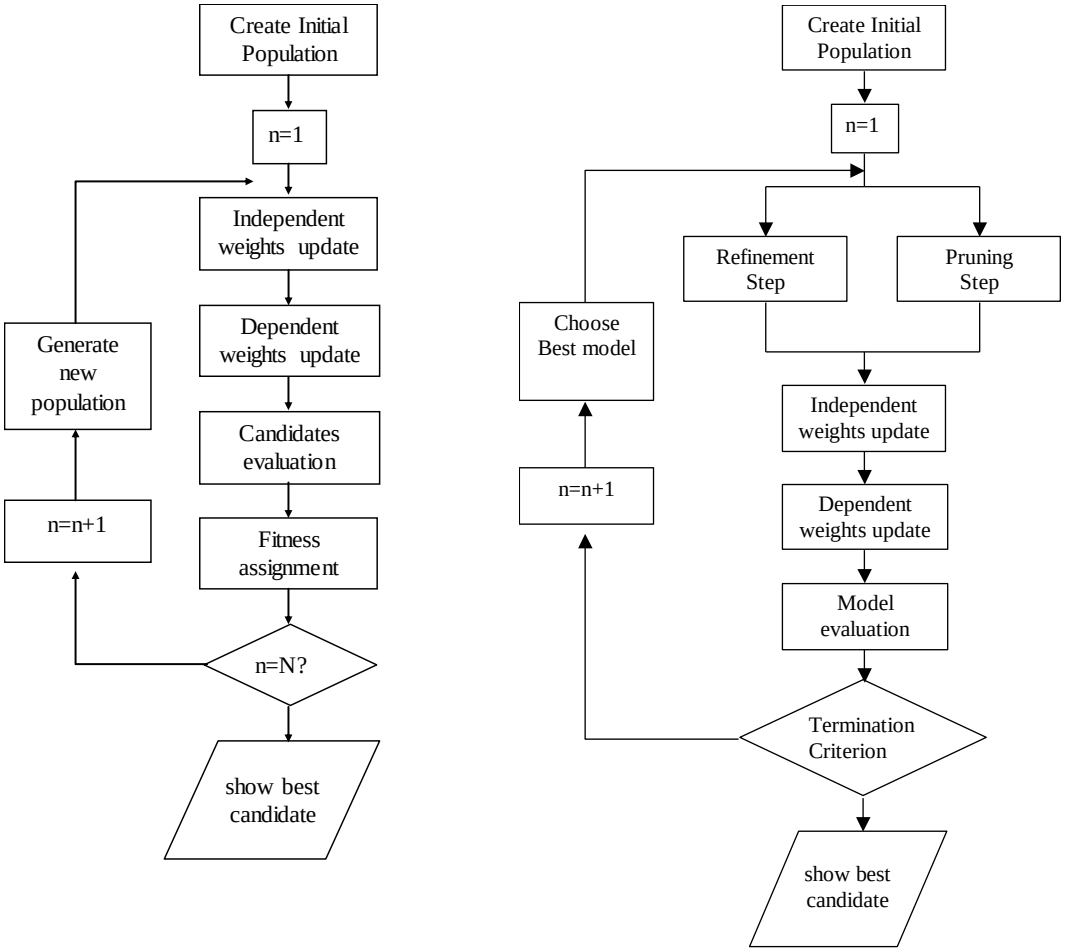
Moreover, the genetic programming operators and population creation procedures have to be changed, as their outcome could be an invalid network structure.

In the case of the RBMOD algorithm, the creation of the initial model structure is of main importance. In the case of the ASMOD algorithm, one builds an initial structure in which a trade-off between the restrictions and the complexity of the model is assumed. This means that the initial models' structure is dependent not only on the spline functions order but also on the number of the restrictions imposed. The number of interior knots must be minimal and sufficient.

As is usual with the ASMOD algorithm, the initial model starts with univariate submodels, with variables of order 3 and zero interior knots.

In this case, additional knots are equally spaced among the restriction patterns. The first submodel's complexity might be greater than the other submodels complexity if there are function restrictions, because it is on the first submodel that the restrictions are accounted for.

The insertion of the restrictions on the first sub-module is arbitrary and, to the best of the author's knowledge, the simplest way of satisfying the restrictions.



a) b)

Fig. 3.1. Flowcharts for a) the RBGEN Algorithm; b) the RBMOD algorithm

Common updates to both algorithms are described below.

3.4.2 Initial model creation

In the presence of function or derivative restrictions, the initial model structure must satisfy some features.

- if there are function restrictions, all input variables must be present in the model;
- For the sake of simplicity, the first variable in the model supplies the necessary basis functions that satisfy the function restrictions;
- in the case of derivative restrictions, each submodel's variable subject to this kind of restrictions, must supply a sufficient amount of basis functions, possibly requiring the addition of interior knots;

- A given variable subject to derivative restrictions, may not be subject to knots addition because, there exists another submodel with the same variable;
- The operation of adding knots requires that each extra knot be placed among (1 knot if the amount to insert is smaller or equal to the number of restriction patterns in the variable, 2 knots otherwise) the restriction patterns. However, the number of extra knots needs not to be greater than the spline order among each pair of restriction patterns (see Fig. 3.2);

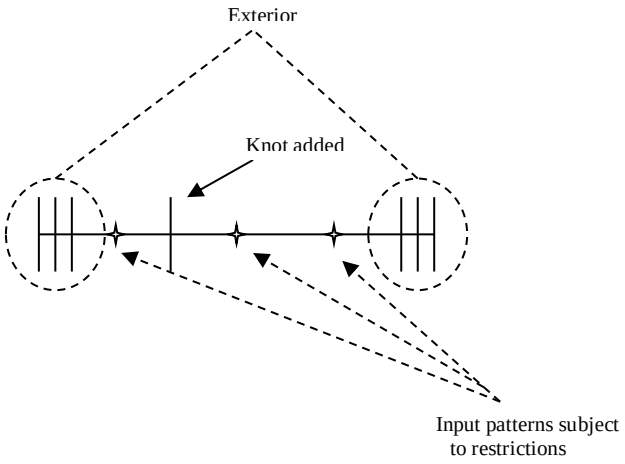


Fig. 3.2. Example of knot addition

- An optimized model structure satisfies the number of restrictions as long as the number of input restriction patterns contained in two adjacent interior knots is not greater than the spline order;
- The extra number of interior knots required is computed from:

N_knots=N_FunctionRestrictions+N_DerivativeRestrictions-OrderOfSpline

3.4.3 Structure pruning

Because this phase aims to simplify the model's structure, it may occur that not every restriction is fulfilled. Therefore, there are some considerations to be accounted for:

3.4.3.1 Variable order reduction

- *Neither function nor derivative restrictions*: the variable order may be reduced or the variable can be removed.
- *Derivative but no function restrictions* or *Derivative and function restrictions*: variable order can be reduced as long as its order is greater than 2.
- *Function but no derivative restrictions*: variable order may be freely reduced, but not removed.

3.4.3.2 Splitting multi-variable submodels.

This procedure has no limitation, because the initial model consisting of univariate submodels only, always corresponds to an adequate structure.

3.4.3.3 Reducing the number of interior knots.

The restrictions are satisfied as long as the number of interior knots generates as many basis functions as the number of restrictions; however, no more than (*OrderOfSpline-1)* input patterns subject to restrictions can be among two adjacent knots, should derivative restrictions exist. See Fig. 3.3 where the right most interior knot is not prone for removal since the splines are quadratic (*OrderOfSpline*=3).

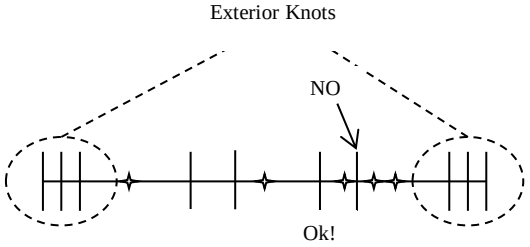


Fig. 3.3. Example of knot removal restrictions

3.4.4 Implications on the genetic operators

Due to the restrictions above listed there is a real chance that an invalid model is obtained from an expression tree. An invalid model is a model which does not comply with the equality restrictions. The way found to avoid such cases is explained next.

3.4.4.1 Initial population creation

i. The creation of the population is done based on the "full" method. This way:

- the model's structure tree representation to contain all the variables is essential to comply with the restrictions set;
- a certain boundary on the model's complexity is established to avoid an overload of multi-variable sub-models.

ii. In order to generate initial low complexity models, the length of the candidates' expression tree is given by $tree\_len = \log_2(n) + 2$, for n inputs.

iii. If the models' evaluation corresponds to an overhead on the complexity, a redefinition of its inner nodes is made, replacing the product function by the sum function, from bottom to top.

iv. If two identical submodels (same variables) are chosen for addition, the output submodel consists of the submodel with the largest number of interior knots.

3.4.4.2 Crossover

Two parents structures are chosen according to the selection algorithm and two associated nodes are randomly selected. Then, the offspring structure is evaluated in order to validate its compatibility with all the restrictions. In case the structure is found to be inappropriate, the crossover procedure is repeated as much as three times, for each pair of offspring. If the structure is still inadequate, the original parents structure is chosen as the final offsprings.

3.4.4.3 Mutation

Mutation is performed in both interior and exterior nodes:

3.4.4.4 External nodes.

Mutation on an external node can be of 5 different types:

Variable replacement: if the variable is not unique, it may be replaced by a new one. The uniqueness property is of main importance, because its presence may ensure the validation of the model's structure, if there are restrictions set.

Variable's order replacement: it is performed at the chosen node, unbounded, as long as the variable is not subject to any kind of restrictions. In case there are restrictions, the new order is randomly chosen from $\{2,3\}$.

Interior knot re-allocation: an interior knot is shifted inside the knot vector, in two possible ways. On one hand, if the selected knot is below the lowest input restriction pattern or above the highest input restriction pattern, it will be re-allocated a random fraction between the two adjacent knots. On the other hand, if the chosen knot is dependent on the input restriction patterns, it is re-allocated a random fraction between the former adjacent input restriction patterns.

Interior knots addition: The same strategy as used in SOGP design is applied, except for the knots position, which is chosen among selected input restrictions.

Interior knots removal: a randomly generated subset of knots scheduled for removal is obtained from the interior knots that do not depend on the restriction patterns (those knots fundamental to ensure the restrictions).

3.4.4.5 Internal nodes

This operation is similar to the function mutation in SOGP. A randomly chosen node in the subtree is replaced by one contained in the set of functions, formed by a validated submodel.

3.5 Simulation results

3.5.1 Comparison between RBMOD and RBGEN

In order to show results, in a graphical manner, a problem composed of 2 input variables was selected. The 2-dimensional function is defined as $f(x,y) = x^2y^2 - xy$, and contains infinite global optima for $f(x,y) = -\frac{1}{4}\Big|_{y=\frac{1}{2x}}$. The input data was generated within the interval [-1,1], with a discretization step of 0.125.

The restrictions were imposed on the patterns $(x_0, y_0) = \{(-1, 0.5), (-0.5, -1.0), (0.5, 1.0), (1, 0.5)\}$. Both function and derivative restrictions were set on the former patterns, for every input variable, such that $\frac{\partial \mathbf{y}(\mathbf{X})}{\partial \mathbf{x}_i} = \underline{\mathbf{0}}$ and $\mathbf{y}(\mathbf{X}_i) = \mathbf{t}_i$.

For RBGEN, the terminal type mutation rate is: [5%, 5%, 20%, 40%, 30%].

The next table shows the parameters definition for the RBGEN algorithm.

The results summarized in Table 3.2 correspond to one run from each of the algorithms. Fig. 3.4 illustrates the 3D plot of the output of the BSNN model for the first and for the final iterations, for RBMOD.

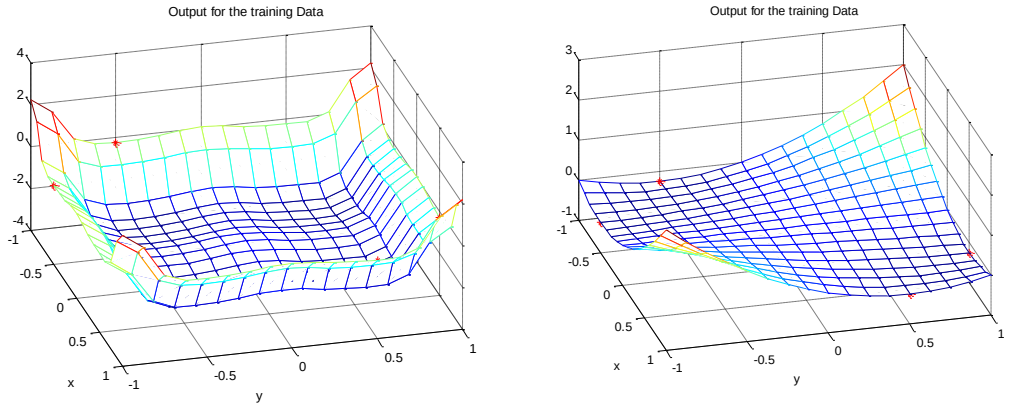
TABLE 3.1: PARAMETERS DEFINITION FOR THE RBGEN ALGORITHM.

Parameters	N_ind	N_ger	Crossover Rate	Mutation Rate
N_ind	10	10	50%	0.8

As can be seen from Table 3.2, there is a perfect match in terms of function approximation (MSE value of zero, in pratical terms). Besides, the RBGEN algorithm is able to generating a model of lower complexity than the one achieved by RBMOD.

TABLE 3.2: FINAL VALUES FOR THE BEST CANDIDATE.

Algorithm	BIC	‖**w**‖	MSE	Number Candidates	Model Complex.	Submodels
RBMOD	-17674	27.8	3.3x10-28	110	108	1+2+[1x2]
RBGEN	-19595	11.3	1.3x10-30	100	52	[1x2]



a) First iteration b) Last iteration

Fig. 3.4. The output of the B-Spline Network for the test function and the best candidate, using RBMOD. The red '*' points indicate the specified optima of the function.

On the other hand, it can be verified in Fig. 3.4 that although starting from a poor initial solution, it is possible reach a perfect final solution. Of course, this fact is achieved by the increase of the interior knots which provided a broader margin for data fitting, showing the correct function approximation and maintaining the restrictions required.

3.5.2 Comparison between RBGEN and SOGP

This section provides a performance comparison between the Genetic Programming Algorithm incorporating restrictions, and, when, no restrictions are imposed. For the latter, SOGP is applied. The examples consist of two benchmarks very commonly used in function optimization problems [177], designated as the *fake banana* function and the *paraboloid* function, which will be formulated subsequently. Average results, concerning 10 (ten) different runs will be shown. The mean training criterion (i.e., the Bayesian Information Criterion) and the mean percentage of mean relative error for the validation data (%MREv), will be shown, as well as mean values for the network complexity, MSE, MSRE and %MRE for the training and validation data.

The *fake banana* function is described as

$$f(x,y) = 100 \times (y - x^2)^2 + (\tfrac{1}{2} - x)^2 . \tag{3.26}$$

The *paraboloid* function is described as

$$f(x,y) = x^4 + 2x^2(y^2 - 1) + (y^2 - 1)^2 \tag{3.27}$$

Equation (3.26) has a global minimum at $(x,y) = (\tfrac{1}{2}, \tfrac{1}{4})$ and equation (3.27) has infinite global minima located at $(x,y) = (x_0, \pm\sqrt{1 - x_0^2})$.

The following two figures show the surfaces concerning the functions used.

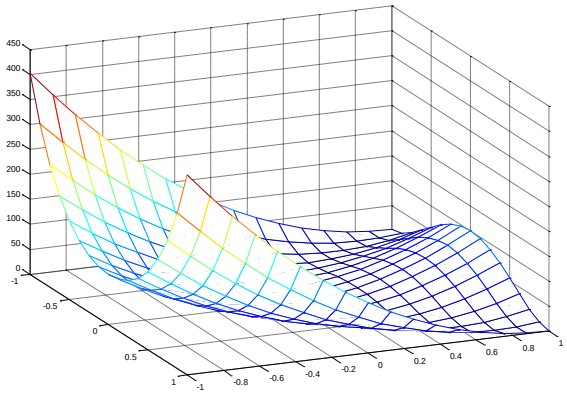


Fig. 3.5. Surface for the *fake banana* function

For the *fake banana* function, $(x_0, y_0) = (\tfrac{1}{2}, \tfrac{1}{4})$ is the input pattern subject to restrictions, clearly the one and only global optimum as observed in Fig. 3.5.

The function and derivative restrictions set is,

$$f(x_0, y_0) = 0, \ \frac{\partial f(x_0, y_0)}{\partial x_0} = 0, \ \frac{\partial f(x_0, y_0)}{\partial y_0} = 0 \tag{3.28}$$

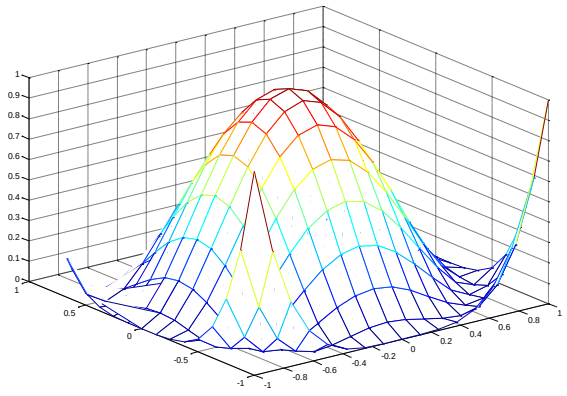


Fig. 3.6. Surface for the *paraboloid* function

For the *paraboloid* function, there are 5 input patterns subject to restrictions:

$$(\mathbf{x_0}, \mathbf{y_0}) = \begin{bmatrix} -\frac{7}{8} & \frac{930}{1921} \\ -\frac{3}{4} & \frac{506}{765} \\ \frac{3}{4} & \frac{506}{765} \\ \frac{7}{8} & \frac{930}{1921} \end{bmatrix} \text{ with restrictions } f(\mathbf{x_0}, \mathbf{y_0}) = \mathbf{0},$$

$$\frac{\partial f(\mathbf{x_0}, \mathbf{y_0})}{\partial \mathbf{x_0}} = \mathbf{0}, \ \frac{\partial f(\mathbf{x_0}, \mathbf{y_0})}{\partial \mathbf{y_0}} = \mathbf{0} \tag{3.29}$$

Fig. 3.6 confirms that the *paraboloid* function is symmetric in respect to the input axis and that there are infinite global minimas.

Note: The derivative restrictions chosen are valid because these input patterns are actually stationary points.

In both problems, the training data set consists of 289 patterns, which lie on a regular grid with input values drawn from $f(x,y): \Re^2 \to \Re, \ x \in [-1+0.125i, 1]\big|_{i=0,1,\ldots}, y = x$.

The validation data set consists of 169 patterns, generated in a way that the input values are drawn from $f(x,y): \Re^2 \to \Re, \ x \in [-0.8+0.125i, 0.8]\big|_{i=0,1,\ldots}, y = x$.

The next table illustrates the performance of the two algorithms when the target function is the *fake banana* function. Table 3.4 shows the results, in average terms, if the target function is the *paraboloid*.

Both tables provide similar conclusions i.e., RBGEN always provides the best generalization capabilities as it can be seen by the lower values shown by the validation criteria. This fact is observed despite the higher neural network complexity returned.

TABLE 3.3: MEAN VALUES FOR MSE, MSRE, %MRE FOR TRAINING AND VALIDATION (V SUBSCRIPT) FOR THE *FAKE BANANA* FUNCTION

Algorithm	MSE	MSRE	%MRE	MSEv	MSREv	%MREv	Complexity
SOGP	3,9	16	20,6	$2{,}6\text{x}10^{10}$	$9{,}4\ \text{x}10^{9}$	$1{,}5\ \text{x}10^{6}$	81,4
RBGEN	$2{,}8\text{x}10^{-3}$	$3{,}4\text{x}10^{-4}$	$1{,}5\text{x}10^{-1}$	$58{,}6\text{x}10^{3}$	$15{,}8\text{x}10^{3}$	703,5	229,4

TABLE 3.4: MEAN VALUES FOR MSE, MSRE, %MRE FOR TRAINING AND VALIDATION (V SUBSCRIPT) FOR THE PARABOLOID FUNCTION

Algorithm	MSE	MSRE	%MRE	MSEv	MSREv	%MREv	Complexity
SOGP	$7{,}3\ \text{x}10^{-3}$	$11{,}6\ \text{x}10^{3}$	886,3	1,6	$1{,}4\ \text{x}10^{6}$	$3{,}6\ \text{x}10^{3}$	114,8
RBGEN	$5{,}4\text{x}10^{-5}$	12,5	53,8	2,4	$23{,}7\ \text{x}10^{3}$	643,7	184,9

3.6 Conclusions

Previous experiments have shown that the performance of neural networks, assessed on its fitting and ability to generalize, even using strategies such as cross validation can, in general, be improved. If extra information about the training data is known, it is helpful to present the network with this a-priori information, allowing the network structure to maintain its approximation and generalization ability, showing a better overall performance.

This chapter has demonstrated how a priori knowledge of the location of the minima of the function to be approximated, or an exact model response for specific inputs, can be accommodated in B-Splines networks, for a fixed structure or using methods that evolve the structure. The results presented here are directly applicable to Mamdani fuzzy models and to Takagi- Sugeno type fuzzy models, satisfying the assumptions described in Chapter 2, Section 2.4.

In the next chapter an alternative evolutionary algorithm is developed to train the BSNN model.

Chapter 4

BACTERIAL PROGRAMMING FOR B-SPLINES DESIGN

4.1 Introduction

Previous work introduced new structured optimization approaches based on genetic programming for the B-spline neural network design [59] and, the use of the bacterial evolutionary algorithm for rule extraction [131]. In [130] these approaches were compared and showed similar performances from the point of view of the black-box identification. Still, they were employed to different model architectures. Though GP can be considered to obtain better results, it was considered of interest to combine benefits from both approaches.

This chapter which is an extended version of [59], [178] and [179], introduces a new technique that combines the Bacterial Algorithm and the Genetic Programming concepts. This technique is called Bacterial Programming (BPA) and is ruled by principles based on the replication of the microbial evolution phenomenon. The performance of this approach is illustrated and compared with existing alternatives. It applies the bacterial operators instead of the original genetic operators. This way, while the bacterial mutation is working on one individual, and tries to optimize this bacterium, the gene transfer is applied to the whole bacteria population, avoiding the local minima solutions. If more clones are being applied in the bacterial mutation, then better results are obtained. The results show that the performance of the algorithm is at least as good as those obtained with GP, or, in the case of the results presented here, slightly better. Also, BPA tuning is much easier than tuning the GP, particularly if different sets of data are used.

This chapter is organized as follows.

Section 4.2 describes the evolution process for the Bacterial Programming Algorithm (BPA). It starts by giving an outline on the BPA algorithm. Then, the coding of this technique is described for the case of the BSNN network, in section 4.3. In this way, a detailed

description on the bacterial operators and bacterial evaluation is given. Results are shown in section 4.4 which is divided in three subsections; firstly, experiments with one problem are conducted to help decide which values should be used for the control parameters; secondly, the performance of the new method is compared with GP for 3 distinct problems; lastly, some statistical tests are presented which show the consistency of the method. Conclusions are drawn in section 4.5.

4.2 The evolutionary process

As noted in chapter 2, section 2.5.2.3.1, designing B-spline neural networks means defining a structure which avoids the *curse of dimensionality* concept. This process can be related to the combination of low dimensional submodels, rather than using a model composed of every input variable. In contrast to the application of GP to other neural networks, for BSNN the node terminals do not represent only the *input variable*, but also the *spline order*, the *number of interior knots*, and *their locations*.

As noted in Chapter 2, Section 2.5.1.4, the Bacterial Evolutionary Algorithm (BEA) introduced a new operator called gene transfer operator. While Pseudo-Bacterial-Genetic-Algorithm (PBGA) [125] incorporated bacterial mutation and crossover operators, the BEA substitutes the classical crossover with the gene transfer operation. Both of these new operators were inspired by bacterial evolution. Bacteria can transfer genes to other bacteria. The bacterial mutation performs local optimization whilst the gene transfer allows the bacteria to directly transfer information to the other individuals in the population.

Based on these bacterial operations and using the tree structures like in the Genetic Programming, the evolution process of BPA involves the following steps:

Algorithm 4.1. BPA algorithm

1. The creation of an initial population, and the determination of the size of the population.
2. Application of bacterial mutation to each bacterium.
3. Application of gene transfer operation to the current population.
4. If the terminal criterion is achieved, the algorithm stops, otherwise it continues from step 2.

The termination criterion is usually the maximum number of generations.

The cycle of evolution is schematized in Fig. 4.1.

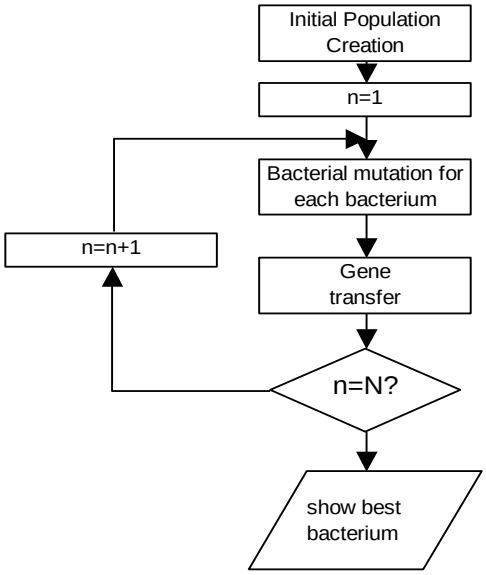


Fig. 4.1. Outline of Bacterial Programming.

To setup the algorithm, an initial population must be created. As this step is problem dependent, a good formulation of the bacteria is required. This is done through encoding. Thus, the next section explains how this is performed.

4.3 The encoding method

Bacterial programming employs the same operators that a bacterial algorithm uses in its search procedure. However, this approach is much useful for this type of neural networks because, instead of encoding the network parameters in bit strings, it uses a tree structure, composed of *function* and *terminal* nodes where one bacterium is represented by one such expression tree. This tree structure, as well as the characteristics of the nodes, evolves from generation to generation.

The function nodes in the BPA algorithm are the same as those in the genetic programming methodology: $F = \{+, *, /\}$.

Also, the same primitive functions are used and the same procedures must be applied when a bacterium is evaluated, or when if it is undergoing bacterial mutation or gene transfer (see section 2.5.1.2.3 in Chapter 2).

4.3.1 Bacterial mutation

Bacterial mutation operation is applied to each bacterium one by one.

First, N_{clones} copies (clones) of the bacterium are generated. Then, a certain part of the bacterium is randomly selected and the parameters of this selected part are randomly changed in each clone (mutation).

In the new method, coding is given by an expression tree which consists of terminal and function nodes. For this reason, mutation can be of two types: function and terminal parts.

Next, all the clones and the original bacterium are evaluated by an error criterion. The error criterion used is (2.117). The best individual transfers the mutated part into the other individuals. This cycle is repeated for the remaining parts until all of the parts of the bacterium have been mutated and tested. At the end, the best bacterium is kept and the remaining N_{clones} are discarded. By use of this operation, the bacteria will be at least as good as before, but in most of the cases it will be better.

The following figures illustrate the mutation procedure. In Fig. 4.2, it can be seen that after a function mutation on a node, all the sub-tree beneath is replaced by a new randomly generated one, which is obtained using the "grow" method. However, terminal mutation affects only the given node, as shown in Fig. 4.3.

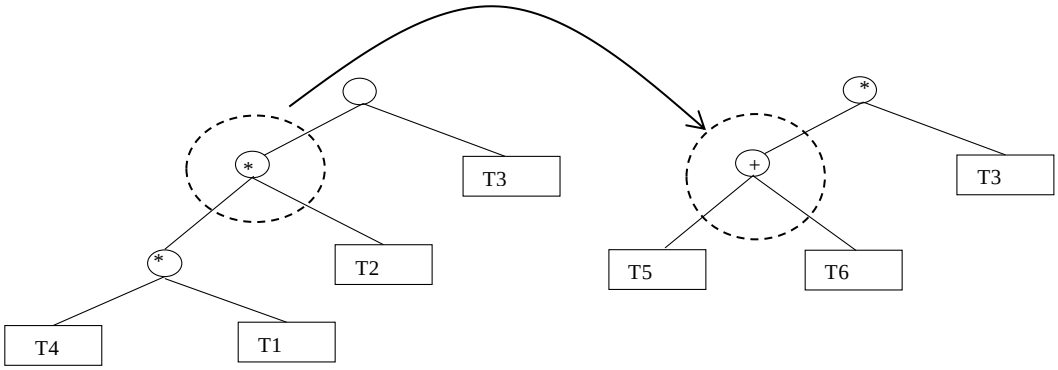


Fig. 4.2. Mutation on a function part: the individual's selected node sub-tree is changed randomly.

Still, the algorithm has to choose between terminal and function node mutation. Regarding experiments taken it was observed that a good choice was to apply the function node mutation more times. This way, function mutation occurs in 70% of the cases in average.

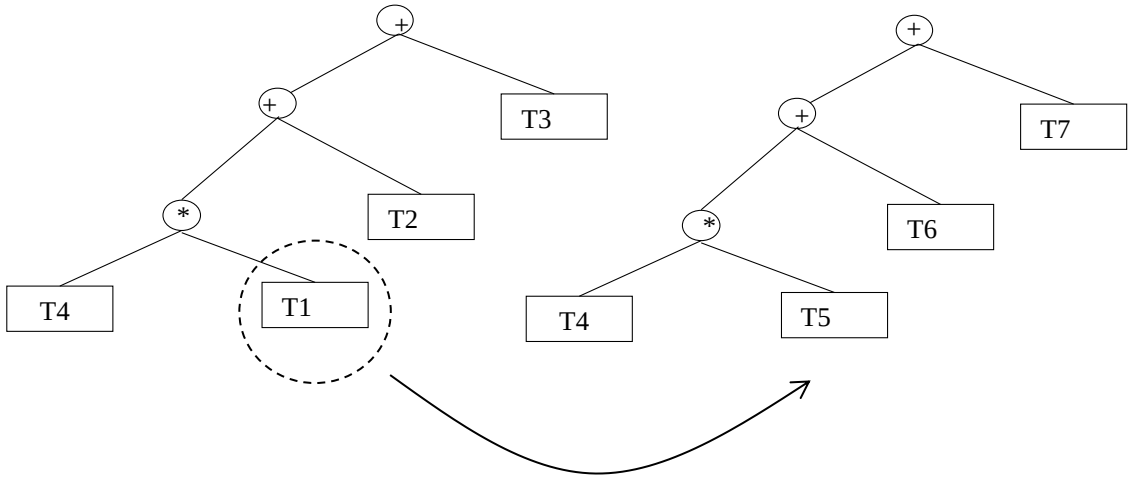


Fig. 4.3. Mutation on a terminal part: only the selected node is changed randomly given the terminal mutation rates.

Terminal mutation is divided into 6 types. These are:

- Submodel replacement. With this operation all information in a terminal node is replaced by a new one.
- Variable Identification replacement.
- Variable order replacement.
- Reallocation of one interior knot by a fraction.
- Addition of interior knots.
- Reduction of the number of interior knots.

A random choice is typically used, for each one of the 6 types. This way, a mutation rate vector is used:

$$t\_mut\_rate = [\%1, \%2, \%3, \%4, \%5, \%6] \tag{4.1}$$

As an example, if $t\_mut\_rate = [0.1, 0.2, 0.2, 0.3, 0.1, 0.1]$ reallocation of one interior knot has a higher chance of being applied.

In bacterial programming, mutation is applied once to the selected part, meaning that neither the selected nodes nor the sub-tree in case of function mutation will be chosen again, in the same generation.

4.3.2 Gene transfer

The aim of the gene transfer operation is to exchange genetic information between two bacteria. The basic procedure of the gene transfer was described in section 2.5.1.4. This procedure is similar to the crossover operation used in genetic programming except for the

fact that the concept of parents and offsprings is not used. Here, one bacterium is selected randomly from the better half of the population (better fitness bacteria) and will infect another bacterium by overwriting a part of its genetic material.

This procedure is repeated for N_{inf} times, where N_{inf} is the number of "infections", in every generation. Since the destination bacterium represents the worst part of the population, by accepting phenotypes from the better part, this operation leads to better solutions. The gene transfer operation is illustrated in Fig. 4.4.

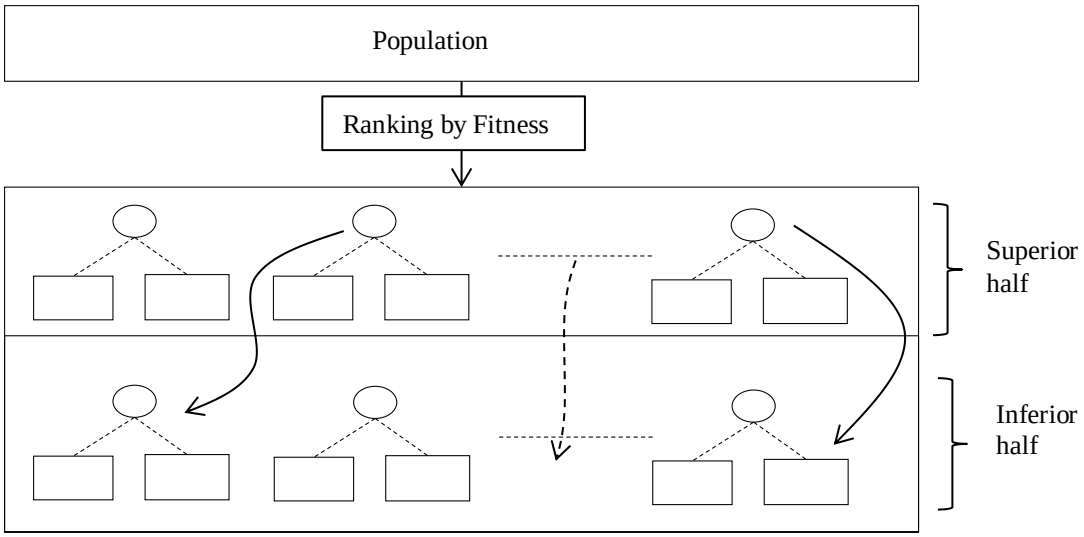


Fig. 4.4. The gene transfer procedure.

4.3.3 Bacterium evaluation

Different criteria could be applied to evaluate the bacteria. One could use *RMS* (Root-Mean-Square) in the training set, or *Cross-Validation*; but the most usual criteria are however (in the single objective approach), *information criteria*, which balance the accuracy obtained by the model against the model complexity.

The Bayesian Information Criterion (2.117) is used in BPA where m denotes the number of training samples and n_z the model complexity (number of basis functions). The measure of modeling error is MSE.

4.4 Simulation results

In this section three problems are used in order to illustrate the strength of this method. The problems in discussion are the pH problem, Inverse Coordinate Transformation and a six dimensional generic function, all described in detail in Appendix A.

An evolutionary computational algorithm needs to be fine-tuned. There are different techniques for the steps described earlier (and also different parameters) that should be

selected for the particular application. A preliminary experimental study was conducted and the main conclusions are summarized here:

- As it is often the case, the larger the size of the population, and the number of generations employed, the better are the results obtained.
- The parameter of bacterial mutation is the number of clones (N_{clones}). The larger the N_{clones}, the more effective is the bacterial mutation.
- From all the terminal mutations referred above, the one related with knots addition appears to be the most important.

4.4.1 BPA parameters values choice

This section presents results used to help deciding on the best values for the BPA parameters. The parameters that influence the BPA performance are the number of individuals (N_{ind}), number of clones (N_{clones}), number of infections (N_{inf}), and number of generations (N_{gen}). The training patterns used belong to the pH problem, only.

These results do not study the role of the number of knots in the algorithm performance. Ten different sessions were executed. For a description on the evaluation criteria see chapter 2, section 2.5.Both the number of individuals, and generations are 20.

Firstly, the number of clones was taken from {5,10,15}, using a number of infections of 5.

The results presented in Table 4.1 show that, in general, better accuracy is accomplished as the number of clones increases. This is translated into more computation as well.

Secondly, the number of infections was adjusted. It was set to 5, 10 and 15 infections, using a number of 8 clones. The results are presented in Table 4.2.

Observing the values in Table 4.2 it can be concluded that the number of infections does not affect the performance as much as the number of clones. More infections may lead the population into local optima because of the premature convergence. A low value for N_{inf} may provide better result in general, and it needs less computation.

TABLE 4.1: MEAN VALUES FOR BIC, MSE, MSRE, PMRE AND COMPLEXITY ADJUSTING PARAMETER N_{CLONES}.

N_{clones}	5	10	15
BIC	-1695.9	-1834.3	-1913
MSE	3.6×10^{-8}	6.9×10^{-9}	2.3×10^{-9}
MSRE	5.8×10^{-2}	2.9×10^{-3}	1.2×10^{-3}
PMRE	1.2	2.7×10^{-1}	1.7×10^{-1}
Complexity	57.8	73	81

TABLE 4.2: MEAN VALUES FOR BIC, MSE, MSRE, PMRE AND COMPLEXITY ADJUSTING PARAMETER N_{INF}

N_{inf}	5	10	15
BIC	-1823.9	-1792.8	-1934.3
MSE	8.4×10^{-9}	4.2×10^{-8}	2.2×10^{-9}
MSRE	7.1×10^{-3}	7.4×10^{-2}	3.4×10^{-3}
PMRE	4.1×10^{-1}	9.1×10^{-1}	3.3×10^{-1}
Complexity	75.9	76.2	87.4

4.4.2 Comparison between BPA and GP

As in the former subsection, 10 sessions were executed and the mean values for the BIC, MSE, MSRE and PMRE were obtained. The number of patterns used is 101, 110, and 200 for the pH, ICT and the six dimensional generic function problem, respectively.

In order to obtain a similar computational complexity by the two algorithms, the values for the parameters used are as follows:

TABLE 4.3: PARAMETERS DEFINITION FOR BOTH ALGORITHMS.

Parameters	GP	BPA
N_{inf}	-	5
N_{clones}	-	8
N_{ind}	160	20
N_{gen}	20	20
Crossover rate	50% of population	-
Mutation rate	0.8	-

The terminal type mutation rate is [5%, 10%, 5%, 10%, 60%, 10%], for both algorithms.

From the values used for the parameters, it can be seen that the size of the population in the BPA is much smaller. However, setting a number of 8 clones gives similar computational complexity because in the bacterial mutation there are 8 clones for each bacterium. One advantage of the BPA approach is that it does not need a large population; the evolution of only 20 bacteria is enough to compare the methods.

4.4.2.1 pH problem

Table 4.4 illustrates the mean values obtained for the pH problem. In this case, the results of GP and BPA seem to be similar. However, the complexity is lower in the BPA case. In

Table 4.5 the model structure for the best individual is also shown in the case of BPA and GP. From this result it can be diagnosed that both algorithms show similar final values.

TABLE 4.4: MEAN VALUES FOR MSE, MSRE, PMRE AND COMPLEXITY OBTAINED FOR THE PH PROBLEM.

	GP	BPA
BIC	-1784.6	-1786.7
MSE	1.1×10^{-8}	1.3×10^{-8}
MSRE	1.3×10^{-2}	2.6×10^{-2}
PMRE	6.1×10^{-1}	6.9×10^{-1}
Complexity	82.6	72.4

TABLE 4.5: MODEL STRUCTURE FOR THE LOWEST BIC VALUE FOUND AFTER ALL SESSIONS FOR THE PH PROBLEM.

	GP	BPA
Submodels	(1)	(1)
Complexity	33	40
BIC	-1903.1	-1874.3
MSE	1.4×10^{-9}	1.8×10^{-9}
MSRE	2.7×10^{-3}	7.5×10^{-5}
PMRE	5.3×10^{-1}	1.0×10^{-1}
||W||	3.3	3.6

4.4.2.2 Inverse Coordinate Transformation problem

Results for the ICT problem can be seen in Table 4.6 and Table 4.7. BPA gives better results not only in average terms but also if the best individual (the individual with the lowest BIC) is considered. Although GP shows lower values for the best individuals regarding the relative errors, the evolution processes are driven by the BIC criterion (which depends on the MSE) both for GP and BP. Thus, the BIC and MSE criteria are more important. According to these criteria the BPA method gives the best results.

TABLE 4.6: MEAN VALUES FOR MSE, MSRE, PMRE AND COMPLEXITY OBTAINED FOR THE ICT PROBLEM.

	GP	BPA
BIC	-1344.4	-1539.7
MSE	2.5×10-7	8.6×10-8
MSRE	1.6×105	2.1×103
PMRE	1331.6	125.8
Complexity	33.4	31.7

TABLE 4.7: MODEL STRUCTURE FOR THE LOWEST BIC VALUE FOUND AFTER ALL SESSIONS FOR THE ICT PROBLEM.

	GP	BPA
Submodels	(1x2)+(1)+(2)	(2x1)+(1)
Complexity	106	106
BIC	-1533.9	-2048.3
MSE	1.3×10^{-8}	8.8×10^{-11}
MSRE	1.6×10^{-7}	22.45
PMRE	1.3×10^{-2}	79.7
‖W‖	686.7	2.6×10^{7}

4.4.2.3 Six dimensional generic function

The main advantage of the bacterial approach can be deduced from the results of the six dimensional problem. The results illustrated in Table 4.8 and Table 4.9 show that the bacterial method achieves a better performance for problems with larger input dimension.

TABLE 4.8: MEAN VALUES FOR MSE, MSRE, PMRE AND COMPLEXITY OBTAINED FOR THE SIX DIMENSIONAL GENERIC FUNCTION PROBLEM.

	GP	BPA
BIC	-380.1	-552.9
MSE	2.0×10^{-1}	2.7×10^{-2}
MSRE	2.1×10^{-3}	5.2×10^{-4}
PMRE	2.1	0.98
Complexity	38.8	44.6

Fig. 4.5 and Fig. 4.6 show the desired output (target) and the error vector, obtained by the best individual for GP and BPA, respectively. Comparing these graphs it can be concluded that the BPA gives the smallest error.

TABLE 4.9: MODEL STRUCTURE FOR THE LOWEST BIC VALUE FOUND AFTER ALL SESSIONS FOR THE SIX DIMENSIONAL GENERIC FUNCTION PROBLEM.

	GP	BPA
Submodels	(5)+(4)+(2)+(3x4)+(5x3x6)+(3x1)	(6x5)+(5x2)+(6x1)+(3x6)+(3x4)+(1)
Complexity	98	156
BIC	-593.2	-702
MSE	3.8×10^{-3}	4.8×10^{-4}
MSRE	7.3×10^{-5}	1.2×10^{-5}
PMRE	6.6×10^{-1}	2.4×10^{-1}
‖W‖	423.8	128.4

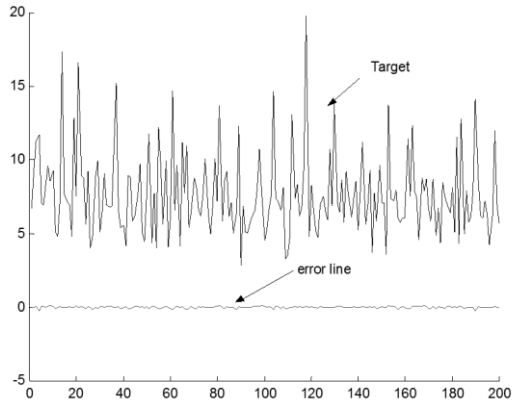


Fig. 4.5. Target output and error vector for the six dimensional generic function problem, using GP.

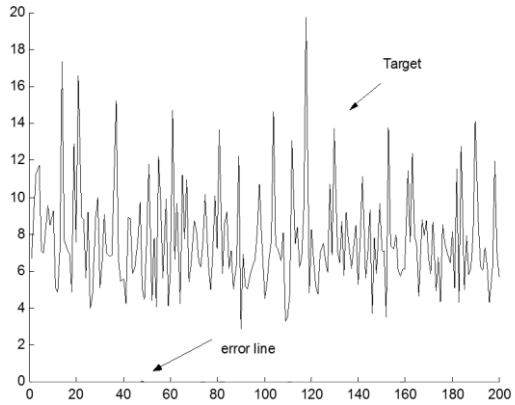


Fig. 4.6. Target output and error vector for the six dimensional generic function problem, using BPA.

4.4.3 Statistical approach

The previous simulations showed that the Bacterial Programming is more efficient than the Genetic Programming. The reason for that is the different nature of the operations in the BPA approach. Bacteria can explore bigger part of the search space because of the interplay of the clones in the bacterial mutation. In this subsection, statistical tests are used and their results are presented as shown in Table 4.10.

A significance level for rejection of $\alpha = 5\%$ was used, giving a 95% rate of confidence on the null hypotheses.

To assess the equality of solutions, the most-popular two sample method was applied (the Mann-Whitney test [180]). This test supplies the *p*-value which represents the probability of obtaining equal valued samples drawn from two different algorithms. Also, to ascertain whether both algorithms have equal medians, the location method known as the Median test was conducted. Given the number of runs and the cumulative sum of ranks from samples

from one of the algorithms, the Median test provides a judgement on the probability of having different, smaller or bigger medians between two populations.

TABLE 4.10: STATISTICAL INFERENCE OBTAINED FOR THE BPA AND GP USING BOTH THE MANN-WHITNEY AND THE MEDIAN TEST METHODS

Problem	Mann-Whitney test (p-value)	Median test
pH	0.7624	Different medians
ICT	0.0156	Different medians
Six dimensional generic function	0.00194	Lower Median for BPA than for GP

The results shown in Table 4.10 indicate how similar the performance is for both algorithms when using the one dimensional pH problem, since the *p*-value is high. When applying multi-variable problems the probability of obtaining similar results is decreased, the p-value being lower than the 5% rejection boundary. Moreover, it happens that the six dimensional problem presents not only a different median but also a lower median value, which means that in most of the times a better evaluation criterion will be obtained with BPA.

Figures Fig. 4.7-Fig. 4.9 show the empirical probability distribution functions for each of the problems, using both algorithms.

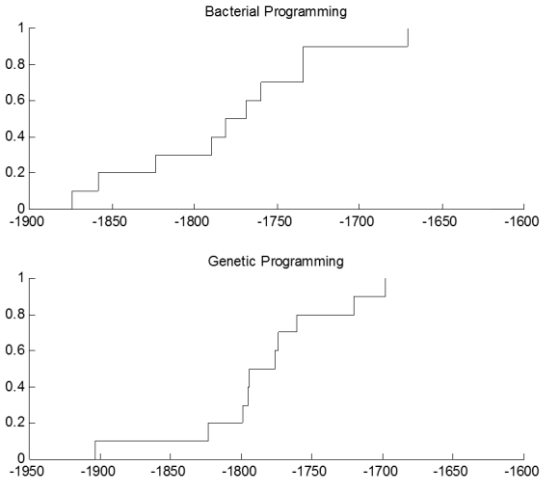


Fig. 4.7. Empirical probability distribution function for the pH problem

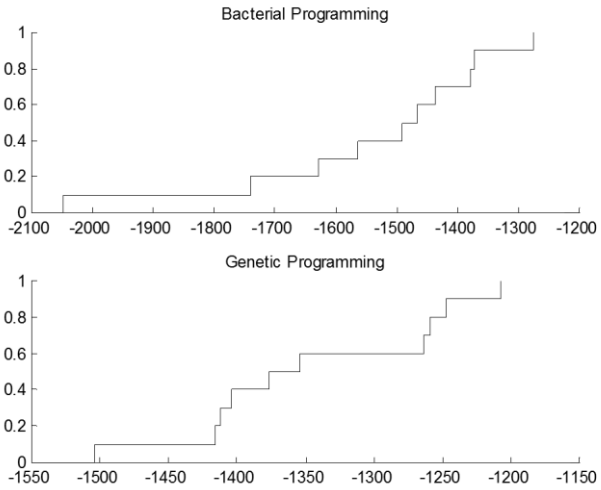


Fig. 4.8. Empirical probability distribution function for the ICT problem

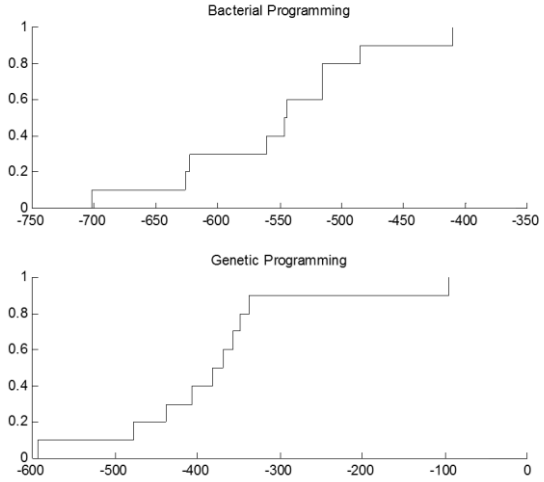


Fig. 4.9. Empirical probability distribution function for the generic six dimensional problem

From these figures the same conclusions can be drawn as in the previous subsection. For the one dimensional problem the two approaches give similar results. For the two dimensional ICT problem, the best individuals' BIC values lie between approximately -2050 and -1750 with BPA, and between approximately -1500 and -1420 with GP, which means that the GP method provides poorer results than BPA. For the six dimensional problem, the best individuals have a BIC between -600 and -500 with GP, while with BPA the BIC lies between -700 and -630.

4.5 Conclusions

In BSNN networks design one important task is to find the best topology. Different algorithms were introduced previously, which try to solve this task. In this chapter, the bacterial programming algorithm was introduced, which applies the bacterial operators instead of the original genetic operators. The bacterial mutation optimizes the local portions of the individual. The gene transfer operator replaces the traditional crossover operator and allows the exchange of information between different individuals. Therefore, while the bacterial mutation is working on one individual, and tries to optimize this bacterium, the gene transfer is applied to the whole bacteria population, avoiding the local minima solutions. If more clones are being applied in the bacterial mutation, better results are expected. In the gene transfer operation, it is important not to use a high infection value because the population can be trapped in local minima. The advantage of the proposed technique is that it is efficient in high dimensional problems.

The promising performance of this new optimization technique can be improved if a local searcher technique for estimating the interior knots is used. Therefore, the next chapter introduces a hybrid methodology where the bacterial programming algorithm is combined with the LM algorithm in BSNN design.

Chapter 5

DEALING WITH LOCAL MINIMA IN B-SPLINE NEURAL NETWORKS

5.1 Introduction

The previous chapter introduced Bacterial Programming evolutionary algorithm (BP) for the determination of the best topology of a BSNN and compared its performance with Genetic Programming (GP). BPA is a fusion of the principles of Bacterial Evolutionary Algorithm (BEA) [70], and GP [59].

However, the design of BSNN networks involves two major phases: the structure determination, and the model parameters estimation, which is the subject of this chapter. In a previous work [84], a completely supervised training algorithm was applied for B-Splines and results show that second order methods are seen as the most promising. In that previous work, the Levenberg-Marquardt (LM) algorithm was compared to the error back-propagation algorithm (first-order derivative method), presenting the best results.

The only drawback with the LM is the fact that its performance depends on the starting points of the training process, as reported earlier.

This chapter describes a method where the BPA algorithm is used, in a hybrid scheme [181], to determine the most suitable starting points to the Levenberg-Marquardt algorithm, therefore increasing the possibility of finding the global minimum.

This chapter is an extended version of [182] and is organized as follows.

In section 5.2 the algorithm guidelines are described. This includes the encoding process and the changes in the bacterial operators used in the evolutionary process. In section 5.3, the performance of the algorithm is evaluated in two distinct models using a bivariate problem.

Conclusions are drawn in section 5.4.

5.2 The proposed algorithm

The idea underlying the proposed algorithm is that of combining an evolutionary technique with a local search algorithm. So, instead of using only bacterial mutation to change the position of the knots (please see Section 4.3), with a hybrid methodology a second order nonlinear technique can be applied simultaneously at the same generation. The proposed algorithm utilizes the Bacterial Programming algorithm to provide suitable starting points for Levenberg-Marquardt. As the structure of the BSNN model is fixed, the bacterial operators must be reformulated. This is explained in the following sections.

This algorithm is denoted BPLM and can attain more efficiency and faster convergence.

5.2.1 The evolutionary process

The evolutionary process involves the following steps, as illustrated in Fig. 5.1.

In step 1, the creation of the initial population consists of starting from a fixed model structure. In this specific methodology (BPLM), every bacterium will be represented by a similar expression tree, except for the interior knots location in the terminal nodes.

Fig. 5.2 represents the tree structure of a BSNN model M consisting of two submodels, for an input set consisting of x_1, x_2, and x_3. At the terminal nodes, the value of the univariate B-spline functions is denoted by T1, T2, T3 and T4. As it can be seen, input variable x_1 is repeated in both submodels meaning that spline information for x_1 must be given by terminals on both tree branches.

So, a possible encoding for the terminals would be:

$$\begin{aligned} T_1 &= \{x_1, 2, 1, \{-2, -1, 0, 1, 2\}\} \\ T_2 &= \{x_2, 2, 0, \{-2, -1, 1, 2\}\} \\ T_3 &= \{x_3, 3, 0, \{-3, -2, -1, 1, 2, 3\}\} \\ T_4 &= \{x_1, 2, 2, \{-2, -1, 0, 0.5, 1, 2\}\} \end{aligned}, \tag{5.1}$$

where the information in the i^{th} terminal, T_i, consists of (by this order), the variable identification, the order of the spline, the number of interior knots, and the knots vector.

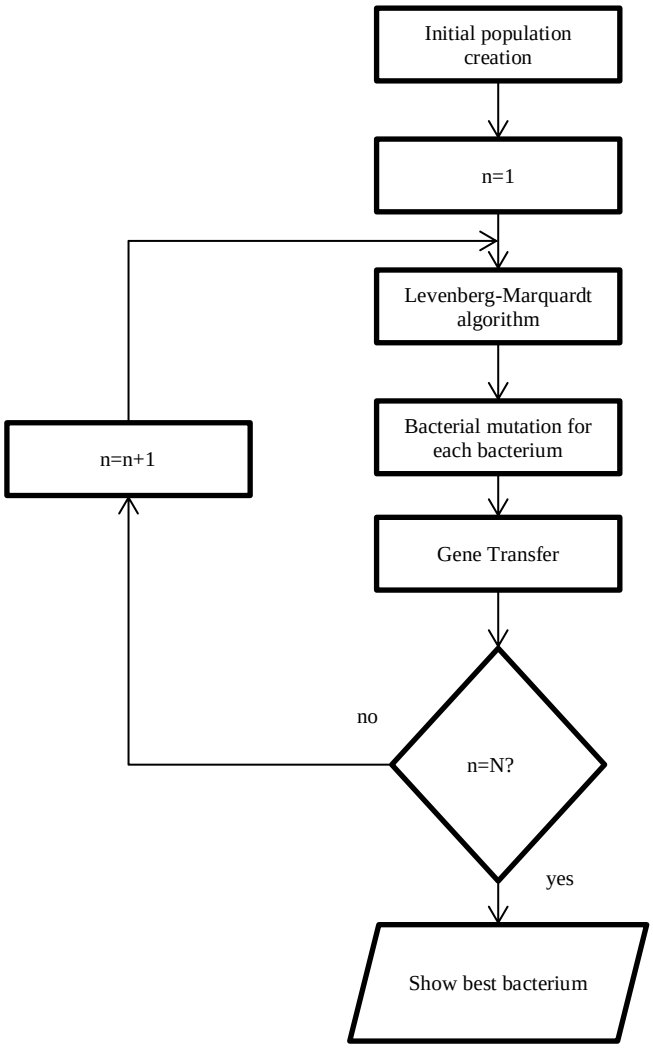


Fig. 5.1 Evolution cycle for the BPLM algorithm.

In order to ensure similar model structures, the same expression tree is assigned to every bacterium. At design time, the algorithm produces slightly different information at the terminal nodes: different knots locations.

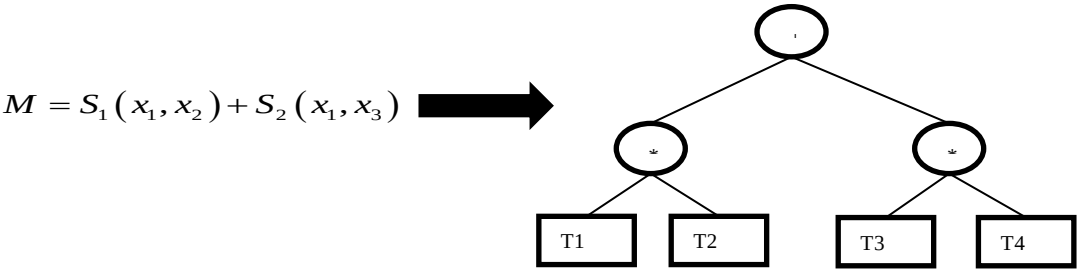


Fig. 5.2 Initial tree structure creation process. A pre-defined BSNN model induces the corresponding tree structure.

All steps above except for step 2 have been described in more detail in section 4.3.

Due to the fixed BSNN structure steps 3 and 4 are revised. This is done next.

5.2.2 Modifications on the bacterial operators

5.2.2.1 Bacterial mutation

As usual, in the bacterial mutation and at every generation, each bacterium is cloned N_{clones} times. A certain part of the bacterium is randomly selected and the parameters of this part are randomly changed in each clone (mutation).

Since the structure is fixed, mutation is carried out at the terminal nodes only and restricted to moving the knots by a random fraction. Hence, the mutation operation does not comprise several types of terminal mutations as described in the previous chapter. Just one terminal mutation is applied, the reallocation of the interior knot.

This is illustrated in Fig. 5.3, where the selected terminal (T2) is subject to a reallocation of its unique interior knots which, for instance, is changed from 0 to 0.6.

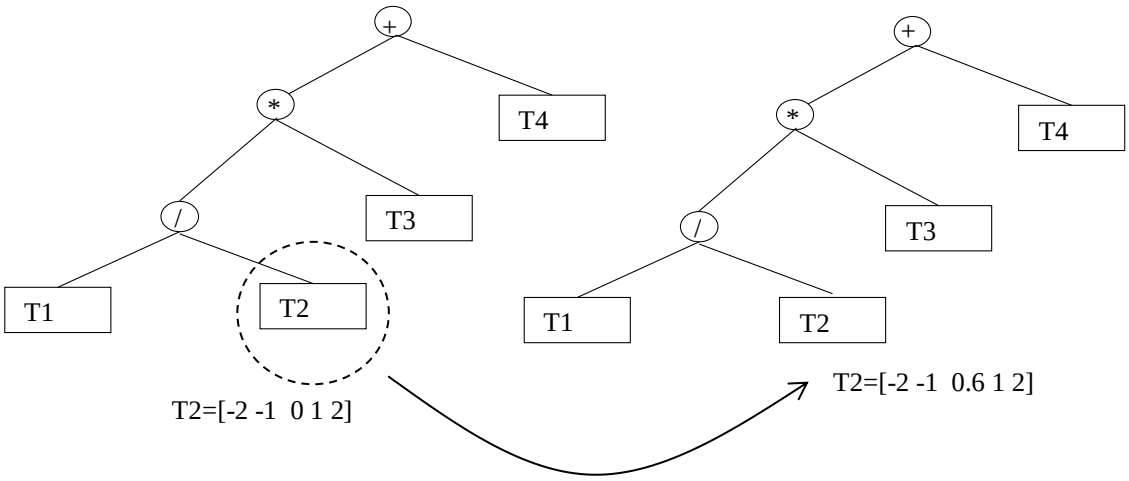


Fig. 5.3 Mutation is performed at the terminal level only: the interior knots' positions are randomly changed.

5.2.2.2 Gene Transfer

During the gene transfer operation the genetic information is exchanged between two bacteria.

All bacteria share identical shape expression trees (with same function nodes and same input variables). The gene transfer operation is modified so that information is exchanged from subtrees located at the same position in the source and destination expression tree. In the example illustrated by Fig. 5.4, a copy of the source bacterium below the selected node is carried out into the destination bacterium, leaving terminal node T3 unchanged.

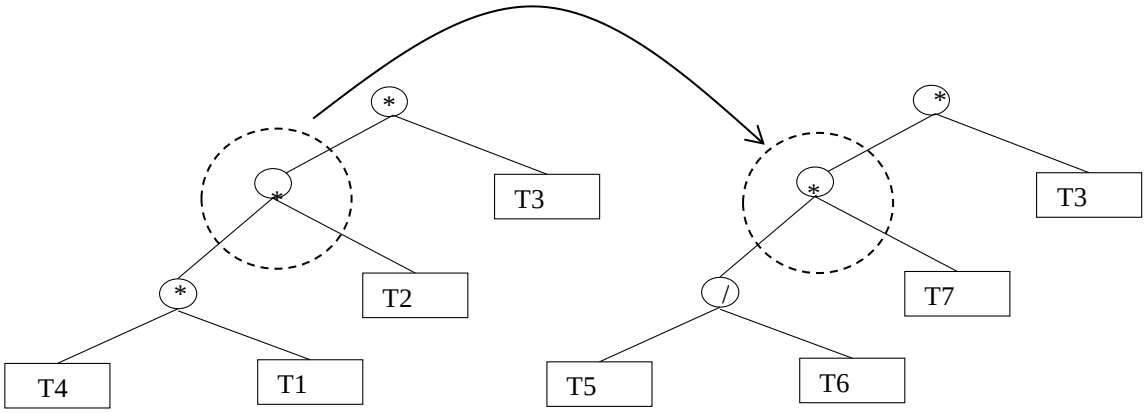


Fig. 5.4 Operation performed during gene transfer.

Due to the methodology of gene transfer, infections are carried out N_{inf} times, in each generation.

Bacteria must be evaluated and re-ranked after each mutation or gene transfer operation. To assess the viability of the bacterium, the error criterion used in BPLM is the SSE.

5.3 Simulation results

The examples relate to the inverse kinematic transformation (ICT) as it is a mapping between two input variables (the Cartesian coordinates) and one output: (one of the angles of a two-links manipulator).

Two distinct B-spline models for fitting the ICT problem will be used. The correspondent structure of these models is given in the next table.

Both models contain 2 quadratic B-Splines one for each dimension. The model on the left column has 1 bivariate submodel ($x_1 \otimes x_2$) while the one on the right is composed of 2 univariate submodels (x_1+x_2). The latter has 3 interior knots, so that the 2 interior knots belonging to the second submodel have restrictions on their ordering. The former has 2 unconstrained (in terms of ordering) interior knots.

In the LM algorithm, the new criterion (2.83) is used and the termination criterion (section 2.3.6.1) with $\tau_f = 10^{-2}$, is employed.

TABLE 5.1.STRUCTURE OF THE MODELS USED FOR THE FITTING

Model Structure	Model 1	Model 2
Submodels input variable	(1x2)	1+2
Complexity	18	12
Splines order	(3x3)	(3+3)
Number of interior_knots	(1x1)	(1+2)

5.3.1 Optimization using LM

To illustrate the need for the proposed algorithm, the LM algorithm for optimizing the knots position of both models, from a collection of different starting points.

For the first model a total of 100 starting points with a uniform distribution in the bi-dimensional space were produced. The performance surface for the training criterion (MSE value) was drawn, where a discretization value of 0.02 was used. Fig. 5.5 illustrates the performance surface for the first model with the superimposition of the MSE value evolution. 7 local minima were found, with this discretization step.

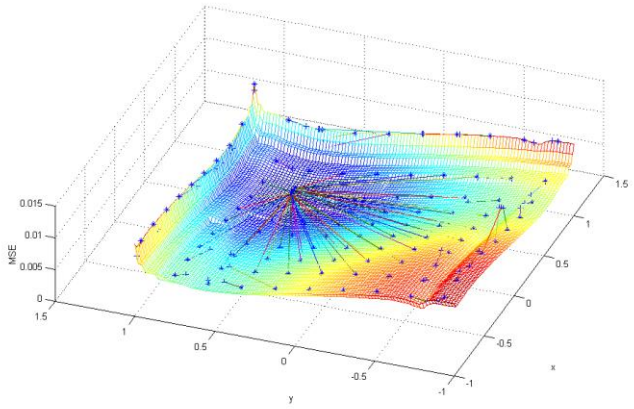


Fig. 5.5 Knots and MSE evolution for 100 different initial starting points using the LM algorithm for model 1: '*' specifies the starting point; '+' specifies the final point

For the second model, the surface shows the location of the optimization knots (x, y, z) when 150 points were used as starting points. A discretization step of 0.05 was employed, and 42 local minima were identified.

Thus, these results confirm the good performance of the Levenberg-Marquardt.

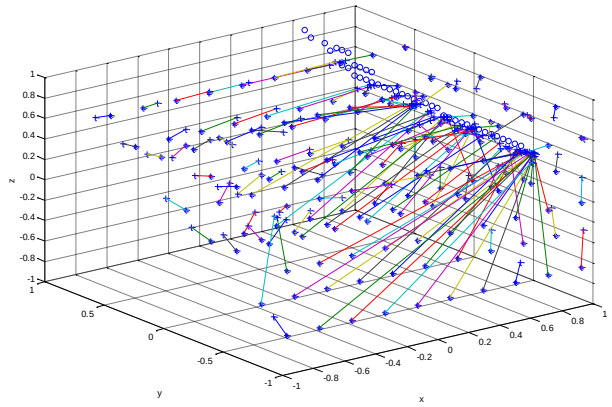


Fig. 5.6 Knots evolution for the 150 different initial positions using the LM algorithm for model 2: '*' specifies the starting point; '+' specifies the final point; local optima are specified by 'o'.

5.3.2 Performance of the BPLM algorithm

This sub-section illustrates the benefits of combining the BPA algorithm and the LM algorithm.

This methodology is applied to both models described in Table 5.1. A population of 10 individuals was used, and the BPLM executed for 10 generations. In the Bacterial mutation, the number of clones employed was 8, and the number of infections for gene transfer was 4.

Model 1 represents an identification problem with 2 parameters located on a bivariate B-spline model. The global optimum is located approximately at $\boldsymbol{\lambda} = [0.4\ 0.5]$ with a MSE value of 1.6×10^{-3}. Alternatively, model 2 has 3 parameters located on a model consisting of 2 univariate submodels.

Fig. 5.7 shows the parameters evolution superimposed on the corresponding performance surface when optimizing the 2 parameters of model 1. In this figure, a variety of starting points given by the population individuals is considered. Despite the initial location of the knots, convergence to local optima is obtained and at a fast rate. The global optimum is never reached, supposedly due to the existence of a very steep slope around the global optimum. This has already happened with the results in section 5.3.1.

Similar conclusions are taken from Fig. 5.8 where the 3 parameters from model 2 are estimated. This problem requires optimizing 3 parameters which does not allow drawing the corresponding performance surface. Nevertheless, it can be observed, from Table 5.3 and Fig. 5.7 that the convergence rate is very fast, since most of the local minima are reached in three or four generations.

Table 5.2 and Table 5.3 list the MSE value for the individuals in the population after the first LM and in the last generation.

Considering Table 5.2, one can see that three of the candidates (4, 7 and 8) are trapped in local optima far from the global optima initially. Despite this fact, the bacterial operators show their ability to step out from these local minima and give LM a better initial solution. The same happens with Table 5.3, where instead of three, four of the candidates are trapped in local minima, at the initial generation. Thanks to the BPLM algorithm every candidate is guaranteed to converge to one of the local optima closer to the global optima.

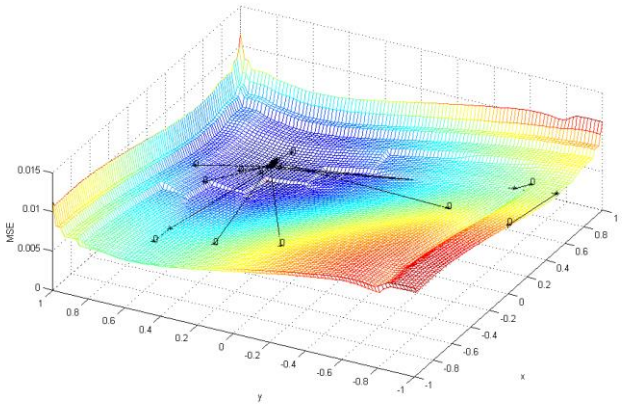


Fig. 5.7. Population evolution for model 1

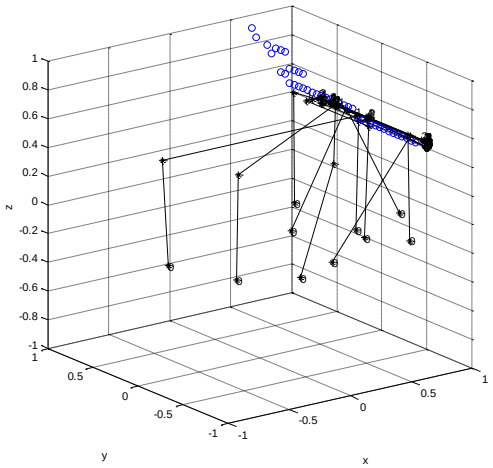


Fig. 5.8. Population evolution for model 2

TABLE 5.2.INDIVIDUALS PERFORMANCE DURING OPTIMIZATION (MODEL 1)

Candidate	Initial MSE	MSE value after first LM procedure	Final Values
1	3.6×10^{-3}	1.7×10^{-3}	1.7×10^{-3}
2	7.6×10^{-3}	1.7×10^{-3}	1.7×10^{-3}
3	6.2×10^{-3}	1.7×10^{-3}	1.7×10^{-3}
4	6.7×10^{-3}	6.4×10^{-3}	1.7×10^{-3}
5	3.2×10^{-3}	1.7×10^{-3}	1.7×10^{-3}
6	6.9×10^{-3}	1.7×10^{-3}	1.7×10^{-3}
7	8.6×10^{-3}	7.0×10^{-3}	1.7×10^{-3}
8	5.6×10^{-3}	5.0×10^{-3}	1.7×10^{-3}
9	1.9×10^{-3}	1.7×10^{-3}	1.7×10^{-3}
10	2.1×10^{-3}	1.7×10^{-3}	1.7×10^{-3}

TABLE 5.3.POPULATION INDIVIDUALS PERFORMANCE DURING OPTIMIZATION (MODEL 2)

Candidate	Initial MSE	MSE value after first LM procedure	Final Values
1	1.0×10^{-2}	8.8×10^{-3}	8.8×10^{-3}
2	1.0×10^{-2}	9.9×10^{-3}	8.7×10^{-3}
3	9.6×10^{-3}	8.9×10^{-3}	8.7×10^{-3}
4	8.8×10^{-3}	8.8×10^{-3}	8.7×10^{-3}
5	1.2×10^{-2}	1.1×10^{-2}	8.7×10^{-3}
6	9.0×10^{-3}	8.9×10^{-3}	8.7×10^{-3}
7	1.2×10^{-2}	1.2×10^{-2}	8.8×10^{-3}
8	1.5×10^{-2}	1.4×10^{-2}	8.7×10^{-3}
9	1.2×10^{-2}	8.8×10^{-3}	8.7×10^{-3}
10	1.0×10^{-2}	8.8×10^{-3}	8.7×10^{-3}

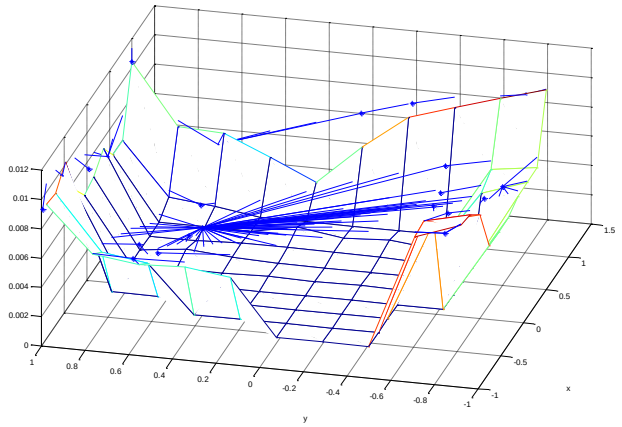


Fig. 5.9. Performance surface seen by BPA when optimizing model 1

Considering Fig. 5.9, which shows the performance surface seen by the BPLM algorithm, it is interesting to observe that, from the point of view of the BPA algorithm, the performance surface is not a smooth one. This would represent a barrier to the LM but it does not. This is effectively confirmed when one considers Fig. 5.10 because a fast convergence to a point near the global optima is achieved when using the LM algorithm in conjunction with an evolutionary algorithm. This is obvious, since at generation 2 the best candidate has already

obtained the MSE value from the last generation. This demonstrates how efficient the usage of both algorithms can be.

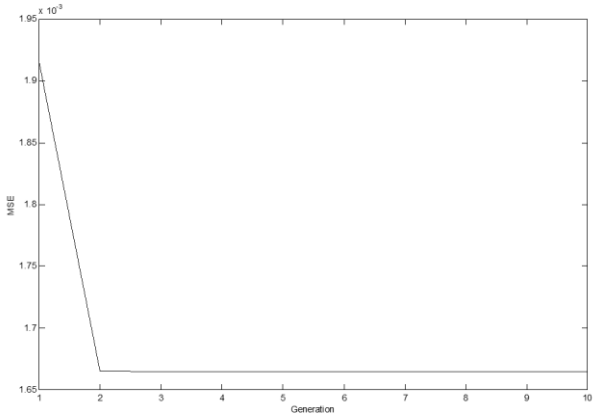


Fig. 5.10. MSE line for best candidate at every generation when optimizing model 1.

5.4 Conclusions

In this chapter a hybrid method for B-spline neural networks parameter estimation was introduced. The Bacterial Programming was combined with the Levenberg-Marquardt (LM) algorithm in order to estimate the optimal interior knots position in a B-Spline neural network (BSNN). The simulation examples have shown that the LM algorithm by itself is much likely to become trapped in local optima. The usage of the evolutionary algorithm (BP) in conjunction with the LM allows achieving solutions closer to the global optima.

Due to the functional equivalence between B-spline neural networks and fuzzy systems, the algorithms presented in Chapters 3 to 5 to B-Splines are applicable to fuzzy systems, provided the conditions described in Section 2.4.1.1 are met. In Chapter 7 the Levenberg-Marquardt algorithm is updated, and applied to fuzzy systems where the fuzzy rules do not need to implement Ruspini-partition. The LM algorithm is also combined with the bacterial evolutionary algorithm [70] forming a hybrid methodology similar to the one described in this chapter.

The next chapter will focus on a new input domain decomposition for the BSNN model, with the purpose of improving the tradeoff between model complexity and accuracy.

CHAPTER 6

INPUT DOMAIN DECOMPOSITION FOR B-SPLINES DESIGN

6.1 Introduction

The previous chapters demonstrated that techniques based on the concepts of evolutionary systems are adequate for optimizing B-spline models. In fact, this is the main goal of system identification, the determination of parsimonious models from data. B-spline Neural Networks (BSNN) offer an interesting number of useful properties. They have, however, an important drawback: they suffer from the *curse of dimensionality*, meaning that the number of its parameters increases exponentially with the number of inputs.

The trademark algorithm for BSNN model training is the ASMOD [58] which is a *mixed* type algorithm. Despite being ASMOD based on a functional decomposition that attempts to reduce the model complexity, the models obtained are typically highly complex. This has to do with the fact that BSNN models assume a regular grid.

To alleviate this drawback, a new partitioning approach is implemented, which goes beyond the usual input decomposition strategies known as noted in chapter 2, subsection 2.5.2.5. With this approach lower complexity models with similar generalization capabilities can be obtained.

The current chapter is an extension of [183] and is organized as follows.

An outline of the proposed input domain decomposition is given in section 6.2. To employ the proposed decomposition, it is necessary to set conditions on the linear weights, which is done in section 6.2.2, for the constant and linear splines. It shows that a reduction of the BSNN model complexity is related to the positioning of the merges in the grid. It illustrates this with an example with a simple *2*-dimensional problem and also shows how to perform the extension to the multivariate case. This includes a new approach for evaluating the BSNN model based on the univariate splines analysis. With this methodology it is quite

straightforward to apply the proposed domain decomposition to any BSNN model. Section 6.3 presents a methodology to evaluate the model complexity when the grid is partitioned, as it is no longer calculated the usual way. Simulation results comparing several partitionings are then shown in 6.4. In section 6.5 the process to automatically evolve an appropriate grid partitioning is described. An appropriate encoding of the chromosome for an evolutionary algorithm is described in subsection 6.5.1. Using the proposed chromosome encoding a general evaluation of the genetic algorithm operators is presented in section 6.5.2. In order to evolve a structure with merged grids the adaptation of GEP to the B-spline structure is described in section 6.5.3. This section includes results for different grids. Next, and to apply a nonlinear local optimization approach, the Levenberg-Marquardt algorithm (LM) is also focused. In this sense, the required update for the LM is described in section 6.5.4. That section includes a study on the performance of the LM for several initial partitioned grids, performance comparison with the error back-propagation algorithm and, the use of another data set for evaluating the quality of a BSNN model.

Finally, conclusions are drawn in section 6.6.

6.2 Proposed input space partitioning

The complexity of BSNN models can be reduced if one could add to an ASMOD decomposition (which is a functional decomposition) some form of domain decomposition. As mentioned in section 2.5.2.5 there are distinct ways to employ input domain decomposition. Still, none fall into the required partitioning proposed in the following. For instance, *k-d* tree partitioning, which is also used by ABBMOD [62], still does not cover the case which is illustrated in Fig. 6.1. As for the sparse partitioning, it is incompatible with the B-spline structure.

The proposed grid partitioning is one where domain decomposition like the following can be obtained:

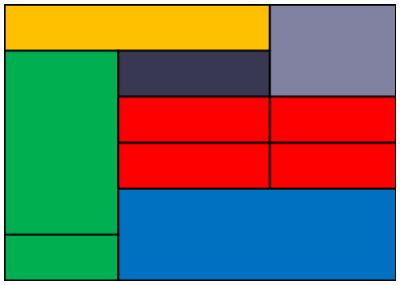


Fig. 6.1. Possible example of proposed input domain decomposition

As it will be shown subsequently, this can be obtained by adding sub-models with different domains, and by merging cells in the sub-models. Consider one multi-dimensional submodel $S(x_1, x_2)$ in the ASMOD expansion. For ease of visualization, a 2-input submodel, such as the one shown in Fig. 6.2, will be considered.

This represents one submodel with inputs x_1 and x_2, with 2 interior knots in the 1st dimension, and 5 interior knots in the 2nd dimension. A regular grid is used.

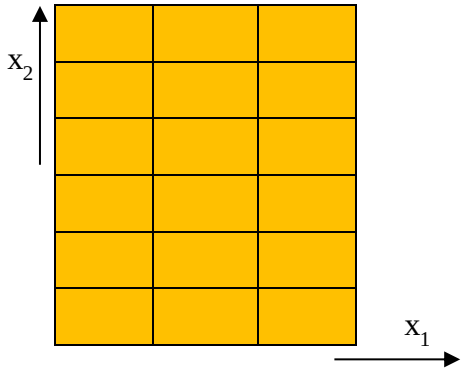


Fig. 6.2. Example of a grid partitioning (the same as used by ASMOD algorithm).

With the same input variables, and the same set of interior knots, different domain decompositions can be obtained, as, for instance, the ones presented in Fig. 6.3 and Fig. 6.4.

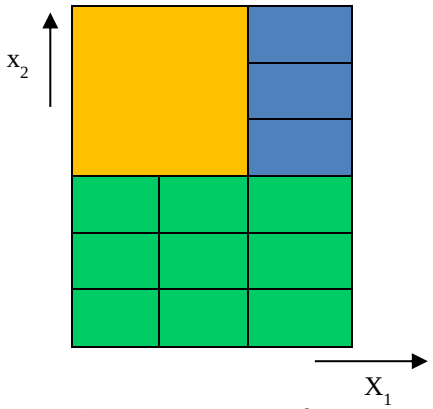


Fig. 6.3. Grid partitioning corresponding to 1 merge of size 2x3; the coloured cells correspond to a possible domain decomposition.

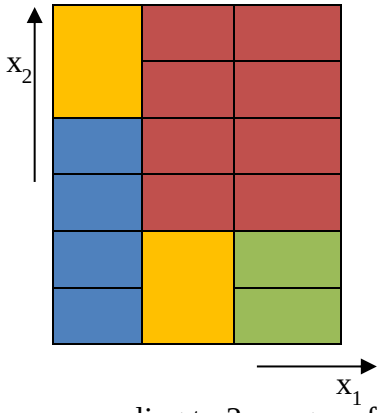


Fig. 6.4. Grid partitioning corresponding to 2 merges of size 1*x*2; the coloured cells correspond to a possible input domain decomposition.

The decompositions shown in Fig. 6.3 and Fig. 6.4 assume the same number of knots per input dimension as in Fig. 6.2, In Fig. 6.3, in comparison with Fig. 6.2, the top-left 2x3 cells are merged into one single cell. The model can be envisaged as a union of 3 sub-models, each one with its own non-overlapping domain, and for which the union of the domains of each sub-model is the original domain (of the model shown in Fig. 6.2).

Fig. 6.4 shows a case where 2 merges of size 1x2 occur. It can be regarded as a sum of 5 non-overlapping sub-models, whose union of domains is the original domain.

Fig. 6.3 and Fig. 6.4 are examples of a (large) number of different partitioning induced by a 2x5 knot matrix in the $x_1 \times x_2$ overall domain shown in Fig. 6.2. All these partitioning are functionally less complex than the full-grid one, and are obtained as the union of a number m_s of submodels SD_i for which the following properties apply:

$$\text{a)} \quad \bigcup_{j=1}^{m_s} Sup\left(SD_j\right) = Sup\left(S_i\right)$$

$$\text{b)} \quad \bigcap_{j=1}^{m_s} Sup\left(SD_j\right) = \varnothing$$

$$\text{c)} \quad \bigcup_{j=1}^{m_s} \boldsymbol{\lambda}\left(SD_j\right) = \boldsymbol{\lambda}\left(S_i\right)$$

(6.1)

In (6.1), *Sup* denotes the support of the model and $\boldsymbol{\lambda}$ the knot matrix.

Notice that, with these definitions the following partitioning is not a valid partitioning induced by $\boldsymbol{\lambda}\left(S_i\right)$, as it does not fulfill (6.1)-c):

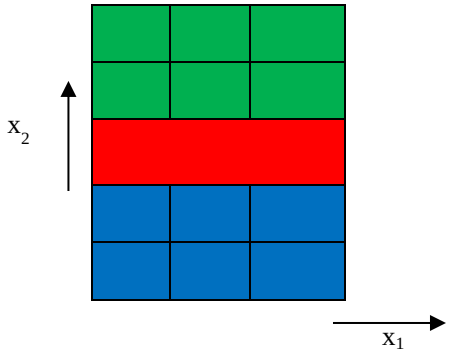


Fig. 6.5. Grid partitioning corresponding to 1 merge of size 3*x1*; Notice that this partitioning is not induced by $\lambda(S_i)$.

In order to have compatibility between the union of the submodels SD_i and the definitions of intervals presented in chapter 2, subsection 2.2.1.3**,** the right limit in the interval is closed only when the upper boundary knot coincides with the upper limit of the support of S, for that dimension.

The sum of the *SD* models must also exhibit the C^{k-1} continuity of a BSNN model. This can be obtained by relations within the weights of the *SD* sub-models. This issue is the purpose of the next subsection.

6.2.1 Linear weights relations among partitioned submodels

Consider the following figure, which represents a bivariate problem, obtained as the sum of 2 sub-models, distinguished by the different cell patterns.

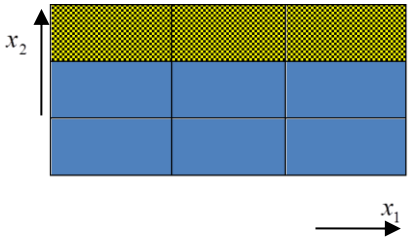


Fig. 6.6. Partitioning of a complete grid with no merges

Sub-model S_1 (on the bottom) has with two interior knots in dimension x_1 and one interior knot in dimension x_2, while sub-model S_2 has zero interior knots in dimension x_2, and the same two in x_1. If the top boundary knot in S_1 equals to the bottom boundary knot in S_2 for dimension x_2 (as it is the case), then conditions (6.1) are fulfilled. This does not guarantee, however, C^{k-1} continuity for the model which is the combination of the two sub-models. This

continuity must also be ensured whenever in one of the sub-models two or more cells are merged, as it is the case of Fig. 6.3 and Fig. 6.4. The continuity can be obtained, as it will be explained subsequently, by relations between the linear weights. To illustrate how this can be obtained, a simple example, shown in Fig. 6.7, will be used.

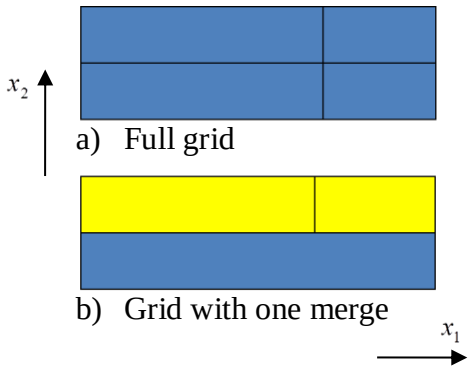


Fig. 6.7. Example of input space partitioning

In the above figure, two 2-dimensional grids are assumed. On the top grid, no cells are merged (therefore a full grid is considered). On the bottom grid, 1 merge is applied in dimension x_1.

Assuming triangular splines (of order 2), the output for Fig. 6.7-a) would be given as

$$y(x_1,x_2)=\begin{bmatrix} N_{2,1}^1(x_1).N_{2,2}^1(x_2) \\ N_{2,1}^1(x_1).N_{2,2}^2(x_2) \\ N_{2,1}^1(x_1).N_{2,2}^3(x_2) \\ \dots \\ N_{2,1}^3(x_1).N_{2,2}^1(x_2) \\ N_{2,1}^3(x_1).N_{2,2}^2(x_2) \\ N_{2,1}^3(x_1).N_{2,2}^3(x_2) \end{bmatrix}^T \times \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{1,3} \\ \dots \\ w_{3,1} \\ w_{3,2} \\ w_{3,3} \end{bmatrix} \tag{6.2}$$

The complexity of the model (in terms of linear parameters) is 9.

In the case of Fig. 6.7-b), there is a bivariate model, composed of two submodels, and three input domains:

$$M(x_1,x_2)=S_1^{2,0}\left(x_1^1\right)\otimes S_2^{2,0}\left(x_2^1\right)+S_1^{2,1}\left(x_1^1\right)\otimes S_2^{2,0}\left(x_2^2\right) \tag{6.3}$$

In (6.3), $M(x_1,x_2)$ denotes the bivariate model, where:

- $S_{\text{var}}^{k,r}\left(x_{\text{var}}^{i}\right)$ stands for the submodel which is represented by a knot vector with *r* interior knots, splines of order *k*, and input dimension *var* from the i^{th} interval.

and,

$$\begin{aligned} x_1^1 &= \left\{ x_1 : x_1 \in [\lambda_{1,0}, \lambda_{1,2}] \right\} \\ x_2^1 &= \left\{ x_2 : x_2 \in [\lambda_{2,0}, \lambda_{2,1}[\right\} \\ x_2^2 &= \left\{ x_2 : x_2 \in [\lambda_{2,1}, \lambda_{2,2}] \right\} \end{aligned} \tag{6.4}$$

Fig. 6.8 illustrates the support for each one of the univariate spline functions for the partitioned grid from Fig. 6.7-b).

As partitioned models will be considered, it is necessary to redefine the notation previously assumed. This way:

- $N_{SD,k,n}^{i}(x_n)$ stands for the i^{th} univariate spline of order *k* in the n^{th} dimension, for sub-model *SD*.
- $w_{SD,i,j}$ stands for the weight corresponding to a bivariate basis function, which is the product between the i^{th} univariate function (from the 1st dimension) and the j^{th} univariate function (from the 2nd dimension) of submodel *SD*.

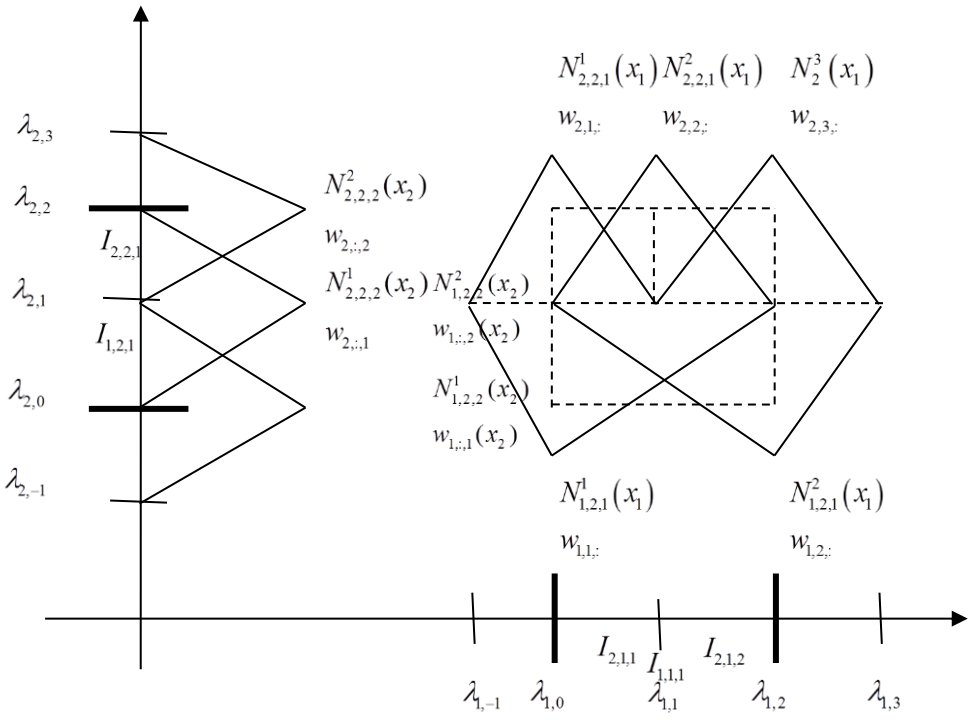


Fig. 6.8. Spline functions support for a grid with one merge of size 2x1

To ensure C^1 (C^{k-1}, k=2) continuity at the boundary points of x_1, when $x_2 = \lambda_{2,1}$:

$$\begin{cases} y\left(\lambda_{1,0}, \lambda_{2,1} - \Delta\right)\Big|_{\Delta \to 0} = y\left(\lambda_{1,0}, \lambda_{2,1}\right) \\ y\left(\lambda_{1,2}, \lambda_{2,1} - \Delta\right)\Big|_{\Delta \to 0} = y\left(\lambda_{1,2}, \lambda_{2,1}\right) \end{cases} \tag{6.5}$$

Noting that the points, $(\lambda_{1,0}, \lambda_{2,1} - \Delta)$ and $(\lambda_{1,2}, \lambda_{2,1} - \Delta)$ are spanned by 4 basis functions but only 1 is activated:

$$\begin{cases} N^1_{1,2,1}(\lambda_{1,0}).N^2_{1,2,2}(\lambda_{2,1} - \Delta)\Big|_{\Delta \to 0}.w_{1,1,2} = N^1_{2,2,1}(\lambda_{1,0}).N^1_{2,2,2}(\lambda_{2,1}).w_{2,1,1} \\ N^{1+1}_{1,2,1}(\lambda_{1,2}).N^2_{1,2,2}(\lambda_{2,1} - \Delta)\Big|_{\Delta \to 0}.w_{1,2,2} = N^3_{2,2,1}(\lambda_{1,2}).N^1_{2,2,2}(\lambda_{2,1}).w_{2,3,1} \end{cases} \tag{6.6}$$

Which yields,

$$\begin{cases} w_{1,1,2} = w_{2,1,1} \\ w_{1,2,2} = w_{2,3,1} \end{cases} \tag{6.7}$$

We shall denote this type of relations (ensuring C^{k-1} continuity at the frontier nodes) as *domain contiguity*).

As in the bottom sub-model the two cells are merged, C^{k-1} continuity must also be ensured at $\left(\lambda_{1,1}, \lambda_{2,1}\right)$. These relations will be denoted as *input merging.* This way:

$$y(\lambda_{1,1}, \lambda_{2,1} - \Delta)\Big|_{\Delta \to 0} = y(\lambda_{1,1}, \lambda_{2,1}) \Leftrightarrow$$

$$N^1_{1,2,1}(\lambda_{1,1}).N^2_{1,2,2}(\lambda_{2,1} - \Delta)\Big|_{\Delta \to 0} w_{1,1,2} + N^2_{1,2,1}(\lambda_{1,1}).N^2_{1,2,2}(\lambda_{2,1} - \Delta)w_{1,2,2} =$$

$$= N^2_{2,2,1}(\lambda_{1,1}).N^1_{2,2,2}(\lambda_{2,1})w_{2,2,1}$$

Solving for $w_{2,2,1}$:

$$w_{2,2,1} = \frac{\left(N^1_{1,2,1}(\lambda_{1,1}).N^2_{1,2,2}(\lambda_{2,1} - \Delta)\Big|_{\Delta \to 0} w_{1,1,2} + N^2_{1,2,1}(\lambda_{1,1}).N^2_{1,2,2}(\lambda_{2,1} - \Delta)\Big|_{\Delta \to 0} w_{1,2,2}\right)}{N^2_{2,2,1}(\lambda_{1,1}).N^1_{2,2,2}(\lambda_{2,1})} \tag{6.8}$$

Since $N^{1+1}_{1,2,2}(\lambda_{2,1} - \Delta)\Big|_{\Delta \to 0} = N^2_{2,2,1}(\lambda_{1,1}).N^1_{2,2,2}(\lambda_{2,1}) = 1$

$$w_{2,2,1} = N^1_{1,2,1}(\lambda_{1,1})w_{1,1,2} + N^{1+1}_{1,2,1}(\lambda_{1,1})w_{1,2,2} \tag{6.9}$$

The model from grid Fig. 6.7-b) has an output given by the combination of the two sub-models, i.e.:

$$y(x_1,x_2)=\begin{bmatrix} N^1_{1,2,1}(x_1).N^1_{1,2,2}(x_2) \\ N^1_{1,2,1}(x_1).N^2_{1,2,2}(x_2) \\ N^2_{1,2,1}(x_1).N^1_{1,2,2}(x_2) \\ N^2_{1,2,1}(x_1).N^2_{1,2,2}(x_2) \end{bmatrix}^T \times \begin{bmatrix} w_{1,1,1} \\ w_{1,1,2} \\ w_{1,2,1} \\ w_{1,2,2} \end{bmatrix} + \begin{bmatrix} N^1_{2,2,1}(x_1).N^1_{2,2,2}(x_2) \\ N^1_{2,2,1}(x_1).N^2_{2,2,2}(x_2) \\ N^2_{2,2,1}(x_1).N^1_{2,2,2}(x_2) \\ N^2_{2,2,1}(x_1).N^2_{2,2,2}(x_2) \\ N^3_{2,2,1}(x_1).N^1_{2,2,2}(x_2) \\ N^3_{2,2,1}(x_1).N^2_{2,2,2}(x_2) \end{bmatrix}^T \times \begin{bmatrix} w_{2,1,1} \\ w_{2,1,2} \\ w_{2,2,1} \\ w_{2,2,2} \\ w_{2,3,1} \\ w_{2,3,2} \end{bmatrix} \tag{6.10}$$

The output in (6.10) has a complexity of 10 basis functions.

However, replacing the weight relations from (6.7) and (6.9) in (6.10) yields a less complex model:

$$y(x_1,x_2)=\begin{bmatrix} N^1_{1,2,1}(x_1).N^1_{1,2,2}(x_2) \\ N^1_{1,2,1}(x_1).N^2_{1,2,2}(x_2)+N^1_{2,2,1}(x_1).N^1_{2,2,2}(x_2)+N^1_{1,2,1}(\lambda_{1,1})N^2_{2,2,1}(x_1).N^1_{2,2,2}(x_2) \\ N^2_{1,2,1}(x_1).N^1_{1,2,2}(x_2) \\ N^2_{1,2,1}(x_1).N^2_{1,2,2}(x_2)+N^3_{2,2,1}(x_1).N^1_{2,2,2}(x_2)+N^2_{1,2,1}(\lambda_{1,1})N^2_{2,2,1}(x_1).N^1_{2,2,2}(x_2) \\ N^1_{2,2,1}(x_1).N^2_{2,2,2}(x_2) \\ N^2_{2,2,1}(x_1).N^2_{2,2,2}(x_2) \\ N^3_{2,2,1}(x_1).N^2_{2,2,2}(x_2) \end{bmatrix}^T \times \begin{bmatrix} w_{1,1,1} \\ w_{1,1,2} \\ w_{1,2,1} \\ w_{1,2,2} \\ w_{2,2,1} \\ w_{2,2,2} \\ w_{2,3,2} \end{bmatrix}$$

(6.11)

The first two terms in rows 2 and 4 are the complementary terms related with the fact that the 2nd B-spline in x_2 is divided between the 2 sub-models. The 3rd term is the contribution of the middle spline in x_1 of the 2nd sub-model, weighted by its projection on each spline in x_1 for the 1st sub-model. Notice also that the number of basis function has decreased to 7 basis functions, compared to 9 corresponding to the complete grid. By removing 1 knot from the 1st sub-model, this sub-model loses 2 basis functions (as it is a bivariate sub-model). This lost is spread to the entire model.

The analysis carried out here showed that the merge of interior knots leads to less complex models. To show this, the C^{k-1} continuity property for B-spline functions was taken into account. Extending this approach to higher order splines or *n*-dimensional grids can be cumbersome.

An alternative methodology is to carry this analysis in a univariate fashion, and as multivariate splines are computed from the tensor product between univariate splines, ensure that every combination between weights in one dimension is projected to the other dimensions.

6.2.2 Methodology used for computing relations between linear weights

Fig. 6.8 sketches the support associated with the spline functions if a merge in dimension x_1 is imposed in one of the sub-models. It also illustrates how spline functions from dimension x_1 from the sub-model with a merge are projected onto the input domain of the other sub-model. Furthermore, the spline functions in dimension x_2 extend across the input domain of both sub-models: specifically, the middle (or second) spline function whose support extends across both sub-models. This fact occurs for *k-1* spline functions.

Because of the relations between basis functions just explained, finding dependences between linear weights based on that fact may prove advantageous. Thus, for the methodology employed, two situations are considered.

The first one deals with the fact that any pair of sub-models is separated by a knot in one of the input dimensions (denoted as *domain contiguity*). In this case, the univariate definition of splines for the other dimensions is irrelevant of the submodel used, i.e., the spline functions common to both sub-models are defined exactly the same way. To ensure equivalence with the complete grid one needs only to guarantee C^{k-1} continuity.

The second situation is relative to the remaining (*n-1*) dimensions. In this specific grid of Fig. 6.7-b), the support for the basis functions in dimension x_1 changes with the submodel at hand. This is denoted by *input merging*. In this case, it is needed to estimate relations between weights that allow output equalities for the shared interior knots.

These two cases will be exploited for constant and B-Splines in the following sub-sections. The quadratic and cubic splines cases are described in Appendix B.

As the analysis in this section is carried out for univariate splines, the notation used will drop the dimension index, so that:

- $N^{j}_{SD,k}(x)$ will denote the j^{th} univariate spline from submodel *SD* of order *k*.
- the weight vector, $w_{SD,i}$ stands for the weight corresponding to the i^{th} spline for submodel *SD*.

6.2.2.1 Constant splines

With these splines, the output is only dependent on the weight value, and so there is no need to set any output or derivative condition. However, a multivariate model probably (if not

always) depends on other order splines in one of the other dimensions, and in this case, special care must be taken. For this reason, only *input merging* requires analysis.

6.2.2.1.1 Input merging

Suppose there are two univariate submodels in the same input variable, *x*:

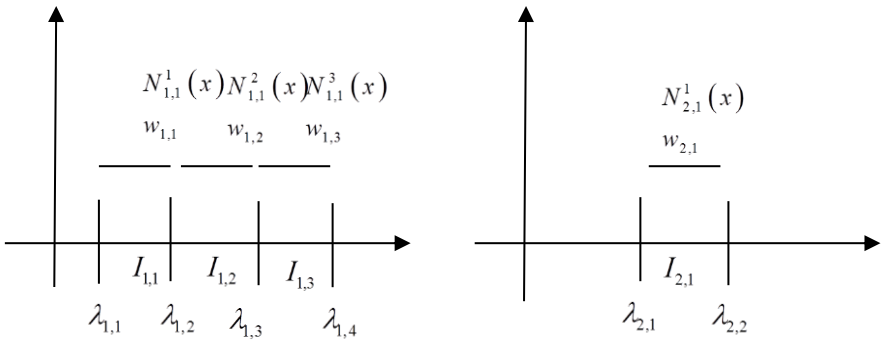


Fig. 6.9. Constant splines representation for a univariate model composed of two univariate submodels, in the same input variable ($I_{1,1} \cup I_{1,2} \cup I_{1,3} = I_{2,1}$).

Assume that $\lambda_{1,1} = \lambda_{2,1}$ and $\lambda_{1,4} = \lambda_{2,2}$.

The requirement is such that

$$y_1 = y_2\big|_{x \in I_{2,1}} \Leftrightarrow \begin{cases} N_{1,1}^1 w_{1,1} = N_{2,1}^1 w_{2,1} \\ N_{1,1}^2 w_{1,2} = N_{2,1}^1 w_{2,1} \\ N_{1,1}^3 w_{1,3} = N_{2,1}^1 w_{2,1} \end{cases} \Rightarrow \begin{cases} w_{1,1} = w_{2,1} \\ w_{1,2} = w_{2,1} \\ w_{1,3} = w_{2,1} \end{cases} \tag{6.12}$$

Relations in (6.12) indicate that the linear weights from both sub-models must be equal as long as the same spline input range is covered.

Therefore, in general:

$$w_{1,j} = w_{2,i}, \quad if\ I_{2,i} = \bigcup I_{1,j}\ , \tag{6.13}$$

6.2.2.2 Triangular splines

Triangular splines are order two splines (k=2). In this sense, C^1 contiguity is required. Both *input merging* and *domain contiguity* require analysis.

6.2.2.2.1 Domain contiguity

Suppose there is a univariate model which is formed as the sum of two sub-models, in the same input variable, *x*:

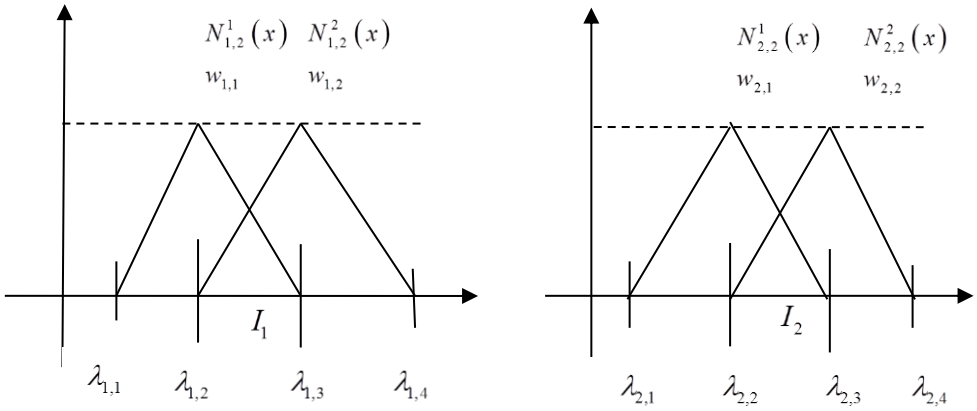


Fig. 6.10. Triangular splines representation for a univariate model composed of two univariate submodels, in the same input variable.

Notice that in the last figure the input domain is defined by intervals I_1 and I_2 where $I_1 \cap I_2 = \varnothing$.

In order to have equivalence between a complete grid model and one consisting of the sum of two submodels in the same input variable, the requirement on the interior knots is: $\lambda_{1,3} = \lambda_{2,2}$. This provides contiguity across the domains.

Under these circumstances the output of the sum of these 2 sub-models is given by:

$$y = y_1 + y_2 = \begin{bmatrix} N_{1,2}^1 & N_{1,2}^2 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix} + \begin{bmatrix} N_{2,2}^1 & N_{2,2}^2 \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \end{bmatrix} \tag{6.14}$$

Since the first term in (6.14) is valid for I_1 and the second term *i*s valid for I_2 the following condition must hold: $y_1(\lambda_{1,3}) = y_2(\lambda_{1,3}) \Rightarrow w_{1,2} = w_{2,1}$.

Therefore, (6.14) can be written as

$$y = \begin{bmatrix} N_{1,2}^1 & N_{1,2}^2 + N_{2,2}^1 & N_{2,2}^2 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} = w_{2,1} \\ w_{2,2} \end{bmatrix} \tag{6.15}$$

Generally, $w_{1,2+r} = w_{2,1}$ if there are *r* interior knots in submodel 1.

6.2.2.2.2 Input merging

Consider now two univariate sub-models that share some part of the input domain (input overlapping). Notice that this situation only makes sense if this input dimension is part of an *n*-dimensional model.

The following picture sketches this case.

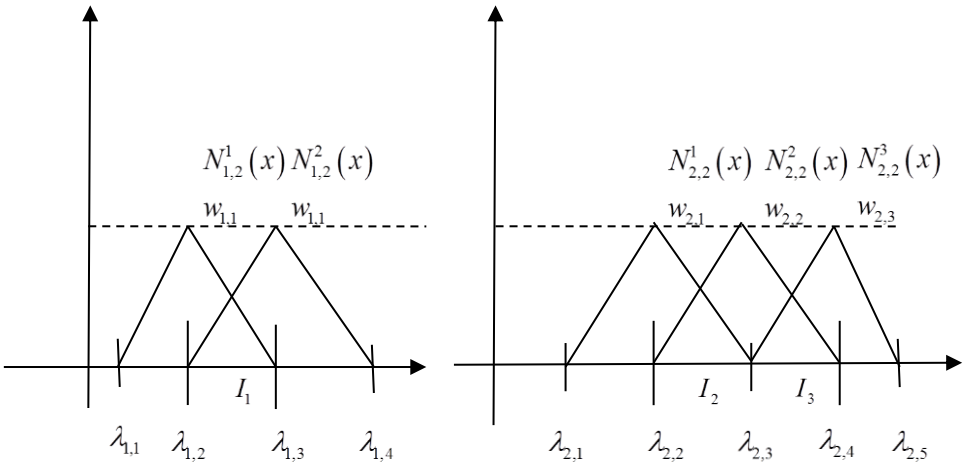


Fig. 6.11. Triangular splines representation of two univariate submodels, in the same input variable ($I_1 = I_2 \cup I_3$).

This is a slightly different situation than the previous one, since one is interested on ensuring output equality for sub-models 1 and 2. Since,

$$y_1 = \begin{bmatrix} N^1_{1,2} & N^2_{1,2} \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix}, y_2 = \begin{bmatrix} N^1_{2,2} & N^2_{2,2} & N^3_{2,2} \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \end{bmatrix} \tag{6.16}$$

Considering $\lambda_{1,2} = \lambda_{2,2}$ and $\lambda_{1,3} = \lambda_{2,4}$:

$$\begin{cases} y_1(\lambda_{1,2}) = y_2(\lambda_{2,2}) \\ y_1(\lambda_{1,3}) = y_2(\lambda_{2,4}) \\ y_1(\lambda_{2,3}) = y_2(\lambda_{2,3}) \end{cases} \Rightarrow \begin{cases} w_{1,1} = w_{2,1} \\ w_{1,2} = w_{2,3} \\ w_{2,2} = \dfrac{N^1_{1,2}(\lambda_{2,3}) w_{1,1} + N^2_{1,2}(\lambda_{2,3}) w_{1,2}}{N^2_{2,2}(\lambda_{2,3})} \end{cases} \tag{6.17}$$

If there are n_1 interior knots in sub-model 1, and n_2 interior knots in sub-model 2, the weights relation can be computed as,

$$\begin{aligned} & w_{1,1} = w_{2,1}, \quad if \ \left(\lambda_{1,2} = \lambda_{2,2}\right) \\ & w_{1,2+n_1} = w_{2,2+n_2}, \quad if \ \left(\lambda_{1,n_1+2} = \lambda_{2,n_2+2}\right) \\ & w_{2,1+i} = \frac{(\lambda_{2,2+i} - \lambda_{1,2+\mathrm{int}_1}) w_{1,\mathrm{int}_1} - (\lambda_{2,2+i} - \lambda_{1,1+\mathrm{int}_1}) w_{1,1+\mathrm{int}_1}}{(\lambda_{1,1+\mathrm{int}_1} - \lambda_{1,2+\mathrm{int}_1})}, i = 1 \ldots n_2 \end{aligned} \tag{6.18}$$

where int_1 is the interval number in sub-model 1 to which $\lambda_{2,2+i}$ belongs to.

6.2.3 Examples

This subsection illustrates the application of the previous methodology to two simple bivariate grids. The first example shows how the formulations presented in the previous subsection corroborate with the values obtained if the usual evaluation approach is applied. The second example shows how to apply the former equations in the case of a BSNN model when a simple partition of the grid is considered (this case was considered in Section 6.2.1).

6.2.3.1 Complete grid

This is an example where there is a (1*2) cells complete grid. The splines are triangular functions and no interior knots are defined. Fig. 6.12 shows the corresponding representation of the input space grid.

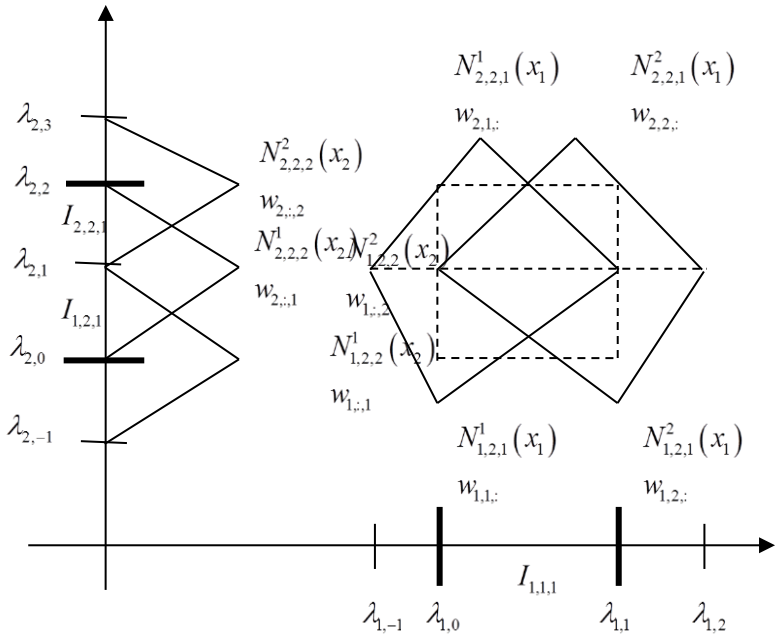


Fig. 6.12. Triangular splines representation for a bi-dimensional submodel with zero interior knots in dimension x_1 and one interior knot in dimension x_2.

The output of the B-Spline model is:

$$y = \begin{bmatrix} N^1_{2,1}N^1_{2,2} \\ N^1_{2,1}N^2_{2,2} \\ N^1_{2,1}N^3_{2,2} \\ N^2_{2,1}N^1_{2,2} \\ N^2_{2,1}N^2_{2,2} \\ N^2_{2,1}N^3_{2,2} \end{bmatrix}^T \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{1,3} \\ w_{2,1} \\ w_{2,2} \\ w_{2,3} \end{bmatrix} \tag{6.19}$$

Assume that the model is the sum of two sub-models in the same input variables, SD_1 and SD_2. Therefore the output is:

$$y = y_1 + y_2 = \begin{bmatrix} N^1_{1,2,1}N^1_{1,2,2} \\ N^1_{1,2,1}N^2_{1,2,2} \\ N^2_{1,2,1}N^1_{1,2,2} \\ N^2_{1,2,1}N^2_{1,2,2} \end{bmatrix}^T \begin{bmatrix} w_{1,1,1} \\ w_{1,1,2} \\ w_{1,2,1} \\ w_{1,2,2} \end{bmatrix} + \begin{bmatrix} N^1_{2,2,1}N^1_{2,2,2} \\ N^1_{2,2,1}N^2_{2,2,2} \\ N^2_{2,2,1}N^1_{2,2,2} \\ N^2_{2,2,1}N^2_{2,2,2} \end{bmatrix}^T \begin{bmatrix} w_{2,1,1} \\ w_{2,1,2} \\ w_{2,2,1} \\ w_{2,2,2} \end{bmatrix} \tag{6.20}$$

It is clear that both definitions (6.19) and (6.20), must generate the same output.

In this way, applying the results of Section 6.2.2.2:

- in dimension x_1:

$w_{1,1,:} = w_{2,1,:}; w_{1,2,:} = w_{2,2,:}$

- in dimension x_2:

$w_{1,:,2} = w_{2,:,1}$

This implies that, for the multivariate model:

$w_{1,1,2} = w_{2,1,1}; w_{1,2,2} = w_{2,2,1}$

And so, the output (6.20) can be re-written as:

$$y = y_1 + y_2 = \begin{bmatrix} N^1_{1,2,1}N^1_{1,2,2} \\ N^1_{1,2,1}N^2_{1,2,2} + N^1_{2,2,1}N^1_{2,2,2} \\ N^2_{1,2,1}N^1_{1,2,2} \\ N^2_{1,2,1}N^2_{1,2,2} + N^2_{2,2,1}N^1_{2,2,2} \\ N^1_{2,2,1}N^2_{2,2,2} \\ N^2_{2,2,1}N^2_{2,2,2} \end{bmatrix}^T \begin{bmatrix} w_{1,1,1} \\ w_{1,1,2} \\ w_{1,2,1} \\ w_{1,2,2} \\ w_{2,1,2} \\ w_{2,2,2} \end{bmatrix} \tag{6.21}$$

Since:

$$\begin{aligned} \{x_1, x_2\} \notin \{I_{1,1,1}, I_{1,2,1}\} &\rightarrow N^1_{1,2,1}N^2_{1,2,2} = 0 \\ \{x_1, x_2\} \notin \{I_{2,1,1}, I_{2,2,1}\} &\rightarrow N^1_{2,2,1}N^1_{2,2,2} = 0 \\ \{x_1, x_2\} \notin \{I_{1,1,1}, I_{1,2,1}\} &\rightarrow N^2_{1,2,1}N^2_{1,2,2} = 0 \\ \{x_1, x_2\} \notin \{I_{2,1,1}, I_{2,2,1}\} &\rightarrow N^2_{2,2,1}N^1_{2,2,2} = 0 \end{aligned} \tag{6.22}$$

The output given by (6.19) and (6.21) are equal.

6.2.3.2 Single merge grid partitioning

As explained before, the output of a multivariate B-spline model consists of the tensor product between univariate splines. In order to explain how to employ the methodology from section 6.2.2 to a bi-dimensional case, one can refer to the example from section 6.2.1.

In that example, the output is given as the sum of two sub-models (denoted by *SD*) both in the same input variables x_1 and x_2:

$$y(x_1, x_2) = \sum_{i=1}^{2} SD_i(x_1, x_2) \tag{6.23}$$

The knot vector for sub-model *SD* and dimension *dim* is defined as:

$$\boldsymbol{\lambda}_{sm,\dim} = \left\{ \lambda_{SD,\dim,1} \quad \lambda_{SD,\dim,2} \quad \lambda_{SD,\dim,i} \quad \ldots \quad \lambda_{SD,\dim,1+2r_i+k_i} \right\} \tag{6.24}$$

Also:

$$\begin{aligned}
&\left\{ \lambda_{1,1,1} \quad \lambda_{1,1,2} \quad \lambda_{1,1,3} \quad \lambda_{1,1,4} \right\} = \left\{ \lambda_{1,-1} \quad \lambda_{1,0} \quad \lambda_{1,2} \quad \lambda_{1,3} \right\} \\
&\left\{ \lambda_{2,1,1} \quad \lambda_{2,1,2} \quad \lambda_{2,1,3} \quad \lambda_{2,1,4} \quad \lambda_{2,1,5} \right\} = \left\{ \lambda_{1,-1} \quad \lambda_{1,0} \quad \lambda_{1,1} \quad \lambda_{1,2} \quad \lambda_{1,3} \right\} \\
&\left\{ \lambda_{1,2,1} \quad \lambda_{1,2,2} \quad \lambda_{1,2,3} \quad \lambda_{1,2,4} \quad \lambda_{1,2,5} \right\} = \left\{ \lambda_{2,-1} \quad \lambda_{2,0} \quad \lambda_{2,1} \quad \lambda_{2,2} \quad \lambda_{2,3} \right\} \\
&\left\{ \lambda_{2,2,1} \quad \lambda_{2,2,2} \quad \lambda_{2,2,3} \quad \lambda_{2,2,4} \quad \lambda_{2,2,5} \right\} = \left\{ \lambda_{2,0} \quad \lambda_{2,1} \quad \lambda_{2,2} \quad \lambda_{2,3} \quad \lambda_{2,4} \right\}
\end{aligned} \tag{6.25}$$

From the methodology:

- in dimension x_1:

$$\begin{aligned}
&w_{1,1} = w_{2,1}, \quad \leftarrow \left(\lambda_{1,1,2} = \lambda_{2,1,2} \right) \\
&w_{1,2} = w_{2,3}, \quad \leftarrow \left(\lambda_{1,1,4} = \lambda_{2,1,4} \right) \\
&w_{2,2} = \frac{(\lambda_{1,3} - \lambda_{1,4}) w_{1,1} - (\lambda_{1,3} - \lambda_{1,2}) w_{1,2}}{(\lambda_{1,2} - \lambda_{1,4})}
\end{aligned} \tag{6.26}$$

- in dimension x_2:

$$w_{1,2} = w_{2,1} \tag{6.27}$$

Revisiting the tensor product between splines, one has that (the notation used now is for multivariate sub-models):

$$\begin{aligned}
&w_{1,1,2} = w_{2,1,1} \\
&w_{1,2,2} = w_{2,3,1} \\
&w_{2,2,1} = \frac{\lambda_{1,3} - \lambda_{1,4}}{\lambda_{1,2} - \lambda_{1,4}} w_{1,1,2} - \frac{\lambda_{1,3} - \lambda_{1,2}}{\lambda_{1,2} - \lambda_{1,4}} w_{1,2,2}
\end{aligned} \tag{6.28}$$

In this way, the output is given as:

$$y = y_1 + y_2 = \begin{bmatrix} N^1_{1,2,1}N^1_{1,2,2} \\ N^1_{1,2,1}N^2_{1,2,2} + N^1_{2,2,1}N^1_{2,2,2} + \dfrac{\lambda_{1,1}-\lambda_{1,2}}{\lambda_{1,0}-\lambda_{1,2}} N^2_{2,1,2}N^1_{2,2,2} \\ N^2_{1,2,1}N^1_{1,2,2} \\ N^2_{1,2,1}N^2_{1,2,2} + N^3_{2,2,1}N^1_{2,2,2} - \dfrac{\lambda_{1,1}-\lambda_{1,2}}{\lambda_{1,0}-\lambda_{1,2}} N^2_{2,2,1}N^1_{2,2,2} \end{bmatrix}^T \begin{bmatrix} w_{1,1,1} \\ w_{1,1,2} \\ w_{1,2,1} \\ w_{1,2,2} \end{bmatrix} + \begin{bmatrix} N^1_{2,2,1}N^2_{2,2,2} \\ N^2_{2,2,1}N^2_{2,2,2} \\ N^3_{2,2,1}N^2_{2,2,2} \end{bmatrix}^T \begin{bmatrix} w_{2,1,2} \\ w_{2,2,2} \\ w_{2,3,2} \end{bmatrix} \tag{6.29}$$

It is clear that (6.29) is equal to equation (6.11).

Considering the methodology described in section 6.2, it is obvious that the complexity of the B-Spline model is strongly dependent on the number and size of the merges. This section describes the methodology for computing the complexity from a given grid partition.

Assume a bi-dimensional grid, with inputs x_1 and x_2. Also, assume order $\{k_1, k_2\}$ and a number of interior knots $\{r_1, r_2\}$, in each dimension. Thus, there will be $\{r_1+1, r_2+1\}$ cells in each dimension.

6.3 Computing the partitioned grid complexity

Considering the methodology described in section 6.2, it is obvious that the complexity of the B-Spline model is strongly dependent on the number and size of the merges. This section describes the methodology for computing the complexity from a given grid partition.

Assume a bi-dimensional grid, with inputs x_1 and x_2. Also, assume order $\{k_1, k_2\}$ and a number of interior knots $\{r_1, r_2\}$, in each dimension. Thus, there will be $\{r_1+1, r_2+1\}$ cells in each dimension.

6.3.1 Single merges

Denote the size for merge Mi in input dimensions x_1 and x_2 as ($s_{1,i}$,$s_{2,i}$). The complexity of this merge is given by (k_1*k_2). As it can be seen, despite of the merges size, the complexity is given solely by the order of the splines.

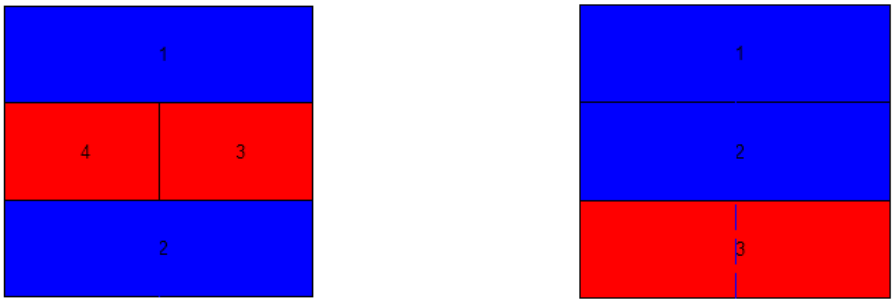
However, a grid partition complexity with one merge is also dependent on the size of the merge, i.e.:

$$o(s_{1,i}, s_{2,i}, k_1, k_2, r_1, r_2) = o(k_1, k_2, r_1, r_2) - o(s_{1,i}, s_{2,i}) = (k_1 + r_1)\times(k_2 + r_2) - \left[(s_{1,i} - 1)\times k_2 + (s_{2,i} - 1)\times k_1\right] \tag{6.30}$$

The first term on the right hand side of (6.30) is the complexity of the full grid and the second term represents the number of splines common between merge Mi and other sub-models.

6.3.2 Multiple merges

This situation is a little more complex since the complexity depends also on the relative positioning of the merges. Consider two distinct grids with two merges of the same size (the merges are identified by the blue colored cells):



a) b)

Fig. 6.13. Two distinct grid partitions using two merges of the same size; complexity of grid in a) is larger than that of grid in b).

Considering linear splines in both dimensions and assuming the same conclusions as in the single merges case, then the complexity would be given as:

$$o(m_1,n_1,m_2,n_1,k_1,k_2,r_1,r_2)=o(1,2,1,2,k_1,k_2,1,2)= \\ =(k_1+1)(k_2+2)-\left[(2-1)\times k_2+(1-1)\times k_1+(2-1)\times k_2+(1-1)\times k_1\right]=12-4=8 \quad (6.31)$$

But, this is true one considers the partition in a).

For the partition in b) the complexity is given as:

$$o(m_1,n_1,m_2,n_1,k_1,k_2,r_1,r_2)=o(1,2,1,2,k_1,k_2,1,2)= \\ =(k_1+1)(k_2+2)-\left[(2-1)\times k_2+(1-1)\times k_1+(2-1)\times k_2+(1-1)\times k_1-1\right]=12-3=9 \quad (6.32)$$

This happens because the same interior knot is removed in both merges and its corresponding removed spline(s) projection onto the other dimensions covers the input domain of the other merge. Therefore, the complexity expression needs reformulation.

A final example, with five merges is analyzed next.

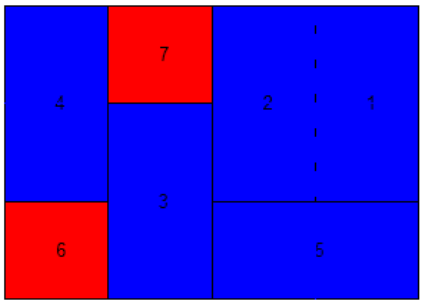


Fig. 6.14. Sample of a Grid partition with five merges.

Consider:

- $\{k_1, k_2\} = \{2,2\}$:

$$o(1,2,1,2,1,2,1,3,2,1,2,2,3,2) = (3+2)(2+2) -$$
$$\left[\begin{matrix}(1-1)\times 2+(2-1)\times 2+(1-1)\times 2+(2-1)\times 2+(1-1)\times 2+(2-1)\times 2+\\ +(1-1)\times 2+(3-1)\times 2+(2-1)\times 2+(1-1)\times 2-1\end{matrix}\right] =$$
$$= 20-9 = 11$$

- $\{k_1, k_2\} = \{3,2\}$:

$$o(1,2,1,2,1,2,1,3,2,1,3,2,3,2) = (3+3)(2+2) -$$
$$\left[\begin{matrix}(1-1)\times 2+(2-1)\times 3+(1-1)\times 2+(2-1)\times 3+(1-1)\times 2+(2-1)\times 3+\\ +(1-1)\times 2+(3-1)\times 3+(2-1)\times 2+(1-1)\times 3-3\end{matrix}\right] =$$
$$= 24-11 = 13$$

6.3.3 General expression

All the above observations can be explained as follows. Consider two grids as given in the next figure.

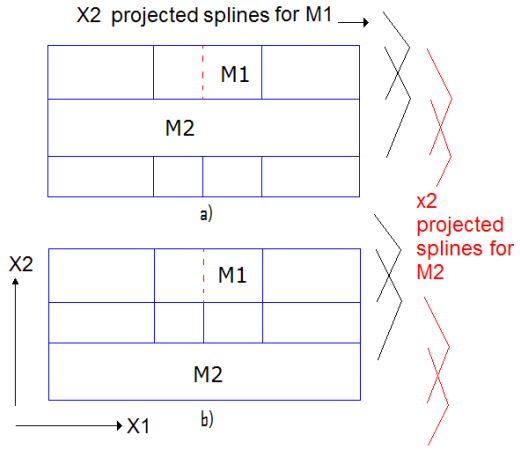


Fig. 6.15. Sample of two grids with two equal merges, but of distinct complexity; the slashed line represents the removed interior knot in merge M1 and M2; triangular splines are considered for dimension x_2.

From Fig. 6.15-a) it is clear that the same interior knot is removed in both merges M1 and M2, but M2 is contiguous to M1, for dimension x_2. In Fig. 6.15-b), the same interior knot is removed, but the two merges are no longer contiguous.

Notice that when the interior knot is removed from M1, all basis functions projected into the other dimensions are removed, as well. Now, if another merge occurs in the same dimension and with the same interior knot, an extra number of splines will be removed.

Consider that the splines are of order k_1 and k_2, respectively in dimension x_1 and x_2.

In Fig. 6.15-b), there are $2*k_2$ less functions than the original grid. However, in Fig. 6.15-a), there are $2*k_2-(k_2-1)= k_2+1$ less functions than the original grid. This can be explained by the fact that a total of (k_2-1) splines among the projected set of functions, are common to both merges.

In order to obtain a general expression one needs to introduce the vertices location for every merge. Thus, for merge Mi, its location is given by (for a bi-dimensional grid):

$$\left\{\mathbf{L}_i, \mathbf{U}_i\right\} = \left\{(x_i, y_i), (x_f, y_f)\right\} \tag{6.33}$$

Extending to an *n*-dimensional grid, the complexity is given by:

$$o\left(s_{:,1}, s_{:,2}, s_{:,n}, k_1, k_2, ..., k_n, r_1, r_2, ..., r_n\right) = \prod_{i=1}^{n}\left(k_i + r_i\right) - \sum_{j=1}^{n_m}\left[\prod_{g=1}^{n}\left(k_g + s_{g,j} - 1\right) - \prod_{g=1}^{n} k_g\right] +$$

$$+ \sum_{\substack{i=1 \\ j=1...n_m \\ j\neq i \\ g=1...n}}^{n_m} \max_{(i,j)\neq(j,i)} \left(\sum_{\mathbf{L}_i(g)+m=\left\{\mathbf{L}_j(g)+1...\mathbf{U}_j(g)-1\right\}}^{s_{g,i}-1} \prod_{\substack{h=1 \\ h\neq g \\ \left|\left\{\mathbf{L}_i,\mathbf{U}_i\right\},\left\{\mathbf{L}_j,\mathbf{U}_j\right\}\right|<k_h-1 \\ s_{h,i}>1 \\ s_{h,j}>1}}^{n} \left[k_h - 1 - \left|\left\{\mathbf{L}_i, \mathbf{U}_i\right\}, \left\{\mathbf{L}_j, \mathbf{U}_j\right\}\right|\right] \right) \tag{6.34}$$

where n_m stands for the number of submodels with merges, $\left|\left\{\mathbf{L}_i, \mathbf{U}_i\right\}, \left\{\mathbf{L}_j, \mathbf{U}_j\right\}\right|$ denotes the distance (number of cells) between Mi and Mj in dimension *h*, and $\left\{\mathbf{L}_j(g)...\mathbf{U}_j(g)\right\}$ is the range of coordinates from the lower to the upper end of the merge, in dimension *g*.

The third term in expression (6.34) returns the maximum number of splines common to other merges and associated with the removal of the interior knot located at coordinate $\mathbf{L}_i(g)+m$ in merge Mi.

6.4 Simulation results

With this example the benefits related to the application of the proposed methodology are shown. At this time, the structure is assumed to be already known. Thus, this example uses a BSNN model with 3 interior knots in the x_1 dimension and 1 interior knot in the x_2 dimension.

The performance of the proposed approach is assessed on the two inputs non-linear function:

$$f(x_1, x_2) = \frac{\sin(10x_1)}{50x_1}(3x_2)^3, \tag{6.35}$$

where $x_1, x_2 \in [-1,1]$.

The simulation data was obtained from (6.35), using 67% of the patterns for training (452), and 33% as validation data (225). The patterns were chosen randomly, ensuring that all input space is covered, and no common patterns exist in the two sets.

The interior knots position for dimension x_1 is $\{\lambda_{1,1},\lambda_{1,2},\lambda_{1,3}\}=\{-0.5,0,0.5\}$ and for dimension x_2, $\lambda_{2,1}=0$. The number and location of interior knots were determined by inspection of the corresponding 3D plot, as the only concern was to have a base model with reasonable approximation ability.

Fig. 6.16 shows the 3D plot for function $f(x_1, x_2)$.

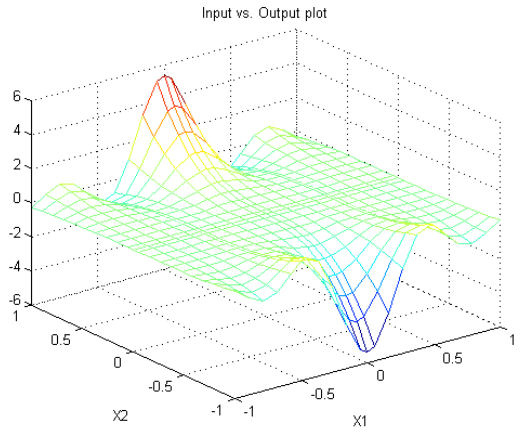


Fig. 6.16. 3D plot of function $f(x_1, x_2)$

The objective is to compare the performance of the grid partitioning approach with five different partitioning, induced by the same knots. For all cases, the weight vector solution was computed using the least-squares solution.

Because each grid partitioning possibility corresponds to a different BSNN model, the models will be designated as Model_i, where *i*=1..5. The regular grid model is designated as Base Model.

Fig. 6.17 presents four different partitioning. Fig. 6.18 indicates that the same approximation to the target function is accomplished using diverse partitions. In this specific case, the merged grids reach similar performance as the regular grid model.

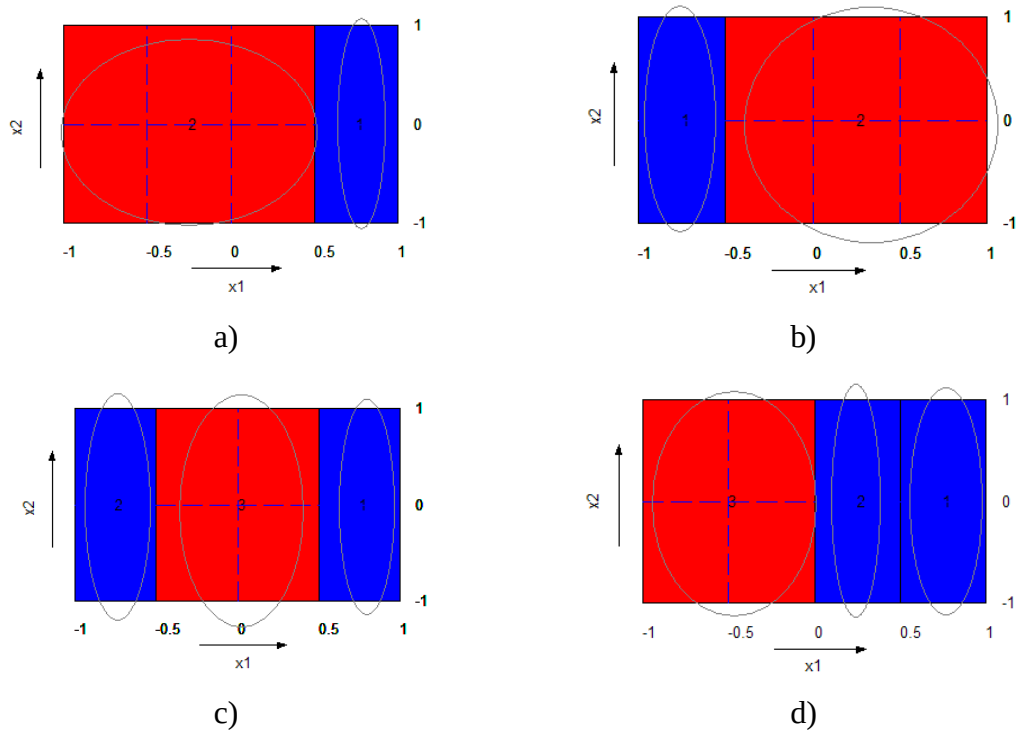


Fig. 6.17. A bi-dimensional input space partitioning with: a) one merge (Model_1); b) one merge (Model_2); c) two merges (Model_3); d) two merges (Model_4)

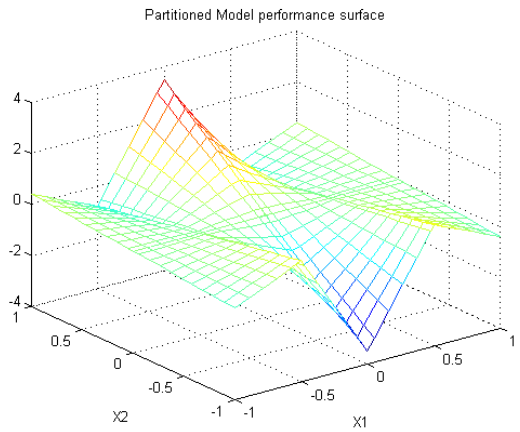


Fig. 6.18. Training data input-output 3D plot for Base Model, Model_1, Model_2 and Model_3.

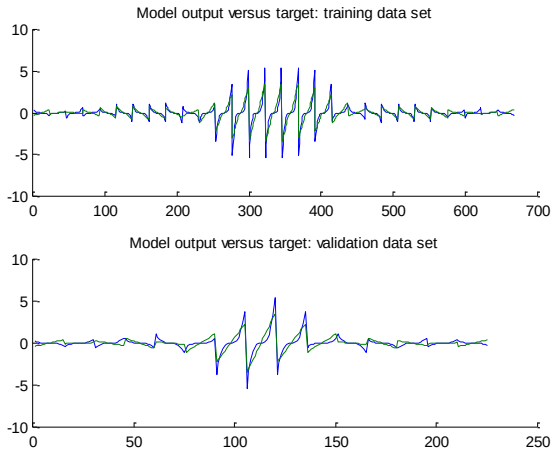


Fig. 6.19. Target versus model output for training and validation sets for Model_1, Model_2, Model_3 and Base Model

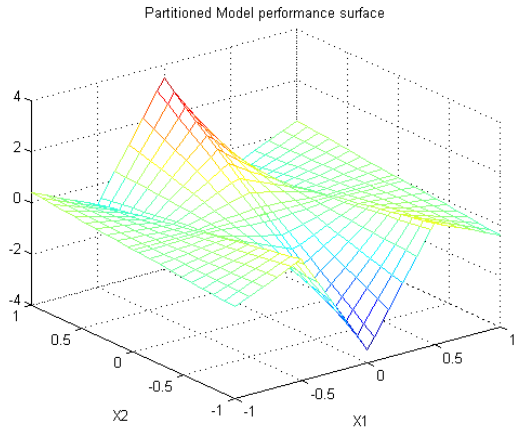


Fig. 6.20. Training data input-output 3D plot for Model_4.

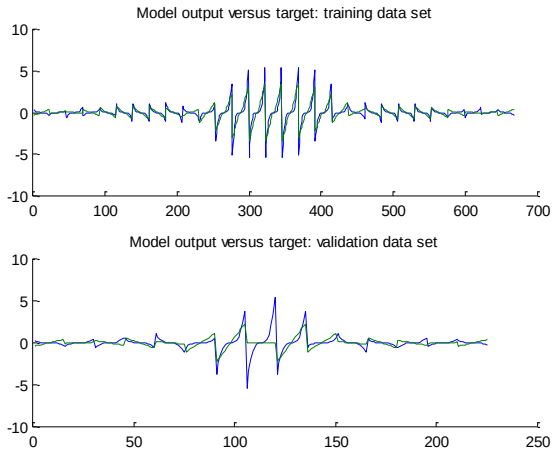


Fig. 6.21. Model_4 output versus target; top lines: training data; bottom lines: validation set; notice how a misplaced merge induces a big deviation between this model's output and the target (validation set).

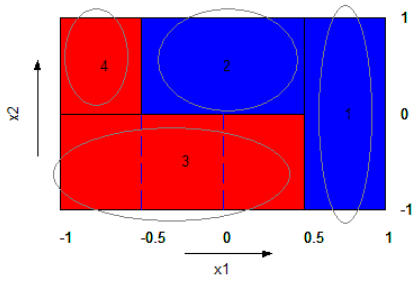


Fig. 6.22. Another bi-dimensional input space partitioning with two merges (Model_5); functionally speaking, it may be regarded as a model consisting of the sum of 4 sub-models, where each sub-model input domain is delimited by the set of grid cells nominated by circles 1-4.

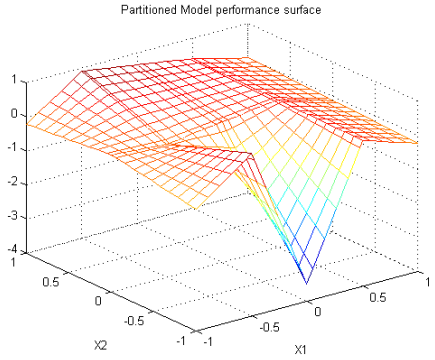


Fig. 6.23. Training data input-output 3D plot for Model_5.

TABLE 6.1. MODELS PERFORMANCE FOR LINEAR SPLINES

Model	MSE	MSEv
Base	$2.1x10^{-1}$	$2.3x10^{-1}$
1 to 3	$2.1x10^{-1}$	$2.3x10^{-1}$
4	$2.1x10^{-1}$	$5.7x10^{-1}$
5	$5.6x10^{-1}$	$4.7x10^{-1}$

From the results in Table 6.1 and in Fig. 6.17 to Fig. 6.23, it can be identified that some domain decompositions, less complex, achieve a performance in terms of function approximation comparable to the full grid case.

As expected, not all grid partition possibilities reach good results (Models 4 and 5), clearly depending on how the merges employed agree with the behavior of the function to approximate.

6.5 Automatic process of grid partitioning

This section proposes the use of GEP for determining the best partitioning of a grid, under the assumptions of the new domain decomposition. It starts by outlining the main steps employed for generating the merges from a fixed grid, including the use of a genetic algorithm. In this respect, both the proposed chromosome encoding and recombination operators are described. Only then, automatic grid partitioning is described by illustrating the linear chromosome encoding using GEP. Experiments for several data sets are carried out where both the grid partitioning and the BSNN model structure are searched using GEP. Additionally, the LM algorithm is used for optimizing a merged grid structure.

6.5.1 Generating merges

The following subsection describes the procedure used to employ partitioning of the proposed domain decomposition from subsection 6.2. It considers a BSNN model with a fixed structure, i.e., where the number of interior knots is fixed.

Two approaches are considered in the next two subsections. In the first, the problem of obtaining the set of all partitions based on a starting grid is addressed. In the latter, an evolutionary algorithm is developed for finding the best partitions, thus avoiding combining all possibilities.

6.5.1.1 Procedure for obtaining all possible combinations of cells

Consider N_i cells along the i^{th} dimension, and $\lambda_{i,j}$ as the j^{th} (j=0…N_i) parameter (or interior knot if the BSNN structure is considered) from the i^{th} dimension.

The outline for the procedure which automatically returns the set of all merges for a given regular grid is summarized by the following steps:

1. Generate $N_i - j$ cell merges, for dimension i, for j=0… $N_i - 2$. Denote this set of partitions $S_i = \left\{ \bigcup P_j \right\} : P_j \doteq \bigcup_{\substack{m=0 \\ j>1}}^{N_i - j} x_i \,|\, x_i \in [\lambda_{i,m}, \lambda_{i,m+j}]$,

   where P_j denotes the partition merge of size j.
2. For the i^{th} dimension, generate combinations of merged cells of distinct cells and sizes. This gives all the partition possibilities in the i^{th} dimension.

   Thus, $S_i = \left\{ \prod P_j \right\}$.

3. Extend the examples obtained by the previous step, in order to include every stripe from other dimensions. Therefore: $S_i^{x_i} = \left\{ \prod S_j \right\}$. These examples refer to merges done solely in dimension *i*.
4. Obtain the overall possibilities, using steps II and III, applying the tensor product among all dimensions: $S = \left\{ \prod S_i^{x_i} \right\}$

Using the above procedure, examples obtained from these steps can be *invalid* partitions (as the partition does not belong to the grid's induced set of parameters). To avoid these unwanted possibilities, it is essential to examine if the generated partition includes every parameter from the grid. Thus, a post evaluation of the partitioned grid is needed.

6.5.2 Evolving merges using a genetic algorithm

In contrast to the preceding section, rather than obtaining all of the partitions from an initial grid, the objective now is to automatically generate the most adequate partitions.

Thus, this section describes the fundamental aspects related to the genetic operators used in a conventional genetic algorithm. Additionally, in order to return appropriate input space partitioning, it also focuses the restrictions on the genetic operations.

6.5.2.1 Encoding

To develop the genetic algorithm, the partition information has to be encoded by a chromosome. In the following, it is assumed that all grids represent the input lattice from a BSNN.

Consider a bi-dimensional input space grid of size 2*6, where there is a merge of size 1*2 located at the top left-hand side. Notice that in the figure, each color grouping refers to a different SD, so there are 3 SDs – yellow (with a merge); green (with a one interior knot in x_2); blue (with one interior knot in x_1 and 3 interior knots in x_2).

The data structure which is used for encoding the chromosome is:

$$\bigcup_{sub=1}^{SD} \mathbf{SC}_{sub}, \mathbf{SC}_i = (\mathbf{sc}_{i,1}, \mathbf{sc}_{i,2}, ..., \mathbf{sc}_{i,n}) \wedge \mathbf{sc}_{i,j} = \begin{Bmatrix} (sc_{i,j,1}, sc_{i,j,2}, ..., sc_{i,j,m}) : sc_{i,j,k} \in [1, N(j)] \wedge \\ \wedge sc_{i,j,k} < sc_{i,j,k+1}, m \le N(j) \end{Bmatrix} \tag{6.36}$$

where *n* stands for the number of input variables, *SD* the number of sub-models, $\mathbf{sc}_{i,j}$ the list of knots for the j^{th} dimension of SD_i.

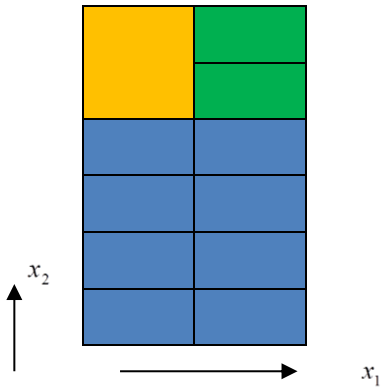


Fig. 6.24. Grid with 2*6 cells

To exemplify, the data structure for the partition in Fig. 6.24 would be given as:

$$\bigcup_{sub=1}^{3} \mathbf{SC}_{sub} = \left(\{1,2,3\},\{1,2,3,4,5\}\right)\bigcup\left(\{1,2\},\{5,7\}\right)\bigcup\left(\{2,3\},\{5,6,7\}\right) \tag{6.37}$$

The merge is identified by a break in the sequence of coordinates, as is the case of {5,7}, where coordinate 6 is missing.

The information in (6.37) can be easily related to the BSNN knots vector:

$$\begin{gathered}\left(\left\{sc_{i,1,1}, sc_{i,1,2}, \ldots, sc_{i,1,m}\right\}, \left\{sc_{i,2,1}, sc_{i,2,2}, sc_{i,2,n}\right\}\right) \\ \downarrow \\ \left(\left\{\lambda_{i,1,0}, \lambda_{i,1,1}, \ldots, \lambda_{i,1,m-1}\right\}, \left\{\lambda_{i,2,0}, \lambda_{i,2,1}, \ldots, \lambda_{i,2,n-1}\right\}\right)\end{gathered} \tag{6.38}$$

where $\lambda_{i,j,k}$ refers to the k^{th} knot in the j^{th} input variable from the i^{th} sub-model.

6.5.2.2 Initial population

In the initial population, all individuals start by one merge, with an arbitrary size and location in the grid. All remaining cells are grouped together in sub-models, providing a final chromosome as in(6.36). This way, no restrictions need to be set on the individuals belonging to the initial population.

6.5.2.3 Mutation

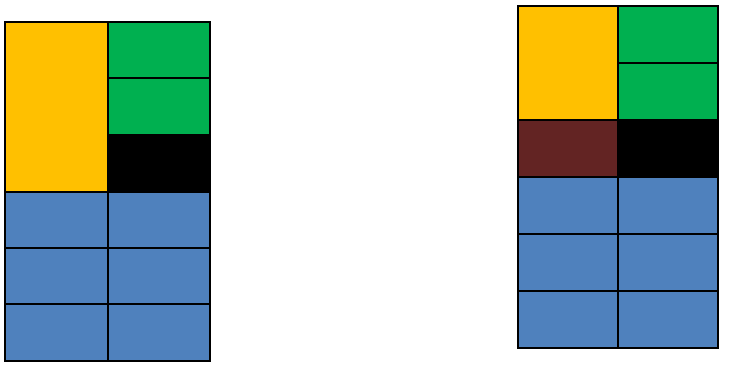
With the mutation operator one aims to change the way that the cells are linked together inside the *SD* sub-models; this is done for two main streams:

- the first stream can be related to addition (type 2) and elimination of edges (type 1), denoted as *Edge Mutation*.
- the second stream deals with the resizing the *SD* submodels – denoted as *Size Mutation*.

6.5.2.3.1 Edge mutation

Addition of edges can be applied only on merged sub-models. For a merged sub-model, adding an edge on a particular dimension requires linking 2*(*n*-1) vertices, for an *n*-dimensional grid. Elimination is applied only on non-merged sub-models containing interior knots.

Examples of mutation are as follows. Consider an interior knot addition in the yellow colored merged sub-model:

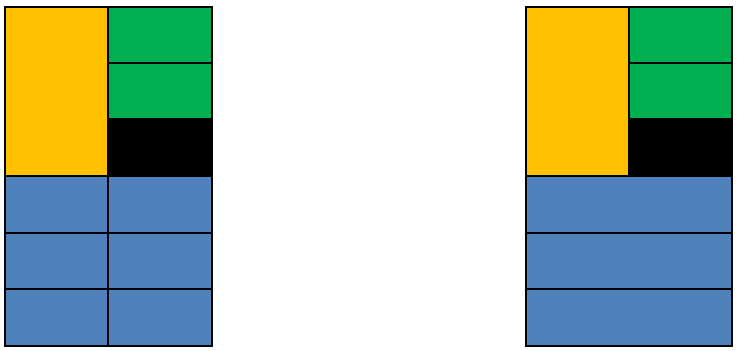


a) Before mutation

b)After mutation; notice that an appropriate grouping of the sub-models will yield only 3 sub-models

Fig. 6.25. Example 1 of edge mutation

On the other hand, removing the vertical edge on the blue colored sub-model:



a) Before mutation

b)After mutation

Fig. 6.26. Example 2 of edge mutation

Edge Mutation is employed by the one point strategy meaning that either type 1 or type 2 will be applied to each candidate.

6.5.2.3.2 Size mutation

Size mutation is applied to any of the sub-models. It either increases or decreases the sub-model size.

The procedure outline is as follows:

1. a part is randomly selected;
2. choose the dimension (dim) to be resized and which reference point (lower or upper vertex);
3. the final size corresponds to changing the reference vertex coordinates, accordingly to this new dimension size.

The next illustration shows as an example of the increase of the number of cells (the sub-model selected is indicated by X):

a) Before mutation b) after size increase mutation

Fig. 6.27. Example 1 of size mutation

In this example, consider the coordinates for the initial grid:

$$\bigcup_{sub=1}^{4} \mathbf{SC}_{sub} = (\{1,2,4,5\},\{1,2\}) \bigcup (\{1,3\},\{2,3,4\}) \bigcup (\{3,5\},\{2,3\}) \bigcup (\{3,4,5\},\{3,4\})$$

The X part corresponds to coordinates $(\{3,5\},\{2,3\})$. Considering an increase of 1 cell in dimension 2, and upper vertex, the coordinates are given as: $(\{3,5\},\{2,4\})$.

The final coordinates for the grid, after mutation is then:

$$\bigcup_{sub=1}^{3} \mathbf{SC}_{sub} = (\{1,2,4,5\},\{1,2\}) \bigcup (\{1,3\},\{2,3,4\}) \bigcup (\{3,5\},\{2,4\})$$

The next illustration shows as an example, the decrease of the number of cells (the submodel selected is indicated by X).

a) Before mutation b) after size decrease mutation

Fig. 6.28. Example 2 of size mutation

In this example, consider the coordinates for the initial grid:

$$\bigcup_{sub=1}^{4} \mathbf{SC}_{sub} = (\{1,2,4,5\},\{1,2\}) \bigcup (\{1,3\},\{2,3,4\}) \bigcup (\{3,5\},\{2,3\}) \bigcup (\{3,4,5\},\{3,4\})$$

The X part corresponds to coordinates $(\{1,3\},\{2,3,4\})$. Considering decrease of 1 cell in dimension 1, and upper vertex, the coordinates are given as: $(\{1,2\},\{2,3,4\})$.

Notice that a new sub-model is generated (red colored cells), which can be grouped together with the green colored cells sub-model.

This way, the final coordinates for the grid, after size decrease mutation is then:

$$\bigcup_{sub=1}^{4} \mathbf{SC}_{sub} = (\{1,2,4,5\},\{1,2\}) \bigcup (\{1,2,3\},\{2,3,4\}) \bigcup (\{3,5\},\{2,3\}) \bigcup (\{3,4,5\},\{3,4\})$$

The mutation rate parameter p_m, forces an individual in the population to have its sub-models changed.

6.5.2.4 Crossover

The aim of the crossover operation is to exchange genetic material between two parents.

In the beginning, two parents are randomly selected and two parts X_1 and Y_1 (merges) from parents 1 and 2 are exchanged (see Fig. 6.29).

The final grid for Offspring 2 is straightforward. The case for offspring 1 is a little more special and is represented by a “gap”.

Three approaches have been considered in the case of crossover, which will be explained next.

The first two represent a solution to the gap problem. The last one uses another principle for crossover but uses the same partitioning as approach 2.

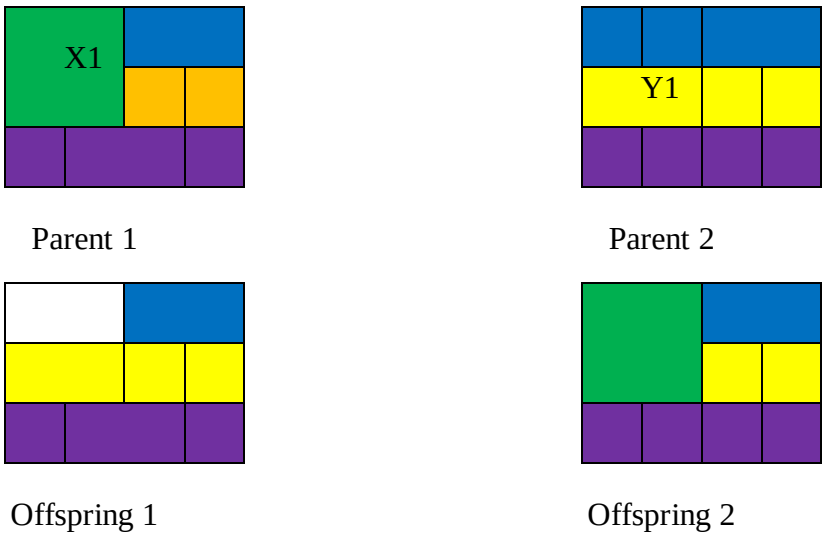
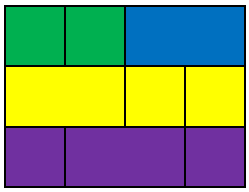


Fig. 6.29. Illustration of the crossover operation

6.5.2.4.1 Approach 1

In this approach, whenever a new partition spans another partition partially, (as it happens above, where the sub-model identified by part X1 is only partially changed by the partition moving from parent 2) the "gap" is filled by a regular grid:



This approach, however, has the disadvantage of increasing the model complexity every time.

6.5.2.4.2 Approach 2

With this approach, the "gap" is replaced by the previous cells partitioning increasing the chance of complexity reduction (which only depends on the parents' cells partitioning):

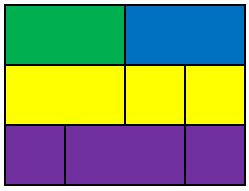
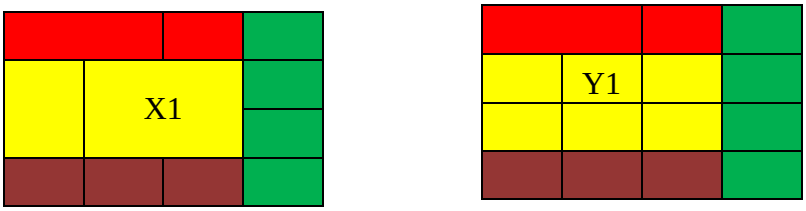


Fig. 6.30. Approach 2 in the crossover operation

6.5.2.4.3 Approach 3

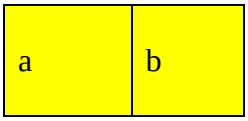
Considering two parents, with similar partitions:



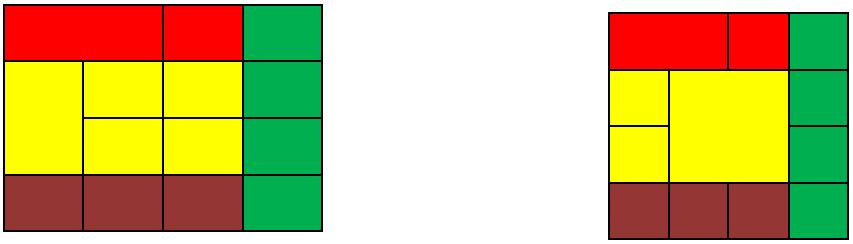
Parent 1 Parent 2

With this proposed approach to crossover, only a fraction of the selected part can be moved to the other parent.

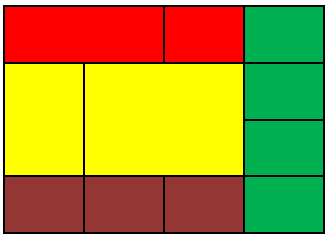
If X1 is composed by fractions *a* and *b*:



This approach will proceed with the random selection of one of the combinations of fractions; in this case, there are three possible combinations:



Possibility 1 Possibility 2



a) Possibility 3

Fig. 6.31. The 3 possibilities for Approach 3 in the crossover operation

6.5.2.5 Remarks to mutation and crossover

In order to ensure that the resulting grid partition is induced by the set of initial knots, trial and test is performed. If an invalid candidate is obtained, it is replaced by its corresponding parent.

6.5.3 Finding the best grid partitioning

Section 6.3 made it clear how a huge number of combinations of merges can be associated with one grid. The hardest task still, is to select the one partitioning which comes up with the best training criterion.

In the current section the process of evolving a B-spline neural network (BSNN) using GEP is proposed. The flowchart of GEP can be seen in chapter 2, subsection 2.5.1.3. GEP has shown to be able to surpass genetic programming by more than four orders of magnitude in some very complex problems [122]. On the other hand, it has also been used for generic feed-forward neural networks optimization [124].

Rather than using the genetic algorithm as described in the previous section, here, the GEP principles are used. The computer programs are represented as linear character strings of fixed length (the chromosomes) which, for subsequent fitness evaluation are expressed as expression trees (ETs) of different sizes and shapes.

6.5.3.1 Chromosome encoding

A one-gene chromosome is encoded as a string of a specific length L, split into two distinct parts, the head and the tail. Similar as used for Genetic Programming [59] or as in Bacterial Programming [179], the set of primitive functions is (*,+,/).

For a 3-dimensional problem, one example of the encoding of the chromosome is:

$$\underbrace{*+12+3/*}_{head}\underbrace{123122311}_{tail} \tag{6.39}$$

For the B-spline structure, the chromosome representation in (6.39) still does not give the specific information on which knots are associated to each input. To accomplish this, the proposed chromosome is augmented so that the data structure includes the following two fields:

- labels: character type string in which each element represents either the input or the primitive function - similar to (6.39).

- terminal: an array of length L where each element contains either the information on the variable identification, splines order and knots selection, or, void if the element points out to a

primitive function. Therefore, a terminal definition holds the variable identification, its spline order (k) and the corresponding knot vector values:

$$\text{terminal} = [x, k, \boldsymbol{\lambda}] \tag{6.40}$$

Encoding of a terminal for input 1, spline order k and knots {-2,-1,0,0.5,1,2} would be given as $ter\min al = [1, k, \{-2, -1, 0, 0.5, 1, 2\}]$. A primitive function is encoded as void, i.e., $\text{terminal} = []$.

An example of such a data structure would be:

$$\begin{aligned} &\text{chr.labels} = \{+1*212111\} \\ &\text{chr.terminal} = \{[], [1, k_2, \{\boldsymbol{\lambda}_2\}], [], [2, k_4, \{\boldsymbol{\lambda}_4\}], [1, k_5, \{\boldsymbol{\lambda}_5\}], [2, k_6, \{\boldsymbol{\lambda}_6\}], ...[1, k_9, \{\boldsymbol{\lambda}_9\}]\} \end{aligned} \tag{6.41}$$

where k_i and λ_i are the spline order and knot vector for the i^{th} label.

6.5.3.2 GEP operators

In GEP, and during population replication, the chromosomes are modified by means of operators such as mutation and transposition, affecting the structure of the corresponding phenotype (the corresponding ET). As such, the evolution process produces sub-models of distinct input variables in the same way as described in section 6.2, as a result of having possible distint merged grids.

Therefore, this new information (at the terminal node) does not change the way that the original GEP is employed. With this chromosome encoding, identical ETs as the ones using the Bacterial Programming or GP are obtained. Thus, both selection and evaluation of individuals is carried out in the same way.

However, and in contrast to the original GEP where the chromosome length is fixed, in this case it can happen that its length may increase as a consequence of the fact that the B-spline model structure may require a valid corresponding phenotype (ET). In other words, after checking the B-Spline model structure for the j^{th} individual, a new phenotype is obtained and, therefore a new genotype is required. If the gene head length is not sufficient to hold the necessary number of function labels, then the chromosome length must be recomputed. This step is required for each one of the individuals since the head of the gene is of fixed size and equal for every individual.

6.5.3.3 Experimental studies

The results shown in this section use data from an inverse coordinate mapping for a two link robot manipulator. The input is the coordinates of the Cartesians coordinates of the arm's

end and the output is the corresponding angle at the end. This example has been used as a benchmark in several studies (see appendix A for more details).

The input data set is composed of 110 input patterns, 33 of which are used for validation purposes. The remaining ones are used as training patterns. Input samples selection is random and repetitions are not allowed.

A simple way to see the nonlinearity of the problem at hand is to draw a plot of the two inputs. The first figure uses the training data, the second, uses the validation data.

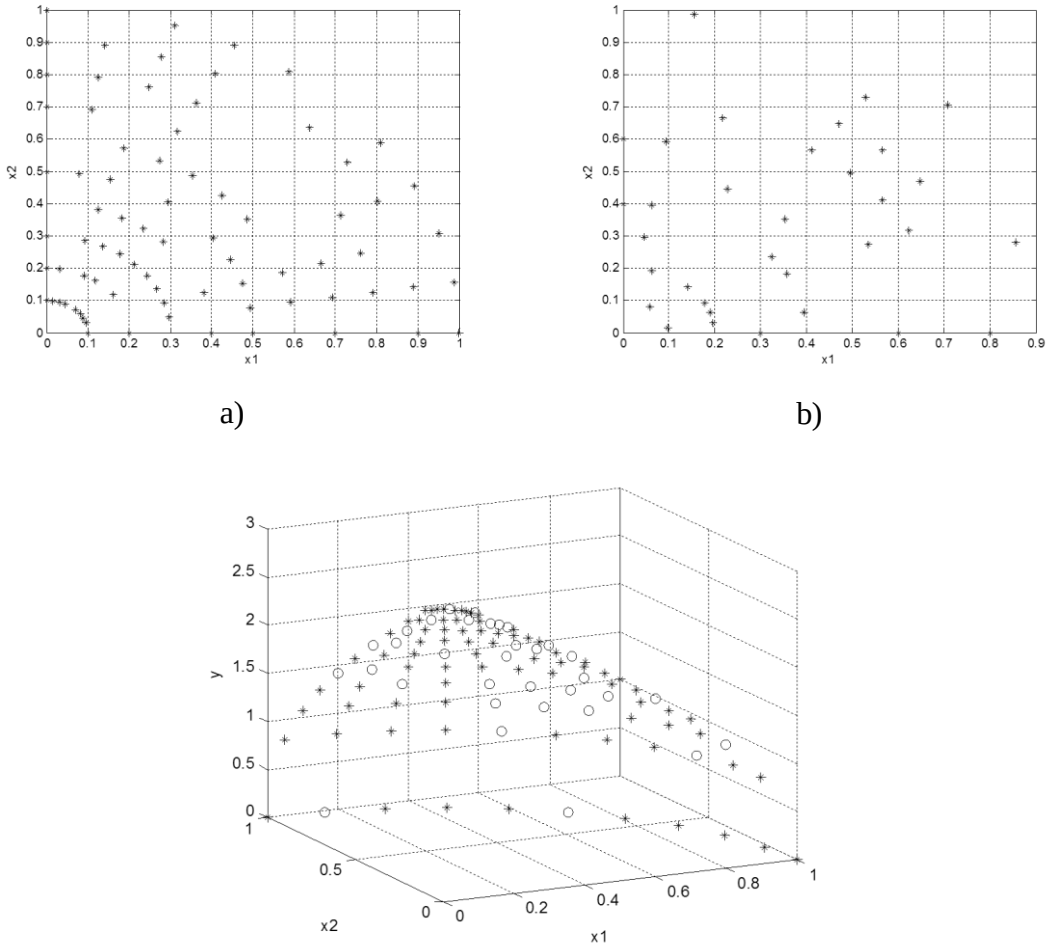


Fig. 6.32. Plot of training versus validation data: a) training data input x_1 versus input x_2; b) validation data input x_1 versus input x_2; input versus target for training ('*') and validation data ('o').

6.5.3.3.1 Performance of the mutation operator

To start, the following results are supported by a first version of the mutation operator (see section 6.5.2.3) in the way that only insertion or removal of interior knots is considered.

The initial grid (a regular grid) is bi-dimensional and corresponds to a size of 6*6 cells. Notice that the interior knots position is equidistant, that quadratic splines are used and so, the

model corresponding to this grid provides performance criteria values as given in the following table.

TABLE 6.2. PERFORMANCE CRITERIA FOR THE INITIAL MODEL

BIC	MSE	MSEv	Complexity	$\|\mathbf{w}\|$
-733	$1{,}98\text{x}10^{-6}$	31242	64	40449

Obviously, the model does not map correctly the validation input samples, although it has extremely good performance with the training data.

Next, the genetic algorithm was employed and used in three different runs in order to estimate the best partitioning of the grid. The performance criteria values along the generations are given in the tables below. Notice how the final grid for the first two runs is similar. In fact, the first generation best model from the 2nd run is the same as the best one from the first run.

The 3rd run exhibited a final model with very well located merges, because this model yields a value of Ψ_v much smaller.

All three runs provide final models with similar values of complexity, around 20% smaller than that from the full grid model.

In every run, complexity is converging to lower values. Thus, the algorithm is searching for parsimonious models. Though not shown, many other runs experimented also gave similar results where values for the BIC, Complexity and Ψ were of the same order.

TABLE 6.3. GENERATION BEST BIC INDIVIDUAL AND PERFORMANCE CRITERIA FOR RUN 1

Generation number	BIC	MSE	MSEv	Complexity	$\|\mathbf{w}\|$
1	-770	2,03x10-6	1,8x10-2	55	428
…	…	…	…	…	…
10	-770	2,03x10-6	1,8x10-2	55	428

TABLE 6.4. GENERATION BEST BIC INDIVIDUAL AND PERFORMANCE CRITERIA FOR RUN 2

Generation number	BIC	MSE	MSEv	Complexity	$\|\mathbf{w}\|$
1	-770	$2{,}03\text{x}10^{-6}$	$1{,}8\text{x}10^{-2}$	55	428
2	-770	$2{,}03\text{x}10^{-6}$	$1{,}8\text{x}10^{-2}$	55	428
3	-771	$2{,}25\text{x}10^{-6}$	$3{,}6\text{x}10^{-2}$	53	214
…	…	…	…	…	…
7	-771	$2{,}25\text{x}10^{-6}$	$3{,}6\text{x}10^{-2}$	53	214
8	-773	$2{,}08\text{x}10^{-6}$	$6{,}8\text{x}10^{-3}$	54	443
9	-780	$2{,}11\text{x}10^{-6}$	$4{,}3\text{x}10^{-2}$	52	436
10	-780	$2{,}11\text{x}10^{-6}$	$4{,}3\text{x}10^{-2}$	52	436

TABLE6.5. GENERATION BEST BIC INDIVIDUAL AND PERFORMANCE CRITERIA FOR RUN 3

Generation	BIC	MSE	MSEv	Complexity	‖w‖
1	-758	$2{,}11\text{x}10^{-6}$	$5{,}2\text{x}10^{-3}$	57	61
2	-750	$2{,}03\text{x}10^{-6}$	$1{,}5\text{x}10^{-2}$	55	284
3	-772	$2{,}11\text{x}10^{-6}$	$5{,}1\text{x}10^{-3}$	54	66
…	…	…	…	…	…
5	-780	$2{,}12\text{x}10^{-6}$	$3{,}6\text{x}10^{-2}$	52	131
8	-784	$2{,}12\text{x}10^{-6}$	$5{,}8\text{x}10^{-3}$	51	306
…	…	…	…	…	…
10	-780	$2{,}12\text{x}10^{-6}$	$5{,}8\text{x}10^{-3}$	51	306

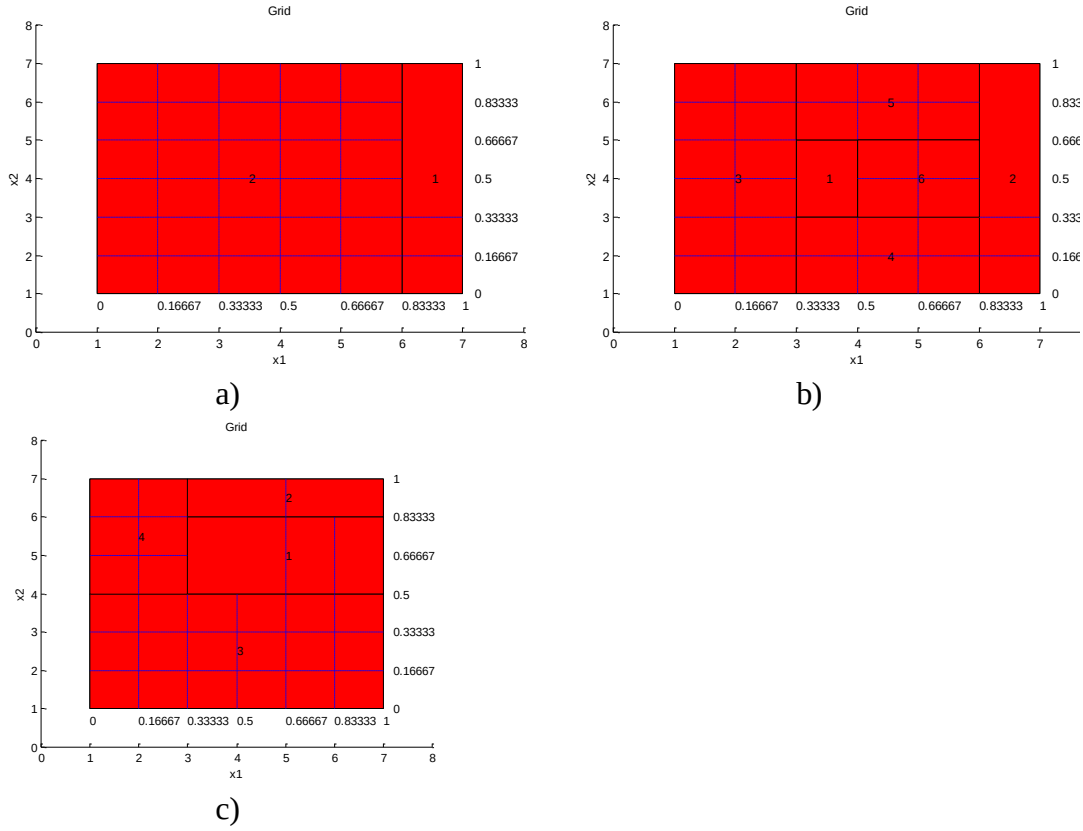


Fig.6.33. Grid partitioning for the final model of: a) run 1; b) run 2; c) run 3.

6.5.3.3.2 Performance of the genetic algorithm

Now, the results are supported by all premises of the genetic algorithm described in section 6.5.2. The algorithm parameters were set as shown in the next table.

TABLE 6.6. OPTIMIZATION PARAMETERS VALUES.

N_{ind}	N_{gen}	Mutation rate	%crossover
20	10	0.3	60%

The initial grid is a regular grid and corresponds to a size of 4*4 cells. Further, the interior knots position are equidistant, quadratic splines are used and the model corresponding to this grid has a performance given in the following table:

TABLE 6.7. REGULAR GRID PERFORMANCE CRITERIA.

BIC	MSE	MSEv	Complexity	‖**w**‖
-466	$3{,}1\text{x}10^{-4}$	$3{,}8\text{x}10^{-2}$	36	425

In order to investigate how well the genetic algorithm performs, several runs were carried out and the final model's performance from each run is summarized in the next table.

TABLE 6.8. BEST (LAST GENERATION) BIC INDIVIDUAL FOR SEVERAL RUNS AND CORRESPONDING PERFORMANCE CRITERIA.

Run	BIC	MSE	MSEv	Complexity	‖**w**‖
1	-487	$3{,}7\text{x}10^{-4}$	$3{,}2\text{x}10^{-2}$	29	338
2	-488	$4{,}8\text{x}10^{-4}$	$1{,}3\text{x}10^{-2}$	23	346
3	-487	$4{,}6\text{x}10^{-4}$	$1{,}2\text{x}10^{-2}$	24	319
4	-485	$5{,}0\text{x}10^{-4}$	$1{,}3\text{x}10^{-2}$	23	312
5	-487	$3{,}7\text{x}10^{-4}$	$1{,}1\text{x}10^{-2}$	28	294
6	-492	$4{,}1\text{x}10^{-4}$	$1{,}0\text{x}10^{-3}$	25	283
7	-485	$5{,}0\text{x}10^{-4}$	$1{,}2\text{x}10^{-2}$	23	322
8	-487	$4{,}4\text{x}10^{-4}$	$1{,}3\text{x}10^{-2}$	25	338

The results show that in average the final BIC criterion is better than the one given by the regular grid model. Likewise, the same happens with all the other criterions except with the MSE. Actually, improvement on the MSE_V value is such that, sometimes is decreases to values as low as 30% of the one obtained with the regular grid model.

Despite the previous good results, the BIC value is greater than the lowest possible, which is the one that can be attained with a more complex regular grid model.

So, in the following, another grid size is used which shows that the lowest BIC criterion obtained by the genetic algorithm obtains a much better MSEv than the regular grid model.

Consider the initial full grid corresponding to a size of 6*7 cells. Equidistant knots are employed, quadratic splines are used and the performance of the model corresponding to this grid is given in the following table.

TABLE 6.9. REGULAR GRID PERFORMANCE CRITERIA.

BIC	MSE	MSEv	Complexity	‖**w**‖
-747	$1{,}05\text{x}10^{-6}$	87	72	1297

Similarly, 3 runs were carried out and the final model's performance from each run is summarized in the next table.

TABLE 6.10. BEST (LAST GENERATION) BIC INDIVIDUAL FOR SEVERAL RUNS AND CORRESPONDING PERFORMANCE CRITERIA.

Run	BIC	MSE	MSEv	Complexity	$\|\|\mathbf{w}\|\|$
1	-800	$1{,}34x10^{-6}$	$1{,}1x10^{-1}$	55	604
2	-806	$1{,}14x10^{-6}$	$2{,}1x10^{-1}$	57	615
3	-797	$1{,}08x10^{-6}$	3,04	60	506

The results in Table 6.10 confirm that a better model in terms of BIC and MSEv criteria is often obtained, with the partitioned grid. This is a very promising result since a lower complexity model returns well-conditioned models and better generalization ability, while keeping accuracy over the training data to similar levels of the full grid model.

6.5.3.3.3 Mackey Glass time series

This problem aims to predict the behavior of a chaotic time series (see appendix A), with prediction horizon of 6 time steps ahead.

This approach is designated as Mackey-Glass type 1, where only 4 of the 6 input variables are deemed to possess complete information about the series. This strategy has been adopted by several other researchers [184][185][186].

The input data set was the target of a linear scale and rotation mechanism that transforms the input space in a way that the input patterns will cover most of the input space [187]. This transformation consists of two steps:

- Find two parallel lines that include all data;
- Rotate the lines in a way that the input data are spread along the *y* axis in a maximum distance equal to the separation of the two lines. In the end, the objective is to leave the two lines parallel to the *x* axis

The data altogether are composed of 1000 input patterns, where the last 500 are used for validation purposes. The first set is used as training data set.

The next figure plots the desired output for the training and validation data sets.

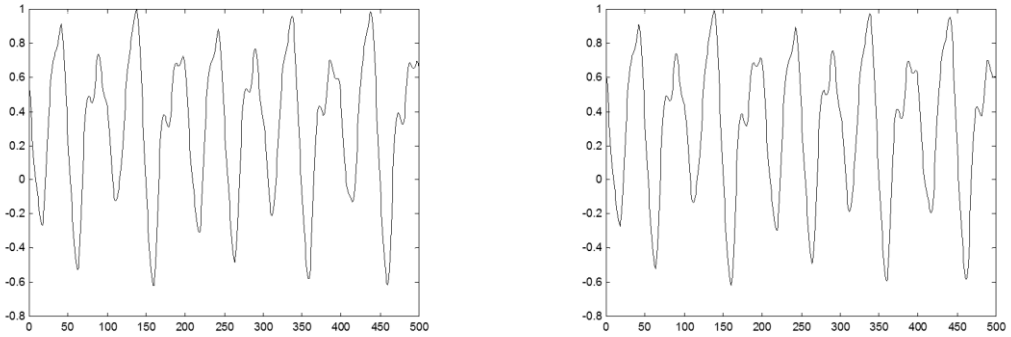


Fig.6.34. left) Target for training input patterns; right) target for validation input patterns

6.5.3.3.4 Estimating the initial model

Instead of arbitrarily imposing the number of cells for this problem (which is of 4 dimensions), the evolutionary algorithm based on GEP for BSNN was adopted.

The best optimization parameters found for the GEP algorithm were used here and are listed in the next table. Because multi-genic chromosome was not used, parameters such as gene recombination rate and gene transposition are not defined.

TABLE 6.11. GEP OPTIMIZATION PARAMETERS

N_{ind}	N_{gen}	h_g	Intknots	Mutation rate	1PR rate
100	20	10	5	0.8	0.4
2PR rate	IS rate	IS Length	RIS rate	RIS Length	Terminal rate
0.4	0.2	[1 2 3]	0.4	[1 2 3]	[0.1 0.05 0.1 0.3 0.3 0.15]

After 3 different runs, the best initial models (best outcome from each run) are summarized in the next table.

TABLE 6.12. DETERMINATION OF THE INITIAL MODEL STRUCTURE USING GEP. SUMMARY OF THE BEST BIC INDIVIDUAL FOR SEVERAL RUNS AND CORRESPONDING PERFORMANCE CRITERIA.

Run	Input variables	BIC	MSE	MSEv	Complexity	$\|\|\mathbf{w}\|\|$
1	3x2+4	-2853	$3{,}1x10^{-4}$	$1{,}2x10^{-3}$	80	809
2	4x3x2	-2602	$7{,}5x10^{-4}$	$7{,}6x10^{-4}$	160	1393
3	3+2x1x4	-2940	$2{,}2x10^{-4}$	$1{,}0x10^{-2}$	205	2067

There are 3 distinct structures considered in Table 6.12. The most complex one is given by run 3 representing the lowest BIC and MSEt criterions. Although representing the lowest BIC criterion (the selected training criterion) this model is by far the worst in terms of generalization, as can be observed from the values in MSEv. In this respect, the best model is

the model from run 2, where a better agreement of MSE and MSEv values is illustrated. Notice that all of these models are based on a full grid.

To investigate the utility of the new input decomposition, the genetic algorithm of section 6.5.2 was used to search a grid partitioning for all the best initial models listed in the last table. The optimization parameters used for that genetic algorithm were as follows.

TABLE 6.13. OPTIMIZATION PARAMETERS VALUES.

N_{ind}	N_{gen}	Mutation rate	%crossover
20	10	0.3	60%

The following tables summarize the results, showing the final models corresponding to the lowest BIC, after optimizing the grid.

TABLE 6.14. LAST GENERATION MODELS AND CORRESPONDING PERFORMANCE CRITERIA OPTIMIZING THE FULL GRID FROM RUN 1 IN TABLE 6.12

Run	BIC	MSE	MSEv	Complexity	$\|\|\mathbf{w}\|\|$
1	-3070	$1{,}43\text{x}10^{-3}$	$1{,}44\text{x}10^{-3}$	33	685
2	-3077	$1{,}46\text{x}10^{-3}$	$1{,}47\text{x}10^{-3}$	30	27
3	-3050	$1{,}42\text{x}10^{-3}$	$1{,}43\text{x}10^{-3}$	37	1661

TABLE 6.15. LAST GENERATION MODELS AND CORRESPONDING PERFORMANCE CRITERIA OPTIMIZING THE FULL GRID FROM RUN 2 IN TABLE 6.12

Run	BIC	MSE	MSEv	Complexity	$\|\|\mathbf{w}\|\|$
1	-3092	$1{,}25\text{x}10^{-3}$	$1{,}22\text{x}10^{-3}$	40	30
2	-3176	$1{,}0\text{x}10^{-3}$	$1{,}0\text{x}10^{-3}$	42	61
3	-3077	$1{,}2\text{x}10^{-3}$	$1{,}2\text{x}10^{-3}$	49	80

TABLE 6.16. LAST GENERATION MODELS AND CORRESPONDING PERFORMANCE CRITERIA OPTIMIZING THE FULL GRID FROM RUN 3 IN TABLE 6.12

Run	BIC	MSE	MSEv	Complexity	$\|\|\mathbf{w}\|\|$
1	-3604	$3{,}3\text{x}10^{-4}$	$3{,}6\text{x}10^{-4}$	65	449
2	-3641	$3{,}0\text{x}10^{-4}$	$3{,}0\text{x}10^{-4}$	69	25
3	-3682	$2{,}8\text{x}10^{-4}$	$2{,}9\text{x}10^{-4}$	66	20

Considering that the training criterion is the BIC, Table 6.14 to Table 6.16 show that using the genetic algorithm a better model is often obtained if grid partitioning is applied. Moreover, there is also a significant complexity reduction, since for some cases a reduction of 25% is reached (if the model from run 2 is considered).

Table 6.16 also shows that it is possible to obtain a model with better generalization while reducing dramatically the complexity. The best model of the third run achieves the best BIC criterion and MSEv, reaching much lower values of MSEv in comparison to any other of the models listed. This is even more remarkable if the network complexity is accounted for, yielding a reduction of nearly 70%.

6.5.4 Optimizing parameters in a merged grid using the Levenberg-Marquardt method

The previous section showed that a less complex model, with a good performance, could be obtained if a partitioned grid was found from the regular grid of an initial model. The procedure therein did not apply a local optimization technique to estimate the parameters of the grid (the interior knots in a B-spline NN). Therefore, the current section shows how to apply the Levenberg-Marquardt (LM) algorithm for optimizing a given fixed structured based on a partitioned grid.

The new training criterion, described in Section 2.3.5. will be used here. The equations introduced therein will be used, with the difference that they will be applied to a partitioned grid, therefore with a small complexity. In order to differentiate the two cases, the symbol ‘r’ will be applied to the partitioned case. As an example the Golub-Pereyra *Jacobian* eq. (2.86) will here be expressed as:

$$\mathbf{J}_{GP}^{r}=\left(\mathbf{I}-\boldsymbol{\Gamma}^{r}\left(\boldsymbol{\Gamma}^{r}\right)^{+}\right)^{-1}\left(\boldsymbol{\Gamma}^{r}\right)_{v}\left(\boldsymbol{\Gamma}^{r}\right)^{+}\mathbf{t}+\left(\left(\boldsymbol{\Gamma}^{r}\right)^{+}\right)^{T}\left(\left(\boldsymbol{\Gamma}^{r}\right)_{v}\right)^{T}\left(\mathbf{I}-\boldsymbol{\Gamma}^{r}\boldsymbol{\Gamma}^{r+}\right)^{-1}\mathbf{t} \tag{6.42}$$

6.5.4.1 Computing the *Jacobian* matrix

Consider the following grid partitioning:

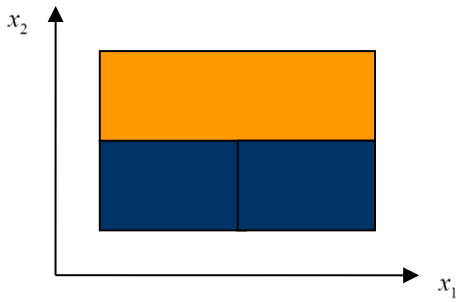


Fig. 6.35. A sample grid partition for a bivariate model

Notice there are two sub-models in the same input variables. The sub-model at the top (yellow) has a merge and the sub-model from below, in the blue cells with one interior knot in input x_1.

Previously it was shown that the model output was the sum of sub-model 1 and sub-model 2. But, to ensure output continuity the output would simply be:

$$y=\begin{bmatrix} N^1_{1,1,2}N^1_{1,2,2}+N^1_{2,1,2}N^2_{2,2,2}+N^2_{2,1,2}N^2_{2,2,2}\dfrac{\lambda_{1,3}-\lambda_{1,2}}{\lambda_{1,2}-\lambda_{1,3}} \\ N^1_{1,1,2}N^2_{1,2,2} \\ N^2_{1,1,2}N^1_{1,2,2}+N^3_{2,1,2}N^2_{2,2,2}-N^2_{2,1,2}N^2_{2,2,2}\dfrac{\lambda_{1,3}-\lambda_{1,2}}{\lambda_{1,2}-\lambda_{1,3}} \\ N^2_{1,1,2}N^2_{1,2,2} \\ N^1_{2,1,2}N^1_{2,2,2} \\ N^2_{2,1,2}N^1_{2,2,2} \\ N^3_{2,1,2}N^1_{2,2,2} \\ \\ \end{bmatrix}^T \begin{bmatrix} w_{1,1,1} \\ w_{1,1,2} \\ w_{1,2,1} \\ w_{1,2,2} \\ w_{2,1,1} \\ w_{2,2,1} \\ w_{2,3,1} \end{bmatrix} \tag{6.43}$$

But, the output can also be seen as:

$$y=y_1+y_2=\begin{bmatrix} N^1_{1,1,2}N^1_{1,2,2} \\ N^1_{1,1,2}N^2_{1,2,2} \\ N^2_{1,1,2}N^1_{1,2,2} \\ N^2_{1,1,2}N^2_{1,2,2} \end{bmatrix}^T \begin{bmatrix} w_{1,1,1} \\ w_{1,1,2} \\ w_{1,2,1} \\ w_{1,2,2} \end{bmatrix} + \\ + \begin{bmatrix} N^1_{2,1,2}N^1_{2,2,2} \\ N^1_{2,1,2}N^2_{2,2,2} \\ N^2_{2,1,2}N^1_{2,2,2} \\ N^2_{2,1,2}N^2_{2,2,2}\dfrac{\lambda_{1,3}-\lambda_{1,2}}{\lambda_{1,2}-\lambda_{1,3}}w_{1,1,1}-N^2_{2,1,2}N^2_{2,2,2}\dfrac{\lambda_{1,3}-\lambda_{1,2}}{\lambda_{1,2}-\lambda_{1,3}}w_{1,2,1} \\ N^3_{2,1,2}N^1_{2,2,2} \\ N^3_{2,1,2}N^2_{2,2,2} \end{bmatrix}^T \begin{bmatrix} w_{2,1,1} \\ w_{1,1,1} \\ w_{2,2,1} \\ 1 \\ w_{2,3,1} \\ w_{1,2,1} \end{bmatrix} \tag{6.44}$$

Now, consider the i^{th} pattern: $\mathbf{x}^{(i)}=\left\{(x_1,x_2)\in[I_{1,1,1}*I_{1,2,1}]\right\}$.

If the partial derivative of the output is taken in respect to $\lambda_{1,3}$, using (6.44) then,

$$J_i = \frac{\partial y_i}{\partial \lambda_{1,3}}$$

$$= \frac{\partial}{\partial \lambda_{1,3}} \begin{bmatrix} 0 + N^1_{2,1,2} N^2_{2,2,2}(x_i) + N^2_{2,1,2} N^2_{2,2,2}(x_i) \dfrac{\lambda_{1,3} - \lambda_{1,2}}{\lambda_{1,2} - \lambda_{1,3}} \\ 0 \\ 0 + 0 - N^2_{2,1,2} N^2_{2,2,2}(x_i) \dfrac{\lambda_{1,3} - \lambda_{1,2}}{\lambda_{1,2} - \lambda_{1,3}} \\ 0 \\ N^1_{2,1,2} N^1_{2,2,2}(x_i) \\ N^2_{2,1,2} N^1_{2,2,2}(x_i) \\ 0 \end{bmatrix}^T \begin{bmatrix} w_{1,1,1} \\ w_{1,1,2} \\ w_{1,2,1} \\ w_{1,2,2} \\ w_{2,1,1} \\ w_{2,2,1} \\ w_{2,3,1} \end{bmatrix}, \tag{6.45}$$

and can be written as

$$J_i = \frac{\partial N^1_{2,1,2} N^2_{2,2,2}(x_i) + N^2_{2,1,2} N^2_{2,2,2}(x_i) \dfrac{\lambda_{1,3} - \lambda_{1,2}}{\lambda_{1,2} - \lambda_{1,3}}}{\partial \lambda_{1,3}} w_{1,1,1} - \frac{\partial N^2_{2,1,2} N^2_{2,2,2}(x_i) \dfrac{\lambda_{1,3} - \lambda_{1,2}}{\lambda_{1,2} - \lambda_{1,3}}}{\partial \lambda_{1,3}} w_{1,2,1} +$$

$$+ \frac{\partial N^1_{2,1,2} N^1_{2,2,2}(x_i)}{\partial \lambda_{1,3}} w_{2,1,1} + \frac{\partial N^2_{2,1,2} N^1_{2,2,2}(x_i)}{\partial \lambda_{1,3}} w_{2,2,1}$$

(6.46)

An identical expression is obtained if equation (6.43) is used, i.e.,

$$J_i = \frac{\partial N^1_{2,1,2} N^1_{2,2,2}(x_i)}{\partial \lambda_{1,3}} w_{2,1,1} + \frac{\partial N^1_{2,1,2} N^2_{2,2,2}(x_i)}{\partial \lambda_{1,3}} w_{1,1,1} + \frac{\partial N^2_{2,1,2} N^1_{2,2,2}(x_i)}{\partial \lambda_{1,3}} w_{2,2,1} +$$

$$\frac{\partial N^2_{2,1,2} N^2_{2,2,2}(x_i) \dfrac{\lambda_{1,3} - \lambda_{1,2}}{\lambda_{1,2} - \lambda_{1,3}}}{\partial \lambda_{1,3}} w_{1,1,1} - \frac{\partial N^2_{2,1,2} N^2_{2,2,2}(x_i) \dfrac{\lambda_{1,3} - \lambda_{1,2}}{\lambda_{1,2} - \lambda_{1,3}}}{\partial \lambda_{1,3}} w_{1,2,1}$$

(6.47)

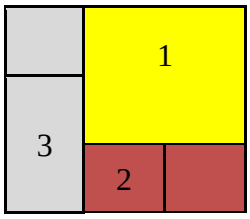
This way, to calculate the *Jacobian* matrix, using the proposed approach it will be required to compute the reduced linear weights vector only.

However, notice that a change in the nonlinear parameters' values imposes a linear weight value re-evaluation, otherwise the splines properties are not kept.

The next subsection describes the way to compute the *Jacobian* matrix, $\mathbf{J}$ and the partial derivative of the regression matrix in respect to the nonlinear parameters, $\left(\mathbf{\Gamma}^r\right)_v$.

6.5.4.2 Computation of J and $\left(\mathbf{\Gamma}^r\right)_v$

Suppose the following partitioned grid. Consider a BSNN with linear splines, for the sake of simplicity (triangular shape basis functions).



This partitioned grid above is made of 2 merged partitions. One identified by the label "1" and a second identified by label "2".

If a full grid is assumed, the complexity is 16. Decomposing the input into the 3 partitions shown, only 9 splines are required.

As previously seen, the output for this model is given by:

$$y\left(x_1, x_2\right) = \sum_{i=1}^{3} S_i\left(x_1, x_2\right) = \mathbf{\Gamma w} \tag{6.48}$$

In this example, $\mathbf{\Gamma}$ is of size (*m*, 16) and **w** is a column vector with 16 elements.

But, the output can be written as:

$$y = \mathbf{\Gamma}^1\mathbf{w}^1 + \mathbf{\Gamma}^2\mathbf{w}^2 + \mathbf{\Gamma}^3\mathbf{w}^3 \tag{6.49}$$

Or, as

$$\mathbf{y} = \sum_{r=1}^{3} \mathbf{\Gamma}^r \mathbf{w}_i^r \tag{6.50}$$

where $(...)^i$ represents the element of the i^{th} partition of a matrix or vector.

In this particular case, the size from $\left\{\mathbf{\Gamma}^i, \mathbf{w}^i\right\} (i = 1...3)$ is:

- {(m, 4), (4,1)};
- {(m, 6), (6,1)};
- {(m, 6), (6,1)}.

But for the partitioned grid model $\mathbf{\Gamma}^r$ and $\mathbf{w}^r$ are of sizes (N, 9) and (9,1), respectively.

Both *Jacobian* and $\left(\mathbf{\Gamma}^r\right)_v$ matrices need to be computed considering the 4 interior knots of the grid being of size (*m*, 4):

$$\mathbf{J} = [\mathbf{J}^{1,1}, \mathbf{J}^{1,2}, \mathbf{J}^{2,1}, \mathbf{J}^{2,2}],$$

$$\left(\mathbf{\Gamma}^r\right)_v = [\left(\mathbf{\Gamma}^r\right)_v^{1,1}, \left(\mathbf{\Gamma}^r\right)_v^{1,2}, \left(\mathbf{\Gamma}^r\right)_v^{2,1}, \left(\mathbf{\Gamma}^r\right)_v^{2,2}] \tag{6.51}$$

In (6.51), $(...)^{\dim,j}$ refers to the j^{th} interior knot from the dim^{th} dimension.

Algorithm 6.1. Computation of the *Jacobian* matrix

1. Calculate the derivatives of the basis functions. For partition p, (p=1…3), compute the basis functions output derivatives in respect to the p^{th} interior knot in the dim^{th} dimension and record them in matrix $\mathbf{\Gamma}'_p$.

2. Repeat step 1 for every partition p and define $\mathbf{\Gamma}'_{\dim,p} = \left[\mathbf{\Gamma}'_1, \mathbf{\Gamma}'_2, ..., \mathbf{\Gamma}'_m\right]$. Note that, here, $\mathbf{\Gamma}'_{\dim,p} = \left[\mathbf{\Gamma}'_1, \mathbf{\Gamma}'_2, ..., \mathbf{\Gamma}'_m\right]$ is still not of the reduced form.

2.1 Obtaining the reduced form. Using weights relations $\mathbf{w}_i^r = f(\mathbf{w}^1, \mathbf{w}^2, ..., \mathbf{w}^{SD})$, obtain the reduced form for $\mathbf{\Gamma}'_{\dim,p}, \left(\mathbf{\Gamma}'_{\dim,p}\right)^r$.

3. Calculate the Jacobian. Apply expression (6.42), where $\left(\mathbf{\Gamma}^r\right)_v = \left(\mathbf{\Gamma}'_{\dim,p}\right)^r$. Notice that this is the *Jacobian* for the p^{th} interior knot in the dim^{th} dimension.

4. Repeat steps 1-3 for every interior knot in the grid.

6.5.4.3 Empirical results

6.5.4.3.1 Data sets

Two data sets from the literature are used in this study to demonstrate the effectiveness of the LM in optimizing a partitioned grid.

In subsections 6.5.4.3.2 and 6.5.4.3.3 a nonlinear system of first order is used which is described in appendix A, section A.7. With this data set comparison between LM and BPA will be carried out first and then the same data set is used to evaluate the performance of the LM for a grid of larger size.

A second example tests the application of the LM algorithm for the Mackey-Glass Chaotic time series, type 1 (only 4 inputs are used).

In all examples, the LM and BP were executed for a maximum of N=20 iterations and $\tau = 10^{-4}$. The new training criterion was employed. The learning rate for the BP algorithm was set to $\eta = 10^{-3}$.

6.5.4.3.2 Comparison between LM and error backpropagation

This subsection compares the performance of LM and BP algorithms.

The structure of the BSNN model was chosen as described in the next table.

TABLE 6.17. STRUCTURE OF THE INITIAL MODEL

Input variables	Splines order	Complexity	Number of knots	Input domain
1x2	3x3	16	1+1	[-1,1]+[-1.465,1.496]

The partitioned grid considered has one merge as illustrated in the figure below.

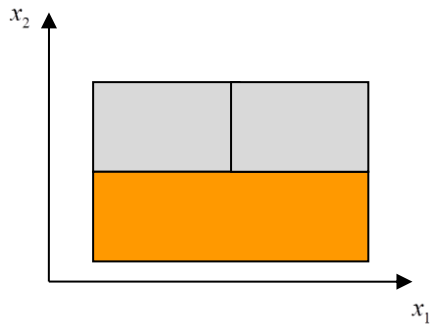


Fig. 6.36. The partitioned grid for the first example

There are two interior knots to be optimized. The next figure illustrates the performance surface for the optimization of the two interior knots in the grid above (denoted by x_1 and x_2 respectively).

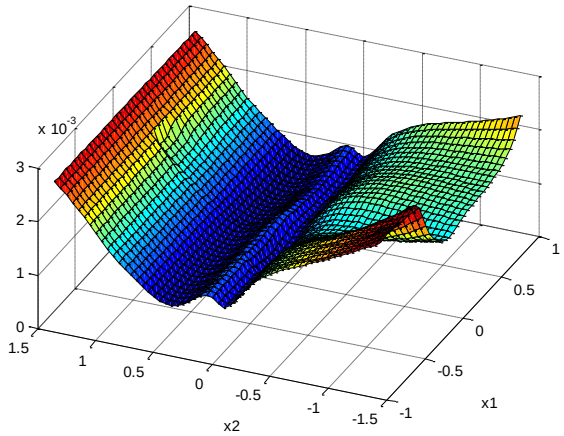


Fig. 6.37. Performance surface for the model given by grid from Fig. 6.36

The local optima are located in various regions of the input space. The lowest, however, are situated in the input range given by $(x_1, x_2) \in \left([-1,1], \{0,385\}\right)$ with values of approximately 7.96x10^{-4}.

Seven different starting interior knots were assumed. The next two tables summarize the results for the LM algorithm and BP.

TABLE 6.18. PARAMETERS ESTIMATION USING LM FOR 7 DIFFERENT STARTING POSITIONS

$\boldsymbol{\lambda}[1]$	$\boldsymbol{\lambda}[N]$	MSE[0]	MSE[*N*]
a) [0.0003, 0.0]	[0.1352 -0.0706]	0.0011	9.8511e-004
b) [-0.7, 1.2]	[-0.9994 0.7166]	0.0023	0.0012
c) [0.0, -1.2]	[-0.0971 -1.1851]	0.0016	0.0015
d) [0.5, 1.0]	[0.4715 0.3600]	0.0018	0.0008
e) [0.5, -1.2]	[0.4183 -1.2042]	0.0017	0.0017
f) [-0.0, -1.0]	[-0.9994 -0.4417]	0.0023	0.0018
g) [-0.5, -1.0]	[-0.2633 -0.0745]	0.0022	0.0009

The results shown in Table 6.19-*a*), *c*), *e*) and *g*) show that local optima are reached. The global optima are reached for case *d*). The cases illustrated by *b*) and *f*) present a peculiar situation, since convergence is not obtained and one of the parameters is likely to converge for -1.

TABLE 6.19. PARAMETERS ESTIMATION USING BP FOR 7 DIFFERENT STARTING POSITIONS

$\boldsymbol{\lambda}[1]$	$\boldsymbol{\lambda}[N]$	MSE[0]	MSE[*N*]
a) [0.0003, 0.00]	[0.0002 0.0608]	0.0011	0.0012
b) [-0.70 1.2]	[-0.7764 1.1169]	0.0023	0.0021
c) [0.00 -1.2]	[-0.0495 -1.2040]	0.0016	0.0016
d) [0.50 1.0]	[0.3633 0.9092]	0.0018	0.0016
e) [0.50 -1.2]	[0.4701 -1.2013]	0.0017	0.0017
f) [-0.60 -1.0]	[-0.6081 -0.9957]	0.0023	0.0023
g) [-0.50 -1.0]	[-0.4984 -0.9957]	0.0022	0.0022

BP convergence is too slow. The learning rate could be adjusted to a larger value, still the results would not be comparable to the ones obtained by the LM above.

6.5.4.3.3 Optimizing a fixed structure

In the current subsection the LM is used to optimize a grid of size 3*4, defined as the union of 3 sub-models with linear splines in dimension x_1 and cubic splines in dimension x_2. Only one of the sub-models has a merge in dimension x_1.

The LM was executed for a maximum of N=10 iterations and $\tau = 10^{-3}$. The new criterion was used. The performance of the algorithm will be observed for 3 different initial interior knots positions.

The table below summarizes the performance results.

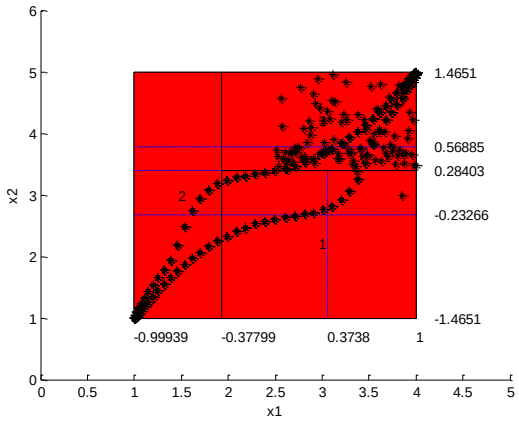
TABLE 6.20. PERFORMANCE CRITERIA USING LM

Grid	a)	b)	c)
N	10	8	10
$\boldsymbol{\lambda}[1]$	[-0.3329 0.3335 -0.7326 0.0000 0.7326]	[0 0.3335 -0.6000 0.0000 0.7326]	[0.1000 0.3335 -0.8000 0.0000 0.7326]
$\boldsymbol{\lambda}[N]$	[-0.3809 0.4022 -0.1864 0.1824 0.6212]	[-0.0308 0.7710 -0.2062 0.2063 0.9102]	[0.5185 0.5598 -0.3604 -0.2758 -0.0045]
MSE[0]	$9.1\text{x}10^{-3}$	$8.9\text{x}10^{-3}$	$8.2\text{x}10^{-3}$
MSE[N]	$8.5\text{x}10^{-3}$	$8.2\text{x}10^{-3}$	$4.8\text{x}10^{-3}$
MSEv[0]	206	9.33	$2.4\text{x}10^{-1}$
MSEv[N]	$3.81\text{x}10^{-1}$	137000	$7.2\text{x}10^{-3}$
BIC	-1771	-1782	-2000

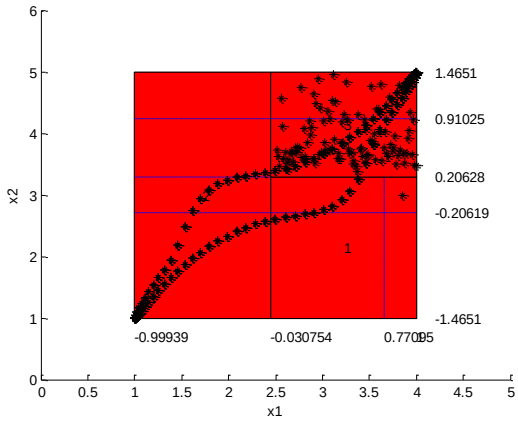
Though each case starts from distinct initial points, the initial MSE value is identical. The final MSE value is also very similar except for case *c*), which is better. One interesting issue is the value of the final MSEv. In general it decreases along the optimization but in case of *b*), it ends with a very high value.

The next figure presents the final grids returned by the LM for each one of the three cases.

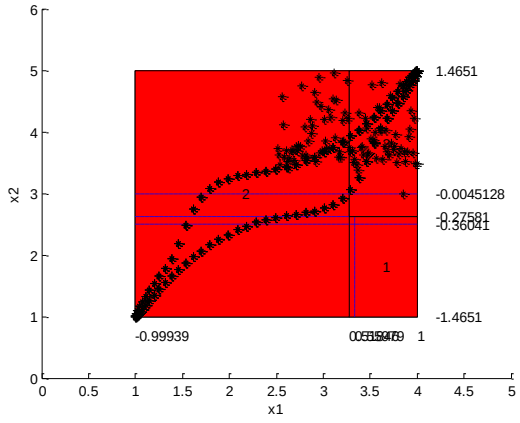
The results show how different starting knots positions lead to distinct final positions, yielding BSNN models of various partitioned grids. Fortunately, some of these partitioned grids present a reasonable input data approximation, either for training and validation. This way, one may find interesting to employ a multi-objective strategy based on cross-validation using a validation data set and a test data set, focusing on the tradeoff between training, validation and model complexity.



a)



b)



c)

Fig. 6.38 Final grids after optimizing grid parameters using LM for three distinct initial points

6.5.4.3.4 Mackey-Glass time series

In this subsection the Mackey-Glass time series, used in 1.5.3.3.3 is employed. The LM is used to optimize interior knots from a BSNN model with two sub-models. The structure of the BSNN model is shown in the next table.

TABLE 6.21. STRUCTURE OF THE INITIAL MODEL

X	***k***	Complexity	nKnots	Input domain
1x3+2	4x2+2	34	2x1+2	[-1,1]x[-0.5,0.5]+[-1,1]

A plot between input dimensions x_1 and x_3 input data is sketched in Fig. 6.39.

The following table shows the results obtained when applying the LM for optimizing the full grid model. In the table, the first element in the cell for the interior knots, refers to the

first interior knot from input x_1, and the 3rd is the first interior knot from input x_3. The last two interior knots are from the 2nd sub-model (input x_2).

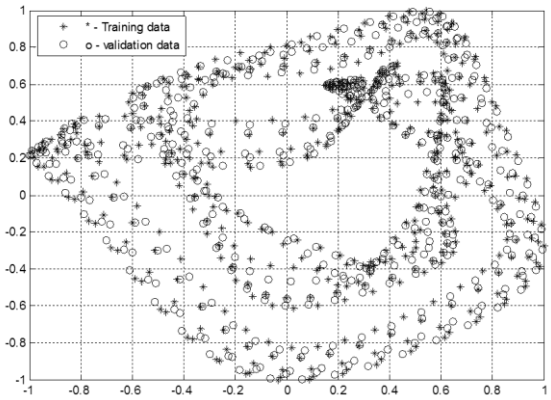


Fig. 6.39. Plot of x_1 versus x_3 for the Mackey-Glass time series

TABLE 6.22. PERFORMANCE CRITERIA FOR A MODEL WITH A FULL GRID USING LM

$\boldsymbol{\lambda}[1]$	[-0.3333, 0.3333, -0.5000, 0, 0.5000, -0.3333, 0.3333]
$\boldsymbol{\lambda}[N]$	[-0.2409, 0.1830, -0.4672, -0.1068, 0.5179, -0.1763, 0.0711]
InitialMSE	1.9×10^{-3}
FinalMSE	1.8×10^{-3}(5 iterations)
InitialMSEv	5.6047e-004
FinalMSEv	5.5081e-004

Next, the LM was applied for optimizing four different grid partitioning as shown in the next figure.

Despite many other partitioned grids could have been chosen, from the results it is clear that two of the partitioned grids tried (the first two) obtain even better results than the full grid. In general, the interior knots are well optimized since their final positions yields models with better mapping abilities (lower MSEv values).

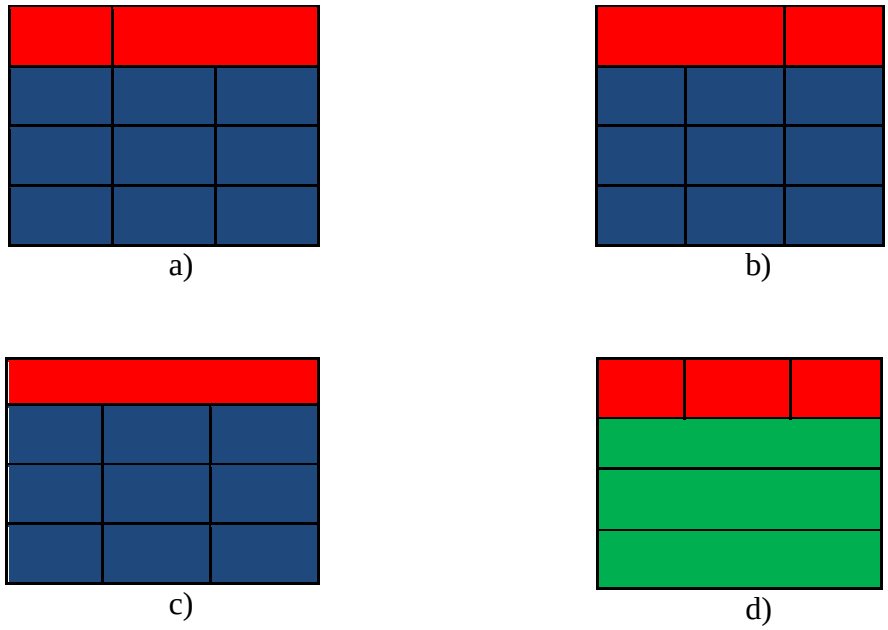


Fig. 6.40. Partitioned grid: a) with 1 merge of size 2*1; b) with 1 merge of size 2*1; c) with one merge of size 3*1; d) with 3 merges of size 3*1

The performance of the LM algorithm is summarized in the next table.

TABLE 6.23. PARAMETERS ESTIMATION FOR THE FULL GRID USING LM

Grid	a)	b)	c)	d)
N	8	5	10	10
$\boldsymbol{\lambda}[N]$	[0.1785, 0.4890, -0.2349, 0.0265, 0.5652, -0.1703, 0.0483]	[-0.241, 0.2357, -0.4360, -0.1002, 0.5446, -0.1760, 0.0659]	[0.0972, 0.1073, -0.3897, 0.1137, 0.7376, -0.2172, 0.0520]	[0.1695, 0.2052, -0.2814, 0.0960, 0.1066, -0.2212, 0.0927]
MSE[0]	5.43e-4	1.9e-3	$3.8x10^{-3}$	3.4e-3
MSE[N]	1.2e-3	5.76e-4	1.2e-3	1.3e-3
MSEv[0]	2.5e-3	1.8e-3	2.8e-3	3.2e-3
MSEv[N]	5.31e-4	5.68e-4	1.2e-3	1.3e-3

6.6 Conclusions

In this chapter a new input domain decomposition was introduced with the objective of obtaining accurate and parsimonious models. For the B-spline neural network, the most common methodology used is the ASMOD algorithm. Typically, this algorithm obtains models of high complexity. This is mainly due to the use of a full grid in the input space.

In the literature methods based on *k-d* tree partitioning have already been applied to similar models and to fuzzy systems. The idea is, rather than using orthogonal axis split, to use a sparse grid where the coupling of the input data is better exploited. Exploiting this approach, a

new input decomposition based on a merged grid was described. Results have shown that this new decomposition not only produces models with a lower complexity, but it also improves the accuracy in terms of generalization.

This scheme, however, requires finding the appropriate grid partitioning which can be cumbersome, especially for high dimensional problems. A genetic algorithm was developed for this purpose and results have shown to provide good results. Additionally, the Levenberg-Marquardt algorithm was applied to the optimization of this type of decomposition. Results have shown that LM when used for optimizing a fixed structure improves the model accuracy.

These last four chapters were dedicated to the design of BSNN models, where several methodologies have been applied. Because BSNN models are functionally equivalent to fuzzy systems (according to some assumptions), all these techniques can be directly employed in such systems. If those assumptions are not met then other strategies have to be undertaken. If the structure determination for a Mamdani-type fuzzy system, based on data, is to be performed global search methods such as evolution-based are the most appropriate. The next chapter investigates the performance of variants of the bacterial evolutionary algorithm for a Mamdani-type FS.

Chapter 7

BACTERIAL ALGORITHMS FOR FUZZY SYSTEMS

7.1 Introduction

The previous chapters applied methods inspired by evolutionary processes to the training of BSNN. As shown, these methodologies proved to be efficient. All of those techniques can be applied to a fuzzy system in the context of neuro-fuzzy learning schemes. As noted in chapter 2, section 2.4, this is valid under certain conditions. In this chapter it is assumed that those assumptions do not hold although the fuzzy system considered is a Mamdani-type FS, the antecedent parts are defined by trapezoidal membership functions, and the inference system does not employ the required conjuction operators. As the main purpose here is to extract the optimal fuzzy rule-base two evolutionary algorithms are compared.

Therefore, in this chapter, the Levenberg-Marquardt (LM) algorithm from [131] is improved (the rule-base partitioning is no longer Ruspini) and is applied in conjunction with the Bacterial Evolutionary algorithm (BEA) [126].

The combination of the evolutionary and the local-search methods is usually referred to as memetic algorithm [188][189]. So, the combination of the bacterial evolutionary algorithm and the LM is called Bacterial Memetic Algorithm (BMA).

The BMA algorithm is then improved in order to allow the determination the number of fuzzy rules, as well.

This chapter presents performance results with this kind of memetic algorithm for fuzzy rule extraction where the bacterial algorithm is improved with the LM technique. This hybrid scheme is then compared with the original BEA.

The class of membership functions investigated is the trapezoidal one as it is general enough and widely used.

This chapter, which is an extension of [190][191][192][193], is organized as follows.

It starts by describing the proposed encoding to implement the BEA for the fuzzy system, in section 7.2. Because the LM will be used in the evolutionary process, the mathematical background necessary to apply the LM algorithm to the Mamdani-type fuzzy system, is outlined in section 7.3.

The purpose of section 7.4 is to provide the reader with the concepts of the bacterial memetic algorithm. So, it includes the outline of the algorithm and the updates required for the evolutionary operators to determine the number of rules automatically. Because sections 7.3 and 7.4 refer to two design approaches, section 7.5 shows a comparison between the performance of the original bacterial evolutionary algorithm (BEA) and the LM algorithm from section 7.3, section 7.6 compares the BEA algorithm with the bacterial memetic algorithm (from section 7.4) and section 7.7 compares the improved BEA with the improved BMA in the quest for the optimal number of rules.

Finally, conclusions are drawn in section 7.8.

7.2 Fuzzy system

The system to optimize is a Mamdani fuzzy system as described in section 2.2.2.6.1. It is probably the most used type of fuzzy system when a fuzzy model is required for control purposes, since it benefits from property of interpretability.

The purpose is to find the optimal fuzzy rule base to a pattern set. Thus, the parameters of the fuzzy rules, i.e., the breakpoints of the trapezoids must be encoded in the bacterium. According to the BEA algorithm described in subsection 2.5.1.4, a segment part will be representing a fuzzy rule:

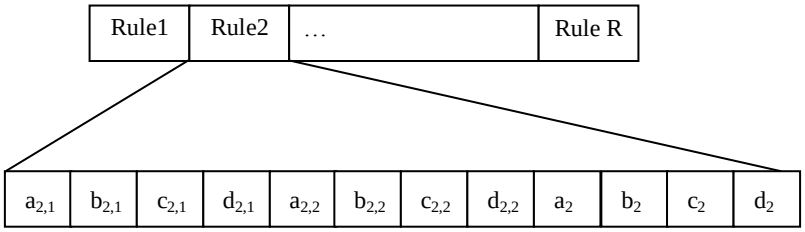


Fig. 7.1. Encoding of the fuzzy rules in BEA

The antecedents of the i^{th} fuzzy rule are $\{a_{i,1}, b_{i,1}, c_{i,1}, d_{i,1}\}$ and $\{a_{i,2}, b_{i,2}, c_{i,2}, d_{i,2}\}$, while the consequents are $\{a_i, b_i, c_i, d_i\}$. Fig. 7.2 shows a trapezoid in the i^{th} rule.

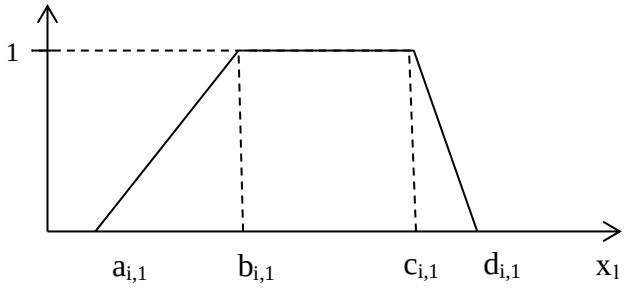


Fig. 7.2. Trapezoidal membership function in the antecedent part of the i^{th} rule

7.3 The training algorithm

The task of the nonlinear optimization technique is to determine the breakpoints of the membership functions. For that purpose, the minimization criterion is related only with the quality of the fitting. The training criterion that will be employed is the usual *Sum-of-the-Square of the Errors* (SSE). The algorithm applied is the LM from chapter 2, subsection 2.3.4.

The *Jacobian* matrix $\mathbf{J}$ is computed in a pattern-by-pattern basis and is given by,

$$\mathbf{J}[k] = \left[\frac{\partial y(\mathbf{x}^{(p)})[k]}{\partial \mathbf{z}[k]} \right] \tag{7.1}$$

where the vector ***z*** contains all membership functions' parameters (all breakpoints in the membership functions), and *k* is the iteration variable.

7.3.1 *Jacobian* computation

Regardless of the method used and of the training criterion employed, the *Jacobian* matrix with respect to the parameters in the rules has to be computed. The computation of the *Jacobian* will be shown below, in a pattern by pattern basis. Eq. (7.1) can be written as follows:

$$\mathbf{J} = \left[\frac{\partial y(\mathbf{x}^{(p)})}{\partial a_{11}} \; \frac{\partial y(\mathbf{x}^{(p)})}{\partial b_{11}} \cdots \frac{\partial y(\mathbf{x}^{(p)})}{\partial a_{12}} \cdots \frac{\partial y(\mathbf{x}^{(p)})}{\partial d_1} \cdots \frac{\partial y(\mathbf{x}^{(p)})}{\partial d_R} \right] \tag{7.2}$$

The number of columns of $\mathbf{J}$ will be *4*(n+1)*R*, where *n* is the number of input variables and *R* is the number of rules.

In (7.2), *y* is the output of the fuzzy system when the COG defuzzification method is applied (please see chapter 2, section 2.2.2.6.1 for more details).

Applying the derivative chain rule,

$$\frac{\partial y(\mathbf{x}^{(p)})}{\partial a_{ij}} = \frac{\partial y}{\partial s_i} \frac{\partial s_i}{\partial \mu_{ij}} \frac{\partial \mu_{ij}}{\partial a_{ij}}$$

$$\frac{\partial y(\mathbf{x}^{(p)})}{\partial b_{ij}} = \frac{\partial y}{\partial s_i}\frac{\partial s_i}{\partial \mu_{ij}}\frac{\partial \mu_{ij}}{\partial b_{ij}}$$

$$\frac{\partial y(\mathbf{x}^{(p)})}{\partial c_{ij}} = \frac{\partial y}{\partial s_i}\frac{\partial s_i}{\partial \mu_{ij}}\frac{\partial \mu_{ij}}{\partial c_{ij}}$$

$$\frac{\partial y(\mathbf{x}^{(p)})}{\partial d_{ij}} = \frac{\partial y}{\partial s_i}\frac{\partial s_i}{\partial \mu_{ij}}\frac{\partial \mu_{ij}}{\partial d_{ij}} \tag{7.3}$$

As the degree of firing of the i^{th} rule uses the *min* as the *t-norm*, s_i depends on the membership functions, and each membership function depends only on four parameters (breakpoints). So, the derivatives of s_i will be:

$$\frac{\partial s_i}{\partial \mu_{ij}} = \begin{cases} 1, \ if \quad \mu_{ij} = \min\limits_{k=1}^{n} \mu_{ik} \\ 0, \ otherwise \end{cases} \tag{7.4}$$

The derivatives of the membership functions will be calculated as follows:

$$\frac{\partial \mu_{ij}}{\partial a_{ij}} = \frac{x_j^{(p)} - b_{ij}}{(b_{ij} - a_{ij})^2}\kappa_{i,j,1}(x_j^{(p)})$$

$$\frac{\partial \mu_{ij}}{\partial b_{ij}} = \frac{a_{ij} - x_j^{(p)}}{(b_{ij} - a_{ij})^2}\kappa_{i,j,1}(x_j^{(p)})$$

$$\frac{\partial \mu_{ij}}{\partial c_{ij}} = \frac{d_{ij} - x_j^{(p)}}{(d_{ij} - c_{ij})^2}\kappa_{i,j,3}(x_j^{(p)})$$

$$\frac{\partial \mu_{ij}}{\partial d_{ij}} = \frac{x_j^{(p)} - c_{ij}}{(d_{ij} - c_{ij})^2}\kappa_{i,j,3}(x_j^{(p)}) \tag{7.5}$$

$\frac{\partial y}{\partial s_i}$ and the derivatives of the output membership functions parameters have to be computed also. From (2.45) the following can be written:

$$\frac{\partial y}{\partial *_{i^*}} = \frac{1}{3}\frac{den\dfrac{\partial F_{i^*}}{\partial *_{i^*}} - num\dfrac{\partial G_{i^*}}{\partial *_{i^*}}}{(den)^2} \tag{7.6}$$

where $*_i = [s_i, a_i, b_i, c_i, d_i]$ *den* is the denominator and *num* is the numerator of (7.6), respectively. F_{i^*} is the i^* member of the sum in the numerator and G_{i^*} is the i^* member in the denominator. The derivatives will be given as follows:

$$\frac{\partial F_i}{\partial a_i} = -6s_i a_i + 6s_i^2 a_i - 3s_i^2 b_i - 2s_i^3 (a_i - b_i)$$

$$\frac{\partial G_i}{\partial a_i} = -2s_i + s_i^2$$

$$\frac{\partial F_i}{\partial b_i} = -3s_i^2 a_i + 2s_i^3 (a_i - b_i)$$

$$\frac{\partial G_i}{\partial b_i} = -w_i^2$$

$$\frac{\partial F_i}{\partial c_i} = 3s_i^2 d_i - 2s_i^3 (d_i - c_i)$$

$$\frac{\partial G_i}{\partial c_i} = w_i^2$$

$$\frac{\partial F_i}{\partial d_i} = 6s_i d_i - 6s_i^2 d_i + 3s_i^2 c_i + 2s_i^3 (d_i - c_i)$$

$$\frac{\partial G_i}{\partial d_i} = 2s_i - s_i^2$$

$$\frac{\partial F_i}{\partial s_i} = 3(d_i^2 - a_i^2)(1 - 2s_i) + 6s_i (c_i d_i - a_i b_i)$$
$$+3s_i^2 \left[\ (c_i - d_i)^2 - (a_i - b_i)^2 \ \right]$$

$$\frac{\partial G_i}{\partial s_i} = 2(d_i - a_i) + 2s_i (c_i + a_i - d_i - b_i) \qquad (7.7)$$

7.4 Bacterial Memetic Algorithm

The previous section presented the equations required to employ the LM algorithm for fuzzy rule optimization. In this approach, the number of rules is assumed to be defined a-priori, condition which is necessary to apply a local nonlinear optimization algorithm. Nevertheless, if the rule base is not known a mechanism must be employed which will return the rule base.

Incorporating the neural network optimization algorithm with the bacterial evolutionary approach, the advantages of both methods can be utilized in the optimization process. The hybridization of these two methods leads to a new kind of memetic algorithm, since the bacterial technique replaces the classical genetic algorithm, while the Levenberg-Marquardt is the local searcher. This algorithm is the Bacterial Memetic Algorithm (BMA). The BMA algorithm is described in the following sections.

7.4.1 Outline of the memetic algorithm

This section discusses the outline and rationale of the hybrid algorithm. Fig. 7.3 depicts the schematic representation of the proposed hybrid algorithm.

The difference between the BEA and the BMA is that the latter contains a local search step, the Levenberg-Marquardt procedure.

Hence, the creation of the initial population, as well as the bacterial mutation and gene transfer operation remain exactly the same as in the original BEA.

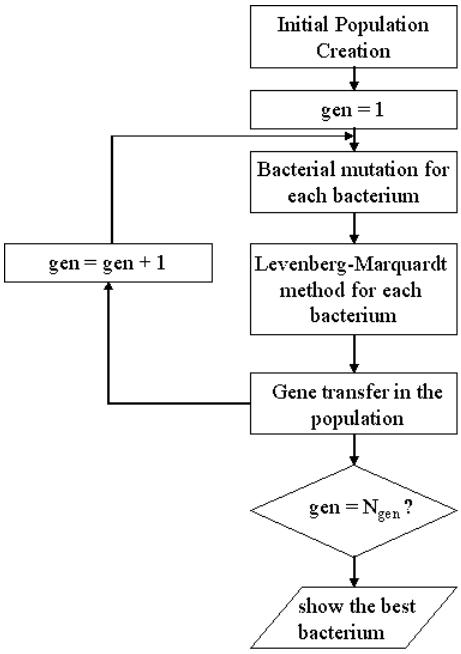


Fig. 7.3. Outline of the bacterial memetic algorithm

However, if besides optimizing the fuzzy rule, the number of fuzzy rules is to be found then both BEA and BMA should be updated. As this affects only the bacterial operators, this will be described in the following subsections. Note that performance comparison with all of these approaches is displayed to the end of this chapter.

7.4.2 Optimizing the number of rules

7.4.2.1 Initial population creation

The process begins with the creation of the initial population, consisting of bacteria where the encoding is similar to the one employed by evolutionary strategies. In other words, the information encoded in the chromosomes is a real value.

The objective of the algorithm is to find the optimal fuzzy rule base. Therefore, the chromosome encoding must be such that breakpoints from all of the trapezoidal membership functions are randomly specified at the time of creation.

Because the number of rules in the chromosomes can be different, the length of the chromosome is not constant. Fig. 7.1 illustrates the encoding of the fuzzy rules for the case of a bivariate problem. In this case, and for every rule, the chromosome needs to specify as many as 12 random values corresponding to the trapezium breakpoints.

7.4.2.2 Bacterial mutation

To find the optimal fuzzy rule base, the number of rules needs also to be defined.

In the classical bacterial mutation procedure N_{clones} clones of a source bacterium are generated. Then, a selected part is randomly changed in each of the clones. But because now it is also required to find the optimal number of rules, a change of the length of the bacterium during this operation should occur. This way, the algorithm proceeds as follows.

For each one of the clones, a random value is generated which will indicate whether the number of rules in the clone will be increased, decreased or stay the same. The chance of obtaining either one of the three options is equal. By default, the number of rules is restricted to a maximum value, *MaxRules* which must be defined by the user at the beginning.

Fig. 7.4 illustrates the decrease of the number of rules during bacterial mutation. In this case, the i^{th} clone had its 2^{nd} rule selected for removal. This operation is applicable as long as the length of the clone is greater than 1. Moreover, the genetic material of the remaining rules is kept in the chromosome.

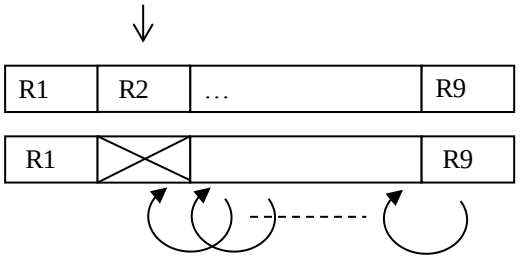


Fig. 7.4. Decrease of the number of rules in bacterial mutation

On the other hand, when the rule base is increased, this means a longer chromosome in the i^{th} clone (See Fig. 7.5).

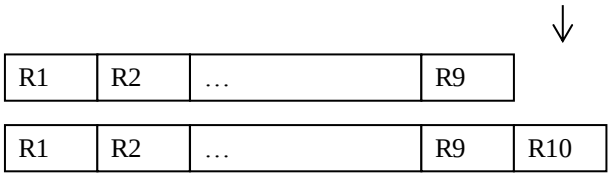


Fig. 7.5. Increase of the number of rules in bacterial mutation

Note that when a new rule is added to the chromosome, the new rule parameters are randomly chosen.

All the clones and the original bacterium are evaluated by an error criterion. The error criterion takes in consideration both the accuracy of the rule base and the complexity of the rule base. For this reason, the criterion chosen is the Bayesian Information Criterion (2.117). The process runs in the usual way. In the end, all parts of the chromosome have been mutated and tested and the best rule base is saved and the remaining N_{clones} are eliminated.

7.4.2.3 Gene transfer

The gene transfer operation allows the recombination of genetic information between two bacteria. For this, a source bacterium will transfer a part of its genetic material to the destination bacterium. The source bacterium belongs to the "best" half of the population. Then, a part (rule) from the source bacterium is chosen and this part will either overwrite a rule of the destination bacterium or will be added to the destination bacterium as a new rule. Each operation has identical chance of occurring. This cycle is repeated for N_{inf} times, where N_{inf} is the number of "infections" per generation.

Note that with gene transfer the rule base never decreases.

7.5 Performance of the LM method

This section illustrates the performance of the LM algorithm when optimizing a fixed fuzzy rule structure. Comparison results are drawn for 2 benchmark problems: the pH uni-dimensional problem and the ICT bi-dimensional problem.

Two cases will be analysed.

7.5.1 First case

This first experiment shows the training capabilities of the Levenberg-Marquardt algorithm optimizing the fuzzy system, using both problems, and for the sake of simplicity, only two membership functions parameters will be adjusted. These are the $\{b,c\}$ parameters belonging

to the first input membership function of the first rule. The LM method is compared with the error backpropagation method (BP), which is usually the most used for this purpose. The initial fuzzy systems structures are composed of three rules and represented by the first two figures. In the BP algorithm the learning rate, η, was set to 0.01.

The global minima of the performance surface for the pH problem is located at aproximately $\{b,c\}=\{0.349,0.800\}$ with a MSE value of 0,010. A local minima of the performance surface for the ICT problem is located at approximately $\{b,c\}=\{0.128,0.245\}$, and the MSE value is 0,865. Three different initial parameters positions are considered, and the initial and final MSE and parameters values are shown in Fig. 7.6 till Fig. 7.9 and in Table 7.1 and Table 7.2. From the next two figures, one can see that even a two parameter dependent performance surface features many local minima, located along the same flat surface. Despite this, from Table 7.1 one can conclude that the LM is much faster than the BP, converging to any of the minima.

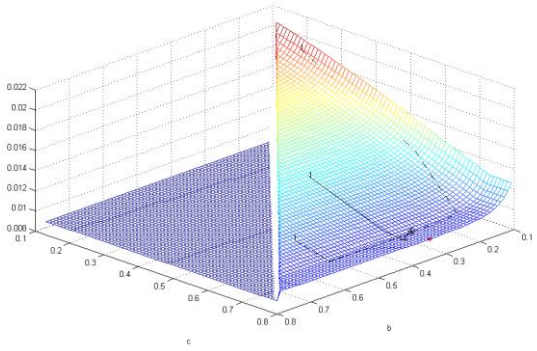


Fig. 7.6 Performance of the LM Method for the pH Problem

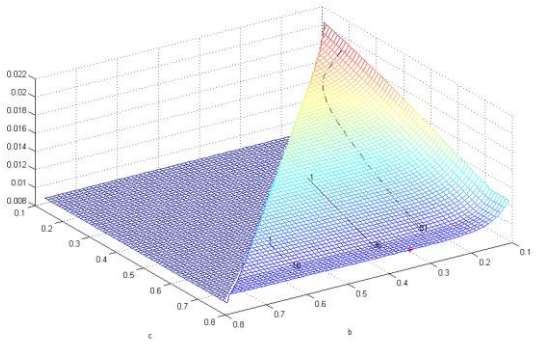


Fig. 7.7. Performance of the BP Method for the pH Problem

TABLE 7.1. SUMMARY OF THE RESULTS, FOR THE PH PROBLEM

Method	I point	F point	I mse	F mse	Iter
LM	[0.12, 0.20]	[0.355, 0.744]	0.019	0.0100	6
LM	[0.40, 0.50]	[0.387, 0.763]	0.013	0.0100	4
LM	[0.60, 0.65]	[0.355, 0.747]	0.011	0.0100	8
BP	[0.12, 0.20]	[0.278, 0.744]	0.019	0.0100	86
BP	[0.40, 0.50]	[0.398, 0.743]	0.013	0.0100	38
BP	[0.60, 0.65]	[0.355, 0.747]	0.011	0.0100	8

Using the ICT problem (as shown in the next two figures), one can observe that, though not reaching the local minima from the surface, both algorithms reach for final positions with similar final MSE, and close to the minima pointed out in the figures. The LM is still much faster than the BP.

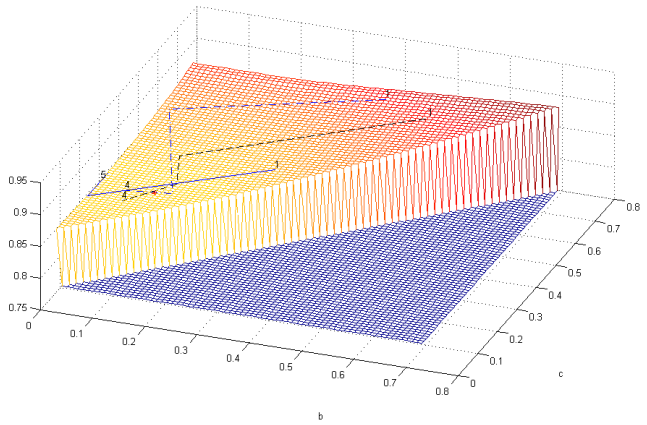


Fig. 7.8 Performance of the LM Method for the ICT Problem

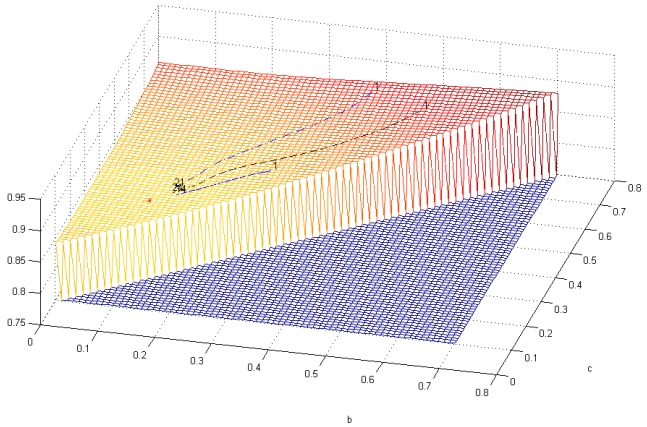


Fig. 7.9. Performance of the BP Method for the ICT Problem

TABLE 7.2. SUMMARY OF THE RESULTS, FOR THE ICT PROBLEM

Method	I point	F point	I mse	F mse	Iter
LM	[0.40, 0.70]	[0.100, 0.239]	0.890	0.8650	4
LM	[0.30, 0.40]	[0.190, 0.256]	0.870	0.8651	4
LM	[0.50, 0.65]	[0.113, 0.218]	0.890	0.8653	7
BP	[0.40, 0.70]	[0.155, 0.301]	0.890	0.8652	21
BP	[0.30, 0.40]	[0.162, 0.279]	0.870	0.8650	14
BP	[0.50, 0.65]	[0.156, 0.283]	0.890	0.8650	24

7.5.2 Second case

In this second case, all parameters are candidates for estimation. This example aims to evaluate the performance of such an approach if the LM method is to be applied in a hybrid scheme, for example the bacterial evolutionary algorithm, where the LM would estimate the parameters of multiple fuzzy rule bases given by each of the individuals of the population. Therefore, the next figures show the MSE line obtained along the LM evolution, and the parameters evolution lines for a total of 15 iterations using the pH problem, for a fuzzy system composed of 5 rules (40 parameters). Also, the MSE, MSRE and PMRE for the final fuzzy system given by the LM, and generated from the Bacterial Algorithm with a population of 10 individuals (N_{inf}=4 and N_{clones}=10) and along 40 generations, is presented.

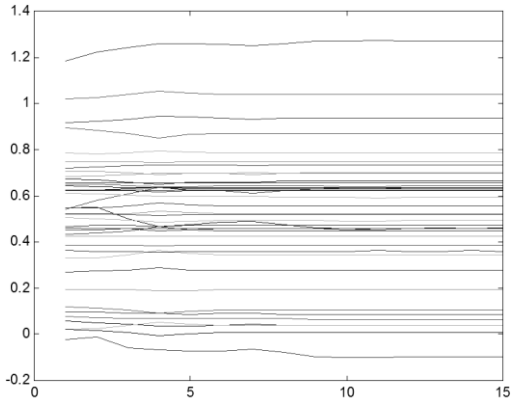


Fig. 7.10. The Fuzzy System Parameters Evolution

Although it is not possible to evaluate the ability to reach a local minima, from the MSE line one may conclude that the algorithm is actually optimizing the fuzzy rule base, and from the parameters evolution lines, it takes about 15 iterations to reach a local minima.

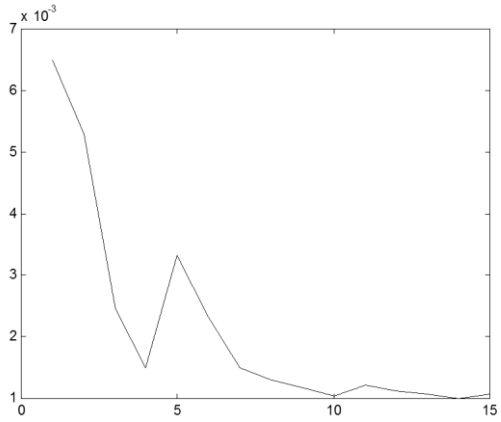


Fig. 7.11. MSE Evolution

TABLE 7.3.SUMMARY OF RESULTS FOR THE LM AND BEA OPTIMIZING A COMPLETE FUZZY RULE BASE, FOR THE PH PROBLEM

Fuzzy rule	MSE	MSRE	PMRE
Initial (LM)	$6.5x10^{-3}$	$1.98x10^{-1}$	25.5
Final (LM)	$9.83x10^{-4}$	$1.42x10^{-1}$	16.7
Bacterial Algorithm (final)	$7.01x10^{-4}$	$1.23x10^{-1}$	14.9

Table 7.3 shows that, although the initial fuzzy rule base is not the most appropriate, it takes only 15 iterations for the LM to reach the performance specifications shown, similar to the one provided by the Bacterial Algorithm, obtained after 40 generations with 10 candidates.

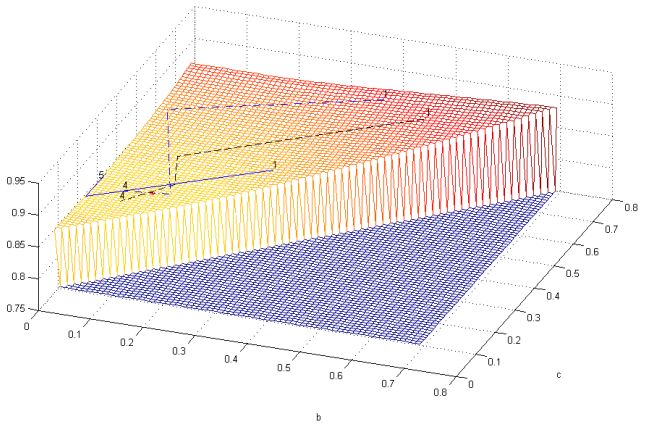


Fig. 7.12. Performance of the LM Method for the ICT Problem

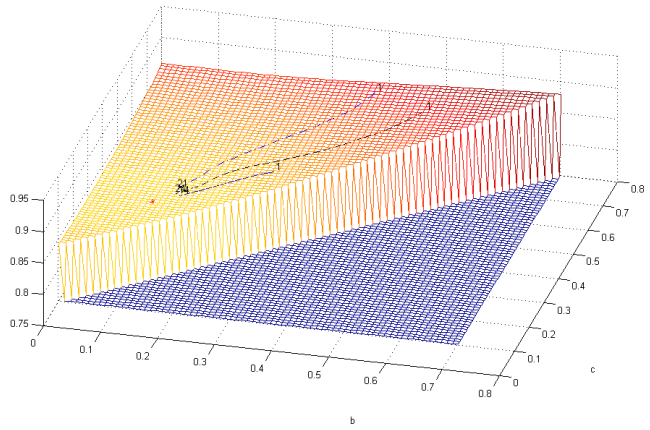


Fig. 7.13. Performance of the BP Method for the ICT Problem

7.6 Comparison between BMA and BEA

This section compares the new memetic algorithm with the BEA algorithm [126].

Five examples have been used as benchmark.

Both algorithms were set to run for 10 sessions of 20 generations each, and the results obtained relate to the following specifications: *MSE*, *MSRE* and *MREP* for both training and validation data (thereby using the v subscript).

The training and validation sets contained the same number of patterns and some of the patterns were identical. 110 patterns were used for the pH problem, 101 patterns for the ICT problem and 200 patterns for the six-dimensional generic function (see details in Appendix A). The algorithm evolution parameters were set as indicated in Table 7.4.

TABLE 7.4. ALGORITHMS EVOLUTION PARAMETERS

Parameter	Value
Population size	10
N_{clones}	8
N_{inf}	4
R	3
N_{gen}	10

7.6.1 Mean values

The mean values obtained for the Bacterial Memetic Algorithm (BMA) after 10 different runs can be seen in Table 7.5, and the mean values for Bacterial Evolutionary Algorithm (BEA) in Table 7.6.

It is clear that the lowest MSE value is obtained using the BMA instead of the BEA, for every problem and specially, when using the pH problem (around 16 times less). In general, BMA obtains better results, except for the case of the pH problem, where the validation results are clearly worse.

Since the training data used for the ICT problem has some input patterns around zero, the relative error results have very high values, in contrary to the results shown by the validation data, where such patterns are inexistent.

TABLE 7.5. MEAN VALUES FROM MSE, MSRE AND MREP FOR TRAINING AND VALIDATION DATA (V) OBTAINED FOR EVERY PROBLEM USING THE BMA ALGORITHM.

Specif.	pH	ICT	Sixth input	Agriculture	Chemical
MSE	$8,9 \times 10^{-4}$	$1,4 \times 10^{-1}$	$3,0 \times 10^{1}$	$8,3 \times 10^{-1}$	$4,64 \times 10^{4}$
MSRE	$4,9 \times 10^{3}$	$2,3 \times 10^{13}$	$5,9 \times 10^{-1}$	$2,1 \times 10^{-2}$	$4,0 \times 10^{-2}$
MREP	$5,8 \times 10^{2}$	$9,8 \times 10^{7}$	$6,6 \times 10^{1}$	10,9	8,5
MSEv	$1,0 \times 10^{-3}$	$9,9 \times 10^{-2}$	$3,3 \times 10^{1}$	3,3	$8,7 \times 10^{4}$
MSREv	$1,9 \times 10^{4}$	$1,3 \times 10^{-2}$	$5,5 \times 10^{-1}$	$4,7 \times 10^{-2}$	$1,8 \times 10^{-3}$
MREPv	$1,2 \times 10^{3}$	9,3	$6,4 \times 10^{1}$	13,4	3,8

TABLE 7.6. MEAN VALUES FROM MSE, MSRE AND MREP FOR TRAINING AND VALIDATION DATA (V) OBTAINED FOR EVERY PROBLEM USING THE BEA ALGORITHM.

Specif.	pH	ICT	Six input	Agriculture	Chemical
MSE	$1,5 \times 10^{-2}$	$5,0 \times 10^{-1}$	$3,4 \times 10^{1}$	$8,7 \times 10^{-1}$	$1,99 \times 10^{5}$
MSRE	$2,8 \times 10^{3}$	$1,1 \times 10^{14}$	$6,8 \times 10^{-1}$	$2,1 \times 10^{-2}$	$8,0 \times 10^{-2}$
MREP	2×10^{2}	$2,4 \times 10^{8}$	$7,2 \times 10^{1}$	11,3	15,8
MSEv	$7,4 \times 10^{-3}$	$6,0 \times 10^{-1}$	$3,5 \times 10^{1}$	4,4	$4,13 \times 10^{5}$
MSREv	$1,1 \times 10^{4}$	$8,2 \times 10^{-2}$	$6,1 \times 10^{-1}$	$6,2 \times 10^{-2}$	$8,5 \times 10^{-3}$
MREPv	$3,9 \times 10^{2}$	$2,5 \times 10^{1}$	$6,8 \times 10^{1}$	14,6	8,4

7.6.2 Overall best fuzzy models

However, it is also important to assess if the mean results reflect the quality of the best fuzzy model given by either one of the algorithms. Therefore, the performance results for the candidate with the lowest MSE value over all sessions are shown in Table 7.7.

TABLE 7.7. SPECIFICATIONS FOR THE FUZZY MODEL WITH THE LOWEST MSE VALUE FOUND AFTER ALL SESSIONS

Specif.	BEA	BMA	ex.
MSE	$4,5 \times 10^{-5}$	$1,5 \times 10^{-5}$	pH
MSRE	$2,1 \times 10^{-1}$	3×10^{2}	pH
MREP	$2,8 \times 10^{1}$	$1,7 \times 10^{2}$	pH
MSEv	$7,6 \times 10^{-3}$	3×10^{-5}	pH
MSREv	$4,2 \times 10^{-1}$	$1,2 \times 10^{3}$	pH
MREPv	$5,1 \times 10^{1}$	$3,5 \times 10^{2}$	pH
MSE	$3,4 \times 10^{-1}$	$8,5 \times 10^{-2}$	ICT
MSRE	$4,7 \times 10^{13}$	$3,6 \times 10^{13}$	ICT
MREP	$1,4 \times 10^{8}$	$1,7 \times 10^{8}$	ICT
MSEv	$2,0 \times 10^{-1}$	$2,0 \times 10^{-2}$	ICT
MSREv	$2,6 \times 10^{-1}$	$2,7 \times 10^{-3}$	ICT
MREPv	$1,5 \times 10^{1}$	4,4	ICT
MSE	$2,5 \times 10^{1}$	$2,0 \times 10^{1}$	6DIM
MSRE	$5,4 \times 10^{-1}$	$4,4 \times 10^{-1}$	6DIM
MREP	$8,3 \times 10^{1}$	$5,4 \times 10^{1}$	6DIM
MSEv	$2,6 \times 10^{1}$	$2,3 \times 10^{1}$	6DIM
MSREv	$4,5 \times 10^{-1}$	$4,7 \times 10^{-1}$	6DIM
MREPv	$5,6 \times 10^{1}$	$5,6 \times 10^{1}$	6DIM
MSE	$7,5 \times 10^{-1}$	$7,3 \times 10^{-1}$	agriculture
MSRE	$1,9 \times 10^{-2}$	$1,9 \times 10^{-2}$	Agriculture
MREP	11	10,6	Agriculture
MSEv	4,4	1,1	Agriculture
MSREv	$6,2 \times 10^{-2}$	$1,7 \times 10^{-2}$	Agriculture
MREPv	15,4	10,5	Agriculture
MSE	$1,0 \times 10^{5}$	$1,2 \times 10^{4}$	Chemical
MSRE	$4,8 \times 10^{-2}$	$9,9 \times 10^{-3}$	Chemical
MREP	12,0	4,2	Chemical
MSEv	$2,0 \times 10^{5}$	$3,0 \times 10^{4}$	Chemical
MSREv	$4,2 \times 10^{-3}$	$6,2 \times 10^{4}$	Chemical
MREPv	6,3	2,3	Chemical

The following table gives the specifications values when the model with lowest percentage of MRE (MREP) is considered. With this criterion the BMA performance is even better than that of BEA for the pH, ICT and six-dimensional problem meaning better ability to generalize

and approximate the function underlying the input data. However, it reveals a slightly worse performance in the case of the agricultural and chemical problems.

TABLE 7.8. SPECIFICATIONS FOR THE FUZZY MODEL WITH THE LOWEST PMRE VALUE FOUND AFTER ALL SESSIONS

Specif.	BEA	BMA	ex.
MSE	$1,5 \times 10^{-2}$	8×10^{-4}	pH
MSRE	$1,6 \times 10^{-1}$	1×10^{-1}	pH
MREP	$2,6 \times 10^{1}$	$1,5 \times 10^{1}$	pH
MSEv	$2,3 \times 10^{-3}$	$1,3 \times 10^{-3}$	pH
MSREv	$2,8 \times 10^{-1}$	2×10^{-1}	pH
MREPv	$3,5 \times 10^{1}$	$2,8 \times 10^{1}$	pH
MSE	$3,5 \times 10^{-1}$	$8,5 \times 10^{-2}$	ICT
MSRE	$8,4 \times 10^{13}$	$3,6 \times 10^{13}$	ICT
MREP	$2,0 \times 10^{8}$	$1,7 \times 10^{8}$	ICT
MSEv	$1,8 \times 10^{-1}$	$2,0 \times 10^{-2}$	ICT
MSREv	$2,3 \times 10^{-2}$	$2,7 \times 10^{-3}$	ICT
MREPv	$1,4 \times 10^{1}$	4,4	ICT
MSE	$2,5 \times 10^{1}$	$2,8 \times 10^{1}$	6DIM
MSRE	$5,4 \times 10^{-1}$	$5,3 \times 10^{-1}$	6DIM
MREP	$8,3 \times 10^{1}$	$6,1 \times 10^{1}$	6DIM
MSEv	$2,6 \times 10^{1}$	$2,9 \times 10^{1}$	6DIM
MSREv	$4,5 \times 10^{-1}$	$4,4 \times 10^{-1}$	6DIM
MREPv	$5,6 \times 10^{1}$	$5,5 \times 10^{1}$	6DIM
MSE	1,2	$7,3 \times 10^{-1}$	Agriculture
MSRE	$2,7 \times 10^{-2}$	$1,9 \times 10^{-2}$	Agriculture
MREP	12,6	10,6	Agriculture
MSEv	$8,1 \times 10^{-1}$	$9,8 \times 10^{-1}$	Agriculture
MSREv	$1,3 \times 10^{-2}$	$1,5 \times 10^{-2}$	Agriculture
MREPv	8,8	9,7	Agriculture
MSE	$5,3 \times 10^{5}$	$2,7 \times 10^{4}$	Chemical
MSRE	$2,4 \times 10^{-1}$	$1,7 \times 10^{-2}$	Chemical
MREP	30,8	6,3	Chemical
MSEv	$1,17 \times 10^{4}$	$1,9 \times 10^{4}$	Chemical
MSREv	$2,4 \times 10^{-4}$	$4,1 \times 10^{-4}$	Chemical
MREPv	1,4	1,9	Chemical

Above all, it is the bacterial memetic algorithm that gives the lowest MSE value fuzzy model. These models also represent the best validation specifications, except for the pH problem. This fact is mainly due to the presence of some input patterns with relative low values (around 10^{-4}).

The following ten figures show the target and error lines for the best fuzzy model in each one of the problems for both algorithms, assuming as the training criterion the MSE value. These figures are complemented with the corresponding fuzzy rule base parameters.

The general conclusion is that BMA achieves the lowest validation error, irrespective of the problem at hand.

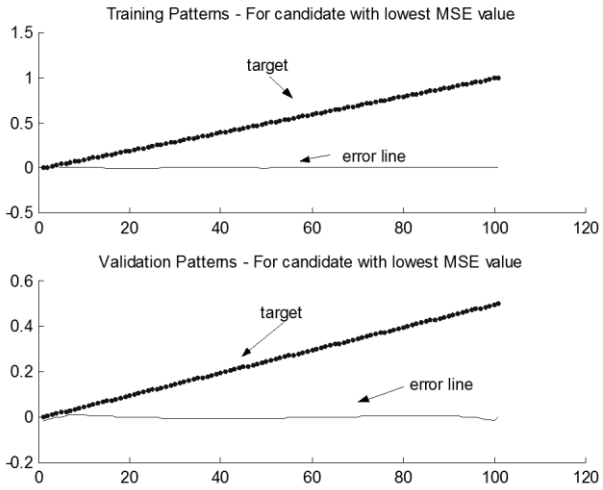


Fig. 7.14. Target and error lines for pH problem using BMA's candidate with lowest MSE value.

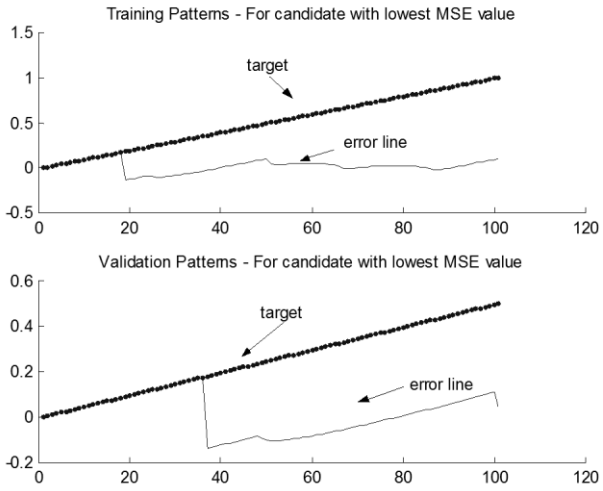


Fig. 7.15. Target and error lines for pH problem using BEA's candidate with lowest MSE value.

The final Fuzzy rules for the pH problem using BMA's candidate with lowest MSE value are:

R1: IF x1 is [0.14845 0.4463 0.64292 0.67123] THEN y is [0.34233 0.51977 0.58192 0.80283]
R2: IF x1 is [-0.10635 0.39081 0.57236 0.74319] THEN y is [-0.4978 0.21469 0.31095 0.45583]
R3: IF x1 is [0.037302 0.23532 0.7653 0.84955] THEN y is [0.6916 0.87295 1.0073 1.3837]

The final Fuzzy rules for the pH problem using BEA's candidate with lowest MSE value are:

R1: IF x1 is [0.059341 0.28597 0.40514 0.7313] THEN y is [0.18589 0.27486 0.52066 0.70602]
R2: IF x1 is [0.38301 0.39652 0.41925 0.76578] THEN y is [0.75291 0.81982 1.0024 1.0372]
R3: IF x1 is [0.051951 0.36896 0.56908 0.70039] THEN y is [0.086398 0.24551 0.37881 0.55217]

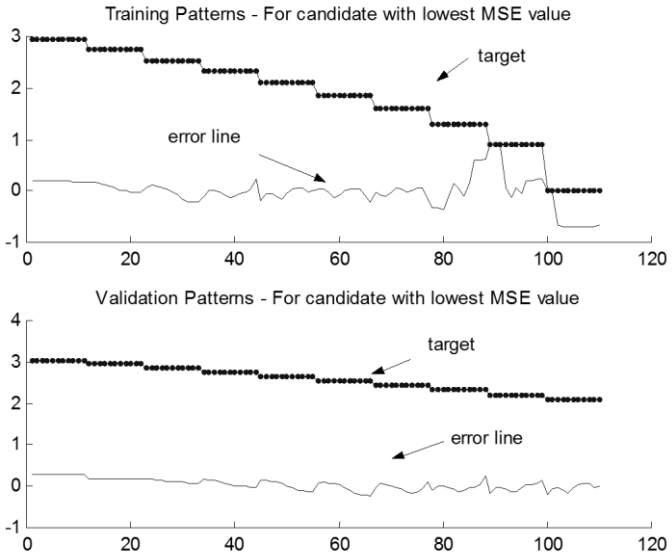


Fig. 7.16. Target and error lines for ICT problem using BMA's candidate with lowest MSE value.

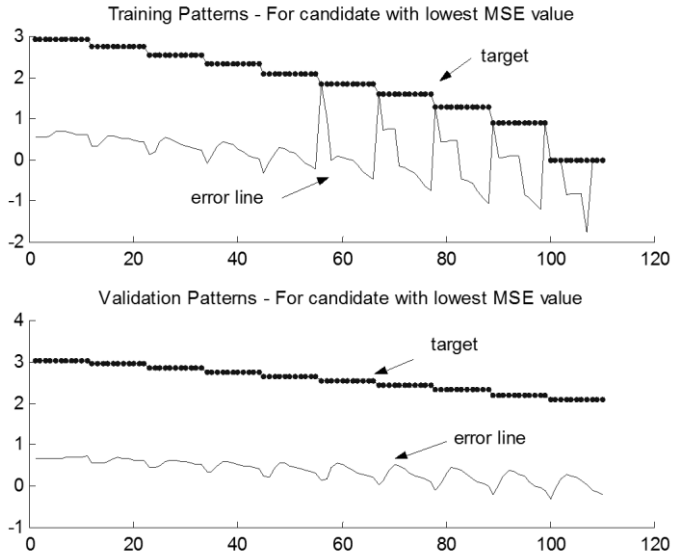


Fig. 7.17. Target and error lines for ICT problem using BEA's candidate with lowest MSE value.

The final Fuzzy rules for the ICT problem using BMA's candidate with lowest MSE value are:

R1: IF x1 is [-0.05557 0.096831 0.096831 0.61764] AND x2 is [-0.38741 0.15828 0.42377 0.75708] THEN y is [1.9286 2.5199 2.841 3.6312]

R2: IF x1 is [-0.13014 0.49711 0.79235 0.99016] AND x2 is [0.30314 0.72457 0.76876 1.114] THEN y is [0.01816 0.38369 1.0447 1.3301]

R3: IF x1 is [0.11687 0.62462 0.69643 0.80899] AND x2 is [-0.083077 0.22048 0.24413 0.70602] THEN y is [1.0631 1.6966 1.9513 2.1227]

The final Fuzzy rules for the ICT problem using BEA's candidate with lowest MSE value are:

R1: IF x1 is [-0.027427 0.013643 0.027407 0.27058] AND x2 is [-0.072697 0.1739 0.43876 0.88877] THEN y is [1.9286 1.9644 2.2663 2.8419]

R2: IF x1 is [0.023605 0.82814 0.83875 0.91526] AND x2 is [0.039025 0.37465 0.76291 0.94203] THEN y is [0.083984 0.599 0.83059 1.6556]

R3: IF x1 is [0.0055574 0.14652 0.57419 0.57709] AND x2 is [-0.090017 0.16984 0.88549 0.89452] THEN y is [1.5032 2.4553 2.6697 3.2334]

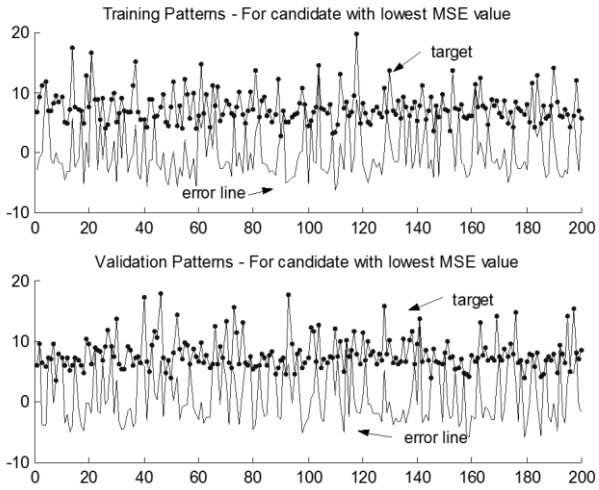


Fig. 7.18. Target and error lines for sixth dimensional problem using BMA's candidate with lowest MSE value.

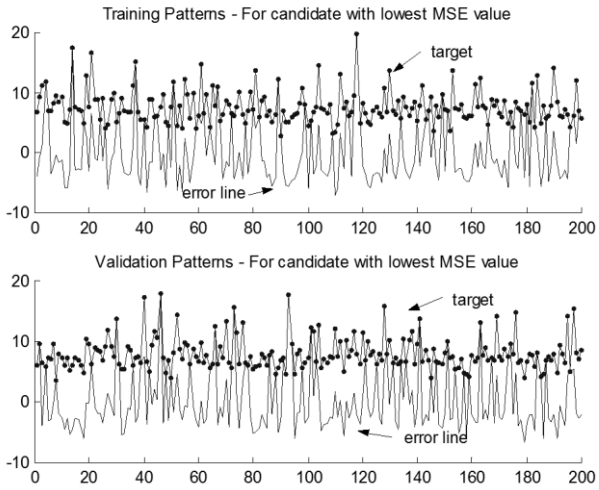


Fig. 7.19. Target and error lines for sixth dimensional problem using BEA's candidate with lowest MSE value.

The final Fuzzy rules for the six dimensional problem using BMA's candidate with lowest MSE value are:

R1: IF x1 is [1.0195 1.0823 2.2247 4.4724] AND x2 is [0.45709 1.3472 3.4677 4.4184] AND x3 is [0.12743 2.4164 3.8943 3.8966] AND x4 is [-0.0092932 2.9735 3.6027 4.597] AND x5 is [0.35931 0.81271 0.93897 3.6913] AND x6 is [0.46638 1.53 4.2455 4.4091] THEN y is [4.14193 9.83104 11.5653 11.6497]

R2: IF x1 is [0.29934 0.42357 4.3054 5.1422] AND x2 is [-0.45532 2.787 4.5064 4.9697] AND x3 is [0.4231 0.77611 1.9622 3.4471] AND x4 is [0.014251 0.36256 0.4216 0.72609]

AND x5 is [0.27691 0.556 0.89273 1.035] AND x6 is [-0.027394 0.061566 0.20996 0.41618] THEN y is [9.16778 10.0454 11.5467 14.1264]

R3:IF x1 is [-0.23359 1.7672 2.9245 4.608] AND x2 is [0.029085 0.14005 3.1426 4.6243] AND x3 is [0.33395 0.62798 2.8702 4.0896] AND x4 is [-0.013502 0.042693 0.51082 0.77926] AND x5 is [-0.040665 0.041077 0.92361 1.0095] AND x6 is [0.017653 0.5352 0.86467 0.87896] THEN y is [3.52437 7.38148 10.069 17.0449]

The final Fuzzy rules for the 6 dimensions problem using BEA's candidate with lowest MSE value are:

R1: IF x1 is [3.2691 3.7803 4.5245 4.9707] AND x2 is [0.45688 1.6764 1.8366 5.2796] AND x3 is [-0.19167 3.0767 3.5534 4.1727] AND x4 is [0.098077 0.32278 0.55876 0.5754] AND x5 is [-0.028333 0.40267 0.99707 1.0857] AND x6 is [-0.0039945 0.36039 0.96744 1.2172] THEN y is [3.77353 8.49603 10.0623 16.1108]

R2: IF x1 is [0.69229 1.4028 4.0655 4.5688] AND x2 is [1.3221 2.2397 3.2113 3.9014] AND x3 is [0.81474 2.5275 3.817 4.7368] AND x4 is [0.019088 0.43315 1.8155 2.9786] AND x5 is [0.80924 0.90894 1.2484 2.6031] AND x6 is [0.13657 1.8188 2.5877 3.1743] THEN y is [3.22641 11.4497 12.7276 15.884]

R3: IF x1 is [0.1308 1.6514 2.2276 4.9155] AND x2 is [-0.46576 0.13091 3.1385 4.4794] AND x3 is [0.37358 1.4722 2.3596 4.1944] AND x4 is [0.0015921 0.40295 0.43801 0.55955] AND x5 is [0.067956 0.10459 0.44455 0.85976] AND x6 is [-0.014537 0.76469 0.93777 1.1354] THEN y is [4.48621 8.30233 15.3879 16.4223]

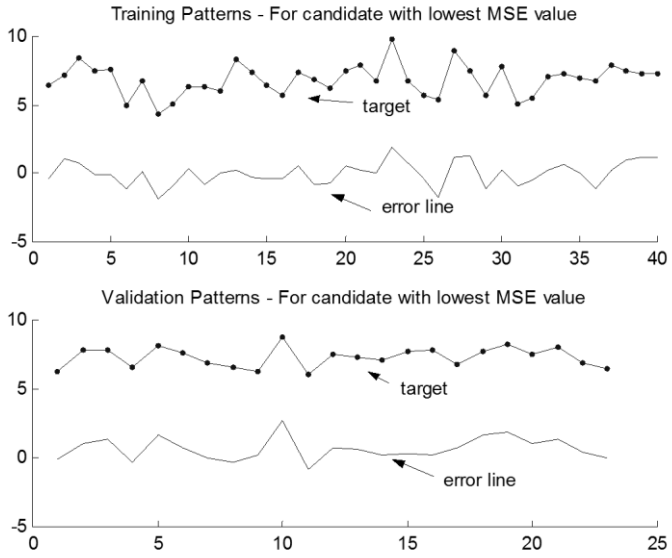


Fig. 7.20. Target and error lines for the agricultural problem using BMA's candidate with lowest MSE value.

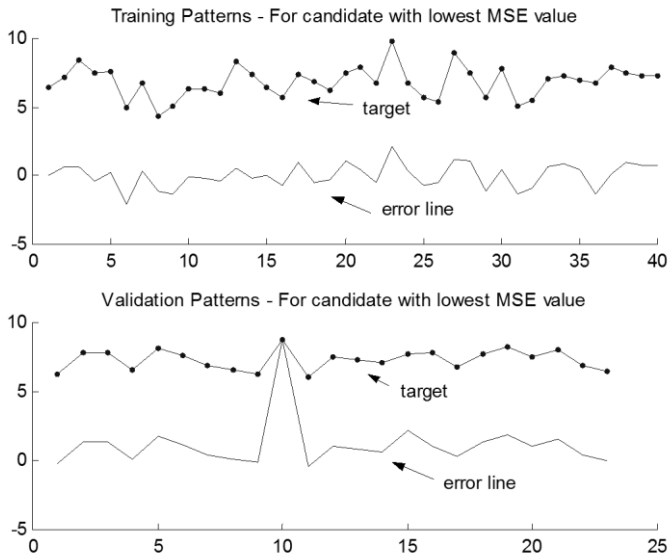


Fig. 7.21. Target and error lines for the agricultural problem using BEA's candidate with lowest MSE value.

The final Fuzzy rules for the agricultural problem using BMA's candidate with lowest MSE value are:

R1: IF x1 is [0.47796 1.6484 1.9865 3.6762] AND x2 is [0.94773 2.79118 12.5175 15.567] AND x3 is [14.0982 17.7734 39.3691 63.3048] AND x4 is [18.12948 135.9387 224.7332 524.1069] AND x5 is [46.75971 50.43507 79.21961 168.7417] AND x6 is [-0.243827 3.81297 10.9592 11.0626] THEN y is [2.5372 3.8271 8.3034 9.576]

R2: IF x1 is [0.12006 1.7442 2.8891 3.1227] AND x2 is [0.261424 1.18546 8.71874 11.1638] AND x3 is [4.25414 7.04729 36.4863 58.7372] AND x4 is [103.0778 104.6215 195.9549 403.3658] AND x5 is [20.37879 33.63517 50.7738 108.1339] AND x6 is [-0.12446 3.8411 8.8812 9.2889] THEN y is [3.67955 8.87042 8.87042 11.4879]

R3: IF x1 is [2.5302 2.7748 3.6654 5.0787] AND x2 is [1.49618 1.57757 8.70348 11.0376] AND x3 is [11.6207 16.1909 35.5542 57.0146] AND x4 is [56.59098 69.52069 146.8576 268.0168] AND x5 is [5.144961 10.33069 60.76468 129.2059] AND x6 is [0.32495 0.9483 2.0883 7.1438] THEN y is [4.43149 9.07721 11.6889 13.0019]

The final Fuzzy rules for the agricultural problem using BEA's candidate with lowest MSE value are:

R1: IF x1 is [1.5142 1.7934 2.9077 5.7027] AND x2 is [0.455225 0.938563 8.55417 10.8499] AND x3 is [11.3405 20.4192 35.1192 56.4783] AND x4 is [6.147601 61.01089 139.3491 235.2254] AND x5 is [22.62425 47.09296 49.69601 105.8604] AND x6 is [0.017295 2.6764 3.0311 6.3414] THEN y is [4.48543 7.08479 10.5221 12.1178]

R2: IF x1 is [0.30569 0.39484 1.9894 3.9017] AND x2 is [0.73 0.93597 5.9333 7.5257] AND x3 is [0.121577 16.6376 32.5122 52.2857] AND x4 is [97.24715 106.2515 219.0978

409.6491] AND x5 is [6.921508 35.79438 64.01843 136.3694] AND x6 is [0.0287838 0.315806 8.8421 10.5541] THEN y is [3.4876 3.9486 7.6912 9.4939]

R3: IF x1 is [0.68888 1.1326 1.4349 2.8142] AND x2 is [1.7471 1.7911 7.1462 9.0641] AND x3 is [3.35896 15.2122 30.9944 49.8447] AND x4 is [65.98891 138.6865 150.962 254.418] AND x5 is [27.99438 89.80611 91.47118 194.8481] AND x6 is [0.018529 0.75039 3.9289 6.7939] THEN y is [1.8734 3.4901 4.781 5.0093]

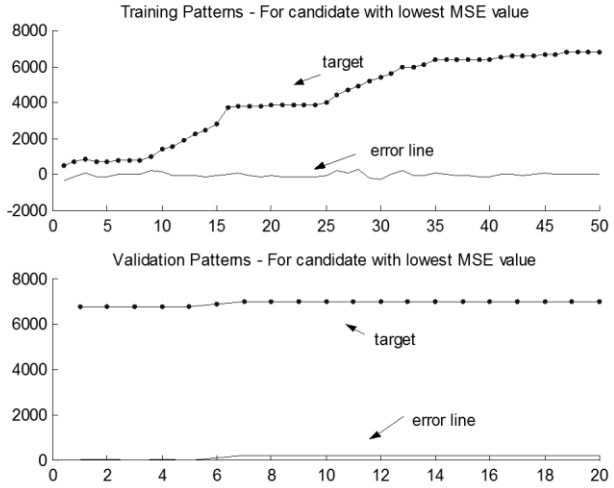


Fig. 7.22. Target and error lines for the chemical problem using BMA's candidate with lowest MSE value.

The final Fuzzy rules for the chemical problem using BMA's candidate with lowest MSE value are:

R1: IF x1 is [0.64389 3.5881 7.8709 8.5822] AND x2 is [-0.33968 -0.28496 0.29334 0.69235] AND x3 is [267.21254 428.59773 1321.6508 6830.8565] AND x4 is [-0.38738 -0.31509 0.30362 0.58666] AND x5 is [-0.20359 -0.053113 0.35114 0.40443] THEN y is [-153.16266 157.7469 1159.5847 2627.3857]

R2: IF x1 is [0.533977 2.97557 6.69248 10.3774] AND x2 is [-0.44198 -0.26061 0.24069 0.56808] AND x3 is [204.99338 297.67489 1135.1634 5479.9971] AND x4 is [-0.59209 -0.22992 0.3832 0.74042] AND x5 is [-0.42551 -0.23569 0.31779 0.36602] THEN y is [-883.87131 -168.00231 1013.4558 2409.1277]

R3: IF x1 is [1.2743 3.3716 6.4247 8.3991] AND x2 is [-0.38969 -0.16209 0.20013 0.22057] AND x3 is [824.239661 9134.08527 11066.4981 12272.1434] AND x4 is [-0.41796 -0.31585 0.30276 0.47112] AND x5 is [-0.26288 -0.096488 0.34021 0.53332] THEN y is [1572.58381 6280.96997 10414.0883 10414.0883]

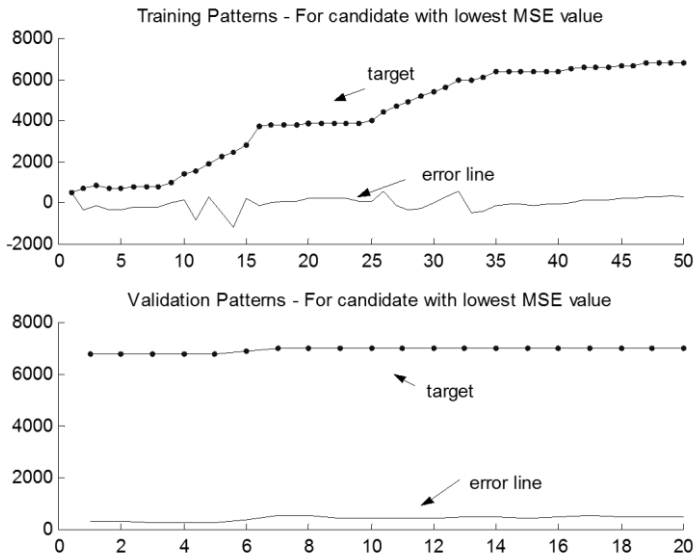


Fig. 7.23. Target and error lines for the chemical problem using BEA's candidate with lowest MSE value.

The final Fuzzy rules for the chemical problem using BEA's candidate with lowest MSE value are:

R1: IF x1 is [0.17002 0.40194 3.3061 6.0523] AND x2 is [-0.29799 0.2135 0.30264 0.39925] AND x3 is [270.80893 3170.9721 4898.488 5649.0938] AND x4 is [-0.47485 0.044146 0.29019 0.68514] AND x5 is [-0.10644 0.30708 0.35014 0.39027] THEN y is [408.357 6297.509 6454.9829 6870.354]

R2: IF x1 is [0.12287 0.62496 2.8252 5.172] AND x2 is [-0.25833 0.081853 0.18509 0.25415] AND x3 is [418.74117 1776.8845 5600.8669 8734.971] AND x4 is [-0.37049 0.038079 0.067936 0.22641] AND x5 is [-0.1374 0.13502 0.38861 0.50378] THEN y is [720.16852 5761.8159 9433.87108 11990.8662]

R3: IF x1 is [1.31078 5.6382 7.49383 13.7185] AND x2 is [-0.37862 0.022461 0.27127 0.5022] AND x3 is [154.20948 1963.9383 2431.6353 3216.8208] AND x4 is [-0.3917 0.16706 0.23937 0.29018] AND x5 is [-0.065146 0.038172 0.15266 0.23887] THEN y is [119.59077 413.20153 1467.155 1900.3928]

The following five figures illustrate the convergence ratio from all algorithms in a training session with 20 generations. In all of the cases, the supremacy of the BMA algorithm is obvious, both in terms of the convergence rate and in the criterion final value.

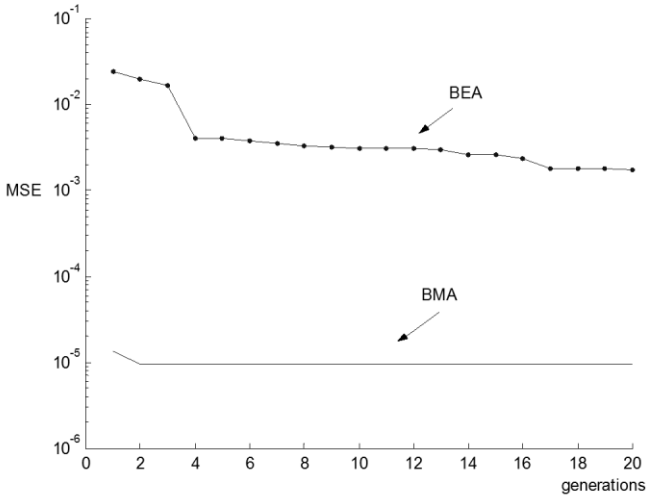


Fig. 7.24. MSE line for pH problem

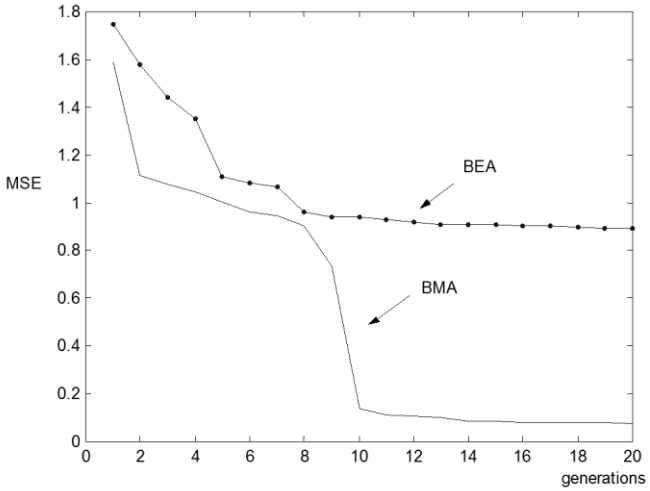


Fig. 7.25. MSE line for ICT problem

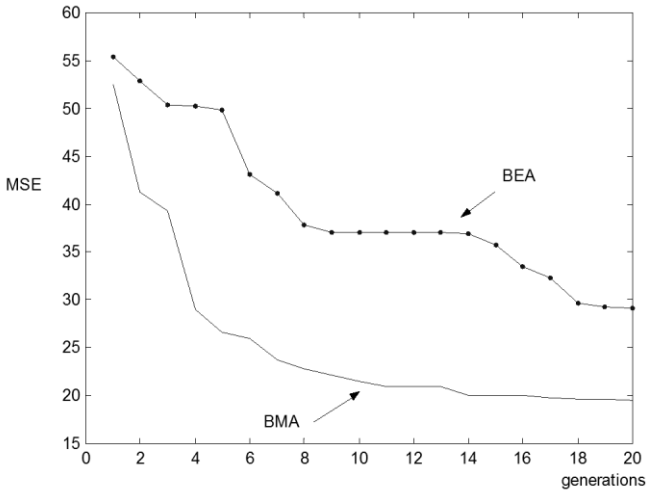


Fig. 7.26. MSE line for 6 dimensions problem

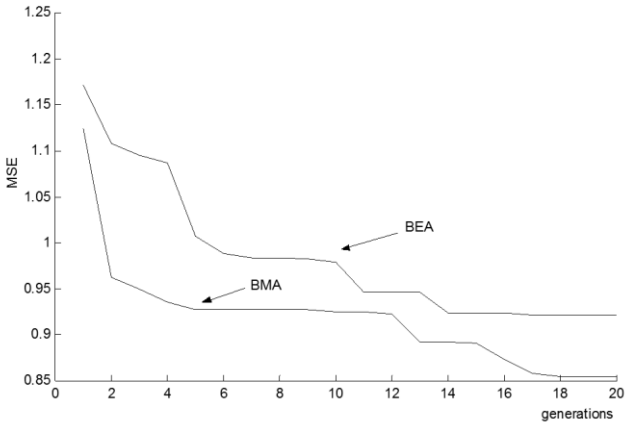


Fig. 7.27. MSE line for the agricultural problem

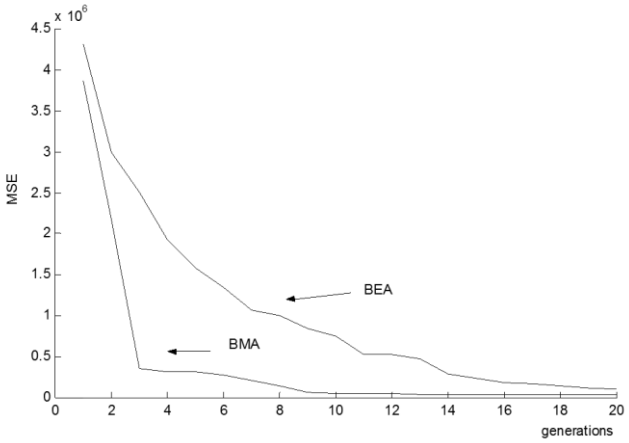


Fig. 7.28. MSE line for the chemical problem

7.7 Optimizing the number of fuzzy rules

This section shows comparison results between the improved BMA and BEA in the sense that both algorithms optimize now the number of rules in the fuzzy system. Only the first three examples from the former section are used: the pH problem, the ICT problem and the six dimensional benchmark problem.

Both algorithms were set to run for 10 sessions of 20 generations each. The training and validation sets contained the same number of patterns and some of the patterns were identical (for the ICT problem only). The parameters of the algorithms were set as follows: population size: 10, number of clones: 8, number of infections: 4. In the BMA, the Levenberg-Marquardt step runs for 10 iterations in every generation and the maximum allowed bacterium length is 10 (maximum number of rules is 10).

For the evaluation of the bacteria, the BIC criterion from (2.117) is used where $N=m$ is the number of patterns and $n_z$is the number of rules. The mean values obtained by the algorithm can be seen in Table 7.9.

It can be seen from the table that the BMA found the lowest value for every problem, and for every error definition.

Besides the mean values it is also important to compare the best models given by the two algorithms. The performance criteria for the best candidate can be seen in Table 7.10. For every study case, the bacterial memetic algorithm produces the fuzzy model with the lowest MSE. These models also represent the best validation specifications. Using the Levenberg-Marquardt method as local searcher, the performance of the bacterial algorithm is improved again.

TABLE 7.9. MEAN VALUES USING IMPROVED BEA AND IMPROVED BMA

Specification	pH		ICT		Six-dimensional	
	BEA	BMA	BEA	BMA	BEA	BMA
MSE	$6{,}28\text{x}10^{-3}$	$2{,}3\text{x}10^{-6}$	$8{,}69\text{x}10^{-1}$	$3{,}67\text{x}10^{-2}$	3.21	1.29
MSRE	$4{,}06\text{x}10^{4}$	$1{,}02\text{x}10^{1}$	$5{,}63\text{x}10^{13}$	$1{,}80\text{x}10^{13}$	$6{,}20\text{x}10^{-2}$	$3{,}07\text{x}10^{-2}$
MREP	$2{,}05\text{x}10^{3}$	$2{,}69\text{x}10^{1}$	$1{,}77\text{x}10^{8}$	$1{,}08\text{x}10^{8}$	$1{,}86\text{x}10^{1}$	$1{,}21\text{x}10^{1}$
MSEv	$7{,}76\text{x}10^{-3}$	$2{,}08\text{x}10^{-6}$	1,30	$1{,}12\text{x}10^{-2}$	4,29	2,22
MSREv	$1{,}63\text{x}10^{5}$	$4{,}18\text{x}10^{1}$	$1{,}92\text{x}10^{-1}$	$1{,}62\text{x}10^{-3}$	$6{,}50\text{x}10^{-2}$	$3{,}37\text{x}10^{-2}$
MREPv	$4{,}12\text{x}10^{3}$	$5{,}46\text{x}10^{1}$	$2{,}56\text{x}10^{1}$	3,04	$1{,}91\text{x}10^{1}$	$1{,}32\text{x}10^{1}$
#rules	4,90	7,40	2,20	5,00	6,50	7,30

TABLE 7.10. BEST CANDIDATES USING IMPROVED BEA AND IMPROVED BMA

Specification	pH		ICT		Six-dimensional	
	BEA	BMA	BEA	BMA	BEA	BMA
MSE	$2{,}73\text{x}10^{-3}$	$4{,}7\text{x}10^{-7}$	$8{,}42\text{x}10^{-1}$	$1{,}04\text{x}10^{-2}$	2.07	$4{,}10\text{x}10^{-1}$
MSRE	$3{,}71\text{x}10^{4}$	3,63	$5{,}32\text{x}10^{13}$	$7{,}09\text{x}10^{12}$	$4{,}78\text{x}10^{-2}$	$9{,}56\text{x}10^{-3}$
MREP	$1{,}97\text{x}10^{3}$	$1{,}92\text{x}10^{1}$	$2{,}03\text{x}10^{8}$	$6{,}39\text{x}10^{7}$	$1{,}59\text{x}10^{1}$	7,06
MSEv	$5{,}12\text{x}10^{-3}$	$6{,}07\text{x}10^{-7}$	1,26	$2{,}60\text{x}10^{-3}$	2,66	$9{,}9\text{x}10^{-1}$
MSREv	$1{,}48\text{x}10^{5}$	$1{,}53\text{x}10^{1}$	$1{,}87\text{x}10^{-1}$	$3{,}74\text{x}10^{-4}$	$4{,}28\text{x}10^{-2}$	$1{,}5\text{x}10^{-2}$
MREPv	$3{,}96\text{x}10^{3}$	$3{,}93\text{x}10^{1}$	$2{,}34\text{x}10^{1}$	1,48	$1{,}47\text{x}10^{1}$	9,28
#rules	9	8	2	5	7	10

7.8 Conclusions

In this chapter the LM algorithm has been employed for fuzzy rule optimization, specifically for non Ruspini partition-like fuzzy rules. The LM algorithm has proven to be feasible on its search for the best fuzzy rule base structure. Its performance is very acceptable as it can estimate the new refined parameters in a reduced number of iterations. Since it can be used to tune a fuzzy model, its incorporation in a population of many candidates (as is the case of evolutionary algorithms), produced a faster convergence to the global optima because each candidate represents the search for a different local minimum, out of the whole search space.

Furthermore, an improved version of the bacterial memetic algorithm (BMA) was presented. The discussion of the applicability of BMA was carried out through a comparison with the BEA algorithm. This approach gives very good results in the determination of an optimized fuzzy rule base. It shows significant improvement over the bacterial evolutionary algorithm. With the improved bacterial operators, one does not need to predefine the number of rules.

As was observed by the multiple experiments conducted along the methodologies employed, nonlinear parameters estimation plays a central role in structure optimization. This is more evident if a *nested* global optimization is adopted, such as the one employed. The convergence of nonlinear local optimization techniques not only depends on a proper selection of initial points, but also on the shape of the performance surface as was seen in section 7.5. Thus, the next chapter exploits the performance surface in a new strategy for optimizing nonlinear parameters where the dependence on the training data set is reduced.

Chapter 8

TOWARDS A MORE ANALYTICAL TRAINING OF NEURAL AND NEURO-FUZZY SYSTEMS

8.1 Introduction

The previous chapters showed that gradient based algorithms are useful when combined with evolutionary algorithms, in a hybrid scheme.

Gradient-based algorithms find the (local) minima of a performance surface. This surface is obtained using the training data, selected from the available data for the problem at hand. A bad selection implies that a "not so good" model will be obtained, as the result of the optimization process.

Having in mind that the ultimate goal of modeling is to obtain a "good" approximation of the function behind the data, and not to the data in itself, the modeling problem can be formulated as the minimization of the integral of the (functional) squared error, along the input domain, and not as the usual sum-of-squares-of-the-error. This approach is referred to as the *functional approach* onwards.

This chapter, which is an extended version of the works in [194][195][196][197][198] , is organized as follows.

It begins by presenting the new definition of the error in section 8.2. Because neuro-fuzzy modeling can employ the concept of parameters separability the training algorithms in the functional approach also explore this concept. Thus, the new error criterion is extended and the new definition for the gradient function is given.

Section 8.3 gives the necessary adaptation to second-order algorithms, focusing on the Levenberg-Marquardt (LM) algorithm. In this way, subsection 8.3.2 describes the required mathematical formulation for employing the functional approach with two versions of the *Jacobian* matrix.

The applicability of this new approach is assessed on three different modeling architectures, in section 8.4. Subsection 8.4.1 covers the Radial Basis function networks whereas subsection 8.4.2 exploits this concept to B-spline networks. In subsection 8.4.3 the application of this methodology to a Takagi-Sugeno fuzzy system is presented.

Finally, conclusions are drawn in section 8.5.

8.2 The methodology

Section 2.3.1 in chapter 2 introduced the class of supervised learning algorithms in which the training techniques for the class of models discussed in this chapter are included. All these training techniques are based on the knowledge provided by the input and output data extracted from the process. Thus, the training data consists of this collection of discretized points. Because the objective is to minimize some error measure between the process and the model behavior, the usual criterion used is:

$$\Omega_d\left(\mathbf{X},\mathbf{v},\mathbf{u},\mathbf{t}\right)=\frac{\left\|\mathbf{e}\left(\mathbf{X},\mathbf{v},\mathbf{u},\mathbf{t}\right)\right\|_2^2}{2} \tag{8.1}$$

In (8.1), the *d* subscript denotes the discretized version of the criterion and **e** is the error vector computed as the difference between the output of the process, **t** and the output of the model, **y**:

$$\mathbf{e}\left(\mathbf{X},\mathbf{v},\mathbf{u},\mathbf{t}\right)=\mathbf{t}-\mathbf{y}\left(\mathbf{X},\mathbf{v},\mathbf{u},\mathbf{t}\right)=\mathbf{t}-\boldsymbol{\varphi}^T\left(\mathbf{X},\mathbf{v}\right)\mathbf{u} \tag{8.2}$$

8.2.1 The training criterion

Alternatively, if the function to approximate was known (and denoted by $t\left(\mathrm{x}\right)$), the model output and criterion (8.1) would be:

$$\mathrm{y}\left(\mathrm{x},\mathbf{v},\mathbf{u}\right)=\boldsymbol{\varphi}^{\mathrm{T}}\left(\mathrm{x},\mathbf{v}\right)\mathbf{u} \tag{8.3}$$

And,

$$\Omega_f\left(\mathrm{x}_{\min},\mathrm{x}_{MAX},t,\mathbf{u},\mathbf{v}\right)=\frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\left(t(\mathrm{x})-\boldsymbol{\varphi}^T\left(\mathrm{x},\mathbf{v}\right)\mathbf{u}\right)^2 d\mathrm{x}}{2}=\frac{\int_{x_{\min}}^{\mathrm{x}_{MAX}}\left(t-\mathbf{u}^T\boldsymbol{\varphi}\right)^2 dx}{2}=\frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}e^2\left(\mathrm{x},t,\mathbf{v},\mathbf{u}\right)d\mathrm{x}}{2} \tag{8.4}$$

Above, x is now a multi-dimensional real variable:

$$\mathbf{x} = x_1, \ldots, x_{i,} \ldots, x_n \tag{8.5}$$

And in (8.4),

$$\int_{\mathbf{x}_{\min}}^{\mathbf{x}_{MAX}} f(\mathbf{x},.)d\mathbf{x} = \int_{x_{1_{\min}}}^{x_{1_{MAX}}} \cdots \int_{x_{k_{\min}}}^{x_{k_{MAX}}} \cdots \int_{x_{n_{\min}}}^{x_{n_{MAX}}} f(x_1, \ldots, x_k, \ldots, x_n, .)dx_1 \ldots dx_k, \ldots, dx_n \tag{8.6}$$

t and e are also real functions. $\boldsymbol{\varphi}$ is a vector of basis functions (and not a matrix, as in the discrete case).

In order to apply this new criterion for first order nonlinear local optimization the *gradient* vector must be computed. In this approach this implies a reformulation of the gradient. The next subsection presents the respective formulations.

8.2.2 Computation of the *gradient* vector

From definition, the gradient of the criterion in (8.4) is:

$$\mathbf{g}_{\Omega_{f_\mathbf{z}}}\left(\mathbf{x}_{\min}, \mathbf{x}_{MAX}, t, \mathbf{z}\right) = \frac{\partial \Omega_f\left(\mathbf{x}_{\min}, \mathbf{x}_{MAX}, t, \mathbf{z}\right)}{\partial \mathbf{z}^T} \tag{8.7}$$

The gradient in the functional case will be decomposed into its partitions in terms of linear and nonlinear parameters[1]:

$$\begin{aligned}
\mathbf{g}_{\Omega_{f_\mathbf{u}}}\left(\mathbf{x}_{\min}, \mathbf{x}_{MAX}, t, \mathbf{u}\right) &= \frac{\partial \Omega_f}{\partial \mathbf{u}^T} \\
&= \frac{1}{2} \frac{\partial \int_{\mathbf{x}_{\min}}^{\mathbf{x}_{MAX}} \left(t - \mathbf{u}^T\boldsymbol{\varphi}\right)^2 d\mathbf{x}}{\partial \mathbf{u}^T} \\
&= \frac{1}{2} \int_{\mathbf{x}_{\min}}^{\mathbf{x}_{MAX}} \frac{\partial\left(t^2 - 2t\mathbf{u}^T\boldsymbol{\varphi} + \mathbf{u}^T\boldsymbol{\varphi}\boldsymbol{\varphi}^T\mathbf{u}\right)}{\partial \mathbf{u}^T} d\mathbf{x} \\
&= -\int_{\mathbf{x}_{\min}}^{\mathbf{x}_{MAX}} \left(t\boldsymbol{\varphi} - \boldsymbol{\varphi}\boldsymbol{\varphi}^T\mathbf{u}\right) d\mathbf{x} \\
&= -\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\left(t - \boldsymbol{\varphi}^T\mathbf{u}\right) d\mathbf{x} \\
&= -\int_{x_{\min}}^{x_{MAX}} \mathbf{j}_\mathbf{u}^T e d\mathbf{x}
\end{aligned} \tag{8.8}$$

[1] In order to simplify, the dependence of the functions on their parameters will only be shown in the function definition.

$$\begin{aligned}
&\mathbf{g}_{\Omega_{f_\mathbf{v}}}\left(\mathrm{x}_{\min},\mathrm{x}_{MAX},t,\mathbf{v}\right)=\frac{\partial\Omega_f}{\partial\mathbf{v}^T}\\
&=\frac{1}{2}\frac{\partial\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\left(t-\boldsymbol{\varphi}^T\mathbf{u}\right)^2 d\mathrm{x}}{\partial\mathbf{v}^T}\\
&=\frac{1}{2}\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\frac{\partial\left(t^2-2t\boldsymbol{\varphi}^T\mathbf{u}+\mathbf{u}^T\boldsymbol{\varphi}\boldsymbol{\varphi}^T\mathbf{u}\right)}{\partial\mathbf{v}^T}d\mathrm{x}\\
&=-\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\left(t\left[\frac{\partial\boldsymbol{\varphi}}{\partial\mathbf{v}^T}\right]^T\mathbf{u}-\mathbf{u}^T\boldsymbol{\varphi}\left[\frac{\partial\boldsymbol{\varphi}}{\partial\mathbf{v}^T}\right]^T\mathbf{u}\right)d\mathrm{x}
\end{aligned}\tag{8.9}$$

$$\begin{aligned}
&=-\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\left(t-\mathbf{u}^T\boldsymbol{\varphi}\right)\left[\frac{\partial\boldsymbol{\varphi}}{\partial\mathbf{v}^T}\right]^T\mathbf{u}d\mathrm{x}\\
&=-\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}e\mathbf{j}_\mathbf{v}^T d\mathrm{x}\\
&=-\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\mathbf{j}_\mathbf{v}^T e d\mathrm{x}
\end{aligned}\tag{8.10}$$

In the expressions above, ***j*** is the *Jacobian* (the partial derivatives of the output function), while for the continuous case it is a row vector of real functions, not a matrix and is computed as:

$$\begin{aligned}
&\mathbf{j}\left(\mathrm{x},\mathbf{v},\mathbf{u}\right)=\frac{\partial\mathbf{y}\left(\mathrm{x},\mathbf{v},\mathbf{u}\right)}{\partial\left[\mathbf{u}^T\ \mid\ \mathbf{v}^T\right]}\\
&=\left[\mathbf{j}_\mathbf{u}\left(\mathrm{x},\mathbf{v}\right)\ \mid\ \mathbf{j}_\mathbf{v}\left(\mathrm{x},\mathbf{v},\mathbf{u}\right)\right]\\
&=\left[\boldsymbol{\varphi}^T\left(\mathrm{x},\mathbf{v}\right)\ \mid\ \left[\frac{\partial\boldsymbol{\varphi}\left(\mathrm{x},\mathbf{v}\right)}{\partial\mathbf{v}^T}\right]^T\mathbf{u}\right]
\end{aligned}\tag{8.11}$$

As the model parameters can be decomposed into linear and nonlinear ones, one can determine the optimal value of the linear parameters with respect to the nonlinear ones.

Equating (8.8) to 0:

$$\hat{\mathbf{u}}_f\left(t,\mathbf{v},\mathrm{x}_{\min},\mathrm{x}_{MAX}\right)=\left[\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}\right]^{-1}\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\boldsymbol{\varphi}t d\mathrm{x}\tag{8.12}$$

If this value is incorporated in the usual criterion, a new criterion (independent on the linear parameters) can be formulated for the functional approach, similarly to the discrete one given in (2.83). In other words,

$$\Psi_f\left(\mathrm{x}, \mathbf{v}, \mathrm{x}_{\min}, \mathrm{x}_{MAX}\right) = \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \left(t - \boldsymbol{\varphi}^T \hat{\mathbf{u}}_f\right)^2 d\mathrm{x}}{2} \tag{8.13}$$

Equation (8.13) defines the values for the optimal values for the linear parameters in the functional version. This solution is of complexity $\mathrm{O}(n_u{}^2)$ instead of $\mathrm{O}(mn_u{}^2)$.

To compute the gradient of this new criterion, expand (8.13):

$$\begin{aligned}
&\Psi_f\left(\mathrm{x}, \mathbf{v}, \mathrm{x}_{\min}, \mathrm{x}_{MAX}\right) = \\
&= \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} t^2 d\mathrm{x} - \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} 2t\boldsymbol{\varphi}^T \hat{\mathbf{u}}_f d\mathrm{x} + \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \hat{\mathbf{u}}_f^T \boldsymbol{\varphi}\boldsymbol{\varphi}^T \hat{\mathbf{u}}_f d\mathrm{x}}{2} \\
&= \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} t^2 d\mathrm{x} - \left[\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} 2t\boldsymbol{\varphi}^T d\mathrm{x}\right]\hat{\mathbf{u}}_f + \hat{\mathbf{u}}_f^T \left[\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}\right]\hat{\mathbf{u}}_f}{2}
\end{aligned} \tag{8.14}$$

The last equation may be derivated in respect to **v**, and the gradient of criterion Ψ_f computed as:

$$\begin{aligned}
&\mathbf{g}_{\Psi_f}\left(\mathrm{x}_{\min}, \mathrm{x}_{MAX}, t, \mathbf{v}\right) = \frac{\partial \Psi_f}{\partial \mathbf{v}^T} \\
&= -\frac{\partial\left[\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} t\boldsymbol{\varphi}^T d\mathrm{x}\right]\hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} + \frac{\partial\left\{\hat{\mathbf{u}}_f^T\left[\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}\right]\hat{\mathbf{u}}_f\right\}}{2\partial \mathbf{v}^T} \\
&= -\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} t\frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} d\mathrm{x}\hat{\mathbf{u}}_f - \left[\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} t\boldsymbol{\varphi}^T d\mathrm{x}\right]\frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} + \hat{\mathbf{u}}_f^T \frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}}{2\partial \mathbf{v}^T}\hat{\mathbf{u}}_f + \\
&+\hat{\mathbf{u}}_f^T \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}}{2}\frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} + \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T}\frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}}{2}\hat{\mathbf{u}}_f
\end{aligned} \tag{8.15}$$

With the appropriate algebraic manipulations it can be shown that $-\left[\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} t\boldsymbol{\varphi}^T d\mathrm{x}\right]\frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} + \hat{\mathbf{u}}_f^T \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}}{2}\frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} + \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T}\frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}}{2}\hat{\mathbf{u}}_f = 0$ [194].

Hence (8.15) is independent on $\frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T}$ and can be simplified to:

$$\mathbf{g}_{\Psi_f} = -\int_{x_{\min}}^{x_{MAX}} t\frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T}dx\hat{\mathbf{u}}_f + \hat{\mathbf{u}}_f^T\frac{\partial\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx}{2\partial \mathbf{v}^T}\hat{\mathbf{u}}_f . \tag{8.16}$$

If $\hat{\mathbf{u}}_f$ is replaced in (8.16) then

$$\begin{aligned}\mathbf{g}_{\Psi_f} = &-\int_{x_{\min}}^{x_{MAX}} t\frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T}dx\left[\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}t dx + \\ &+\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}^T t dx\left\{\left[\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}\frac{\partial\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx}{2\partial \mathbf{v}^T}\left[\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}\right\}\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}t dx\end{aligned} \tag{8.17}$$

In (8.17), the terms

$$\left[\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1} \tag{8.18}$$

$$\left[\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}\frac{\partial\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx}{2\partial \mathbf{v}^T}\left[\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1} \tag{8.19}$$

are independent on the function to approximate, and can be obtained analytically for the model at hand.

Alternatively, the only terms involving the function to approximate, and consequently the training data, are

$$\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}t dx \tag{8.20}$$

$$\int_{x_{\min}}^{x_{MAX}} t\frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T}dx \tag{8.21}$$

Remark:

In a practical application the underlying function is not known (otherwise it should be used). The integrals in (8.20) and (8.21) must be numerically approximated using the training data.

Besides being an elegant solution – the gradient is computed with a set of terms that only depend on the model and the input domain, and another set of terms which are the projection of the function on the basis functions and on their partial derivatives, over the input domain –

the use of this approach also reduces the computational complexity. Inspection of the computation of the discrete gradient (equation (2.80) and (2.81)) shows that it involves the computation of a pseudo-inverse (which has, at least a complexity of $O\left(mn_u^2\right)$ and matrix multiplications, whose complexity is $O\left(mn_u n_v\right)$.

If (8.17) is used, 4 matrix-vector multiplications are needed, but the quantities involved have only size n_u, and are independent on the number of the training patterns, m. The numerical computation of the projections has a complexity of $O\left(mn_u\right)$.

8.3 Training algorithms

8.3.1 Error backpropagation

The application of the error back-propagation algorithm is straightforward since the rotation matrix is simply the *Identity* matrix and all is required is to apply (8.17) in (2.54).

8.3.2 Levenberg-Marquardt algorithm

As for the LM algorithm, (using either criterion (8.4) or (8.13)), one requires the calculation of the corresponding *Jacobian* matrix.

Consider the new training criterion. The *Jacobian* is given as the partial derivative of $\boldsymbol{\varphi}^T\hat{\mathbf{u}}_f$ with respect to the nonlinear parameters, i.e.:

$$\mathbf{j}_{\Psi_f} = \frac{\partial\left[\boldsymbol{\varphi}^T\hat{\mathbf{u}}_f\right]}{\partial\mathbf{v}^T} \tag{8.22}$$

In other words,

$$\mathbf{j}_{\Psi_f} = \frac{\partial\boldsymbol{\varphi}^T}{\partial\mathbf{v}^T}\hat{\mathbf{u}}_f + \boldsymbol{\varphi}^T\frac{\partial\hat{\mathbf{u}}_f}{\partial\mathbf{v}^T} \tag{8.23}$$

The derivative of the left term in (8.23) is dependent on the model used. Nevertheless, the derivative in the last term of (8.23) is

$$\frac{\partial\hat{\mathbf{u}}_f}{\partial\mathbf{v}^T} = \partial\frac{\left[\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}\int_{x_{\min}}^{x_{MAX}}\boldsymbol{\varphi}t dx}{\partial\mathbf{v}^T} \tag{8.24}$$

Equation (8.24) simplifies to

$$\frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} = \frac{\partial \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1}}{\partial \mathbf{v}^T} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx + \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} t dx \tag{8.25}$$

After some algebraic manipulations (8.25) is simplified as

$$\begin{aligned} \frac{\partial \left[\boldsymbol{\varphi}^T \hat{\mathbf{u}} \right]}{\partial \mathbf{v}^T} &= \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} \hat{\mathbf{u}}_f - \boldsymbol{\varphi}^T \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T dx \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx - \\ &- \boldsymbol{\varphi}^T \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} dx \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx + \boldsymbol{\varphi}^T \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} t dx \end{aligned} \tag{8.26}$$

Noting that the optimal values for the linear parameters is $\hat{\mathbf{u}}_f = \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx$, a simplified version of expression (8.26) is as follows:

$$\begin{aligned} \frac{\partial \left[\boldsymbol{\varphi}^T \hat{\mathbf{u}} \right]}{\partial \mathbf{v}^T} &= \left[\frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} - \boldsymbol{\varphi}^T \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \left\{ \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T dx + \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} dx \right\} \right] \hat{\mathbf{u}}_f + \\ &+ \boldsymbol{\varphi}^T \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} t dx \end{aligned} \tag{8.27}$$

This is the functional equivalent to the *Jacobian* form introduced by Golub and Pereyra (2.86).

The functional equivalent to the *Jacobian* form as developed by Ruano (2.88) is given as a solution of

$$\mathbf{j}_{\Psi_{f_R}} = \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} \hat{\mathbf{u}}_f \tag{8.28}$$

Thus, the last expression does not require any further calculation.

Notice also that, if any of these two Jacobians are used in (8.4), the same gradient is obtained. To prove this, subtract (8.28) from (8.27) and transpose the result:

$$\begin{aligned} &\left[-\hat{\mathbf{u}}_f^T \left\{ \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}} \boldsymbol{\varphi}^T dx + \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}} dx \right\} \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} + \right. \\ &\left. + \left\{ \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}} t dx \right\}^T \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \right] \boldsymbol{\varphi} \end{aligned} \tag{8.29}$$

Notice that the quantities within the square brackets are independent of x. Integrating (8.29) with the optimal error vector, i.e. $t - \boldsymbol{\varphi}^T \hat{\mathbf{u}}_f$, one has

$$\left[-\hat{\mathbf{u}}_f^T \left\{ \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}} \boldsymbol{\varphi}^T dx + \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}} dx \right\} \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} + \left\{ \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}} t dx \right\}^T \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \right] \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \left[t - \boldsymbol{\varphi}^T \hat{\mathbf{u}}_f \right] dx \tag{8.30}$$

If one focuses attention to the last integral of (8.30),

$$\begin{aligned} &\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \left[t - \boldsymbol{\varphi}^T \hat{\mathbf{u}} \right] dx = \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx - \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T \hat{\mathbf{u}}_f dx = \\ &= \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx - \left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right] \left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right]^{-1} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx = 0 \end{aligned} \tag{8.31}$$

This last equation proves that both *Jacobian* matrices ((8.27) and (8.28)) produce the same gradient vector.

The functional version of Levenberg-Marquardt (LM) update, is the solution of

$$\left(\int_{x_{min}}^{x_{max}} \mathbf{j}^T(k)\mathbf{j}(k) dx + \rho(k)\mathbf{I} \right) \mathbf{s}(k) = -\mathbf{g}_{\Psi_f}(k) . \tag{8.32}$$

The terms involved in the computation of the gradient vector have already been given above. As two *Jacobian* matrices have been introduced, the terms to have available will differ according to the *Jacobian* used.

8.3.2.1 Functional version of $\mathbf{J_R}$

If the *Jacobian* from (8.28) is employed:

$$\begin{aligned} &\int_{x_{min}}^{x_{MAX}} \mathbf{j}^T \mathbf{j} dx = \\ &= \int_{x_{min}}^{x_{MAX}} \hat{\mathbf{u}}_f^T \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} \hat{\mathbf{u}}_f dx \\ &= \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}^T t dx \left\{ \left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right]^{-1} \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} dx \left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right]^{-1} \right\} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx \end{aligned} \tag{8.33}$$

which requires availability of:

$$\int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} dx \tag{8.34}$$

Additionally 4 matrix-vector multiplications will be needed. Notice, however, that the dimensions involved are only n_u, not mn_u as is the case with the discretized approach.

8.3.2.2 Functional version of J_{GP}

Replacing (8.23) in (8.34):

$$\int_{x_{min}}^{x_{MAX}} \mathbf{j}^T \mathbf{j} dx = \int_{x_{min}}^{x_{MAX}} \left[\hat{\mathbf{u}}_f^T \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} + \frac{\partial \hat{\mathbf{u}}_f^T}{\partial \mathbf{v}^T} \boldsymbol{\varphi} \right] \left[\frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} \hat{\mathbf{u}}_f + \boldsymbol{\varphi}^T \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} \right] dx$$

$$= \int_{x_{min}}^{x_{MAX}} \left(\hat{\mathbf{u}}_f^T \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} \hat{\mathbf{u}}_f + \hat{\mathbf{u}}_f^T \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} + \left[\hat{\mathbf{u}}_f^T \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} \right]^T + \frac{\partial \hat{\mathbf{u}}_f^T}{\partial \mathbf{v}^T} \boldsymbol{\varphi} \boldsymbol{\varphi}^T \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} \right) dx \tag{8.35}$$

Notice that 3rd term is the transpose of the 2nd. Let's focus on the 2nd and on the 4th. Using (8.12) and (8.25), the 2nd term,

$$\int_{x_{min}}^{x_{MAX}} \hat{\mathbf{u}}_f^T \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} dx$$

is equivalent to

$$\begin{aligned} &- \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}^T t dx \left\{ \left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right]^{-1} \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T dx \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)^{-1} \frac{\partial \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)}{\partial \mathbf{v}^T} \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)^{-1} \right\} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx + \\ &+ \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}^T t dx \left\{ \left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right]^{-1} \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T dx \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)^{-1} \right\} \int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} t dx \end{aligned} \tag{8.36}$$

The 4th term in (8.35) is

$$\begin{aligned} &\int_{x_{min}}^{x_{MAX}} \frac{\partial \hat{\mathbf{u}}_f^T}{\partial \mathbf{v}^T} \boldsymbol{\varphi} \boldsymbol{\varphi}^T \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} dx = \\ &\int_{x_{min}}^{x_{MAX}} \left\{ \left[\int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} t dx - \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}^T t dx \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)^{-1} \frac{\partial \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)}{\partial \mathbf{v}^T} \right] \times \right. \\ &\times \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)^{-1} \boldsymbol{\varphi} \boldsymbol{\varphi}^T \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)^{-1} \times \\ &\left. \times \left[\int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} t dx - \frac{\partial \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)}{\partial \mathbf{v}^T} \left(\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} \boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx \right] \right\} dx \end{aligned} \tag{8.37}$$

Expanding (8.37),

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \hat{\mathbf{u}}_f^T}{\partial \mathbf{v}^T} \boldsymbol{\varphi}\boldsymbol{\varphi}^T \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} dx = \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} tdx \left\{ \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \right\} \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} tdx -$$

$$- \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}^T tdx \left\{ \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \frac{\partial \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)}{\partial \mathbf{v}^T} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \right\} \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} tdx -$$

$$- \left[\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}^T tdx \left\{ \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \frac{\partial \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)}{\partial \mathbf{v}^T} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \right\} \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} tdx \right]^T +$$

$$+ \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}^T tdx \left\{ \begin{array}{l} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \frac{\partial \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)}{\partial \mathbf{v}^T} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \times \\ \times \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \frac{\partial \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)}{\partial \mathbf{v}^T} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right)^{-1} \end{array} \right\} \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi} tdx$$

(8.38)

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \hat{\mathbf{u}}_f^T}{\partial \mathbf{v}^T} \boldsymbol{\varphi}\boldsymbol{\varphi}^T \frac{\partial \hat{\mathbf{u}}_f}{\partial \mathbf{v}^T} d\mathrm{x} = \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} td\mathrm{x} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)^{-1} \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} td\mathrm{x} -$$

$$- \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}^T td\mathrm{x} \left\{ \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)^{-1} \frac{\partial \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)}{\partial \mathbf{v}^T} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)^{-1} \right\} \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} td\mathrm{x} -$$

$$- \left[\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}^T td\mathrm{x} \left\{ \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)^{-1} \frac{\partial \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)}{\partial \mathbf{v}^T} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)^{-1} \right\} \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} td\mathrm{x} \right]^T +$$

$$+ \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}^T td\mathrm{x} \left\{ \begin{array}{l} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)^{-1} \frac{\partial \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)}{\partial \mathbf{v}^T} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)^{-1} \times \\ \frac{\partial \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)}{\partial \mathbf{v}^T} \left(\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x} \right)^{-1} \end{array} \right\} \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi} td\mathrm{x}$$

(8.39)

Equation (8.39) denotes the complexity of computing the functional version of the LM update according to the *Jacobian* as given by Golub and Pereyra.

8.3.2.3 Prediction error

To update the regularization parameter (section 2.3.4), the LM algorithm relies on predicting the value of the integral of the training criterion at iteration k, $\int_{x_{\min}}^{x_{MAX}} \left(e^P(k)\right)^2 dx$.

Since,

$$e^P(k) = e(k) + \mathbf{j}(k)e(k)\mathbf{s}(k) \tag{8.40}$$

$$\int_{x_{\min}}^{x_{MAX}} \left(e^P(k)\right)^2 dx = \int_{x_{\min}}^{x_{MAX}} \left(e(k)\right)^2 dx + 2\left(\mathbf{s}(k)\right)^T \int_{x_{\min}}^{x_{MAX}} \mathbf{j}^T(k)e(k)dx + \mathbf{s}(k)^T \left(\int_{x_{\min}}^{x_{MAX}} \mathbf{j}^T(k)\mathbf{j}(k)dx \right) \mathbf{s}(k) \tag{8.41}$$

The first term is equivalent to the double of (8.13).

The second term is the equivalent to $-2\left(\mathbf{s}[k]\right)^T \mathbf{g}_{\Psi_f}(k)$.

The computation of the integral for the last term is either given by (8.35) or (8.39).

8.4 Application to different model types

The previous sections provided with the basic mathematical formulation for applying the functional approach for models where the parameters separability concept can be applied. This section shows how to apply this approach to the RBF, BSNN and TS-type fuzzy systems. Comparison to the usual discretized methodologies is given, as well. As was noted, the terms independent on the target function can be computed beforehand for a specific model. In appendix C, the corresponding mathematical formulas are given.

8.4.1 Radial Basis Function networks

8.4.1.1 Univariate problem

This subsection illustrates the use of the functional approach with a very simple example. The RBF network is composed of one neuron and a bias term.

Thus, the basis function vector is

$$\boldsymbol{\varphi}\left(x, c, v\right) = \begin{bmatrix} e^{-\frac{(x-c)^2}{2v}} \\ 1 \end{bmatrix}. \tag{8.42}$$

The nonlinear parameters are $\mathbf{v}^T = [c \;\; v]$ and the model has 2 linear parameters.

The fact of having only two non-linear parameters exist allows the illustration of the training criterion performance surface through a 3D plot. The input domain considered is [-1, 1].

Only 6 sample points will be used as training data, given by Gaussian Quadrature (see chapter 2, section 2.6.1.1.2 for more details) within the input domain, i.e.,

$$\mathbf{x} = [\text{-}0.9325 \;\; \text{-}0.6612 \;\; \text{-}0.2386 \;\; 0.2386 \;\; 0.6612 \;\; 0.9325]^T \tag{8.43}$$

As test data, 66 points were generated in range [-1, 1], at increments of 0.03.

The target function resembles a RBF network with 2 neurons in one dimensional input space and is given by

$$t(x) = 0.5\mathrm{e}^{-x^2} - 0.2\mathrm{e}^{-\frac{(x-0.2)^2}{0.5}} \tag{8.44}$$

For the error backpropagation algorithm, the learning rate is set to one, $\eta = 1$. In both algorithms, the maximum number of iterations is fixed to 4000. Also, $\tau = 10^{-8}$ and the stopping criteria in chapter 2, section 2.3.6 were combined with the maximum number of iterations as stopping criterion.

Notice that in the case of the Levenberg-Marquardt algorithm, two versions of the *Jacobian* matrix were employed, denoted by J_{R} and J_{GP}. In the discrete version they are defined respectively by (2.88) and (2.86) whereas for the functional version they are given by (8.28) and (8.23) .

In all trainings, Ψ_f refers to the integral of the square of errors over the domain using a quadrature technique. When the true function is also known, the analytical value for the functional approach is denoted as Ψ_a, and the usual discrete sum-of-square errors is denoted as Ψ_d.

The analytical solution, i.e., using the functional approach and (8.44), obtaining a training criterion $\hat{\Psi}_a = 4.7 \times 10^{-5}$, is

$$\hat{\mathbf{p}}_a = \begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{v}} \end{bmatrix} = [0.219 \;\; 0.116 \;\; | \;\; \text{-}0.198 \;\; 0.314]^T \tag{8.45}$$

The optimum for the discrete approach and data (8.43) is

$$\hat{\mathbf{p}}_d = \begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{v}} \end{bmatrix} = [0.223 \;\; 0.113 \;\; | \;\; \text{-}0.187 \;\; 0.309]^T \tag{8.46}$$

These values will produce $\hat{\Psi}_d = 1.58 \times 10^{-4}$ and $\Psi_a\big|_{\hat{\mathbf{p}}_d} = 5.31 \times 10^{-5}$.

With the functional approach and data (8.43),

$$\hat{\mathbf{p}}_f = \begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{v}} \end{bmatrix} = \left[0.215 \quad 0.121 \,|\, -0.200 \quad 0.299 \right]^T \tag{8.47}$$

and $\Psi_a\big|_{\hat{\mathbf{p}}_f} = 4.76 \times 10^{-5}$.

In order to assess the success of optimization, a summary on evaluation criteria for 4 different initial starting points is presented in the tables below.

In addition, the evolution of the training is complemented by figures illustrating the 3D surface of the criteria.

TABLE 8.1: TRAININGS WITH BP (DISCRETE VERSION)

v[1]	[-1 0.8]	[0.4 0.4]	[0.3 0.4]	[0.5 0.8]
N	1248	349	260	1386
v[N]	[-0.186, 0.316]	[0.863, 0.036]	[-0.187, 0.309]	[0.863, 0.036]
$\Psi_d\big\|_{\mathbf{p}_d}[\mathrm{N}]$	1.58e-4	5.1e-3	1.58e-4	5.1e-3
$\Psi_a\big\|_{\mathbf{p}_d}[\mathrm{N}]$	5.3e-5	1.65e-3	5.3e-5	1.65e-3
$\Psi_d\big\|_{\mathbf{p}_d}[\mathrm{N}]$ (test data)	1.69e-3	5.3e-2	1.69e-3	5.3e-2

TABLE 8.2: TRAININGS WITH BP (FUNCTIONAL VERSION)

v[1]	[-1 0.8]	[0.4 0.4]	[0.3 0.4]	[0.5 0.8]
N	3597	442	549	2212
v[N]	[-0.197, 0.328]	[0.923, 0.107]	[-0.197, 0.328]	[1.000, 0.133]
$\Psi_d\big\|_{\mathbf{p}_f}[\mathrm{N}]$	1.84e-4	5.8e-3	1.84e-4	5.8e-003
$\Psi_a\big\|_{\mathbf{p}_f}[\mathrm{N}]$	4.74e-5	1.31e-3	4.74e-5	1.31e-3
$\Psi_d\big\|_{\mathbf{p}_d}[\mathrm{N}]$ (test data)	1.46e-3	4.02e-2	1.46e-3	4.02e-2

TABLE 8.3: TRAININGS WITH LM USING RUANO *JACOBIAN* DEFINITION (DISCRETE VERSION)

v[1]	[-1 0.8]	[0.4 0.4]	[0.3 0.4]	[0.5 0.8]
N	114	64	73	51
v[N]	[-0.187 0.311]	[0.852 0.030]	[-0.187 0.311]	[0.852 0.030]
$\Psi_d\big\|_{\mathbf{p}_d}[\mathrm{N}]$	1.58e-4	5.1e-3	1.58e-4	5.1e-3
$\Psi_a\big\|_{\mathbf{p}_d}[\mathrm{N}]$	5.30e-5	1.71e-3	5.30e-5	1.71e-3
$\Psi_d\big\|_{\mathbf{p}_d}[\mathrm{N}]$ (test data)	1.69e-3	5.4e-2	1.69e-3	5.4e-2

TABLE 8.4: TRAININGS WITH LM USING GOLUB-PEREYRA *JACOBIAN* DEFINITION (DISCRETE VERSION)

v[1]	[-1 0.8]	[0.4 0.4]	[0.3 0.4]	[0.5 0.8]
N	17	64	15	49
v[N]	[-0.187 0.309]	[0.807 0.0056]	[-0.187 0.309]	[0.807 0.0054]
$\Psi_d\big\|_{\mathbf{p}_d}[\mathrm{N}]$	1.58e-4	5.1e-3	1.58e-4	5.1e-3
$\Psi_a\big\|_{\mathbf{p}_d}[\mathrm{N}]$	5.30e-5	1.19e-2	5.30e-5	1.32e-2
$\Psi_d\big\|_{\mathbf{p}_d}[\mathrm{N}]$ (test data)	1.69e-3	3.94e-1	1.69e-3	3.94e-1

TABLE 8.5: TRAININGS WITH LM USING RUANO *JACOBIAN* DEFINITION (FUNCTIONAL VERSION)

V[1]	[-1 0.8]	[0.4 0.4]	[0.3 0.4]	[0.5 0.8]
N	96	23	74	45
v[N]	[-0.199 0.301]	[0.950 0.116]	[-0.199 0.301]	[0.950 0.115]
$\Psi_d\big\|_{\mathbf{p}_f}[\mathrm{N}]$	1.84e-4	5.8e-3	1.84e-4	5.8e-3
$\Psi_a\big\|_{\mathbf{p}_f}[\mathrm{N}]$	4.74e-5	1.31e-3	4.74e-5	1.31e-3
$\Psi_d\big\|_{\mathbf{p}_d}[\mathrm{N}]$ (test data)	1.45e-3	4.02e-02	1.45e-3	4.02e-2

TABLE 8.6: TRAININGS WITH LM USING GOLUB-PEREYRA *JACOBIAN* DEFINITION (FUNCTIONAL VERSION)

v[1]	[-1 0.8]	[0.4 0.4]	[0.3 0.4]	[0.5 0.8]
N	17	22	17	28
v[N]	[-0.200 0.299]	[0.950 0.116]	[-0.200 0.299]	[0.950 0.116]
$\Psi_d\big\vert_{\mathbf{p}_f}$ [N]	1.84e-4	5.8e-3	1.84e-4	5.8e-3
$\Psi_a\big\vert_{\mathbf{p}_f}$ [N]	4.76e-5	1.31e-3	4.76e-5	1.31e-3
$\Psi_d\big\vert_{\mathbf{p}_d}$ [N] (test data)	1.45e-3	4.02e-2	1.45e-3	4.02e-2

The key point is that, although the functional approach uses the same input data as the discrete version (with 6 input points), it produces, at the optimum, a better approximation to the function underlying the data.

Regardless of the version employed (whether functional or discrete), it is obvious that the LM algorithm is faster and takes much less iterations to converge to the final parameters values, in comparison to the back-propagation algorithm. It should also be noted that in case of the functional approach, the Golub-Pereyra *Jacobian* version exhibits better convergence rate than that of Ruano's *Jacobian* version. However, the same is not true for the discrete case, since the convergence to the local optima is not always attained, as seen in Table 8.3 and Table 8.4. In addition, the Golub-Pereyra version requires more computational effort.

An important difference between the functional and discrete versions in terms of performance is the fact that criterion Ψ_f in the functional version is lower than that of the discrete version, for all the final points considered.

Another fact that distinguishes the ability of the functional version in pursuing the local optima is the ability to explore the nonlinearity of the function. This is clearly seen if Fig. 8.1-c) is compared to Fig. 8.1-d). The path chosen by the algorithm in Fig. 8.1-d) agrees more with the surface than that from Fig. 8.1-c). The same conclusions are drawn from the remaining figures except that the path is longer (more steps are required).

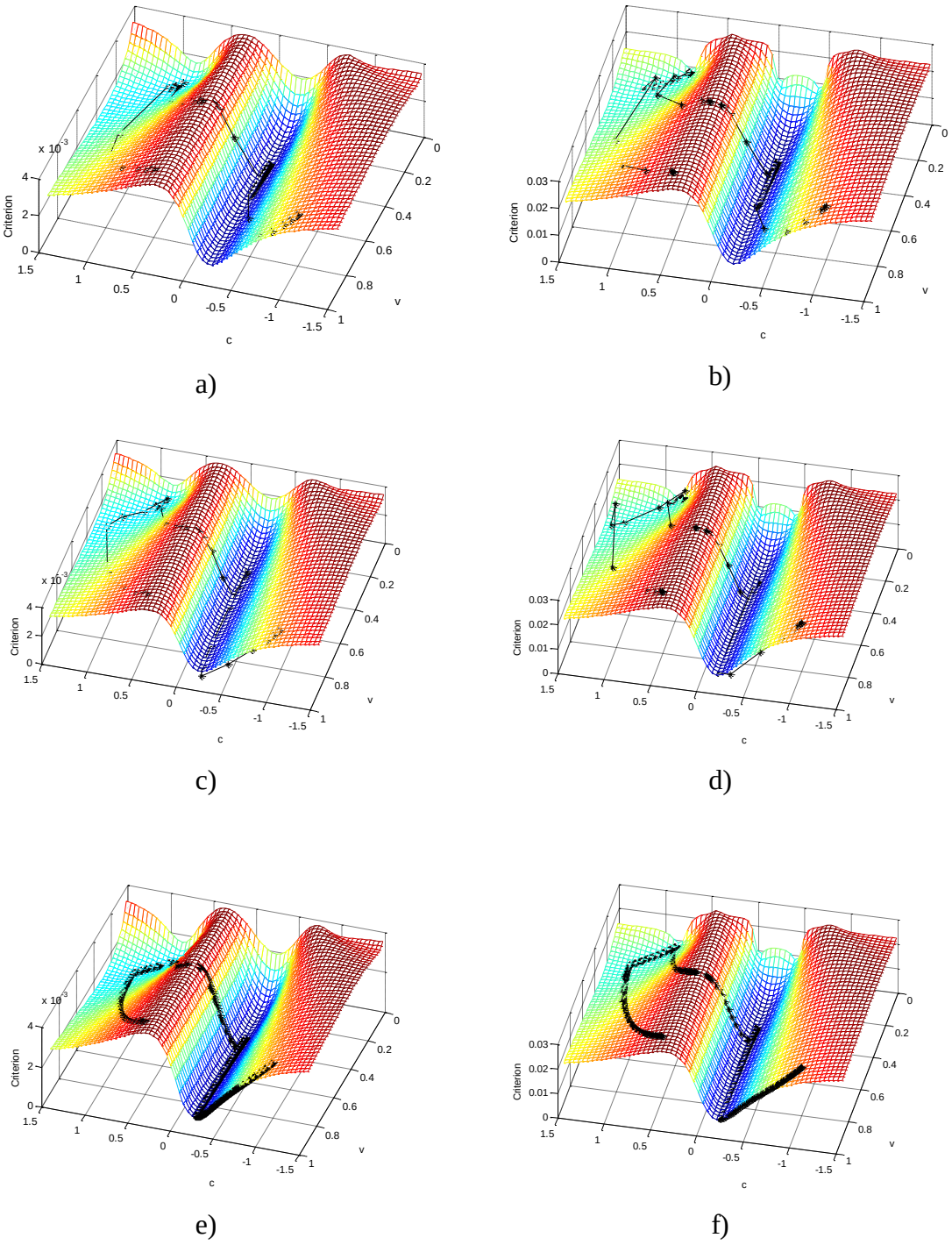


Fig. 8.1. Evolution of 4 different trainings with a) LM using Ruano *Jacobian* (functional version); b) LM using Ruano *Jacobian* (discrete version); c) LM using Golub-Pereyra *Jacobian* (functional version); d) LM using Golub-Pereyra *Jacobian* (discrete version); e) BP (functional version); f) BP (discrete version)

8.4.1.2 Identification with noisy data

Consider that the target function is corrupted with Gaussian distributed noise so that the measured process output is given by

$$y_n(i) = y(i) + n(i), \tag{48}$$

where $n(i)$ is noise with a Gaussian distribution with zero mean and variance, σ^2.

To investigate how well the proposed algorithms deal with noise, four different Signal to Noise Ratios (*SNR*) were considered. For each one of them, a total of 50 runs were executed.

Two different input data sizes, 6 and 20 were used given by Gaussian Quadrature. 50 runs were conducted and the mean values were obtained.

The results summarized in the next tables complement the criteria in previous tables with the mean and the variance of the error between $\hat{\Psi}_a$ and Ψ_a, between $\hat{\mathbf{p}}$ and $\mathbf{p}$, and between $\hat{\Psi}_d$ and Ψ_d, for each *SNR*, using either the optimal parameters assuming the functional LM, and the optimal parameters obtained when the standard (discrete) approach is employed.

TABLE 8.7 TRAININGS WITH LM USING GOLUB-PEREYRA *JACOBIAN* DEFINITION (FUNCTIONAL VERSION). THE STARTING POINT IS [-1, 0.8] AND INPUT DATA SIZE IS 6.

SNR (dB)	74	54	34	31
N	18	18	18	19
$\overline{\mathbf{v}}[N]$	[-0.19948 0.29881]	[-0.199 0.299]	[-0.20252 0.30471]	[-0.20057 0.29533]
$\overline{\psi}_d[N]$	1.83e-4	1.76e-4	1.24e-4	1.71e-4
$\overline{\psi}_f[\mathrm{N}]$	4.76e-5	4.77e-5	6.57e-5	9.21e-5
$\overline{\psi}_d[\mathrm{N}]$ (test data)	1.45e-3	1.47e-3	3.21e-3	2.37e-3
$\overline{\left(\Psi_a\big\vert_{\mathbf{p}_f} - \hat{\Psi}_a\right)}$	5.213e-7	7.03e-7	1.87e-5	4.50e-5
$\mathrm{var}\left(\Psi_a\big\vert_{\mathbf{p}_f} - \hat{\Psi}_a\right)$	1.34e-15	1.37e-13	1.20e-10	1.03e-9
$\overline{\left(\mathbf{p}_f[N] - \hat{\mathbf{p}}_f[N]\right)}$	[-1.67e-3 -1.53e-2]	[-1.61e-3 -1.52e-2]	[-4.71e-3 -9.37e-3]	[-2.75e-3 -1.88e-3]
$\mathrm{var}\left(\mathbf{p}_f[N] - \hat{\mathbf{p}}_f[N]\right)$	[2.43e-8 2.43e-7]	[1.76e-6 2.07e-5]	[1.68e-4 2.97e-3]	[4.28e-4 5.92e-3]

The results clearly show that, for all SNRs considered, smaller means and variances are obtained using the functional version. The functional version gives a better approximation to

the underlying function, using the same data, also with noisy data. Furthermore, some other conclusions can be drawn:

- Convergence to optima is more probable with the functional version, in general.
- With the increase of noise power, the guarantee of convergence to optima reduces. This is more significant if SNR<27, where both algorithms struggle to find the best parameters.

TABLE 8.8 TRAININGS WITH LM USING GOLUB-PEREYRA *JACOBIAN* DEFINITION (DISCRETE VERSION). THE STARTING POINT IS [-1, 0.8] AND INPUT DATA SIZE IS 6.

SNR (dB)	74	54	34	31.2
N	17	17	16.72	17
$\overline{\mathbf{v}}[N]$	[-0.18687 0.30888]	[-0.18684 0.30904]	[-0.19004 0.31529]	[-0.18732 0.30899]
$\overline{\psi}_d[N]$	1.57e-4	1.51e-4	1.09e-4	1.41e-4
$\overline{\psi}_f[N]$	5.3075e-5	5.325e-5	6.78e-5	9.38e-5
$\overline{\psi}_d[\mathrm{N}]$ (test data)	1.69e-3	1.73e-3	3.12e-3	2.37e-3
$\overline{\left(\Psi_a\big\|_{\mathbf{p}_d}-\hat{\Psi}_a\right)}$	6.04e-6	6.22e-6	2.07e-5	4.68e-5
$\mathrm{var}\left(\Psi_a\big\|_{\mathbf{p}_d}-\hat{\Psi}_a\right)$	9.74e-15	1.06e-12	1.71e-10	1.13e-9
$\overline{(\mathbf{p}_d[N]-\hat{\mathbf{p}}_d[N])}$	[1.09e-2 -5.2e-3]	[1.10e-2 -5.04e-3]	[7.77e-3 1.21e-3]	[1.05e-2 -5.09e-3]
$\mathrm{var}(\mathbf{p}_d[N]-\hat{\mathbf{p}}_d[N])$	[2.19e-8 1.84e-7]	[1.56e-6 1.60e-5]	[1.76e-4 2.80e-3]	[3.91e-4 5.38e-3]
$\overline{(\psi_d-\hat{\psi}_d)}$	1.81e-9	-3.72e-7	1.08e-5	3.94e-5
$\mathrm{var}(\psi_d-\hat{\psi}_d)$	8.41e-13	9.99e-11	6.35e-9	2.49e-8

If the data size is increased to 20, the input points are,

$$\mathbf{x}=\begin{bmatrix} 0.9931 & 0.9640 & 0.9122 & 0.8391 & 0.7463 & 0.6361 & 0.5109 & 0.3737 \\ 0.2278 & 0.0765 & -0.0765 & -0.2278 & -0.3737 & -0.5109 & -0.6361 & -0.7463 \\ -0.8391 & -0.9122 & -0.9640 & -0.9931 & & & & \end{bmatrix}^T \tag{8.49}$$

For that data the functional version with (8.49) yields $\hat{\Psi}_f = \hat{\Psi}_a = 4.70\times10^{-5}$, and $\begin{bmatrix}\hat{\mathbf{u}}\\ \hat{\mathbf{v}}\end{bmatrix} = \left[0.116\ \ 0.219\ \mid\ -0.198\ \ 0.314\right]^T$. The optima with the discrete version and the same input points in (8.49) is $\begin{bmatrix}\hat{\mathbf{u}}\\ \hat{\mathbf{v}}\end{bmatrix} = \left[0.107\ \ 0.230\ \mid\ -0.182\ \ 0.322\right]^T$, yielding a $\hat{\Psi}_d = 6.10\times10^{-4}$ and $\hat{\Psi}_f = 5.83\times10^{-5}$.

Notice that despite the increase in data size, the discrete version seems less capable of converging to the analytical optimal parameters. On the contrary, the functional version generalization capability increases with the increase on the input data size.

The next tables validate the results obtained before, with the functional version. Besides being significantly better than the discrete version in terms of lower value of Ψ_f, the functional version gets close to the analytical optima despite the higher SNR value.

In the next subsection the application of the functional approach was extended to the approximation of the inverse of a non-linearity which relates the pH with the concentration (x) of chemical substances.

TABLE 8.9: TRAININGS WITH LM USING GOLUB-PEREYRA *JACOBIAN* DEFINITION (FUNCTIONAL VERSION). THE STARTING POINT IS [-1, 0.8] AND INPUT DATA SIZE IS 20.

SNR (dB)	37	27	17	15.3
N	17	17	18	18
$\bar{\mathbf{v}}[N]$	[-0.19782 0.31407]	[-0.19775 0.31396]	[-0.20008 0.31334]	[-0.19694 0.31898]
$\bar{\psi}_d[N]$	7.79e-4	7.69e-4	1.34e-3	1.80e-3
$\bar{\psi}_f[N]$	4.7032e-5	4.71e-5	5.368e-5	5.95e-5
$\bar{\psi}_d$[N] (test data)	1.44e-3	1.45e-3	1.69e-3	2.17e-3
$\overline{\left(\Psi_a\big\vert_{\mathbf{p}_f} - \hat{\Psi}_a\right)}$	1.093e-11	5.16e-8	6.65e-6	1.24e-5
$\mathrm{var}\left(\Psi_a\big\vert_{\mathbf{p}_f} - \hat{\Psi}_a\right)$	3.04e-19	8.00e-16	2.01e-11	8.21e-11
$\overline{\left(\mathbf{p}_f[N] - \hat{\mathbf{p}}_f[N]\right)}$	[-9.95e-6 -1.05e-5]	[5.60e-5 -1.22e-4]	[-2.27e-3 -7.41e-4]	[8.68e-4 4.90e-3]
$\mathrm{var}\left(\mathbf{p}_f[N] - \hat{\mathbf{p}}_f[N]\right)$	[5.65e-9 9.26e-8]	[4.84e-7 8.15e-6]	[4.24e-5 6.34e-4]	[1.34e-4 1.54e-3]

TABLE 8.10: TRAININGS WITH LM USING GOLUB-PEREYRA *JACOBIAN* DEFINITION (DISCRETE VERSION). THE STARTING POINT IS [-1, 0.8] AND INPUT DATA SIZE IS 20.

SNR (dB)	37	27	17	15.3
N	15	15	15	15
$\overline{\mathbf{v}}[N]$	[-0.18195 0.32243]	[-0.18188 0.32231]	[-0.18414 0.32274]	[-0.18106 0.32831]
$\overline{\psi}_d[N]$	6.12e-4	6.10e-4	1.05e-3	1.47e-3
$\overline{\psi}_f[N]$	5.83e-5	5.84e-5	6.37e-5	7.07e-5
$\overline{\psi}_d$[N] (test data)	1.89e-3	1.92e-3	2.13e-3	2.77e-33
$\overline{\left(\Psi_a\big\vert_{\mathbf{p}_d} - \hat{\Psi}_a\right)}$	1.12e-5	1.14e-5	1.67e-5	2.37e-5
$\mathrm{var}\left(\Psi_a\big\vert_{\mathbf{p}_d} - \hat{\Psi}_a\right)$	3.94e-15	3.672e-13	5.21e-11	1.72e-10
$\overline{(\mathbf{p}_d[N]-\hat{\mathbf{p}}_d[N])}$	[1.59e-2 8.35e-3]	[1.59e-2 8.23e-3]	[1.37e-2 8.66e-3]	[1.67e-2 1.42e-2]
$\mathrm{var}(\mathbf{p}_d[N]-\hat{\mathbf{p}}_d[N])$	[5.11e-9 7.56e-8]	[4.71e-7 8.27e-6]	[4.76e-5 5.59e-4]	[1.21e-4 1.37e-3]
$\overline{(\psi_d - \hat{\psi}_d)}$	4.52e-8	2.06e-6	1.84e-4	4.17e-4
$\mathrm{var}(\psi_d - \hat{\psi}_d)$	3.83e-12	3.23e-10	3.73e-8	8.85e-8

8.4.1.3 pH problem

The strategy here employed separates the input data into the training set and the test set, where points from one another are distinct. The usual purpose of the use of the test data is to help evaluating the ability of generalization of the estimated model, upon presence of new data. With the investigation carried out next, it will be shown that the use of the test data error as a validation criterion can be misleading and does not help on finding the correct analytical function underlying the input data.

The inverse of a titration-like (pH) curve to approximate is given by Fig. 8.2. The input-output plot shows clearly the existence of two nonlinear regions connected by a quasi-linear region, where the number of input patterns is scarce. It is assumed that the function

underlying the data is one such that the points in input range [0.2,0.6] should be connected through a straight line.

To explore the ability of the functional version of the LM using this problem, the input data was split into a training set and a test set, whose size is 2/3 and 1/3 of the original set, respectively.

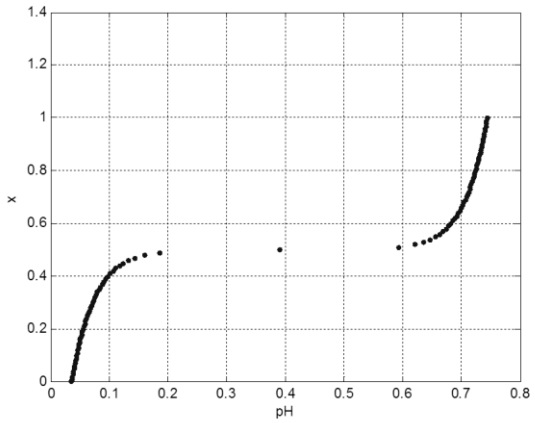


Fig. 8.2. Non-linearity relating the pH concentration with chemical substances

In order to provide results in different situations (5 runs), the input points for the two sets were obtained through a uniform random selection of the input points in the original set.

Two neurons and a bias constitute the RBF network, requiring optimization of 3 linear parameters and 4 non-linear parameters.

The initial values chosen (arbitrarily) for the nonlinear parameters are:

$$\mathbf{v}[1] = \left\{\mathbf{C}[1], \mathbf{v}[1]\right\} = \left\{\begin{bmatrix}-0.4 & -0.4\end{bmatrix}^T, \begin{bmatrix}0.3 & 0.4\end{bmatrix}^T\right\} \tag{8.50}$$

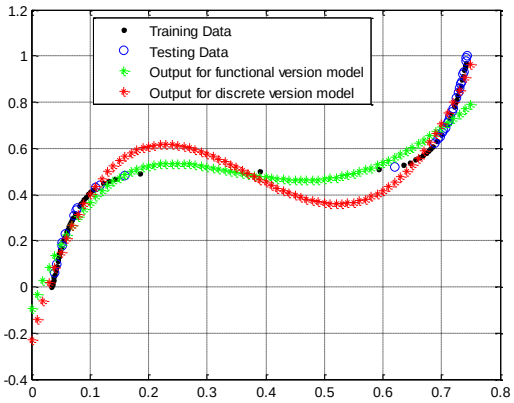
The maximum number of iterations for both algorithms was set to 100.

The following table summarizes the results where final values for the integral of the training data error $\hat{\Psi}_f\big|_{\hat{\mathbf{v}}}$, sum of the squared test error $\hat{\Psi}_d\big|_{\hat{\mathbf{v}}}$ (test data) are presented.

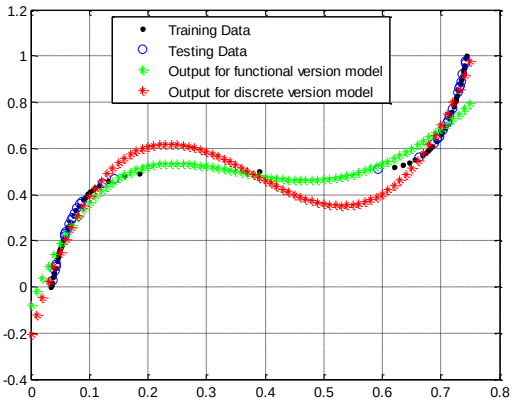
TABLE 8.11: TRAININGS WITH LM USING GOLUB-PEREYRA JACOBIAN

Run	$\hat{\Psi}_f\big\|_{\hat{\mathbf{v}}}$		$\hat{\Psi}_d\big\|_{\hat{\mathbf{v}}}$ (Test data)	
	$\hat{\mathbf{v}} = \hat{\mathbf{v}}_f$	$\hat{\mathbf{v}} = \hat{\mathbf{v}}_d$	$\hat{\mathbf{v}} = \hat{\mathbf{v}}_f$	$\hat{\mathbf{v}} = \hat{\mathbf{v}}_d$
1	4.80e-4	2.21e-3	1.37e-1	2.63e-2
2	4.48e-4	2.03e-3	9.97e-2	2.03e-3
3	4.91e-4	2.38e-3	1.21e-1	2.45e-2
4	4.35e-4	2.00e-3	1.48e-1	2.97e-2
5	1.00e-3	2.03e-3	2.17e-1	4.89e-2

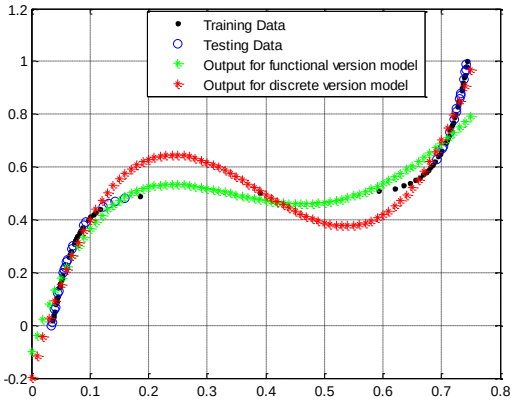
Moreover, and from the same initial starting points (and for the 5 runs), a plot over the input space covering the performance of both methodologies was drawn and shown in Fig. 8.3.



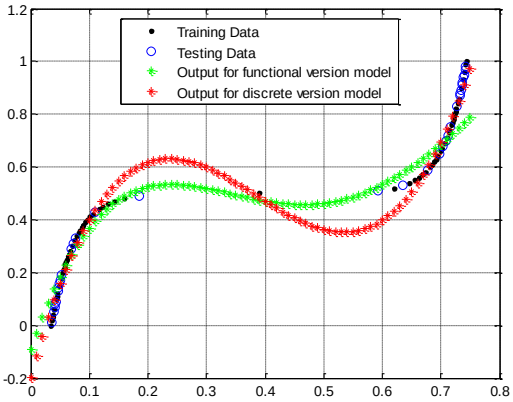
a)



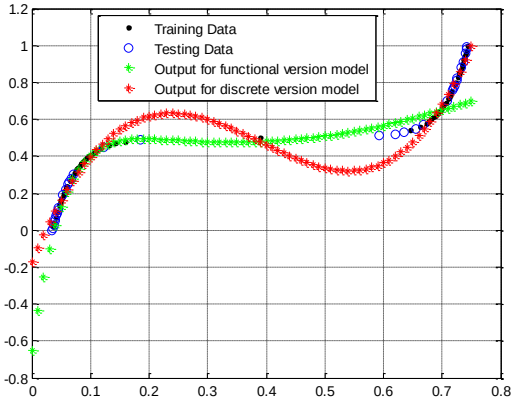
b)



c)



d)



e)

Fig. 8.3. Comparison of input-output plots between the discrete and functional versions, illustrating a better capacity of function approximation by the functional version with different training data sets (from a) till e))

As is seen, all curves produced by the functional version models, with no exception, exhibit a closer shape to the analytical function underlying the input data, in particular around the quasi-linear region of the input domain. This means that, despite the better results for ψ_d (test data) in case of the discrete version (viz.Table 8.12), the function approximation is worse. In other words, this can be regarded as an indication of how misleading model evaluation can be if test data is used (which does not cover adequately the domain). Instead, the decrease of the ψ_f value (computed from the training data only) is related to a better ability on function approximation.

TABLE 8.12: MEAN VALUES FOR ψ_f, ψ_d, ψ_d (TEST DATA) AND $\bar{\mathbf{v}}[\mathrm{N}]$ AFTER 5 RUNS

Criteria/Version	Functional version	Discrete version
$\bar{\psi}_f$	5.73e-4	2.13e-3
$\bar{\psi}_d$	2.04e-1	4.35e-2
$\bar{\psi}_d$ (Test data)	1.45e-1	2.63e-2
$\bar{\mathbf{v}}[\mathrm{N}]$	[-0.40577 0.23853 0.32836 0.4794]	[-7.91631 6.17776 0.71295 29.8391]
N	62.4	100

The mean values shown in the previous table concour with the conclusions in the last paragraph. Apart from confirming the better function approximation ability, it also states that less number of iterations are required, in the functional version case.

8.4.2 B-spline neural networks

The use of the functional approach for the B-Splines is shown in this section.

The first example shows that, besides great computational complexity savings, this approach obtains better results than the standard, discrete technique, as the performance surface employed is more similar to the one obtained with the function underlying the data. In some cases, as shown in the example, a complete analytical solution can be found.

8.4.2.1 Univariate example

For this example triangular splines (order 2), with 2 internal knots are used. The nonlinear parameters are $\mathbf{v}^T = [\lambda_1, \lambda_2]$ and there are 4 linear weights, associated with the 4 basis

functions For this model type, the objective is to approximate the function $t(x)=x^2$ - similar to a 3rd order spline, with no internal knots - by a 2nd order B-spline, with 2 interior knots. The domain considered will be $x\in[-1,1]$.

This example is chosen as it can be completely solved analytically using the equations in Section 8.2.2.

The gradient (8.17) is given by:

$$\mathbf{g}_{\Psi_f}=\frac{\begin{bmatrix}(\lambda_1-1)(\lambda_2+1)^2\left(\lambda_1+(\lambda_1-2)\lambda_2\right)\left(3+5\lambda_1+3(\lambda_1-1)\right)\lambda_2\\(\lambda_1-1)^2(\lambda_2+1)\left(3-5\lambda_2+3\lambda(\lambda_2+1)\right)\left(\lambda_1(\lambda_2+2)-\lambda_2\right)\end{bmatrix}}{6\left(3+5\lambda_2-\lambda_1(5+3\lambda_2)\right)^2} \tag{8.51}$$

Equation (8.51) is null for the point $\hat{\mathbf{v}}=\left[\hat{\lambda}_1,\hat{\lambda}_2\right]^T=[-1/3,1/3]^T$. Actually, if $x_{\min}$ and $x_{\max}$ were also parameters, the global optimum would be located at $\hat{\boldsymbol{\lambda}}=\begin{bmatrix}\frac{2x_{\min}-x_{MAX}}{3}\\\frac{x_{\min}+2x_{MAX}}{3}\end{bmatrix}$. For the case at hand, the optimal value of the integral of the squared error, has the value $\hat{\Psi}_a=\frac{4}{3645}=1.097e^{-3}$.

In this particular case, the analytical solution is known. Typically, when this is not possible (i.e., an analytically solution for $\mathbf{g}_{\Psi_f}=\mathbf{0}$ is not given), a gradient-based algorithm can be used.

In order to differ this from the case where (8.20) and (8.21) are approximated using sampled data, one shall denote the performance surface using the true function as $\Psi_a(\lambda_1,\lambda_2)$, and the performance surface using a numerical integration method as $\Psi_f(\lambda_1,\lambda_2)$.

Fig. 8.4 shows the performance surface of $\Psi_f(\lambda_1,\lambda_2)$, when the true function is used, together with four different evolutions of BP, starting from different initial points. In all simulations, the learning rate (η) is set to 1. Training stops if the maximum number of 200 iterations is reached or, if the termination criteria (see section 2.3.6.1) are verified.

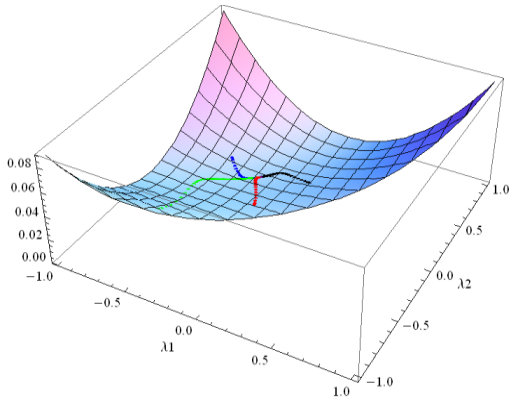


Fig. 8.4. Performance surface of Ψ_a, with 4 different trainings

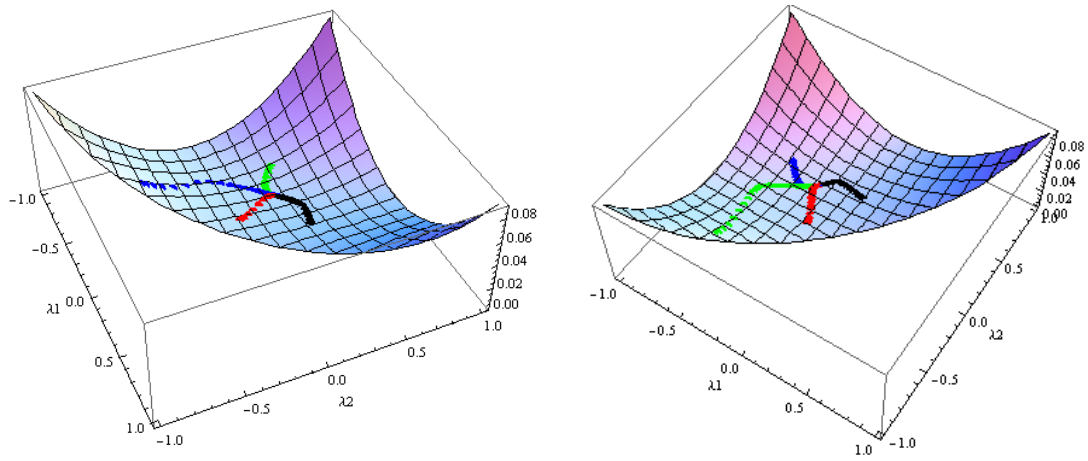
As it can be seen, the performance surface is a nice, smooth function with just 1 (global) minimum, as pointed out before. Some statistics related to the four different trainings can be found in Table 8.13.

TABLE 8.13: TRAININGS WITH BP (ANALYTICAL)

v[1]	[0 0.4]	[-0.2 0]	[-0.6 0.4]	[-0.6 -0.5]
N	195	148	179	200
$\hat{\mathbf{v}}_a$	[-0.330 0.337]	[-0.337 0.330]	[-0.337, 0.3230]	[-0.341 0.326]
$\hat{\Psi}_a$	**1.098 e-3**	**1.098 e-3**	**1.098 e-3**	**1.098 e-3**
$\Psi_f\big\vert_{\hat{\mathbf{v}}_a}$	4.322 e-4	4.322 e-4	4.322 e-4	4.353 e-4
$\Psi_d\big\vert_{\hat{\mathbf{v}}_a}$	3.132 e-3	3.132 e-3	3.132 e-3	3.261 e-3

As it can be seen the first three trainings reach the optimum (with $\tau = 1e^{-8}$) in a number of iterations smaller than the designated maximum (200). The fourth run will also converge to the global minimum, but will need a few more iterations. Better results are attained with the LM algorithm, exploiting the Golub-Pereyra and Ruano's versions of the *Jacobian* matrix. This is verified in Table 8.14 and Table 8.15, respectively.

The analytical version of the LM is, as shown, faster in convergence than the BP version. The Golub-Pereyra's *Jacobian* version of the LM, for this particular example has the faster convergence rate of all.



a) b)

Fig. 8.5. Evolution of 4 different trainings using a) Golub-Pereyra LM (analytical version); b) Ruano LM (analytical version)

TABLE 8.14: TRAININGS WITH GOLUB-PEREYRA LM (ANALYTICAL VERSION)

$\mathbf{v}[1]$	[0 0.4]	[-0.2 0]	[-0.6 0.4]	[-0.6 -0.5]
N	38	34	35	44
$\hat{\mathbf{v}}_a$	[-0.333 0.334]	[-0.334 0.333]	[-0.334 0.334]	[-0.334 0.333]
$\hat{\Psi}_a$	**1.098e-3**	**1.098e-3**	**1.098e-3**	**1.098e-3**
$\Psi_f\big\|_{\hat{v}_a}$	4.31e-4	4.31e-4	4.31e-4	4.31e-4
$\Psi_d\big\|_{\hat{v}_a}$	3.13e-3	3.13e-3	3.13e-3	3.13e-3

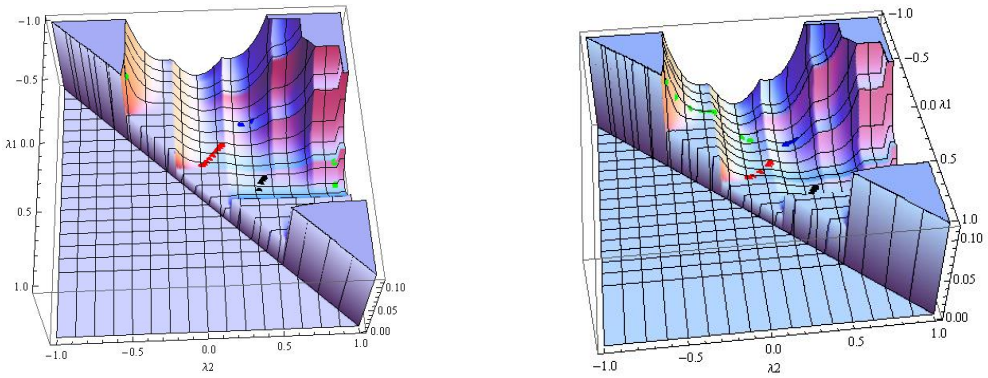
TABLE 8.15: TRAININGS WITH RUANO LM (ANALYTICAL VERSION)

$\mathbf{v}[1]$	[0 0.4]	[-0.2 0]	[-0.6 0.4]	[-0.6 -0.5]
N	90	69	86	118
$\hat{\mathbf{v}}_a$	[-0.335 0.331]	[-0.335 0.331]	[-0.335 0.331]	[-0.335 0.331]
$\hat{\Psi}_a$	**1.098e-3**	**1.098e-3**	**1.098e-3**	**1.098e-3**
$\Psi_f\big\|_{\hat{v}_a}$	4.31e-4	4.31e-4	4.31e-4	4.31e-4
$\Psi_d\big\|_{\hat{v}_a}$	3.13e-3	3.13e-3	3.13e-3	3.13e-3

In the following, it is assumed that the training function was unknown and only data was available. Consider then that 8 sample points were given by Gaussian Quadrature within the range $[-1,1]$:

$$\mathbf{x} = [0.9603, 0.8000, 0.5255, 0.1834, -0.1834, -0.5255, -0.8000, -0.9603]^T \qquad (8.52)$$

Next, the BP was applied to minimize (2.83), where the gradient is obtained using (2.84). This is the standard approach. Its performance surface, together with the evolution of the 4 different trainings, is presented in Fig. 8.6-a).



a) b)

Fig. 8.6. Performance surface of Ψ_d, with 4 different trainings a) with BP; b) with LM

As it can be seen, the discrete performance surface presents several local minima. As the model is a 2nd order spline, each pattern only activates 2 out of the 4 basis functions. This has the effect that, when the knots change, the number of patterns within each cell also varies. The performance surface is therefore not smooth.

Consider the results presented in the last line of Table 8.16, where each one of the 4 different trainings converges to different local minimum. The third line in this table shows the value obtained for Ψ_a, i.e., evaluated with the true function, for the local minimum achieved in each training. Clearly, this is the most important objective in terms of the training, as it is desired to obtain the best approximation to the function underlying the data, and not to the data itself.

TABLE 8.16: TRAININGS WITH BP (DISCRETE)

v[1]	[0 0.4]	[-0.2 0]	[-0.6 0.4]	[-0.6 -0.5]
N	11	18	13	10
$\hat{\mathbf{v}}_d$	[-0.116 0.447]	[-0.405 0.159]	[-0.578, 0.311]	[-0.028 0.943]
$\Psi_a\big\vert_{\hat{v}_d}$	2.394 e-3	2.120 e-3	2.838 e-3	6.297 e-3
$\Psi_f\big\vert_{\hat{v}_a}$	2.282 e-3	2.282 e-3	1.986 e-3	3.013 e-3
$\hat{\Psi}_d$	**8.953 e-3**	**8.953 e-3**	**5.891 e-3**	**1.112 e-2**

On the other hand, when using the LM algorithm, the evolution for the 4 same starting points is smoother (see Fig. 8.6-b)).

Table 8.17 shows the results with the LM discrete version. It is obvious that all final points diverge from the global optima and all except one diverge from the ones obtained with the BP version. Also, the speed of convergence is faster as expected.

TABLE 8.17: TRAININGS WITH LM (DISCRETE)

v[1]	[0 0.4]	[-0.2 0]	[-0.6 0.4]	[-0.6 -0.5]
N	8	8	6	10
$\hat{\mathbf{v}}_d$	[-0.114 0.450]	[-0.392 0.179]	[-0.578 0.311]	[-0.638 0.073]
$\Psi_a\big\vert_{\hat{v}_d}$	2.28e-3	2.28e-3	1.99e-3	2.15e-3
$\Psi_f\big\vert_{\hat{v}_d}$	2.42e-3	2.08e-3	2.84e-3	3.13e-3
$\hat{\Psi}_d$	**8.95e-3**	**8.95e-3**	**5.89e-3**	**8.31e-3**

Since only the training data is available, a numerical integration method for approximating (8.20) and (8.21) is required. The performance surface will be denoted as Ψ_f, then. The latter is shown in Fig. 8.7, together with the evolution of the trainings, for the same initial points.

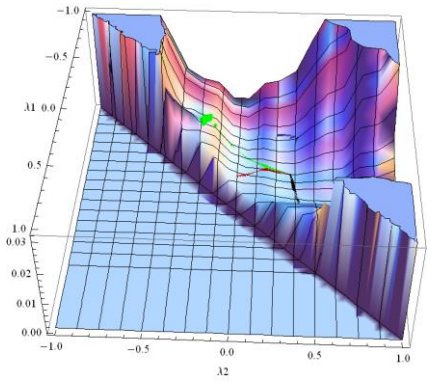


Fig. 8.7. Performance surface of Ψ_f, with 4 different trainings

Comparing the 3 performance surfaces, it is clear that Ψ_f is smoother than Ψ_d, and closer to Ψ_a, than Ψ_d. For this reason, 3 out of the 4 BP runs converge to the global minimum of Ψ_f, as shown in Table 8.18.

Analyzing the four Tables, it should first be underlined that the line in bold denotes the objective function that is actually minimized. For this reason, the values in the Ψ_f line of Table 8.18 and Table 8.19 are smaller than the corresponding line in Table 8.15, where Ψ_a is

minimized. For 3 out of the 4 initial points, the trainings using Ψ_f and BP achieve the global optima of Ψ_d. The same does not happen in any case for Ψ_d. In fact, comparing the two last lines in Table 8.16 and Table 8.18, the functional approach achieves, in 3 out of the 4 cases, better values for Ψ_d, although minimizing Ψ_f. As pointed out earlier, this is due to the fact that the functional approach achieves a better approximation to the function underlying the data, than the discrete approach.

TABLE 8.18: TRAININGS WITH BP (FUNCTIONAL)

$\mathbf{v}[1]$	[0 0.4]	[-0.2 0]	[-0.6 0.4]	[-0.6 -0.5]
N	52	53	22	131
$\hat{\mathbf{v}}_f$	[-0.338 0.338]	[-0.338 0.338]	[-0.657, 0.274]	[-0.338 0.338]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	1.108 e-3	1.108 e-3	3.460 e-3	1.108 e-3
$\hat{\Psi}_f$	**3.694 e-4**	**3.694 e-4**	**2.381 e-3**	**3.694 e-4**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	2.538 e-3	2.538 e-3	9.197 e-3	2.538 e-3

TABLE 8.19: TRAININGS WITH LM (FUNCTIONAL)

$\mathbf{v}[1]$	[0 0.4]	[-0.2 0]	[-0.6 0.4]	[-0.6 -0.5]
N	12	12	15	31
$\hat{\mathbf{v}}_f$	[-0.338 0.338]	[-0.338 0.338]	[-0.657, 0.274]	[-0.663 0.161]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	1.108 e-3	1.108 e-3	3.460 e-3	2.733 e-3
$\hat{\Psi}_f$	**3.694 e-4**	**3.694 e-4**	**2.381 e-3**	**3.247 e-3**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	2.538 e-3	2.538 e-3	9.197 e-3	1.350 e-2

As it was emphasized in sections 8.2 and 8.3, the functional approach is more efficient than the discrete approach in terms of computational complexity if some of the analytical terms can be computed beforehand. This can be relatively complex for some neural models but will require less matrix products, aside from avoiding computing pseudo-inverse which complexity grows with the size of the training data set. The next subsection will address this issue, presenting results comparing the time spent by the discrete and the main functional approaches.

8.4.2.2 Assessing computational complexity

The computational complexity of the functional version is assessed using the one-dimensional example from the last subsection. Comparison will be made in terms of time spent (in seconds) with the standard discrete approach and with the functional version, according to different sizes of input data sets. These results have been obtained with a Pentium Intel Core 2CPU@1.83Ghz, running Mathematica version 7.0.0.

Table 8.20 shows the time spent by the analytical version (i.e., when the real function was used), the functional version (BPf, LMf) and the discrete version (BPd, LMd), when there are two nonlinear parameters (two interior knots) and four linear weights.

TABLE 8.20: TIME PER ITERATION SPENT OPTIMIZING 2 PARAMETERS, USING DIFFERENT TRAINING DATA SET SIZES

Algorithm	Number of samples			
	8	100	500	1000
BPana	0,016	0,016	0,016	0,016
BPf	0,006	0,027	0,090	0,188
BPd	0,002	0,015	0,157	0,780
LMana	0,020	0,020	0,020	0,020
LMf	0,008	0,027	0,116	0,208
LMd	0,001	0,024	0,174	0,808

As expected, the analytical approaches do not depend on the training data size; the LM algorithm spends slightly more time than BP and above 100 input samples, the analytical trainings are increasingly more efficient than the other approaches. The same trend is observed with the functional versions of the algorithms; the functional version of the LM spends substantially more time than the BP version for small training data sets but, with the increase of data sets size the influence in computing time diminishes.

Comparing the results between the functional and discrete versions, the latter is faster for small sample sizes, but is much slower for larger number of samples. The time taken by the functional version increases almost linearly with the samples set size. The same is not verified for the discrete version, since the time spent increases exponentially with the number of training input samples.

Increasing the model complexity to four interior knots gives similar conclusions. The corresponding results are listed in Table 8.21. When comparing the two tables one verifies

that the functional version time increases, on average, less than the discrete version, with an increasing number of parameters.

TABLE 8.21: TIME PER ITERATION SPENT OPTIMIZING 4 PARAMETERS, USING DIFFERENT TRAINING DATA SET SIZES

Algorithm	Number of samples			
	8	100	500	1000
BPana	0,024	0,024	0,024	0,024
BPf	0,010	0,028	0,184	0,322
BPd	0,004	0,034	0,290	1,506
LMana	0,027	0,027	0,027	0,027
LMf	0,019	0,052	0,197	0,393
LMd	0,006	0,036	0,306	1,754

8.4.2.3 Performance with other input data

The example discussed in 8.4.2.1 showed that the performance of the functional approach to a B-spline model is more efficient than its discrete counterpart. Notice though, that the input samples used for the trainings had pre-defined locations over the input domain, since they correspond to the nodes of the Gaussian Quadrature methodology.

As this can be advantageous to the functional approach (because it chooses the most adequate input samples for the purpose of numerical integration) this subsection investigates the application of the functional and discrete approaches when distinct input training data sets are used. Moreover, this will be done using two different target functions:

I - $t(x) = \dfrac{\sin(10x)}{x}$

II - pH problem, where the objective is to approximate the inverse of a titration-like curve.

From these examples it is possible to assess the ability to approximate the function underlying the sample points, since they are completely solved analytically using the equations in Appendix C.

In the following subsections, triangular splines of order 2 with 2 internal knots will be considered. The nonlinear parameters are $\mathbf{v}^T = [\lambda_1, \lambda_2]$. There are 4 linear weights, associated with the 4 basis functions The domain considered will be $x \in [-1,1]$.

In the functional approach, training stops when the maximum number of 30 iterations is reached, or if the set of termination criteria (chapter 2, section 2.3.6.1).

The numerical integration method used is the trapezoidal rule. The LM update is performed using Golub-Pereyra's *Jacobian* matrix.

8.4.2.3.1 Case I

Note that using the analytical target function definition then the analytical value for the training criterion is 4.4 for either one of the 2 optima locations [-0.361, -0.116] or [0.116, 0.361].

Fig. 8.8 illustrates the analytical performance surface. Notice that the shape of such surface, for the discrete case, is severely dependent on the number of input samples used for training and their locations. In the following experiments only the location of the input samples is shown, not the performance surfaces.

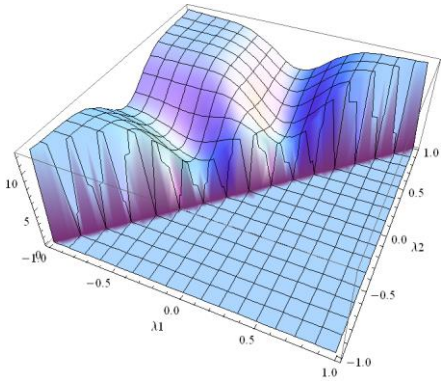


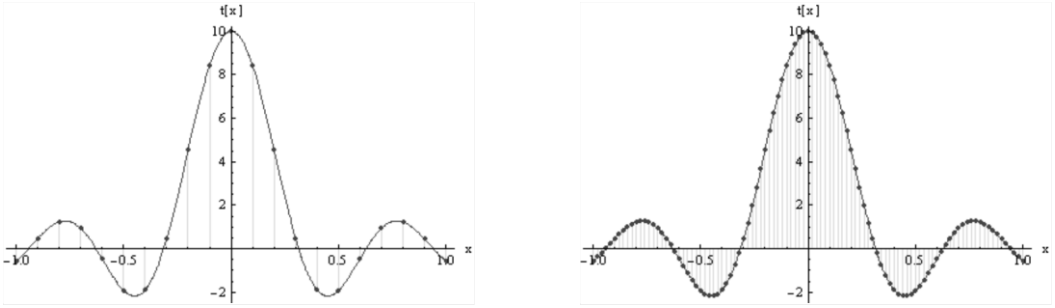
Fig. 8.8. Performance surface for the analytical approach

8.4.2.3.1.1 Evenly distributed samples

There are two experiments considered. The first one consists of 21 input points spread over the input domain [-1, 1], resulting from a discretization step of 0.1. The second one consists of 101 samples.

Table 8.22-Table 8.24 summarize the trainings using 21 input samples, whereas Table 8.25 and Table 8.26 refer to the case when 101 input samples are used.

The following figure illustrates the samples used for training.



a) b)

Fig. 8.9. Plot of target function, with the discrete samples marked by a dot. a) 21 samples resulting from a discretization step of 0.1; b) 101 samples resulting from a discretization step of 0.02;

TABLE 8.22: TRAININGS WITH LM (ANALYTICAL)

v[1]	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	8	13	8	10
$\hat{\mathbf{v}}_a$	[-0.361, 0.120]	[-2e04, 2e-4]	[0.120, 0.361]	[0.120, 0.361]
$\hat{\Psi}_a$	**4.397**	**7.44**	**4.397**	**4.397**
$\Psi_f\big\vert_{\hat{\mathbf{v}}_a}$	4.397	7.44	4.398	4.398
$\Psi_d\big\vert_{\hat{\mathbf{v}}_a}$ (21 samples)	46	73	46	46
$\Psi_d\big\vert_{\hat{\mathbf{v}}_a}$ (101 samples)	222	373	222	222

TABLE 8.23: TRAININGS WITH LM (FUNCTIONAL)

v[1]	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	7	15	6	10
$\hat{\mathbf{v}}_f$	[-0.356, -0.122]	[-0.021, 0.021]	[0.122, 0.356]	[0.122, 0.356]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	4.40	7.40	4.40	4.40
$\hat{\Psi}_f$	**4.40**	**7.21**	**4.40**	**4.40**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	46	73	46	46

TABLE 8.24: TRAININGS WITH LM (DISCRETE) USING 21 SAMPLES

v[1]	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	6	30	5	9
$\hat{\mathbf{v}}_d$	[-0.351, -0.125]	[-0.0, 0.0]	[0.125, 0.351]	[0.125, 0.351]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_d}$	4.43	7.47	4.43	4.43
$\Psi_f\big\vert_{\hat{\mathbf{v}}_d}$	4.42	7.30	4.42	4.42
$\hat{\Psi}_d$	**46**	**74**	**46**	**46**

TABLE 8.25: TRAININGS WITH LM (FUNCTIONAL) USING 101 SAMPLES

v[1]	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	7	14	6	9
$\hat{\mathbf{v}}_f$	[-0.356, -0.122]	[-0.021, 0.021]	[0.122, 0.356]	[0.122, 0.356]
$\Psi_a\|_{\hat{\mathbf{v}}_f}$	4.40	7.43	4.40	4.40
$\hat{\Psi}_f$	**4.40**	**7.43**	**4.40**	**4.40**
$\Psi_d\|_{\hat{\mathbf{v}}_f}$	222	373	222	222

TABLE 8.26: TRAININGS WITH LM (DISCRETE) USING 101 SAMPLES

v[1]	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	17	30	15	23
$\hat{\mathbf{v}}_d$	[-0.362, -0.120]	[-0.0, 0.0]	[0.120, 0.362]	[0.120, 0.362]
$\Psi_a\|_{\hat{\mathbf{v}}_d}$	4.40	7.44	4.40	4.40
$\Psi_f\|_{\hat{\mathbf{v}}_d}$	4.40	7.44	4.40	4.40
$\hat{\Psi}_d$	**222**	**373**	**222**	**222**

The results confirm that any of the approaches can have a good approximation to the function underlying the data, as long as the input points are well spread along the input domain.

The following subsection, will investigate the performance of the functional approach if the input points are randomly drawn from a sample distribution probability function.

To investigate this issue, the performance of both approaches will be analyzed, for input samples either drawn from uniform and Gaussian distribution functions.

8.4.2.3.1.2 Randomly generated samples from a uniform distribution

These results show the mean values for the training criteria, obtained from 10 runs. The training data set consists of 60 input samples.

Using a random uniform distribution function for the selection of the input samples allows any of the approaches to obtain, in average, good approximation to the true function. In the case of the functional approach, two out of the four final points coincide with the optima (1st and 3rd).

TABLE 8.27: MEAN VALUES WITH LM (FUNCTIONAL)

v[1]	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	7	15	8	11
$\hat{\mathbf{v}}_f$	[-0.357, -0.122]	[-0.0087, 0.033]	[0.122, 0.362]	[0.298, 0.480]
$\Psi_a\|_{\hat{\mathbf{v}}_f}$	4.41	6.63	4.41	7.88
$\hat{\Psi}_f$	**4.30**	**6.42**	**4.23**	**7.57**
$\Psi_d\|_{\hat{\mathbf{v}}_f}$	121	176	117	206

Comparatively, the discrete has poorer performance though being close to the optima in the same two trainings (but around 8% worse). The discrete approach yields a better model when the 4th training is carried out, which may be related to the lack of input points around that specific input range (65% better approximation).

TABLE 8.28: MEAN VALUES WITH LM (DISCRETE)

v[1]	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	7.3	14	9	13
$\hat{\mathbf{v}}_d$	[-0.355, -0.130]	[-0.136, 0.178]	[0.131, 0.368]	[0.116, 0.427]
$\Psi_a\|_{\hat{\mathbf{v}}_d}$	4.76	7.13	4.78	5.08
$\Psi_f\|_{\hat{\mathbf{v}}_d}$	4.62	6.83	4.54	4.83
$\hat{\Psi}_d$	**113**	**165**	**108**	**116**

8.4.2.3.1.3 Randomly generated samples from normal distribution

In this example, 20 input points are drawn from a normal distribution with zero mean and standard deviation 0.25. In this particular case, the plot of input points versus output is as follows:

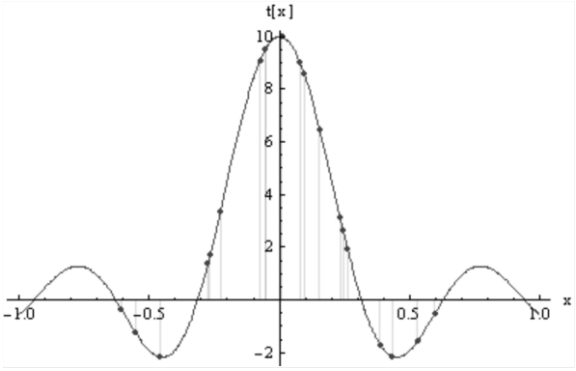


Fig. 8.10. Plot of target function, with the discrete samples marked by a dot

The following tables give a summary of the results provided by both approaches.

TABLE 8.29: TRAININGS WITH LM (FUNCTIONAL)

$\mathbf{v}[1]$	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	10	15	16	14
$\hat{\mathbf{v}}_f$	[-0.696, -0.004]	[0.0322, 0.689]	[0.0149, 0.669]	[0.0335, 0.701]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	6.77	6.71	6.59	6.81
$\hat{\Psi}_f$	**4.75**	**4.52**	**4.53**	**4.51**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	67	61	59	62

TABLE 8.30: TRAININGS WITH LM (DISCRETE)

$\mathbf{v}[1]$	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	3	2	4	7
$\hat{\mathbf{v}}_d$	[-0.7, 0.0013]	[-0.6, 0.6]	[0.0013, 0.6]	[0.0013, 0.6]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_d}$	22.2	4673	16.4	16.4
$\Psi_f\big\vert_{\hat{\mathbf{v}}_d}$	5.53	14.18	5.53	5.53
$\hat{\Psi}_d$	**31**	**185**	**31**	**31**

Regarding the input/output plot, at least two regions lack the presence of input samples. This means there is no way of predicting how the function behaves in those regions since the training patterns do not carry that information. Despite this, the functional approach performs reasonably well, as it converges to final points whose analytical integral of the errors is around 50% above the optima but 220% below (in the worst case) the ones given by the discrete approach.

For this target function, any one of the performance surfaces, say the functional or the discrete approach, as well as the analytical approach, illustrate the existence of 2 different optima, with opposite locations and defined by the symmetry of the surface. Both provide $\Psi_a = 4.397$.

The next figure illustrates the plot of the input samples versus the output when the analytical approach is used (or the true function is known).

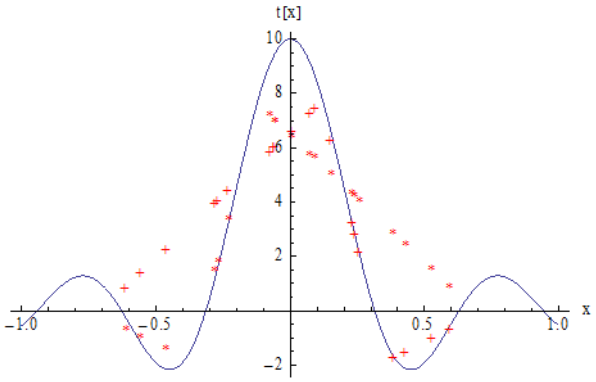


Fig. 8.11. Solid line: Plot of target function; the "+" line corresponds to the input-output plot for the model with optima at $\hat{\mathbf{v}}_1 = [0.120,\ 0.361]$; the "*"line corresponds to the input-output plot for the model with optima at $\hat{\mathbf{v}}_2 = [-0.361, -0.120]$

The following figures compare the output provided by the models from the functional and discrete cases for each one of the final points according to Table 8.29 and Table 8.30.

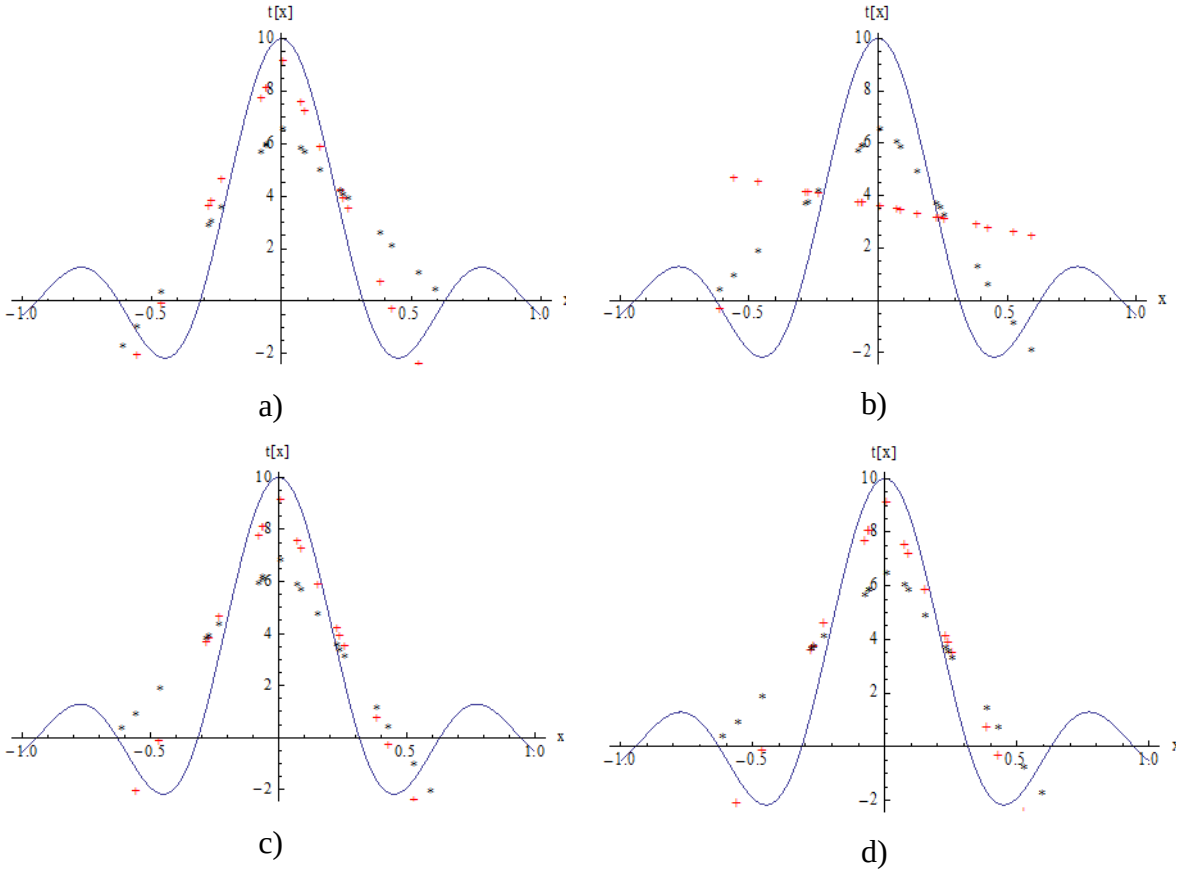


Fig. 8.12. Solid line: Plot of target function; the "+" line corresponds to the input-output plot for the discrete model; the "*"line corresponds to the input-output plot for the functional model. Subheading a),b),..d) denotes the final parameters for the training with the 1st, 2nd, …,4th starting points.

From the previous figures, one can conclude the following. None of the plots given by the discrete version show a behavior anywhere close to the shape of that illustrated by the analytical approach. Hence, if the input samples happen to be more concentrated around a specific input range, the performance of the discrete model is highly biased (an example of this is training illustrated by the plot in Fig. 8.12-b)). The reason for this performance is the fact that the input samples contributing most to the training criterion (sum of square of errors) are concentrated around zero and are not sufficient to represent the function over the input domain. On the other hand, though not really reaching any of the two optima, the functional approach has a very meritorious performance not only because of the shape of the curves resulting from the trainings but also because the functional error value is close to the lowest possible value.

In order to confirm the former results, the following two tables summarize results corresponding to 10 runs with trainings for the same starting points but where the random input samples are drawn from a normal distribution with zero mean and standard deviation 0.5.

TABLE 8.31: MEAN VALUES WITH LM (FUNCTIONAL)

$\mathbf{v}[1]$	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	8	17	6	9
$\hat{\mathbf{v}}_f$	[-0.368, -0.120]	[-0.0083, -0.015]	[0.117, 0.366]	[0.155, 0.382]
$\Psi_a\|_{\hat{\mathbf{v}}_f}$	4.41	6.78	4.41	5.08
$\hat{\Psi}_f$	**4.40**	**6.84**	**4.43**	**5.11**
$\Psi_d\|_{\hat{\mathbf{v}}_f}$	236	385	244	297

TABLE 8.32: MEAN VALUES WITH LM (DISCRETE)

$\mathbf{v}[1]$	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	8	23	7	11
$\hat{\mathbf{v}}_d$	[-0.381, -0.095]	[-0.052, 0.0088]	[0.0088, 0.376]	[0.078, 0.430]
$\Psi_a\|_{\hat{\mathbf{v}}_d}$	5.66	8.81	6.26	7.52
$\Psi_f\|_{\hat{\mathbf{v}}_d}$	5.60	8.84	6.26	7.52
$\hat{\Psi}_d$	**202**	**321**	**196**	**214**

In all cases, the functional approach leads to lower values of the analytical training error criterion. This improvement is within the range [28%, 47%].

Suppose now an increase of the number of random samples to 200, using the same normal distribution, for the same number of 10 runs. This will be provide a sufficient number of input training patterns spread across the input domain, though the majority of them is concentrated around zero. Overall, one would expect the trainings to converge to a good solution. This is not the case for the discrete approach, as will be illustrated in the next tables.

TABLE 8.33: MEAN VALUES WITH LM (FUNCTIONAL)

$\mathbf{v}[1]$	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	7	18	6	9
$\hat{\mathbf{v}}_f$	[-0.364, -0.120]	[-0.0038, -0.033]	[0.119, 0.363]	[0.119, 0.363]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	4.39	7.12	4.39	4.39
$\hat{\Psi}_f$	**4.48**	**7.21**	**4.46**	**4.46**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	501	843	484	484

TABLE 8.34: MEAN VALUES WITH LM (DISCRETE)

$\mathbf{v}[1]$	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	8	18	7	10
$\hat{\mathbf{v}}_d$	[-0.374, -0.094]	[-0.096, 0.0095]	[0.088, 0.374]	[0.088, 0.374]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_d}$	5.66	8.10	5.65	5.65
$\Psi_f\big\vert_{\hat{\mathbf{v}}_d}$	5.56	8.10	6.61	5.61
$\hat{\Psi}_d$	**440**	**661**	**410**	**410**

From the former two tables one can say that:

- minimizing the sum of square errors does not signify the best approximation to the true function.
- with this distribution of the samples, the functional approach converges to the optima in the same way as the analytical approach does. The discrete fails to do so in all of the trainings.

8.4.2.3.2 Case II

The titration-like (pH) curve to approximate is:

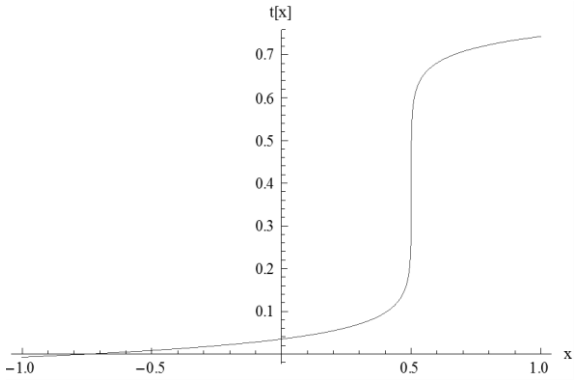


Fig. 8.13. Output-Input plot for the titration-like curve

The usual approach in control is to approximate the inverse of this titration-like curve. To show the advantages of using the functional approach on the direct mapping, this subsection studies the effects on the trainings, if the training data is concentrated on certain regions of the input space. The

Though seeming to be a relatively simple problem, the analytical performance surface (see Fig. 8.14) has a clear distinct optima located at [0.486, 0.512] with a value of $\hat{\Psi}_a = 2.42e-4$.

8.4.2.3.2.1 Input samples generated from Gaussian quadrature

Suppose a training data set of 50 samples generated from Gaussian quadrature.

The next figure shows trainings for 4 starting points using the functional approach and using the fact the true function is known (denoted as the analytical approach).

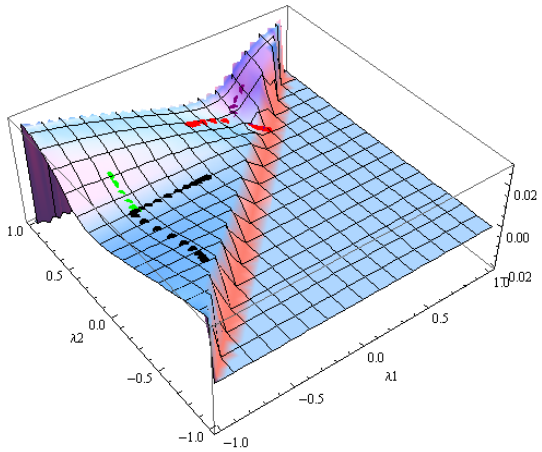


Fig. 8.14. Evolution of trainings for 4 starting points (LM Analytical approach)

TABLE 8.35: SUMMARY OF TRAININGS WITH LM (ANALYTICAL)

$\mathbf{v}[1]$	[-0.7, -0.5]	[-0.7, 0.5]	[0, 0.7]	[0.5, 0.8]
N	30	9	22	20
$\hat{\mathbf{v}}_a$	[-0.178, 0.240]	[-0.681, 0.234]	[0.484, 0.513]	[0.484, 0.513]
$\hat{\Psi}_a$	**7.36e-3**	**7.36e-3**	**2.44e-4**	**2.44e-4**
$\Psi_f\big\vert_{\hat{v}_a}$	7.38e-3	7.38e-3	2.91e-4	2.98e-5
$\Psi_d\big\vert_{\hat{v}_a}$	0.215	0.215	6.33e-3	6.47e-3

Now, if the true function is not known and the input samples are used to compute the integral of the errors, the functional approach depends on the numerical integration method (the trapezoidal in this case) and convergence is illustrated in the following figure.

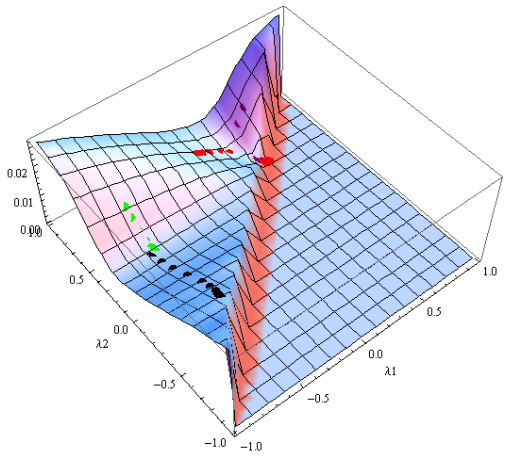


Fig. 8.15. Evolution of trainings for 4 starting points (LM functional approach)

On the other hand, performing the usual sum of the squares of errors, the trainings are illustrated in the following figure.

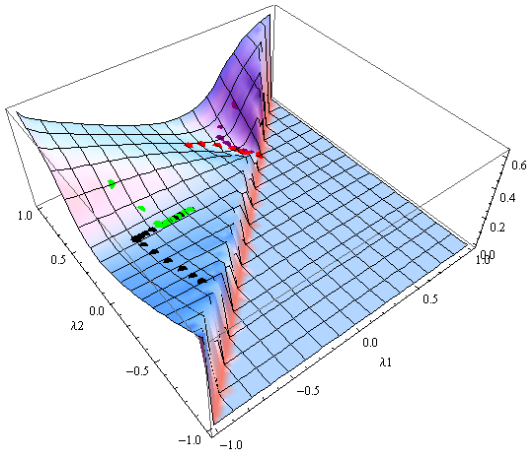


Fig. 8.16. Evolution of trainings for 4 starting points (LM discrete approach)

The following tables give a summary of the results provided by both approaches.

TABLE 8.36: TRAININGS WITH LM (FUNCTIONAL)

$\mathbf{v}[1]$	[-0.7, -0.5]	[-0.7, 0.5]	[0, 0.7]	[0.5, 0.8]
N	12	9	16	12
$\hat{\mathbf{v}}_f$	[-0.726, 0.210]	[-0.678, 0.219]	[0.490, 0.507]	[0.443, 0.517]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	7.42e-3	7.41e-3	2.65e-4	9.28e-4
$\hat{\Psi}_f$	**7.34e-3**	**7.34e-3**	**1.49e-4**	**8.33e-5**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	2.13e-1	2.16e-1	4.65e-3	2.88e-3

TABLE 8.37: TRAININGS WITH LM (DISCRETE)

$\mathbf{v}[1]$	[-0.7, 0]	[-0.6, 0.6]	[0.1, 0.6]	[0.5, 0.6]
N	19	17	8	11
$\hat{\mathbf{v}}_d$	[-0.351, 0.152]	[-0.351, 0.152]	[0.466, 0.495]	[0.443, 0.518]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_d}$	8.29e-3	8.29e-3	1.44e-3	9.72e-4
$\Psi_f\big\vert_{\hat{\mathbf{v}}_d}$	8.17e-3	8.17e-3	3.43e-4	7.18e-5
$\hat{\Psi}_d$	**1.76e-1**	**1.76e-1**	**6.89e-3**	**1.43e-3**

8.4.2.3.2.2 Randomly generated samples from uniform distribution

The following results show the mean values for the training criteria, obtained from 10 runs. The training data set consists of 60 input samples.

A random uniform distribution of the input samples seems to bring more benefits to the discrete approach. As observed in the next 2 tables, the functional approach leads to worse models in 3 out of the 4 trainings. None of the methods reach the optima.

TABLE 8.38: MEAN VALUES WITH LM (FUNCTIONAL)

$\mathbf{v}[1]$	[-0.7, -0.5]	[-0.7, 0.5]	[0, 0.7]	[0.5, 0.8]
N	16	9	14	10
$\hat{\mathbf{v}}_f$	[-0.584, 0.081]	[-0.684, 0.232]	[0.408, 0.566]	[0.431, 0.582]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	9.83e-3	7.46e-3	2.14e-3	2.18e-3
$\hat{\Psi}_f$	**9.53e-3**	**7.18e-3**	**1.23e-3**	**1.17e-3**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	2.18e-1	2.02e-1	3.20e-2	3.20e-2

TABLE 8.39: MEAN VALUES WITH LM (DISCRETE)

$\mathbf{v}[1]$	[-0.7, -0.5]	[-0.7, 0.5]	[0, 0.7]	[0.5, 0.8]
N	18	16	10	9
$\hat{\mathbf{v}}_d$	[-0.310, 0.184]	[-0.277, 0.252]	[0.435, 0.547]	[0.439, 0.544]
$\Psi_a\|_{\hat{\mathbf{v}}_d}$	9.0e-3	7.87e-3	1.29e-3	1.49e-3
$\Psi_f\|_{\hat{\mathbf{v}}_d}$	8.67e-3	7.54e-3	9.80e-4	1.59e-4
$\hat{\Psi}_d$	**2.18e-1**	**1.87e-1**	**2.74e-3**	**3.70e-3**

8.4.2.3.2.3 Randomly generated samples from normal distribution

In this example, 20 input points are drawn from a normal distribution with mean value 0.5 and standard deviation 0.2. In this particular case, the plot of input points versus output is as follows:

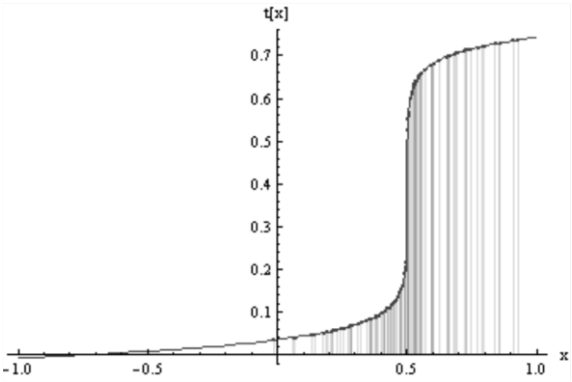


Fig. 8.17. Plot of target function, with the discrete samples marked by a dot

As it can be seen, only one of the selected samples is below the origin which means there is no information about the input-output relation for this input range.

Using these samples the performance of the functional and discrete approaches is summarized in the following tables.

TABLE 8.40: TRAININGS WITH LM (FUNCTIONAL)

$\mathbf{v}[1]$	[-0.7, -0.5]	[-0.7, 0.5]	[0, 0.7]	[0.5, 0.8]
N	11	9	18	16
$\hat{\mathbf{v}}_f$	[-0.625, -0.137]	[-0.686, 0.238]	[0.477, 0.517]	[0.447, 0.517]
$\Psi_a\|_{\hat{\mathbf{v}}_f}$	1.22e-2	7.62e-3	7.38e-3	7.38e-3
$\hat{\Psi}_f$	**1.15e-2**	**7.59e-3**	**1.10e-3**	**1.10e-3**
$\Psi_d\|_{\hat{\mathbf{v}}_f}$	1.80	1.23	9.87e-2	9.87e-2

TABLE 8.41: TRAININGS WITH LM (DISCRETE)

v[1]	[-0.7, -0.5]	[-0.7, 0.5]	[0, 0.7]	[0.5, 0.8]
N	2	8	4	15
$\hat{\mathbf{v}}_d$	[-0.7, -0.5]	[-0.7, 0.318]	[0, 0.699]	[0.492, 0.506]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_d}$	0.135	1.23e-2	123	1.12e-2
$\Psi_f\big\vert_{\hat{\mathbf{v}}_a}$	5.77e-2	1.06e-2	8.96e-3	6.87e-4
$\hat{\Psi}_d$	**1.20**	**0.99**	**1.04**	**1.28e-2**

These results confirm the similar experiments taken in the previous section where the function from case I was used as target. Now, in all of the trainings the functional approach gives fairly better models.

Remember that at the optima $\hat{\mathbf{v}}^T = [0.486,\ 0.512]$ and $\hat{\Psi}_a = 2.42e-4$.

The next figure illustrates the plot of the input samples versus the output when the analytical approach is used (or the true function is known).

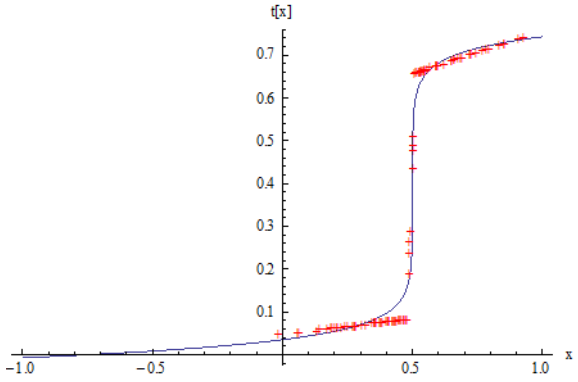
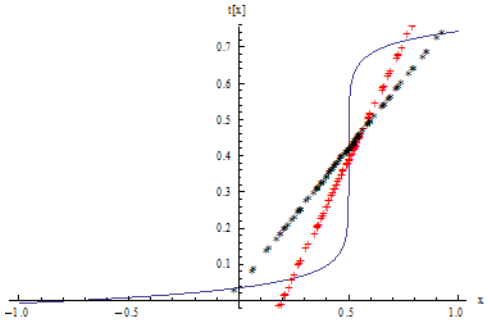


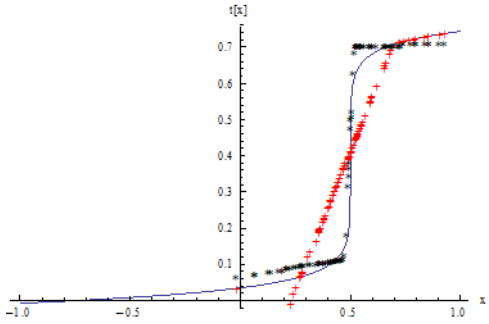
Fig. 8.18. Drawing of the Input-output plot for the samples used in the training (analytical approach)

Fig. 8.19 compares the input-output plots for the two worse trainings (1st and 3rd) in terms of the discrete approach.

Observing the plot, the 1st and the 3rd trainings look similar, but produce much different values in terms of Ψ_a. The functional approach returns its best fitness in the 3rd training, very close to the one shown by the analytical approach (see Fig. 8.18).



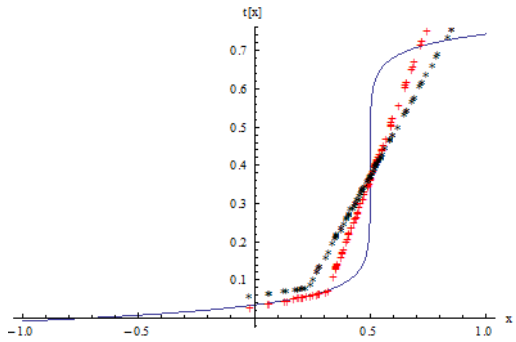
a) Plot for model given by 1st training



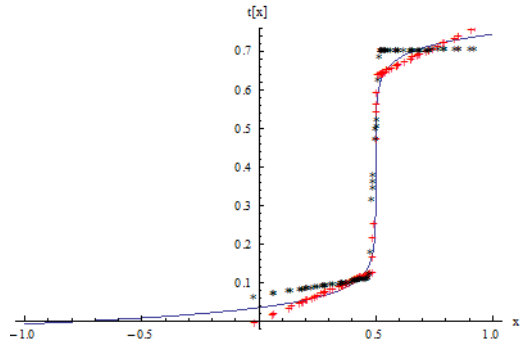
b) Plot for model given by 3rd training

Fig. 8.19. The "+" line corresponds to the input-output plot for the discrete model; the "*"line corresponds to the input-output plot for the functional model.

Similar conclusions are drawn when observing the other 2 models behavior:



a) Plot for model given by 2nd training



b) Plot for model given by 4th training

Fig. 8.20. The "+" line corresponds to the input-output plot for the discrete model; the "*"line corresponds to the input-output plot for the functional model.

As a matter of fact, the 2nd and 4th trainings yield similar models in terms of Ψ_a, but very different when comparing the plots.

When comparing the 4th training, the parameters of the discrete model are almost coincident to those given by the functional approach, but with a difference of 50% in the value of Ψ_a. This is due to the different slopes of the interpolating lines, which may be explained by the linear parameters computed in each of the approaches (values not shown here).

8.4.2.4 Two dimensional problems

This section introduces the functional approach to two bivariate problems, using the Gaussian quadrature integration technique for the two-dimensional case (2.171). It is divided into two 2 main subsections.

The trainings will be carried out with the LM method, where the trainings stop if the maximum number of 200 iterations is reached or if the criteria in section 2.3.6.1 with $\tau = 10^{-8}$, are satisfied.

8.4.2.4.1 First example

In the first example the BSNN model is composed of triangular splines, with 1 interior knots in both dimensions. The nonlinear parameters are $\mathbf{v}^T = \left[\lambda_{1,1}, \lambda_{2,1}\right]$, hence there are 9 linear weights, associated with the tensor product between 3 basis functions from each dimension. For this example, the function to be approximated will be $t_1(\mathrm{x}) = t(x_1, x_2) = x_1^2 x_2^2$ which means that a bivariate spline of 3rd order in each dimension, with no internal knots, will be approximated by a bivariate spline, 2nd order with 1 interior knot for each dimension. The domain considered will be $\{x_1, x_2\} \in [-1,1]$.

The input data form a bivariate grid with 8 samples in each dimension:

$$\mathbf{x}_{1,2} = \begin{Bmatrix} 0.96029, 0.796666, 0.525532, 0.183435, \\ -0.183435, -0.525532, -0.796666, -0.96029 \end{Bmatrix} \tag{8.53}$$

The target function to be approximated is depicted in the 3D plot of Fig. 8.21. The analytical, functional and discrete performance surfaces are shown in Fig. 8.22, which also illustrate the evolution of 4 different trainings, starting from 4 different initial points.

The analytical optimum is located at $\left[\lambda_{1,1}, \lambda_{2,1}\right] = [0,0]$ with an analytical error criterion $\hat{\Psi}_a = 4,381 \times 10^{-3}$. As it can be seen in Fig. 8.22-a), the analytical performance surface has one global optimum, and the 4 different trainings converge to this optimum. These analytical trainings are summarized in Table 8.42. The corresponding trainings with the functional and discrete versions are shown in Table 8.43 and Table 8.44, respectively. In all the tables, the line with a bold typeface refers to the criterion that is minimized.

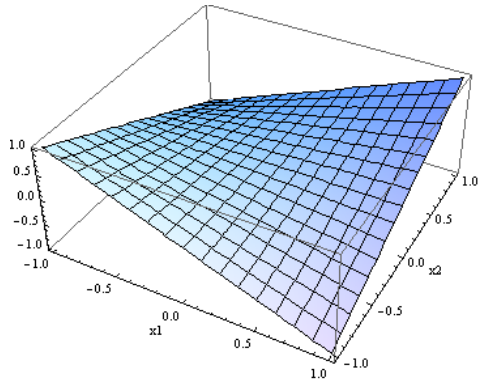
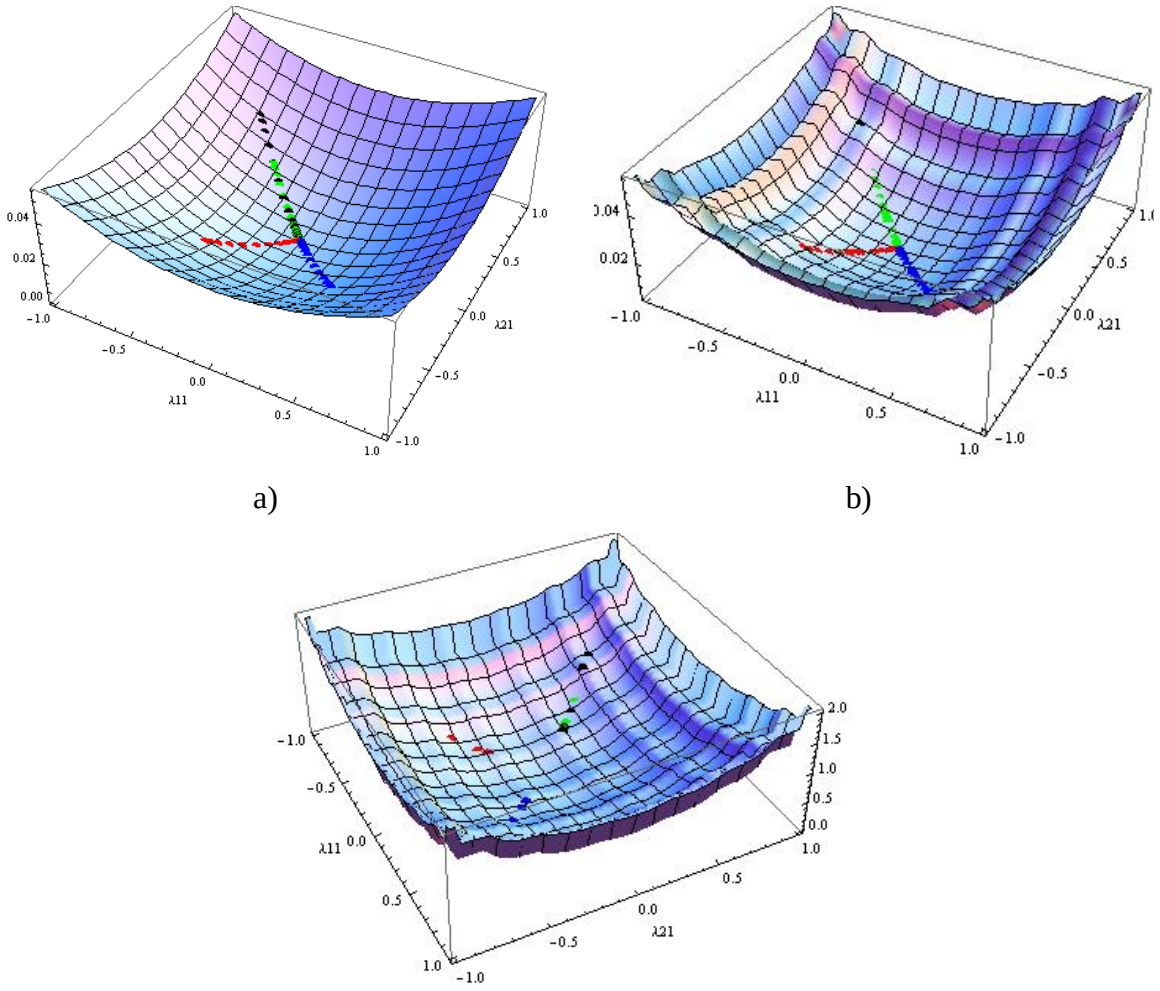


Fig. 8.21. Input-Output 3D plot for $t_1(x)$



a) b)

c)

Fig. 8.22. Evolution of 4 different trainings for target function $t_1(x_1,x_2)$: a) with LM analytical version; b) with LM functional version; c) with LM discrete version.

In this example, the performance surfaces of the functional and discrete versions exhibit many local optima in contrast to that from the analytical approach. This is due to the fact that

B-Splines are piece-wise linear polynomials, the location of the training input samples being decisive on the smoothness of the surface. Nevertheless, the functional version is smoother than its discrete counterpart which is in line with the trainings convergence shown by the corresponding tables.

TABLE 8.42: TRAININGS WITH LM (ANALYTICAL)

v[1]	[-0.6 0.6]	[-0.4 -0.4]	[0.4 -0.4]	[-0.4 0.4]
N	26	25	25	25
$\hat{\mathbf{v}}_a$	[-1e-4 1e-4]	[-1e-4, -1e-4]	[1e-4, -1e-4]	[-1e-4 1e-4]
$\hat{\Psi}_a$	**4.38e-3**	**4.38e-3**	**4.38e-3**	**4.38e-3**
$\Psi_f\big\vert_{\hat{\mathbf{v}}_a}$	3.12e-3	3.12e-3	3.12e-3	3.12e-3
$\Psi_d\big\vert_{\hat{\mathbf{v}}_a}$	1.48e-1	1.48e-1	1.48e-1	1.48e-1

Table 8.43 shows that the functional version has a very reasonable performance since 3 out of the 4 cases reach the global optimum. The same is not achieved by the discrete version, as it is verified in Table 8.44.

TABLE 8.43: TRAININGS WITH LM (FUNCTIONAL)

v[1]	[-0.6 0.6]	[-0.4 -0.4]	[0.4 -0.4]	[-0.4 0.4]
N	8	26	26	26
$\hat{\mathbf{v}}_f$	[-0.587 0.587]	[-1e-4, -1e-4]	[1e-4, -1e-4]	[-1e-4 1e-4]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	2.51e-2	4.38e-3	4.38e-3	4.38e-3
$\hat{\Psi}_f$	**2.47e-2**	**4.61e-3**	**4.61e-3**	**4.61e-3**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	3.35	1.49e-1	1.49e-1	1.49e-1

TABLE 8.44: TRAININGS WITH LM (DISCRETE)

v[1]	[-0.6 0.6]	[-0.4 -0.4]	[0.4 -0.4]	[-0.4 0.4]
N	7	6	6	6
$\hat{\mathbf{v}}_d$	[-0.236 0.236]	[-0.236 -0.236]	[0.236 -0.236]	[-0.236 0.236]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_d}$	9.61e-3	9.61e-3	9.61e-3	9.61e-3
$\Psi_f\big\vert_{\hat{\mathbf{v}}_d}$	1.054e-2	1.054e-2	1.054e-2	1.054e-2
$\hat{\Psi}_d$	**2.35e-1**	**2.35e-1**	**2.35e-1**	**2.35e-1**

8.4.2.4.2 Second example

In this example, the target function to be approximated is now:

$$t_2(x_1, x_2) = e^{-10\left((-0.7+x_1)^2+(-0.7+x_2)^2\right)} + e^{-5\left((0.7+x_1)^2+(0.7+x_2)^2\right)} \tag{8.54}$$

This target function is equivalent to a RBF with 2 neurons. The corresponding 3D plot is given in Fig. 8.23. The same B-spline model is used here.

The analytical performance surface is shown in Fig. 8.24-a), together with 4 different training evolutions. The global optimum is located at $\left[\lambda_{1,1}, \lambda_{2,1}\right] = [0.079, 0.079]$ with a value of $\hat{\Psi}_a = 0.0303$. The functional performance surface is shown in Fig. 8.24-b) and the performance surface for the discrete version in Fig. 8.24-c).

The trainings are summarized in Table 8.45-Table 8.47.

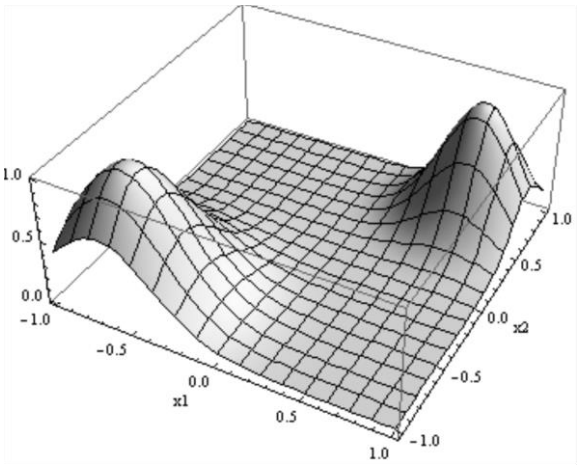


Fig. 8.23. Input-Output 3D plot for $t_2(x)$

TABLE 8.45: TRAININGS WITH LM (ANALYTICAL)

v[1]	[-0.6 0.6]	[-0.4 -0.4]	[0.4 -0.4]	[-0.4 0.4]
N	13	11	10	10
$\hat{\mathbf{v}}_a$	[0.079 0.079]	[0.079 0.079]	[0.079 0.079]	[0.079 0.079]
$\hat{\Psi}_a$	**3.03e-2**	**3.03e-2**	**3.03e-2**	**3.03e-2**
$\Psi_f\big\vert_{\hat{\mathbf{v}}_a}$	2.98e-2	2.98e-2	2.98e-2	2.98e-2
$\Psi_d\big\vert_{\hat{\mathbf{v}}_a}$	9.80e-1	9.80e-1	9.80e-1	9.80e-1

TABLE 8.46: TRAININGS WITH LM (FUNCTIONAL)

v[1]	[-0.6 0.6]	[-0.4 -0.4]	[0.4 -0.4]	[-0.4 0.4]
N	21	10	11	11
$\hat{\mathbf{v}}_f$	[0.069 0.069]	[0.069 0.069]	[0.069 -0.184]	[-0.184 0.069]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_f}$	3.03e-2	3.03e-2	3.35e-2	3.35e-2
$\hat{\Psi}_f$	**2.95e-2**	**2.95e-2**	**3.36e-3**	**3.36e-3**
$\Psi_d\big\vert_{\hat{\mathbf{v}}_f}$	9.84e-1	9.84e-1	1.15	1.15

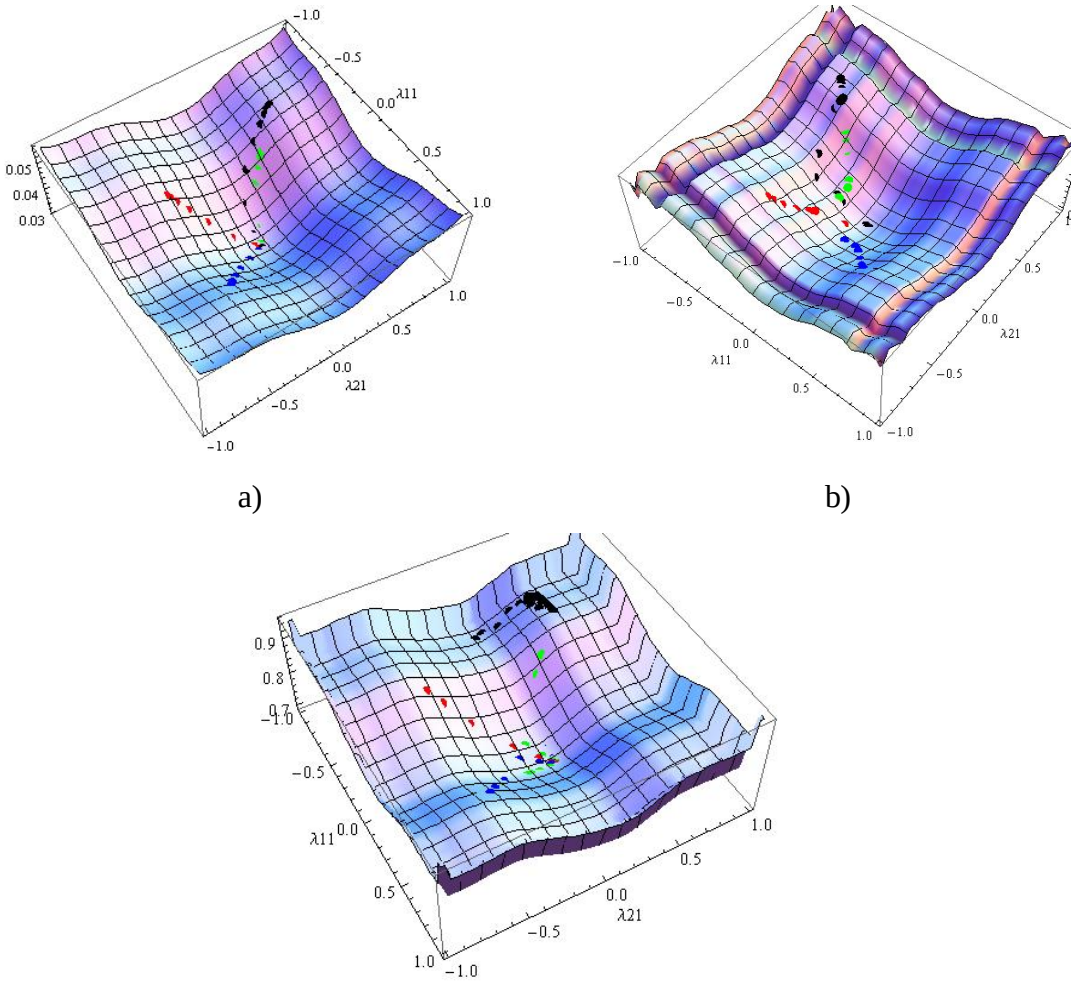


Fig. 8.24 Evolution of 4 different trainings for target function $t_2(x_1,x_2)$: a) with LM analytical version; b) with LM functional version; c) with LM discrete version

The functional version fails to converge to the global optimum in two of the trainings. This is mainly explained by the shape of the graph in this region, which looks similar to a saddle,

leading the estimated parameters into saddle points, i.e. $\left[\lambda_{1,1}, \lambda_{2,1}\right] = \left[-0.184, 0.065\right] \vee \left[\lambda_1, \lambda_2\right] = \left[0.065, -0.184\right]$.

Note that the starting point located furthest from the global optimum, though requiring more iterations, converges smoothly, being able to skip the saddle points in its way towards the global optimum.

On the other hand, the discrete version fails to converge to the minima within the maximum number of iterations. The final points, for all cases, are far from the global analytical optimum.

TABLE 8.47: TRAININGS WITH LM (DISCRETE)

v[1]	[-0.6 0.6]	[-0.4 -0.4]	[0.4 -0.4]	[-0.4 0.4]
N	200	200	200	200
$\hat{\mathbf{v}}_d$	[-0.803 0.183]	[0.183 0.183]	[0.175 0.183]	[0.183 0.175]
$\Psi_a\big\vert_{\hat{\mathbf{v}}_d}$	4.81e-2	3.96e-2	3.96e-2	3.9 e-2
$\Psi_f\big\vert_{\hat{\mathbf{v}}_d}$	4.67e-2	3.83e-2	3.83e-2	3.83e-2
$\hat{\Psi}_d$	**7.78e-1**	**6.90e-1**	**6.90e-1**	**6.90e-1**

8.4.3 Takagi-Sugeno fuzzy systems

As it was shown in previous sections, the functional approach employed to neural models is a viable option. The goal in this section is to show that similar performance can be attained for fuzzy systems, and in particular to Takagi-Sugeno (TS) fuzzy systems.

8.4.3.1 Example

In this simple example, the target function to be approximate is $t(x) = \dfrac{sin(10x)}{x}$, over the domain $x \in \left[-1,1\right]$ with a TS fuzzy system.

The fuzzy system employed is explained next.

8.4.3.1.1 The fuzzy system

The fuzzy rule for the fuzzy system model employed is described by the 2 rules:

$$\begin{aligned} R_1 &: \text{if } (x \text{ is } A_1) \text{ then } f_1 = a_0^{(1)} + a_1^{(1)} x \\ R_2 &: \text{if } (x \text{ is } A_2) \text{ then } f_2 = a_0^{(2)} + a_1^{(2)} x \end{aligned} \tag{8.55}$$

Trapezoidal membership functions will be used. Noting that the j^{th} interval on the i^{th} input is defined as:

$$I_{i,j} = \begin{cases} [\lambda_{i,j-1}\ \lambda_{i,j}[& j = 1,\ldots,p_i - 1 \\ [\lambda_{i,j-1}\ \lambda_{i,j}] & j = p_i \end{cases}, \tag{8.56}$$

the j^{th} membership in the i^{th} dimension is defined as:

$$\mu_{i,j}(x_i) = \begin{cases} 1, & x_i \in I_{i,2j-1} \\ \dfrac{x_i - \lambda_{i,2j-3}}{\lambda_{i,2j-2} - \lambda_{i,2j-3}}, & x_i \in I_{i,2j-2} \\ \dfrac{\lambda_{i,2j} - x_i}{\lambda_{i,2j} - \lambda_{i,2j-1}}, & x_i \in I_{i,2j} \\ 0, & otherwise \end{cases} \tag{8.57}$$

As it can be seen, from the previous equations, the **u** parameters appear linearly in the output, and the $\boldsymbol{\lambda}$ parameters nonlinearly.

Given a crisp input *x*, assuming that for implementing logic connectives such as the conjunction and implication, the *t-norm* used is the algebraic product, the output, *y*, of this model is:

$$y = \frac{\sum_{i=1}^{R} \mu_{A_i}(x) y^{(i)}}{\sum_{i=1}^{R} \mu_{A_i}(x)}, \tag{8.58}$$

where μ_{A_i} is the membership function for the input linguistic term A_i. Assuming that the membership functions form a Ruspini partition, the denominator of (8.58) is unitary. Therefore, (8.58) can be written as $y = \boldsymbol{\varphi}^T \mathbf{u}$, with

$$\boldsymbol{\varphi}^T = \left[\mu_{1,1}(x)\ \ \mu_{2,1}(x)\ \ \mu_{1,1}(x)x\ \ \mu_{2,1}(x)x \right], \tag{8.59}$$

and

$$\mathbf{u}^T = \left[a_0^{(1)}\ \ a_0^{(2)}\ \ a_1^{(1)}\ \ a_1^{(2)} \right]. \tag{8.60}$$

8.4.3.1.2 Terms for the functional approach

In this example, the functional approach employing the backpropagation algorithm will be compared to the discrete approach. In this way, the gradient vector (8.16) will be required.

For this domain [-1,1] and noting (8.59), $\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx$ is given by:

$$\begin{bmatrix} \frac{3+2\lambda_1+\lambda_2}{3} & \frac{\lambda_2-\lambda_1}{6} & \frac{-6+3\lambda_1^2+2\lambda_1\lambda_2+\lambda_2^2}{12} & \frac{\lambda_2^2-\lambda_1^2}{12} \\ & \frac{3-\lambda_1-2\lambda_2}{3} & \frac{\lambda_2^2-\lambda_1^2}{12} & \frac{6-\lambda_1^2-2\lambda_1\lambda_2-3\lambda_2^2}{12} \\ & & \frac{10+4\lambda_1^3+3\lambda_1^2\lambda_2+2\lambda_1\lambda_2^2+\lambda_2^3}{30} & \frac{-3\lambda_1^3-\lambda_1^2\lambda_2+\lambda_1\lambda_2^2+3\lambda_2^3}{60} \\ & & & \frac{10-\lambda_1^3-2\lambda_1^2\lambda_2-3\lambda_1\lambda_2^2-4\lambda_2^3}{30} \end{bmatrix} \tag{8.61}$$

Notice that, as the matrix is symmetric, only the upper-diagonal elements are shown. Having (8.61), the calculation of its inverse is straightforward, hence not shown here.

Another term which can be previously determined is $\frac{\partial \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx}{\partial \mathbf{v}^T}$:

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx}{\partial \lambda_1} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{6} & \frac{3\lambda_1+\lambda_2}{6} & -\frac{\lambda_1}{6} \\ & -\frac{1}{3} & -\frac{\lambda_1}{6} & -\frac{\lambda_1+\lambda_2}{6} \\ & & \frac{6\lambda_1^2-3\lambda_1\lambda_2+\lambda_2^2}{15} & \frac{-9\lambda_1^2-2\lambda_1\lambda_2+\lambda_2^2}{60} \\ & & & \frac{-3\lambda_1^2-4\lambda_1\lambda_2-3\lambda_2^2}{30} \end{bmatrix} \tag{8.62}$$

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx}{\partial \lambda_2} = \begin{bmatrix} \frac{1}{3} & \frac{1}{6} & \frac{\lambda_1+\lambda_2}{6} & \frac{\lambda_2}{6} \\ & -\frac{2}{3} & \frac{\lambda_2}{6} & -\frac{\lambda_1+3\lambda_2}{6} \\ & & \frac{3\lambda_1^2+4\lambda_1\lambda_2+3\lambda_2^2}{30} & \frac{-\lambda_1^2+2\lambda_1\lambda_2+9\lambda_2^2}{60} \\ & & & \frac{-\lambda_1^2-3\lambda_1\lambda_2-6\lambda_2^2}{15} \end{bmatrix} \tag{8.63}$$

As the target function is known, the analytical error backpropagation algorithm is employed here, too. Notice that under those assumptions, the terms involving the function to approximate need to be determined too:

$$\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi} t dx = \begin{bmatrix} \dfrac{\cos(10\lambda_1)-\cos(10\lambda_2)+10(\lambda_1-\lambda_2)\times SI(10)+10\lambda_1 SI(10\lambda_1)-10\lambda_2 SI(10\lambda_2)}{10(\lambda_1-\lambda_2)} \\ \dfrac{-\cos(10\lambda_1)+\cos(10\lambda_2)+10(\lambda_1-\lambda_2)\times SI(10)+10(\lambda_2 SI(10\lambda_2)-\lambda_1 SI(10\lambda_1))}{10(\lambda_1-\lambda_2)} \\ \dfrac{10\cos(10)(\lambda_1-\lambda_2)-\sin(10\lambda_1)+\sin(10\lambda_2)}{100(\lambda_1-\lambda_2)} \\ \dfrac{10\cos(10)(\lambda_2-\lambda_1)+\sin(10\lambda_1)-\sin(10\lambda_2)}{100(\lambda_1-\lambda_2)} \end{bmatrix}$$

$$\int_{x_{min}}^{x_{MAX}} t\frac{\partial \boldsymbol{\varphi}^T}{\partial \lambda_1} dx = \begin{bmatrix} \dfrac{-\cos(10\lambda_1)+\cos(10\lambda_2)-10\lambda_2\left[SI(10\lambda_1)-SI(10\lambda_2)\right]}{10(\lambda_1-\lambda_2)^2} \\ \dfrac{\cos(10\lambda_1)-\cos(10\lambda_2)+10\lambda_2\left[SI(10\lambda_1)-SI(10\lambda_2)\right]}{10(\lambda_1-\lambda_2)^2} \\ \dfrac{-10\cos(10\lambda_1)(\lambda_1-\lambda_2)+\sin(10\lambda_1)-\sin(10\lambda_2)}{100(\lambda_1-\lambda_2)^2} \\ \dfrac{10\cos(10\lambda_1)(\lambda_1-\lambda_2)-\sin(10\lambda_1)+\sin(10\lambda_2)}{100(\lambda_1-\lambda_2)^2} \end{bmatrix}$$

$$\int_{x_{min}}^{x_{MAX}} t\frac{\partial \boldsymbol{\varphi}^T}{\partial \lambda_2} dx = \begin{bmatrix} \dfrac{\cos(10\lambda_1)+\cos(10\lambda_2)+10\lambda_1\left[SI(10\lambda_1)-SI(10\lambda_2)\right]}{10(\lambda_1-\lambda_2)^2} \\ \dfrac{-\cos(10\lambda_1)+\cos(10\lambda_2)-10\lambda_1\left[SI(10\lambda_1)-SI(10\lambda_2)\right]}{10(\lambda_1-\lambda_2)^2} \\ \dfrac{10\cos(10\lambda_2)(\lambda_1-\lambda_2)+\sin(10\lambda_1)-\sin(10\lambda_2)}{100(\lambda_1-\lambda_2)^2} \\ \dfrac{10\cos(10\lambda_2)(\lambda_2-\lambda_1)+\sin(10\lambda_1)-\sin(10\lambda_2)}{100(\lambda_1-\lambda_2)^2} \end{bmatrix}$$

(8.64)

In the expressions above, *SI* refers to the Sine Integral function.

In the following, the BP algorithm was used with a learning rate $\eta = 0.02$, and trainings were set for 5 different starting points. Training stops when a maximum number of 200 iterations is reached, or when the criteria from section 2.3.6.1, with $\tau = 10^{-6}$ are satisfied.

Because the rule base forms a Ruspini Partition the order relation must be preserved (see chapter 2, section 2.3.6). Therefore, when there is a violation of the order relation, the learning rate for the BP algorithm is updated at that iteration.

Fig. 8.25 shows the analytical performance surface, together with the results for the five different trainings.

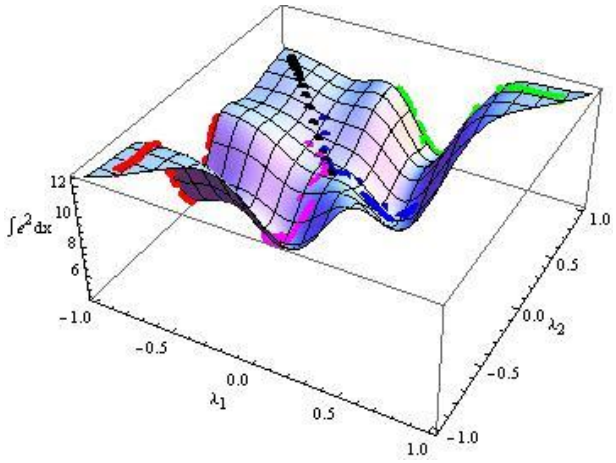


Fig. 8.25 Analytical performance surface

Even for this simple case, several local minima exist. The surface is symmetric, and there are two global minima, located at [-0.357,-0.116] and [0.116, 0.357], with a value of 4.4 for criterion (8.13).

The following table summarizes the results for the different trainings.

TABLE 8.48. ANALYTICAL TRAININGS

$\mathbf{v}_a[1]$	[-0.90,-0.85]	[-0.90,0.90]	[0.85,0.90]	[-0.5,0.6]	[-0.5,0.4]
N	55	26	55	200	200
$\mathbf{v}_a[N]$	[-1.0,-0.087]	[-0.28,0.28]	[0.088,1.0]	[0.182,0.395]	[-0.28,0.06]
$\hat{\Psi}_a[N]$	**5.22**	**7.01**	**5.22**	**4.82**	**6.15**

The first line shows the starting points, and N is the number of iterations taken by the training. Notice that no training reaches the global minima. This is due to the setting of the maximum number of iterations (200). The two last evolutions would attain the global minima with a few more iterations.

Next, assuming that the target function is not known, and only training data is available, the same methodology can be employed. A solution to the integrals in (8.64) requires a numerical integration technique then.

The following experiment relates to the case where Gaussian quadrature is used for approximating the integral.

With this technique, the input samples $x^{(i)}$ are chosen by the method.

The next figure and table illustrate the results obtained with the number of input patterns, *m*=25. The performance surface seen in Fig. 8.26 is very similar to the analytical one. The different evolutions go to the same minima, as seen in Table 8.49.

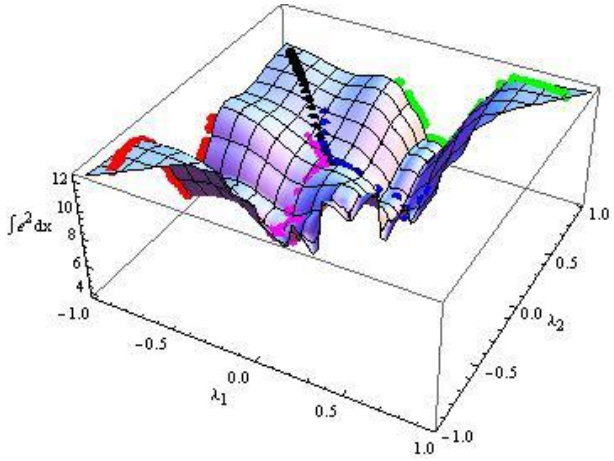


Fig. 8.26. Gaussian quadrature performance surface

TABLE 8.49 FUNCTIONAL APPROACH USING GAUSSIAN QUADRATURE

$\mathbf{v}_G[1]$	[-0.90,-0.85]	[-0.90,0.90]	[0.85,0.90]	[-0.5,0.6]	[-0.5,0.4]
N	102	39	55	29	32
$\hat{\Psi}_G[N]$	**5.02**	**7.01**	**4.99**	**4.27**	**4.15**
$\mathbf{v}_G$[N]	[-1.0,-0.118]	[-0.29,0.29]	[0.121,1.0]	[0.12,0.41]	[-0.31,-0.16]
$\Psi_a\|_{v_G[N]}$	5.34	7.01	5.36	4.49	4.49

As stated before, the goal here is to investigate the quality of the methodology in terms of the analytical criterion. This is seen in the last row of the tables, evaluated for the last parameters' values obtained by the training. In this last table the evaluation is performed with the data, using Gaussian quadrature. Comparing with Table 8.48, similar values were obtained (actually for the two last columns it attains lower values, which are much closer to the global optima).

8.4.3.2 Different integration techniques

As pointed above, although the last training was performed with data, the training inputs were obtained with Gaussian quadrature. As in practice one has no control over the training data values, this subsection investigates the methodology performance in cases where the input data is randomly generated. Moreover, it is desirable to see how different numerical integration techniques affect this performance. In all experiments, 25 input patterns were used. In order to compare the different methods, the extreme points were fixed to -1 and +1. Therefore, only 23 of the 25 values were randomly generated. The discrete version of the BP

algorithm is applied for that same data. The results for the discrete trainings are summarized in Table 8.50 and the corresponding performance surface is depicted in Fig. 8.27. The subscripts indicate the integration techniques used, and the subscript "d" denotes the discrete training algorithm.

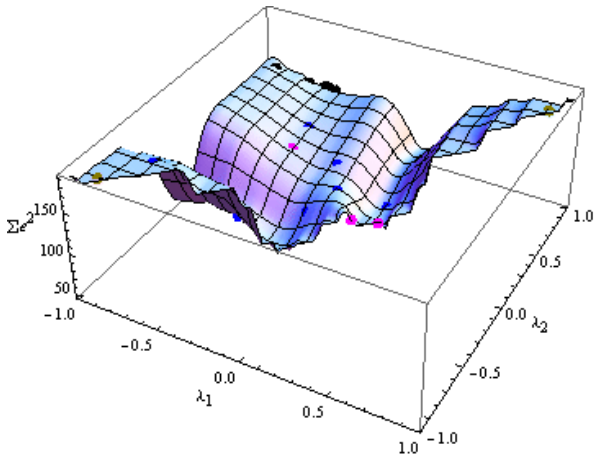


Fig. 8.27. Discrete performance surface

TABLE 8.50. DISCRETE TRAININGS USING RANDOM DATA

$\mathbf{v}_d[1]$	[-0.90,-0.85]	[-0.90,0.90]	[0.85,0.90]	[-0.5,0.6]	[-0.5,0.4]
N	2	200	2	3	4
$\hat{\Psi}_d[N]$	181.31	112.81	172.82	63.43	72.54
$\mathbf{v}_d$[N]	[-0.90,-0.85]	[-0.58,1]	[0.85,0.90]	[0.12,0.39]	[0.17,0.18]
$\Psi_a\big\vert_{\mathbf{v}_d[N]}$	12.21	9.00	12.21	4.65	5.35

Despite using the same input, the discretized criterion (Fig. 8.27) provides a smoother surface than the one provided by the functional technique (Fig. 8.26). Despite this, and using the same input samples and comparing the results (Table 8.48 and Table 8.49) there are 3 (the 3 leftmost columns) starting points out of the 5 starting points, for which very different (worse) results were obtained.

Results for the different numerical trainings were carried out and are summarized in Table 8.51. The bold typeface is used to show the lowest Ψ_a values obtained in each one of the trainings.

The results in Table 8.51 provide distinct interpretations. First, many dissimilar results are obtained with the integration technique employed. For instance, the results show worse performance of the functional approach when using the trapezoidal technique.

TABLE 8.51 FUNCTIONAL APPROACH WITH DIFFERENT INTEGRATION TECHNIQUES

Integration method	**v**[1]	[-0.90,-0.85]	[-0.90,0.90]	[0.85,0.90]	[-0.5,0.6]	[-0.5,0.4]
Forward	N	15	7	42	46	3
	$\hat{\Psi}_f[N]$	4.82	9.54	4.69	7.02	5.21
	$\mathbf{v}_f$ [N]	[-0.43,-0.14]	[-0.85,0.95]	[0.08,0.81]	[-0.99,0.12]	[-0.87,-0.04]
	$\Psi_a\big\vert_{v_f[N]}$	**4.70**	9.75	5.71	7.85	5.67
Backward	N	7	9	59	7	3
	$\hat{\Psi}_b[N]$	11.60	9.16	5.14	4.70	5.09
	$\mathbf{v}_b$ [N]	[-0.88,-0.75]	[-0.80,0.98]	[0.08,0.32]	[-0.99,-0.05]	[-0.87,-0.09]
	$\Psi_a\big\vert_{v_b[N]}$	12.16	9.64	**4.58**	**5.32**	5.51
trapezoidal or tustin	N	6	2	24	22	12
	$\hat{\Psi}_t[N]$	5.59	9.43	4.74	5.11	5.80
	$\mathbf{v}_t$[N]	[-0.98,-0.16]	[-0.95,0.86]	[0.03,0.49]	[-0.99,-0.02]	[-0.99,0.18]
	$\Psi_a\big\vert_{v_t[N]}$	5.81	9.77	5.08	5.60	6.06
3rd order polynomial	N	35	12	45	35	15
	$\hat{\Psi}_p[N]$	6.46	9.77	4.31	3.88	5.66
	$\mathbf{v}_p$[N]	[-0.91,0.03]	[-0.91,0.74]	[0.071,0.36]	[0.08,0.19]	[-0.39,0.03]
	$\Psi_a\big\vert_{v_p[N]}$	6.34	**9.51**	4.65	5.59	**5.32**

The trapezoidal approach is a technique which tries to approximate the integral combining the principles from backward and the forward techniques and so, it is expected to obtain a better approximation to the function integral. Nevertheless, the results show that it is the backward technique that reaches the lowest analytical criterion value in 2 out of the 5 starting points, and the trapezoidal technique does not achieve any best solution. Despite of this fact, the trapezoidal approach seems to obtain the more consistent results across the different starting points.

Secondly, and using the Mathematica 3rd order polynomial a closer approximation to the function underlying the input data is expected. However, the results only confirm this fact in 2 out of the 5 points.

Lastly, if the discrete trainings are compared with the functional approach, then only in 2 out of the 5 starting points is the discrete approach better. Whereas in the 2nd and the 5th

trainings the results are similar, in the 4th training the discrete version is clearly better (around 15% lower value then the best functional approach).

8.4.3.2.1 Random data

The training data in the previous subsection corresponds to a particular set of input patterns. The results in this subsection correspond to 20 experiments where performance of each version of the functional approach will be evaluated in terms of mean (μ) and standard deviation (σ). The same 5 starting points will be used. The training set consists of 25 input patterns where only 23 of the 25 values were randomly generated.

Table 8.52 provides these statistics for the optimal values of the criteria optimized, and Table 8.53 shows the same statistics, but in terms of the analytical criterion, evaluated at the final points for each different optimization.

The discrete criterion obtains the better results only for 1 out of the 5 initial points, and even for this case, with results very similar to the polynomial version of the functional approach. Comparing the different integration schemes, one concludes that, for this particular example, the polynomial integration technique is the best method, as it obtains 2 best results overall, and 3 out of 5, comparing only the functional approaches.

TABLE 8.52. MEAN AND STANDARD DEVIATION OF THE CRITERIA OPTIMIZED

	[-0.90,-0.85]	[-0.90,0.90]	[0.85,0.90]	[-0.5,0.6]	[-0.5,0.4]
$\mu_d \pm \sigma_d$	119±56	71±28	136±37	56±13	54±14
$\mu_f \pm \sigma_f$	5.6±2.6	9.5±1.5	7.8±4.3	5.2±2.1	4.2±1.9
$\mu_b \pm \sigma_b$	9.7±3.0	7.7±3.7	3.5±2.4	4.1±3.1	4.0±2.5
$\mu_t \pm \sigma_t$	10.9±3.2	9.0±1.5	4.4±1.1	5.0±1.1	4.9±1.5
$\mu_p \pm \sigma_p$	10.2±3.5	7.5±2.7	9.1±3.3	3.1±2.3	3.2±2.5

TABLE 8.53 MEAN AND STANDARD DEVIATION OF THE ANALYTICAL CRITERIA, FOR THE OPTIMUM FOUND FOR EACH CRITERION

	[-0.90,-0.85]	[-0.90,0.90]	[0.85,0.90]	[-0.5,0.6]	[-0.5,0.4]
$\mu_a \pm \sigma_a\big\vert_{v_d[n]}$	11.8±1.7	**7.5±2.2**	12.2±0.03	5.7±1.3	5.5±0.84
$\mu_a \pm \sigma_a\big\vert_{v_f[n]}$	**6.0±2.1**	9.3±1.0	8.3±3.3	6.0±1.2	5.4±0.2
$\mu_a \pm \sigma_a\big\vert_{v_b[n]}$	11.6±1.2	9.7±0.1	5.4±0.5	5.9±1.1	5.8±0.6
$\mu_a \pm \sigma_a\big\vert_{v_t[n]}$	11.5±2.1	9.7±0.1	**4.9±0.4**	5.8±0.9	5.4±0.2
$\mu_a \pm \sigma_a\big\vert_{v_p[n]}$	9.4±3.5	7.7±2.4	8.8±3.5	**5.3±1.0**	**5.2±1.6**

8.5 Conclusions

This chapter showed how to apply the new concept of the minimization of the integral of the error in the context of function approximation, for a fixed model structure. The usual criterion used for parameter estimation is the sum-of-squares-of-the-error (SSE). This criterion is totally dependent on the training data and, even using a test set in an early-stopping scheme, it was shown that bad models, over the input domain can be obtained.

In this chapter an alternative was presented and its applicability to several architectures was shown. The objective of this new approach is to minimize a performance surface which is more independent on the training data.

There are some advantages clearly identified with this approach:

i) If the training criterion exploits the separability of the parameters, it reduces the computational complexity in the calculation of the gradient, as some terms are independent of the data (and therefore its size), and avoids the inverse operation (or pseudo-inverse) in the computation of the optimal linear parameters;

ii) As some of the terms involved in the calculation of the gradient are only dependent on the limits of the input domain and are independent of the training data, the gradient uses a performance surface which is closer to the one obtained if the true function was actually used. The degree of similarity depends on the error obtained in approximating the projection of the function on the basis functions and on their partial derivatives, over the domain, with the training data.

iii) If the function generating the data is known, the local minima and the performance of a specified model can be determined. In the case where integrals (8.20), (8.21) and the gradient zeros can be analytically computed, an analytical solution is obtained, as shown in section 8.4.2.1. In other cases, a numerical solution can be found.

This methodology was applied for the different architectures: Radial Basis Function networks, B-Spline networks, and Takagi-Sugeno fuzzy models. One and two-dimensional problems were considered. It was experimentally demonstrated that the use of Gaussian quadrature integration achieves models with a performance very similar to the completely analytical approach, i.e., the case where the function modeling the data is known. For generic training data, further studies on integration methods, especially in the multi-dimensional case, should be conducted. The former chapters showed that the hybrid use of evolutionary algorithms and gradient-based algorithms are useful for model design. The discrete gradient-

based algorithms used in previous chapters, can be replaced by the functional version, which was introduced in this chapter.

Chapter 9

CONCLUSIONS AND FUTURE WORK

Neuro-fuzzy models offer the propriety of transparency, found in FS, while using algorithms, introduced in the context of neural networks, for their design. This thesis has introduced new algorithms for the design of this type of models, concentrating on parameter estimation and model structure selection algorithms.

Due to the functional equivalence between BSNNs and FS (meeting some assumptions), a series of algorithms were introduced in the context of BSNNs but are also applicable to fuzzy models.

It was shown how to incorporate a-priori knowledge in BSNNs design, first for a fixed structure and afterwards integrated with Genetic Programming, which evolves the model structure. It was demonstrated that, in general, the incorporation of a-priori knowledge results in models with better generalization ability.

In a previous work [59], the author proposed a design methodology specific to BSNN models, which was based on genetic programming [14]. Nearly at the same time, in the context of fuzzy systems modeling, an evolutionary alternative, BEA, was introduced by Nawa and Furuhashi [126]. In this work, it was proposed to improve [59] by combining the principles of BEA and GP. Thus, a new technique called Bacterial Programming (BPA) was proposed here, which has been found to avoid local minima. This approach was compared with existing alternatives for B-Spline networks and Mamdani Fuzzy systems, presenting a performance at least as good as GP. Also, tuning of the BPA is much easier than tuning the GP, particularly if one is to use different sets of data.

For parameters estimation, a completely supervised training algorithm was applied to BSNN in a previous work [84]. Results showed that the performance of these gradient-based techniques depends much on the starting points of the searching process. To remedy this problem, the BPA algorithm was used in a hybrid scheme, where it is used to determine the

most suitable starting points to the gradient-based algorithm. As it was seen, a faster convergence to a point near the global optima is often obtained with this technique.

One problem associated with neuro-fuzzy (and fuzzy) models is that, typically, in order to obtain a good performance the model complexity is high. In order to alleviate this problem, a new input decomposition approach has been proposed, which can be used together with functional decomposition techniques, such as ASMOD. Experiments show, that in average a complexity reduction of around 30% can be obtained, when considering a fixed model structure. However, finding the best grid partitioning is not straightforward, with increasing difficulty as the number of input variables grows. For this reason, evolutionary (GA and GEP) techniques, for finding the best grid partitioning were also proposed, and examples showed that less complex models, with similar generalization quality, are achievable with this approach. These models can improve their performance in a hybrid scheme, where the GEP algorithm is combined with the LM algorithm.

Hybrid schemes were also proposed to generic, non-Ruspini Mamdani fuzzy systems. In this sense, the original bacterial evolutionary algorithm was used in conjunction with the LM algorithm resulting in a technique coined Bacterial Memetic Algorithm. BMA was used for fuzzy rule extraction in five benchmark problems, exhibiting consistently better performance, in terms of accuracy and convergence rate than the BEA alone. Furthermore, to improve both algorithms, a reformulation encoding was employed that, besides obtaining very good results, allows the use of the same bacterial operators with minor changes.

The parameters separability concept [82] can be applied for most of the models used in this work. Using this principle, and reformulating the training criterion (from a discrete to a continuous version), a new training methodology, coined functional approach, has been introduced. This methodology offers important advantages, over the conventional (discrete) concept:

- If the function generating the data is known, it offers a complete analytical solution for the design problem;
- As some of the terms involved in the calculation of the criterion and its derivatives are only dependent on the model used and on the limits of the input domain, and are independent of the training data, the performance surface used by the training algorithm is closer to the one obtained if the true function was actually used. This means that the resulting model has more capability of generalization, within that domain that the one designed with the same algorithm, using only the training data.

The degree of similarity between the analytical and the functional solution depends on the error obtained in approximating the projection of the function on the basis functions and on their partial derivatives, over the domain, with the training data;

- In the training algorithms, the terms that depend only on the model and on the domain can be computed in advance, and applied whenever needed in the algorithm execution. This, together with the avoidance of pseudo-inverse computation of the basis function matrix, translates into important computational savings, especially when model complexity increases.

It was shown how to apply this technique to RBFs, BSNNs and TS FS, both using the BP and the LM algorithms. It can be extended to all models where the parameters separability concept can be applied, provided that an analytical solution can be found for the needed integrals. Obviously, this is applicable to all gradient-based algorithms introduced in this Thesis for models with that characteristic, where the discrete gradient-based algorithms can be replaced, with advantage, by their functional counterpart.

9.1.1 Future work

There are some topics in this thesis which have not been fully investigated. In the following, some research proposals are listed with the purpose of improving the proposed approaches.

Integration of the different proposals for neuro-fuzzy systems

In the context of neuro-fuzzy systems, the proposals which can be found in Chapters 3 to 7 can be integrated. As examples:

- Instead of using RBGEN for incorporating a-priori knowledge in BSNNs, BPA could be used, with possible advantages. Alternatively, the models evolved by BPA could have their linear weights restricted as result of the application of the function and derivative restrictions.
- BMA could be employed in a outer loop, with an inner loop where near optimal initial values for the gradient algorithms could be supplied by BPLM, introduced in Chapter 5;
- BPA and GP can also be incorporated into the structure optimization process for input domain decomposition, described in Chapter 6;
- In all cases where (discrete) gradient-based algorithms are employed, they can be replaced by their functional alternative, provided issues described below are solved.

Extension of the functional approach

As this was the last topic addressed in this thesis, only one and two-dimensional problems could be considered. In order for this approach to be used for generic problems, it must be extended to more input dimensions. The major problem lies in the use of appropriate techniques for multi-dimensional numerical integration. The majority of the existing techniques assume the availability of the function to integrate, in order to evaluate the function (and possible derivatives) at specified points. This, however, does not happen in model training where what is available is a set of data.

In this sense, it would be of interest to apply an integration technique based on the same principles found in the trapezoidal rule for the one-dimensional problems. In the case of two-dimensional problems one way to calculate the integral of a function from the set of input samples available, is the use Delaunay triangulation [199][200]. This way, a numerical integration can be computed from the volume of the surface below the subset of disjoint triangles given by Delaunay triangulation. This is conceptually simple and should be easily employed. If results justify, an extension to higher dimensional problems should be straightforward.

One other aspect that should be explored is the computation of the integrals over the convex hull of the data, instead of over hyper-rectangles (as was used in this work). In most cases, and definitely when a static model is used, with external feedback, to approximate a dynamical system, the input data only covers a small subset of the hyper-rectangle defined by the input data range.

Use of a multi-objective approach

All the evolutionary algorithms used in this thesis were formulated as a single-objective minimization (typically the BIC criterion was used to balance model accuracy, complexity and training size). Different (possibly better) balances between these and other criteria can be obtained using a multi-objective approach.

BIBLIOGRAPHY

[1] A. Konar, *Artificial Intelligence and Soft Computing, Behavioral and Cognitive Modelling of the Human Brain*, CRC Press, 2000.

[2] C. M. Bishop, *Neural Networks for pattern recognition*, Clarendon Press, Oxford, 1995.

[3] D.O. Hebb, *The Organization of Behavior*, Wiley: New York, 1949.

[4] F. Rosenblatt. *The Perceptron: a perceiving and recognizing automaton*. Report 85-460-1, Cornell Aeronautical Laboratory Spartan Books, 1957.

[5] B. Widrow, and F. Smith. Pattern recognizing control systems. *Computer and Information Sciences*, Symposium Proceedings, Spartan Books, 1963.

[6] B. Widrow. An adaptive "Adaline", neuron using chemical "memistors". *Stanford Electronics Labs.*, Stanford, USA, Tech. Report 1553-2, 1960.

[7] M. Minsky, S. Papert, *Perceptrons*. MIT Press, 1969.

[8] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning internal representations by error propagation, In: *Parallel distributed processing: explorations in the microstructure of cognition*, Vol. 1, MIT Press, pp. 318-362, 1986.

[9] D.E. Rumelhart, J. McLelland, and the PDP Research Group, *Parallel distributed processing*, Vol. 1, MIT Press, 1986.

[10] S. Karner, The Laws of Thought, *Encyclopedia of Philosophy*, Vol 4, MacMillan, pp. 414-417, 1967.

[11] C. Lejewski, and J. Lucaziewicz, *Encyclopedia of Philosophy*, Vol 5, MacMillan, pp. 104-107, 1967.

[12] L. A. Zadeh, Fuzzy Sets, *Information and Control*, 8:338-353, 1965.

[13] J. H. Holland , *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, Cambridge, 1992.

[14] J. R. Koza, *Genetic Programming, On the programming of computers by means of natural selection*, 6th ed., MIT Press, 1998.

[15] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*, Wiley, 1996.

[16] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Evolution*, Frammann-Hozboog Verlag, Stuttgart, 1973.

[17] I. Renners, *Genetic Topology Optimization of B-spline Networks*, Department of Mathematics, The University of Paderborn, Paderborn, Germany, 2000.

[18] Y-G. Leu, Nonlinear System Modelling Using GA-based B-spline Membership Fuzzy-Neural Networks, *2nd International Conference on Autonomous Robots and Agents*, December 13-15, 2004 Palmerston North, New Zealand, 2004.

[19] O. Nelles, M. Fisher, and B. Müller, fuzzy rule extraction by a genetic algorithm and constrained nonlinear optimization of membership functions, In *International IEEE Conference on Fuzzy Systems* (FUZZ-IEEE), pp. 213-219, New Orleans, USA, September 1996.

[20] L. S. Coelho, and F. A. Guerra, B-spline neural network design using improved differential evolution for identification of an experimental nonlinear process, *Applied Soft Computing* 8 (2008) 1513–1522.

[21] S-K Oh, W. Pedrycz, Genetically Optimized Self-organizing Neural networks based on Polynomial and Fuzzy Polynomial Neurons:Analysis and Design, *Engineering Evolutionary Intelligent Systems Studies in Computational Intelligence*, Volume 82, pp 59-108, 2008.

[22] A. E. Ruano, P. M. Ferreira and, C. M. Fonseca, An overview of Nonlinear identification and control with Neural Networks, In *intelligent Control Systems using Computational Intelligence Techniques*, A. E. Ruano, Editor, 2005, Institution of Electrical Engineers, pp. 37-78, 2005.

[23] R. Babuška, and H. Verbruggen, Neuro-fuzzy methods for nonlinear system identification, *Annual Reviews in Control*, 27, 73–85, 2003.

[24] K. Funuhashi, On the approximate realization of continuous mappings by neural networks, *Neural Networks*, Vol. 2, Nº. 3, pp. 183-192, 1989.

[25] K. Ornik, M. Stinchcombe, and H. White, Multilayer perceptrons are universal approximators, *Neural Networks*, Vol. 2, Nº. 5, pp. 359-366, 1989.

[26] DS. Broomhead and D. Lowe, Multivariable Function interpolation and adaptive networks, *Complex systems*, vol 2, pp.321-355, 1988.

[27] S. Chen, C. F. N. Cowan, and P. M. Grant, Orthogonal least squares learning algorithm for radial-basis-function networks, *IEEE Trans. on neural networks*, vol. 3,pp. 302-309, 1991.

[28] C. Drioli, and D. Rocchesso, Orthogonal least squares algorithm for the aproximation of a map and its derivatives with a RBF network, *Elsevier signal processing*, pp.283-296, 2003.

[29] J.A.S. Freeman, and D. Saad., Learning and generalization in radial basis function networks, *Neural Computation*, vol. 7, pp.1601-1622, 1997.

[30] A. Webb, and S. Shannon, Shape-adaptive radial basis functions, *IEEE Trans. Neural Networks*, vol. 9, November 1998.

[31] T. Kohonen, *Self-organizing Maps*, Second Edition, Springer, 1997.

[32] O. Nelles, and R. Isermann., A new technique for determination of hidden layer parameters in RBF networks, In *IFAC World Congress*, pp. 453-457, San Francisco, USA June 1996.

[33] S. Sarimveis, A. Alexandridis, and G. Bafas, A fast training algorithm for RBF networks based on subtractive clustering, *Neurocomputing* 51, 501–505, 2003.

[34] M. J. L. Orr, Regularization in the selection of radial basis function centers, *Neural computation*, 7 (3), pp. 606-623, 1995.

[35] J. Platt, A resource allocating network for function interpolation, *Neural computation*, 3:213-225, 1991.

[36] V. Dadirkamanathan, and M. Niranjan, A function estimation approach to sequential learning with neural networks, *Neural computation*, 5, pp. 954-75, 1993.

[37] L. Yingwei, N. Sundararajan, and O. Saratchandran, A sequential learning scheme for function approximation using minimal radial basis function neural networks, *Neural computation*, 9, pp. 461-478, 1997.

[38] L. Yingwei, N. Sundararajan, and O. Saratchandran, Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm, *IEEE Transactions on Neural Networks*, 8 (2), pp. 308-318, 1998.

[39] H.S. Park, W. Pedrycz, S.K. Oh, Granular neural networks and their development through context-based clustering and adjustable dimensionality of receptive fields, *IEEE Trans. Neural Networks* 20 (10), pp. 1604–1616, 2009.

[40] S.B. Roh, T.C. Ahn, and W. Pedrycz, The design methodology of radial basis function neural networks based on fuzzy K-nearest neighbors approach, *Fuzzy Sets and Systems*, 161 (13), 1803–1822, 2010.

[41] W. Pedrycz, H.S. Park, S.K. Oh, A granular-oriented development of functional radial basis function neural networks, *Neurocomputing*, 72, pp. 420–435, 2008.

[42] A.D. Niros, G.E. Tsekouras, On training radial basis function neural networks using optimal fuzzy clustering, *17th IEEE Mediterranean Conference on Control and Automation*, pp. 395–400, 2009.

[43] C.-B. Cheng, E.S. Lee, Fuzzy regression with radial basis function network, *Fuzzy Sets Syst.* 119 (2), pp. 291–301, 2001.

[44] M.J. Er, W. Chen, S. Wu, High-speed face recognition based on discrete cosine transform and RBF neural networks, *IEEE Trans. Neural Networks* 16 (3), pp. 679–691, 2005.

[45] Y. Li, S. Qiang, X. Zhuang, O. Kaynak, Robust and adaptive back stepping control for nonlinear systems using RBF neural networks, *IEEE Trans. Neural Networks* 15 (5), pp. 693–701, 2004.

[46] P. M. Ferreira, E. A. Faria, and A. E. B. Ruano, Comparison of on-line learning algorithms for radial basis function models in greenhouse environmental control, In *Proceedings of fourth Portuguese conference on Automatic control*, Guimarães, Portugal, 2000.

[47] P. M. Ferreira, and A. E. Ruano, Choice of RBF model structure for predicting greenhouse inside air temperature, *preprints from IFAC 15th world Congress*, 2002.

[48] V.M. Rivas, J.J. Merelo, P.A. Castillo, M.G. Arenas, J.G. Castellano, Evolving RBF neural networks for time-series forecasting with EvRBF, *Information Sciences*, 165 (3–4) pp. 207–220, 2004.

[49] A.E. Ruano, and A.B. Azevedo, B-Splines neural networks assisted PID autotuning, *International Journal of Adaptive Control & Signal Processing* 13 (4) (1999) 291–307,1999.

[50] H. Jin, C.W. Chan, H.Y. Zhang, and W.K. Yeung, Fault detection of redundant systems based on B-spline neural network, In *Proceedings of American Control Conference*, vol. 2, Seattle, USA, pp. 1215–1219, 2000.

[51] H. Deng, D. Srinivasan, R. Oruganti, A B-spline network based neural controller for power electronic applications, *Neurocomputing*, 73, 593–601, 2010.

[52] C. De Boor, A Practical Guide to Splines, *Applied Mathematical Sciences* 27, Revised Edition, 2001.

[53] G. Lightbody, P. O'Reilly, G.W. Irwin, K. Kelly and J. McCormick, Neural modeling of chemical plant using mlp and b-spline networks, *Control Eng. Practice*, Vol. 5, No. 11, pp. 1501 - 1515, 1997.

[54] C. H. Wang, W. Y. Wang, T. T. Lee, and P.S. Tseng, Fuzzy B-spline membership function (BMF) and its applications in fuzzy-neural control, *IEEE Transactions on Systems, Man, and Cybernetics* 25 (5) (1995) 841–851, 1995.

[55] M. Figueiredo, J. Leitão, and A. K. Jain, Unsupervised contour representation and estimation using B-splines and a minimum description, *IEEE Transactions on Image Processing*, vol. 9, no. 6, pp.1075-1087, 2000.

[56] R.E. Bellman, *Adaptive control Processes*, Princeton University Press, New Jersey, 1961.

[57] M. Brown, and C. Harris, *Neurofuzzy Adaptive Modelling and Control, Systems and control engineering*, Prentice Hall, 1994.

[58] E. Weyer, T. Kavli, *The ASMOD Algorithm. Some New Theoretical and Experimental Results*, SINTEF Report STF31 A95024, Oslo, 1995.

[59] C. L. Cabrita, A. E. Ruano, and C.M. Fonseca, Single and Multi-Objective Genetic Programming Design for B-Spline Neural Networks and Neuro-Fuzzy Systems,

IFAC Workshop on Advanced Fuzzy/Neural Control 2001, (AFNC'01), Valencia, Spain, October, 15-16, pp. 93-98, 2001.

[60] C. J. Harris, C. G. Moore, and M. Brown, *Intelligent Control: Some Aspects of Fuzzy Logic and Neural Networks*, World Scientific Press, London & Singapore, 1993.

[61] J. Abonyi, *Fuzzy model identification for control*, Birkhauser, 2003.

[62] O. Nelles. *Nonlinear System Identification*, Springer Verlag, 1st edition, 2002.

[63] L. A. Zadeh , Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Tr. Systems, Man and Cybernetics **3***, pp. 28-44, 1973.

[64] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Systems Man Cybemet. 15* (1985) 116-132, 1985.

[65] M. Sugeno, and M. Nishida, fuzzy control of model car, *Fuzzy sets and systems* 16, pp.103-113, 1985.

[66] T.J. Procyk, E.H. Mamdani, A linguistic self-organizing process controller, *Automatica*, 15, 15-30, 1979.

[67] W.Z. Qiao, W.P. Zhuang, T.H. Heng, and S.S. Shan, A rule self-regulating fuzzy controller, *Fuzzy Sets and Systems* 47, 13-21, 1992.

[68] M. Sugeno, and T. Yasukawa, Fuzzy-logic-based approach to qualitative modeling, *IEEE Trans. On Fuzzy Sets*, Vol 1, 7-31, February 1993.

[69] E. D. Lughofer, FLEXFIS: A Robust Incremental Learning Approach for Evolving Takagi-Sugeno Fuzzy Models, *IEEE Trans. Fuzzy Systems*, Vol 16 (6), December 2008.

[70] J. Botzheim, B. Hámori, L. T. Kóczy, and A. E. Ruano, Bacterial algorithm applied for fuzzy rule extraction, *IPMU 2002*, Annecy, France pp. 1021-1026, 2002.

[71] L.X. Wang, and J.M. Mendel, Back-propagation fuzzy system as nonlinear dynamic system identifiers, *Proceedings IEEE Int. Conf. on Fuzzy Systems*, San Diego, pp. 1409-1416, 1992.

[72] F. J. de Souza, M. M.R. Vellasco, and M. A.C. Pacheco, Hierarchical neuro-fuzzy quadtree models, *Fuzzy Sets and Systems* 130, 189–205, 2002.

[73] Y. Shi, M. Mizumoto, N. Yubazaki, and M. Otani, A learning algorithm for tuning fuzzy rules based on the gradient descent method, *Proc. 5th IEEE Int. Conf. on Fuzzy Systems*, New Orleans, vol. 1, pp. 55-61, 1996.

[74] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD. Dissertation, Appl. Math. Harvard University, USA, 1974.

[75] Z. Zainuddin, N. Mahat, and Y. Abu Hassan, *Improving the Convergence of the Backpropagation Algorithm Using Local Adaptive Techniques*, World Academy of Science, Engineering and Technology 1, 2005.

[76] B. Riedmiller, A direct adaptive method for faster backpropagation learning: The RPROP algorithm, 1993.

[77] S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, *1988 Connectionist Models Summer School*, San Mateo, CA, Morgan Kaufmann, 1988.

[78] Gill, P. E., Murray, W. and Wright, M. H. *Practical Optimization*. Academic Press, Inc., 1981.

[79] D. Marquardt, An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *SIAM J. Appl. Math.*, 11, pp. 431-441, 1963.

[80] K. Levenberg, A Method for the Solution of Certain Problems in Least Squares, *Quarterly Applied Mathematics*, (2), 164–168, 1944.

[81] G. H. Golub, and V. Pereyra, Differentiation of Pseudo-Inverses and Nonlinear Least-Squares Problems Whose Variables Separate, *Siam Journal on Numerical Analysis*, Vol 10, pp. 413-432, 1973.

[82] G. H. Golub, and V. Pereyra, Separable Nonlinear Least Squares: The Variable Projection Method and Its Applications, *Inverse Problems*, (19), R1-R26, 2003.

[83] A. E. B. Ruano, D. I. Jones, and P. J. Fleming, A New Formulation of the Learning-Problem for a Neural Network Controller. In (ed.), *Proceedings of the 30th Ieee Conference on Decision and Control*, Vols 1-3, 865-866, 1991.

[84] A. E. Ruano, C. L. Cabrita, , J. V. Oliveira, and L. T. Koczy, Supervised Training Algorithms for B-Spline Neural Networks and Neuro-Fuzzy Systems. *International Journal of Systems Science*, (33), 689-711, 2002.

[85] P. M. Ferreira, A. E. B. Ruano, Exploiting the Separability of Linear and Nonlinear Parameters in Radial Basis Function Networks. In *Proceedings IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 321-326, 2000.

[86] J. Sjoberg, and M. Viberg, Separable Non-Linear Least-Squares Minimization - Possible Improvements for Neural Net Fitting. *Neural Networks for Signal Processing* VII, 345-354, 1997.

[87] F.T. Krogh, Efficient implementation of a variable projection algorithm for nonlinear least squares problems, *Comm.* ACM 17:167-169, 1974.

[88] L. Kaufman, Variable projection method for solving separable nonlinear least squares problems, *BIT*, 15, pp. 49-57, 1975.

[89] A. E. B. Ruano, P. J. Fleming, and D. I. Jones, A Connectionist Approach to PID Autotuning. In *International Conference on Control 91*, Vols 1 and 2, 762-767, 1991.

[90] R. R. Picard., and R. D. Cook, Cross-Validation of Regression Models, *Journal of the American Statistical Association*, Vol. 79, No. 387, pp. 575-583, 1984.

[91] T. Sonderstrom, and P. Stoica, On covariance function tests used in system identification, *Automatica*, vol. 26, pp.125-133, 1990.

[92] Quan Min Zhu, Li Feng Zhang, Ashley Longden, Development of omni-directional correlation functions for nonlinear model validation, *Automatica, Volume 43*, Issue 9, pp. 1519-1531, 2007.

[93] S. A. Billings, and W. S F. Voon, Correlation based model validity tests for nonlinear models, *Int. J. Control*, vol. 44, pp.235-244, 1986.

[94] S. A. Billings, and W. S F. Voon, Structure detection and model validity tests in the identification of nonlinear system, *Proc. Inst. Electron. Eng.*, Vol 130, pt. D, pp. 193-199, 1983.

[95] R. Haber, Nonlinearity tests for dynamic processes, *IFAC Symposium on Identification and System Paramater Estimation*, pp. 409-414, York, UK, 1885.

[96] L. F. Zhang, Q. M. Zhu, and A. Longden, Correlation-test-based validation procedure for identified neural networks, *IEEE Trans. On Neural Networks*, vol 20 (1), Jan, 2009.

[97] M. Y. Mashor, Model Validity Tests for RBF Network, *International Journal of the Computer, the internet and management*, vol.7, nº2, 1999.

[98] J. S. R. Jang, and C. T. Sun, Functional Equivalence between Radial Basis Function Networks and Fuzzy Inference Systems, *IEEE Trans. on Neural Networks*, vol. 4, no. 1, pp. 156-159, Jan. 1993.

[99] J. H. Holland, and J. S. Reitman, Cognitive systems based on adaptive algorithms, In D. A. Waterman and F. Hayes-Roth, editors, *Pattern directed inference systems*, pages 313–329, Academic Press, New York, NY, 1978.

[100] J. Kennedy, and R. C. Eberhart, *Swarm Intelligence*, Morgan Kaufmann, 2001.

[101] M. Dorigo, and L.M. Gambardella, *Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem*, IEEE Transactions on Evolutionary Computation, Vol 1, Nº. 1, pp. 53-66, 1997.

[102] M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano, Italy, 1992.

[103] J. O. Kephart, A biologically inspired immune system for computers. *Proceedings of Artificial Life IV: The Fourth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press. pp. 130–139, 1994.

[104] L. N. de Castro, and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer. pp. 57–58, 2002.

[105] E. Ülker, and A. Arslan, Automatic knot adjustment using an artificial immune system for B-spline curve approximation, *Information Sciences* 179 (2009), pp. 1483–1494, 2009.

[106] L. S. Coelho, M. W. Pessoa, Nonlinear identification using a B-spline neural network and chaotic immune approaches, *Mechanical Systems and Signal Processing* 23 (2009), pp. 2418–2434, 2009.

[107] T. Bäck, U. Hammel, H-P. Schwefel, Evolutionary Computation: comments on the History and Current State, *IEEE Trans. Evolutionary Computation*, vol 1(1), April 1997.

[108] L. J. Fogel. *On the organization of intellect*. PhD thesis, UCLA University of California, Los Angeles, California, USA, 1964.

[109] R. Storn, and K. Price, *Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces*, Technical report TR-95–012, ICSI, March, 1995.

[110] J. Liu, and J. Lampinen, A Fuzzy Adaptive Differential Evolution Algorithm, *Soft Computing* 9: 448–462, 2005.

[111] J. N. Thompson, *The coevolutionary process*, University of Chicago Press, 1994.

[112] T. Bäck, Optimal mutation rates in genetic search, in *Proc. 5th Int. conf. On Genetic Algorithms*, S. Forrest, Ed. San Mateo, CA: Morgan Kaufmann, pp.2-8, 1993.

[113] D. E. Goldberg, *Genetic Algorithms in search, optimization and machine learning*, Addison-Wesley, Reading, MA, 1989.

[114] Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Programs*, Springer, Berlin, 1992.

[115] O. Cordón, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena, Ten years of genetic fuzzy systems: current framework and new trends, *Fuzzy Sets and Systems 141* (1), 5–31, 2004.

[116] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena, *Genetic fuzzy systems, Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*, World Scientific, 2001.

[117] S-K. Oh, and W. Pedrycz, The design of self-organizing Polynomial Neural Networks, *Information Science*, Vol. 141, pp. 237-258, 2002.

[118] K. Rodriguez-Vázquez, C. M. Fonseca, and P. J. Fleming, Identifying the structure of nonlinear dynamic systems using multiobjective genetic programming, *IEEE Transactions on Systems Man and Cybernetics – Part A: Systems and Humans*, 34 (4), pp. 531-545, July 2004.

[119] A. E. Ruano, P. J. Fleming, C. Teixeira, K. Rodriguez-Vazquez, and C. M. Fonseca, Nonlinear identification of aircraft gas-turbine dynamics, *Neurocomputing*, 55, pp. 551-579, 2003.

[120] C. Ferreira, Function Finding and the Creation of Numerical Constants in Gene Expression Programming, *Advances in soft computing*, pp. 257-265, 2003.

[121] C. Ferreira, Genetic Representation and Genetic Neutrality in Gene Expression Programming, *Advances in Complex Systems*, Vol. 5, No. 4, pp.389-408, 2002.

[122] C. Ferreira, Gene Expression Programming: A New Adaptive Algorithm for Solving Problems, *Complex systems*, Vol. 13 (2), pp. 87-129, 2001.

[123] X. Li, C. Zhou, W. Xiuao, and P. R C. Nelson, Prefix Gene Expression Programming, *Late breaking paper at Genetic and Evolutionary Computation Conference* {(GECCO'2005), 2005.

[124] C. Ferreira, Designing Neural Networks Using Gene Expression Programming, *9th Online World Conference on Soft Computing in Industrial Applications*, September 20-October 8, 2004.

[125] N. E. Nawa, and T. Furuhashi, A Study on Nonlinear Model Identification Using Pseudo-Bacterial Genetic Algorithm, *Proc. of the 1997 Int'l Conf. on Neural Information Processing and Intelligent Information System* (ICONIP/ANNES'97), pp. 408-411, 1997.

[126] N. E. Nawa, and T. Furuhashi, Fuzzy System Parameters Discovery by Bacterial Evolutionary Algorithm, *IEEE Tr. Fuzzy Systems* **7**, pp. 608-616, 1999.

[127] K. Yamamoto, T. Yoshikawa, T. Furuhashi, T. Shinogi, and S. Tsuruoka, Evaluation of Search Performance of Bacterial Evolutionary Algorithm, CD-ROM *Proc. of 11th IEEE Int'l Conf. on Fuzzy Systems* (FUZZ-IEEE2002), 2002.

[128] M. Miwa, T. Furuhashi, M. Matsuzaki, and S. Okuma, Cerebellar Model Arithmetic Computer with Pseudo-Bacterial Genetic Algorithm and Its Hardware Acceleration, *Trans. of IEICE*, 185-D-II, 8, pp.1332-1340, 2002.

[129] S. Das, A. Chowdhury, and A. Abraham, A Bacterial Evolutionary Algorithm for automatic data clustering, In *preprints of IEEE congress on Evolutionary Computation*, CEC '09, 18-21 May, 2009.

[130] C. L. Cabrita, J. Botzheim, A. E. Ruano, and L. T. Kóczy, Genetic programming and bacterial algorithm for neural networks and fuzzy systems design: a comparison, *IFAC international conference on intelligent control systems and signal processing* (ICONS 2003), Faro, Portugal, 8-11 Abril, pp. 500-505, 2003.

[131] J. Botzheim, L. T. Kóczy, and A. E. Ruano, Extension of the Levenberg-Marquardt algorithm for the extraction of trapezoidal and general piecewise linear fuzzy rules, *WCCI 2002*, Honolulu, Hawaii pp. 815-821, 2002.

[132] A. P. Engelbrecht, *Computational Intelligence: An introduction*, John Wiley & Sons, 2005.

[133] A. Wetzel, *Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization*, PhD thesis, University of Pittsburgh, Pittsburgh, PA, Unpublished manuscript, technical report, 1983.

[134] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Proceedings of Foundations of Genetic Algorithms*, pp. 69–93, 1990.

[135] J. Mingjun, T. Huanwen, Application of chaos in simulated annealing, *Chaos, Solitons, and Fractals* 21 (4), 933–941, 2004.

[136] K.F. C. Yiu, S. Wang, K. L. Teo, and A. C. Tsoi, Nonlinear System Modeling via knot optimizing B-Spline networks, *IEEE Trans. NEURAL NETWORKS*, vol. 12, no. 5, pp.1013-1022, 2001.

[137] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997.

[138] S-M. Chou, T.-S. Lee, Y. E. Shao, and I-F. Chen, Mining the breast cancer pattern using artificial neural networks and multivariate adaptive regression splines, *Expert Systems with Applications*, Volume 27, Nº 1, pages 133–142, July 2004.

[139] J. S. Roger Jang, ANFIS: adaptive-network-based fuzzy inference systems, *IEEE Trans. Systems Man Cybernetics*. 23, 665-685, 1993.

[140] J.-S. Roger Jang, and E. Mizutani, Levenberg-Marquardt method for ANFIS learning, *Proc. Biennial Conf. of the North American Fuzzy Information Processing Society NAFIPS'96*, Berkeley, CA, IEEE, New York, pp. 87-91, 1996.

[141] Rasit Ata, An adaptive neuro-fuzzy inference system approach for prediction of power factor in wind turbines, *Journal of Electrical and Electronics Engineering*, Vol9, 1, pp. 905-912, 2009.

[142] E.H. Mamdani, and S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, *Int. J. Man Machine Studies 7*, 1-13, 1975.

[143] D. Nauck, and R. Kruse, Neuro-fuzzy systems for function approximation, *Fuzzy Sets and Systems 101*, pp.261-271, 1999.

[144] B. D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge, 1996.

[145] H. R. Maier, T. Sayed, B. J. Lence, Forecasting cyanobacterium Anabaena spp. in the River Murray, South Australia, using B-spline neurofuzzy models, *Ecological Modelling* 146, 85–96, 2001.

[146] L.J. Herrera, H. Pomares, I. Rojas, O. Valenzuela, and M. Awad, MultiGrid-based fuzzy systems for function approximation, in: Proceedings of the Mexican International Conference on Artificial Intelligence 2004, *Lecture Notes in Computer Science*, vol. 2972, pp. 252–261, 2004.

[147] A. G. Ivakhnenko, Polynomial theory of complex systems, *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-1, pp. 364-378, 1971.

[148] T. Kavli, ASMOD an algorithm for adaptive spline modelling of observation data, *International Journal of Control Volume* 58, Nº 4, 1993.

[149] N.K. Treadgold and T.D. Gedeon, Exploring constructive cascade networks, *IEEE Transactions on Neural Networks*, vol. 10, issue 6, pp. 1335-1350, 1999.

[150] K. Balázs, J. Botzheim, and L T. Kóczy, Hierarchical Fuzzy System Modeling by Genetic and Bacterial, Programming Approaches, *WCCI 2010 IEEE World Congress on Computational Intelligence*, July, 18-23, 2010 - CCIB, pp. 1866-1871, Barcelona, Spain, 2010.

[151] G. V. S. Raju, and J. Zhou, Adaptive hierarchical fuzzy controller, *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4):973-980, 1993.

[152] M. Sugeno, M. F. Griffin, and A. Bastian: Fuzzy hierarchical control of an unmanned helicopter, *Proceedings of the 5th IFSA World Congress* (IFSA93), Seoul, South Korea, pp. 1262-1265, 1993.

[153] L-X. Wang, Analysis and Design of Hierarchical Fuzzy Systems, *IEEE Transactions on fuzzy systems*, Vol. 7, Nº. 5, October 1997.

[154] Y. Chen, Q. Meng, and Y. Zhang, Optimal Design of Hierarchical B-Spline Networks for Nonlinear System Identification, in *Proceedings of the International Conference on Sensing, Computing and Automation*, pp. 3263-3268, 2006.

[155] Y. Chen, M. Liu, and B. Yang, Breast Cancer Detection Using Hierarchical B-Spline Networks, *ICNC 2006*, Part I, LNCS 4221, pp. 25–28, 2006.

[156] S. Rafal, and S. Jurgen, Probabilistic Incremental Program Evolution, *Evolutionary Computation*, vol.5, pp. 123-141, 1997.

[157] M. Rawski, H. Selvaraj, and P. Morawiecki, Efficient method of input variable partitioning in functional decomposition based on evolutionary algorithms, *Euromicro Symposium on Digital System Design*, DSD 2004, 136-143, 2004.

[158] R.A. Finkel, J.L. Bentley, Quad trees, a data structure for retrieval on composite keys, *Acta Inform*, 4, 1–9, 1974.

[159] M. Sun, and R. E. Steuer, InterQuad: An interactive quad tree based procedure for solving the discrete alternative multiple criteria problem, *European Journal of Operational Research*, 89,, 462-472, 1996.

[160] C. A. Duncan, M. T. Goodrich, and S. Kobourov, Balanced Aspect Ratio Trees: Combining the Advantages of k-d Trees and Octrees, *Journal of Algorithms* 38, 303-333, 2001.

[161] S. Arya, D. M. Mount, Approximate range searching, *Computational Geometry*, 17, 135–152, 2000.

[162] H. J. Haverkort, M. Berg B., and J. Gudmundsson, Box-trees for collision checking in industrial installations, *Computational Geometry* 28,113–135, 2004.

[163] G. Vigueras, M. Lozano, J.M. Orduña, F. Grimaldo, A Comparative Study of Partitioning Methods for Crowd Simulations, *Applied Soft Computing Journal* (2008), doi:10.1016/j.asoc.2009.07.004, 2008.

[164] J.H. Friedman, Multivariate adaptive regression splines, *The annals of Statistics*, 19(1):1-141, March 1991.

[165] A. Lemos, W. Caminhas, and F. Gomide, Evolving Fuzzy Linear Regression Trees, *WCCI 2010 IEEE World Congress on Computational Intelligence*, July, 18-23, 2010 - CCIB, Barcelona, Spain, 2010.

[166] O. Nelles, and R. Isermann., Basis function networks for interpolation of local linear models, In *IEEE Conference on Decision and Control* (CDC), pp. 470-475, Japan, 1996.

[167] S. Ernst, Hinging hyperplane trees for approximation and identification, *IEEE conference on Decision and Control* (CDC), pp. 1261-1277, Tampa, USA, 1998.

[168] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*, Wadsworth, Belmont, 1984.

[169] R. A. Jacobs, M. I. Jordan, Learning Piecewise Control Strategies in a Modular Neural Network, *IEEE Tran. On Systems Man and Cybernetics.*, Vol. 23,(2), pp. 337-345, 1993.

[170] G. Dalquist, and Å. Björck, *Numerical Methods in Scientific Computing*, Society for Industrial and Applied Mathematics, Volume 1, 2008.

[171] F. Heiss, and V. Winschel, Estimation with Numerical Integration on Sparse Grids. *Discussion Papers in Economics*, 2006-15, 2006.

[172] M. J. Miranda, and P. L. Fackler, *Applied Computational Economics and Finance*. MIT Press, Cambridge MA, 2002.

[173] Wolfram Mathematica®, Tutorial Collection: ADVANCED NUMERICAL INTEGRATION IN MATHEMATICA, 2008.

[174] J.M.G. Lima, and A.E. Ruano, Neuro-genetic PID autotuning: time invariant case, *Mathematics and Computers in Simulation*, **51**(3-4): p. 287-300, 2000.

[175] C. L. Cabrita, and A. E. Ruano, B-spline and neuro-fuzzy models design with function and derivative equalities, In *Proceedings World Automation Congress* (WAC2004), June 28-July 1, Sevilha, Spain, 2004.

[176] W. Neuhaus, Optimal Estimation Under Linear Constraints, *Astin Bulletin*, Vol 26, no.2, pp. 233-245, 1996.

[177] N. Noman, and H. Iba, Enhancing Differential Evolution Performance with Local Search for High Dimensional Function Optimization, *GECCO'05*, June 25–29, Washington, DC, USA, 2005.

[178] J. Botzheim, C. L. Cabrita, L. T. Koczy, and A. E. Ruano, Genetic and Bacterial Programming for B-Spline Neural Networks Design, *Journal of Advanced Computational Intelligence & Intelligent Informatics*, Vol 11, nº2, 2007, pp. 220-231, 2007.

[179] C. L. Cabrita, J. Botzheim, A. E. Ruano, and L. T. Kóczy, Design of B-spline Neural Networks using a Bacterial Programming Approach, *International Joint Conference on Neural Networks (IJCNN 2004) and IEEE International Conference on Fuzzy Systems (Fuzz-IEEE 2004)*, Budapest, Hungary, 25July-29July, 2004.

[180] K. G. Bhattacharyya, R. A. Johnson, *Statistical Concepts and Methods*, John Wiley & Sons, 1977.

[181] Y-T. Kao, E. Zahara, A hybrid genetic algorithm and particle swarm optimization for multimodal functions, *Applied Soft Computing* 8 (2008), 849–857, 2008.

[182] C. L. Cabrita, J. Botzheim, A. E. Ruano, and L. T. Koczy, A Hybrid Method for B-Spline Neural Networks Training, *International Symposium on Intelligent Signal Processing*, Faro, Portugal, 2005.

[183] C. L. Cabrita, A. E. B. Ruano, and L. T. Kóczy, A new domain decomposition for B-spline Neural Networks, *WCCI 2010 IEEE World Congress on Computational Intelligence*, July 18-23, Barcelona, pp. 308-315, 2010.

[184] A. Saranliy, and B. Baykal, Chaotic time-series prediction and the relocating LMS (RLMS) algorithm for radial basis function networks, In *Proc. European Signal Processing Conference* (EUSIPCO)2, 1996.

[185] A. Gholipour, B. N. Araabi, and C. Lucas, Predicting Chaotic Time Series Using Neural and Neurofuzzy Models: A Comparative Study, *Neural Processing Letters* (2006) 24:217–239, Springer, 2006.

[186] O. Valenzuela, I. Rojas, F. Rojas,, H. Pomares, L.J. Herrera, A. Guillen, L. Marqueza, and M. Pasadasa, Hybridization of intelligent techniques and ARIMA models for time series prediction, *Fuzzy Sets and Systems* 159, 821 – 845, 2008.

[187] A. E. B. Ruano, First Report Controlo de Fluxo – Bytronics, *Univ. Algarve*, 2000.

[188] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. In *Technical Report 826, Caltech Concurrent Computation Program*. California Institute of Technology, Pasadena, California, 1989.

[189] Y-S. Ong, and A. J. Keane, Meta-Lamarckian Learning in Memetic Algorithms, *IEEE Trans. on Evolutionary Computation*, Vol 8. No. 2. 2004, pp. 99-110, 2004.

[190] J. Botzheim, C. L. Cabrita, A. E. Ruano, and L. T. Kóczy, Estimating Fuzzy Membership Functions Parameters by the Levenberg-Marquardt Algorithm, *International Joint Conference on Neural Networks (IJCNN 2004) and IEEE International Conference on Fuzzy Systems (Fuzz-IEEE 2004)*, pp 1667–1672, Budapest, Hungary, 25July-29July, 2004.

[191] C. L. Cabrita, J. Botzheim, T. Gedeon, A. E. Ruano, L. T. Kóczy, and C. Fonseca, Bacterial memetic algorithm for fuzzy rule base optimization, 2006 *World Automation Congress (WAC)- ISSCI Symposium*, Budapest, Hungary, 24-26July, 2006.

[192] J. Botzheim, C. L. Cabrita, L. T. Kóczy, and A. E. B. Ruano, Fuzzy rule extraction by bacterial memetic algorithms, *International Journal of Intelligent Systems*, Vol.24, pp.312-339, 2009.

[193] J. Botzheim, C. L. Cabrita, L. T. Kóczy, and A. E. Ruano, Fuzzy rule extraction by bacterial memetic algorithms, In *Proceedings of the 11th World Congress of International Fuzzy Systems Association*, IFSA 2005, pp. 1563–1568, Beijing, China, July, 2005.

[194] A. E. Ruano, C. L. Cabrita, and P. M. Ferreira, Towards a More Analytical Training of Neural Networks and Neuro-Fuzzy Systems, *IEEE International Symposium on Intelligent Signal Processing 2011* (WISP11), Floriana, Malta, 19-21 September 2011.

[195] A. E. Ruano, C. L. Cabrita, and P. M. Ferreira, Exploiting the functional training approach in B-Spline, *1st IFAC Conference on Embedded Systems, Computational Inteligence and Telematics in Control*, Wurzburg, Germany, 3-5 April 2012.

[196] C. L. Cabrita, A. E. Ruano, and P. M. Ferreira, Exploiting the functional training approach in Radial Basis Function Networks, *IEEE International Symposium on Intelligent Signal Processing 2011*, WISP11, 19-21 September 2011.

[197] C. L. Cabrita, A. E. Ruano, P. M. Ferreira, and Lásló T. Kóczy, Extending the functional training approach for B-Splines, *IEEE World Congress on Computational Intelligence*, Brisbane, Australia,10-15 June, pp. 2702-2709, 2012.

[198] C. L Cabrita, A. E. Ruano, P. M. Ferreira, and L. T. Kóczy, Exploiting the Functional Training Approach in Takagi-Sugeno Neuro-fuzzy Systems. In *Soft Computing Applications*, ed. Valentina Emilia BalasJános FodorAnnamária R. Várkonyi-KóczyJoszef DombiLakhmi C. Jain, 543 - 559. ISBN: 978-3-642-33940-0. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[199] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars. *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 2008.

[200] D. T. Lee, and B. J. Schachter, Two Algorithms for Constructing a Delaunay Triangulation, *International Journal of Computer and Information Sciences*, Vol. 9, No. 3, 1980.

[201] M. Mackey and L. Glass, *Oscillation and chaos in physiological control systems*, Science (197), 287, 1977.

[202] R.H. Abiyev, O. Kaynak, T. Alshanableh, and F. Mamedov, A Type-2 Neuro-fuzzy System Based on Clustering and Gradient Techniques Applied to System Identification and Channel Equalization, *Applied Soft Computing Journal*, doi:10.1016/j.asoc.2010.04.011, 2008.

Appendices

Appendix A

BENCHMARKS

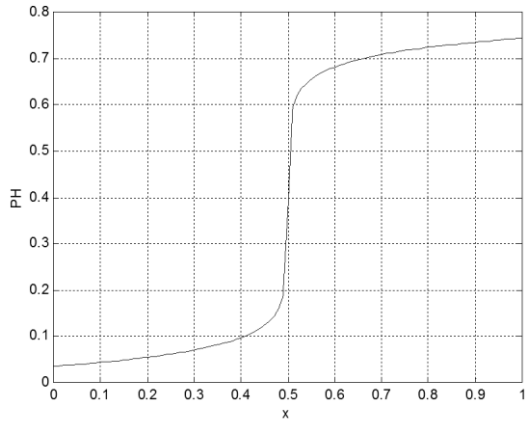
This Appendix describes the problems and the methodologies employed for obtaining the data used in the examples. 7 different problems were considered:

A.1 pH problem

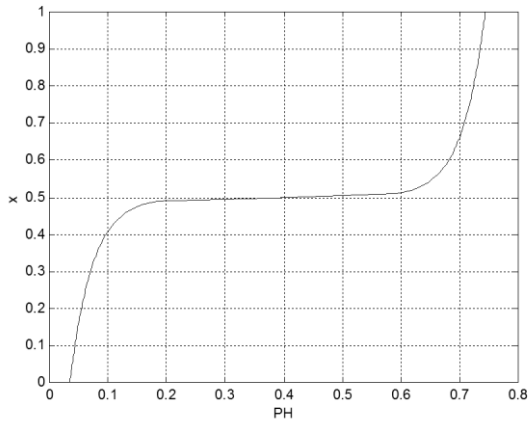
The aim of this example is to approximate the inverse of a titration-like curve. This type of non-linearity relates the *pH* (a measure of the activity of the hydrogen ions in a solution) with the concentration (*x*) of chemical substances.

The data patterns are obtained from the following equation..

$$PH = -\frac{\log\left(\sqrt{\frac{y^2}{4}+10^{-14}}-\frac{y}{2}\right)+6}{26}, y = 2\times10^{-13}x - 10^{-3} \quad \text{(A.1)}$$



a) direct pH nonlinearity



a) inverse pH nonlinearity

Fig.A.1. pH titration like curves.

A.2 Inverse Coordinate Transformation problem (ICT)

This example illustrates an inverse kinematic transformation between 2 Cartesian coordinates and one of the angles of a two-links manipulator.

Regarding the following figure, the angle of the second joint is connected with the Cartesian coordinates on the end of the arm, by the expression:

$$\theta_2 = \pm tg^{-1}\left(\frac{s}{c}\right)$$

$$c = \frac{x^2 + y^2 + {l_1}^2 + {l_2}^2 = \cos(\theta_2)}{2l_1 l_2} \tag{A.2}$$

$$s = \sqrt{1-c^2} = sen(\theta_2)$$

where l_1 e l_2 refer to the length of the corresponding arm segments.

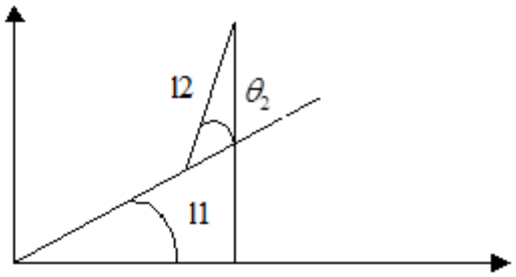


Fig.A.2. A two-link Robot arm

The purpose of this example is to proceed with the mapping $(x, y) \rightarrow \theta_2$ in the first quadrant assuming a length of 0.5m for each arm segment. The data input for training consist of 110 pairs of patterns generated from the intersection of 10 arcs of a circle centered at the origin and 11 radial lines. For each one of the 110 input patterns, the respective solution was determined and used as training patterns.

A.3 A six dimensional generic function

This example is widely used as a target function and the output is given by the following expression:

$$y = x_1 + {x_2}^{0.5} + x_3 x_4 + 2e^{2(x_5 - x_6)} \tag{A.3}$$

where

$x_1 \in [1,5]$, $x_2 \in [1,5]$, $x_3 \in [0,4]$, $x_4 \in [0,0.6]$, $x_5 \in [0,1]$, $x_6 \in [0,1.2]$.

Both training and test data sets consist of 200 input samples, drawn from a uniform random distribution. Because of the large number of samples, the corresponding tables are not shown here.

A.4 Agricultural data

This data set that describes relationships between the properties of soil and the yield of maize. In this work, six attributes of the soil are considered. The data set is presented below. Constructing a model of the data will provide a more accurate way for planning the productivity by being able to distribute fertilizers and other chemicals according to the real needs.

TABLE A.1: DATA SET FOR THE AGRICULTURAL PROBLEM

x_1	x_2	x_3	x_4	x_5	x_6	y
2.5	7.29	46	216	75	8.78	6.48
3.3	7.24	48	320	85	0.03	7.14
2.65	7.42	45	189	70	1.68	8.46
3	7.23	50	207	88	1.96	7.5
3.25	7.15	48	177	97	5.73	7.56
3.25	7.17	46	179	95	0.21	4.92
3.25	7.24	49	123	88	6.84	6.78
2.5	7.33	44	204	104	2.98	4.38
2.5	7.32	45	305	114	4.49	5.1
3.3	7.21	50	192	104	7.69	6.36
3.1	7.2	46	174	117	4.29	6.3
3.25	7.25	49	168	88	7.42	6.06
3.25	7.3	48	207	92	0.64	8.28
2.75	7.28	43	207	79	2.1	7.38
2.85	7.26	45	303	79	0.84	6.48
3.3	7.18	47	319	101	6.25	5.7
3	7.23	42	322	88	1.71	7.38
2.65	7.26	41	204	70	4.49	6.84
2.5	7.31	38	290	73	2.57	6.18
3	7.28	44	290	85	9	7.5
3.1	7.27	46	194	82	0.46	7.92
3.45	7.2	44	193	95	0.35	6.78
3	7.23	42	202	92	2.72	9.78
2.15	7.28	42	388	120	2.75	6.78
2.85	7.25	42	213	107	7.68	5.7
2.7	7.26	44	178	82	6.56	5.34
2.75	7.29	44	177	79	2.21	9
2.5	7.26	40	194	104	7.72	7.5
2.15	7.31	40	193	70	4.44	5.7
2.7	7.28	39	213	88	2.76	7.8
3.2	7.23	42	321	85	0.56	5.1
3.3	7.28	42	293	82	2.01	5.52
2.95	7.26	40	332	82	2.83	7.08

TABLE A.1: DATA SET FOR THE AGRICULTURAL PROBLEM (CONT)

x_1	x_2	x_3	x_4	x_5	x_6	y
2.6	7.32	40	313	88	3.47	7.32
2.95	7.35	38	293	85	7.79	6.96
3.05	7.4	42	208	82	1.33	6.72
2.95	7.36	38	218	92	4.95	7.86
2.6	7.4	38	314	97	2.48	7.44
2.95	7.42	41	352	122	2.39	7.26
2.85	7.37	38	313	117	0.8	7.26
2.7	7.42	39	302	101	5.13	6.24
2.45	7.36	38	322	85	4.34	7.8
2.8	7.38	40	287	101	7.72	7.8
2.9	7.33	42	304	88	2.43	6.54
3.1	7.26	41	319	88	0.89	8.16
2.8	7.35	40	292	85	4.38	7.56
2.8	7.38	40	333	75	1.15	6.9
2.4	7.4	38	291	85	1.66	6.54
2.65	7.34	40	375	117	3.69	6.3
2.6	7.36	42	364	162	8.82	8.76
2.65	7.38	39	358	97	0.32	6
2.35	7.4	39	313	73	3.6	7.5
2.5	7.4	40	317	79	3.34	7.32
2.6	7.36	42	340	70	1.13	7.08
2.7	7.37	42	236	65	4.26	7.74
3.05	7.25	43	233	85	1.16	7.8
3.25	7.19	47	333	92	0.86	6.78
3.25	7.26	47	344	97	0.69	7.74
3.1	7.31	42	332	75	3.5	8.28
3.1	7.31	45	327	92	2.97	7.5
2.5	7.34	42	329	85	5.91	8.04
3.1	7.32	38	357	85	5.61	6.9
2.8	7.36	41	377	97	4.22	6.42

A.5 Human operation at a chemical plant

This is a 5-D problem that realizes an operator's control of a chemical plant.

There are five inputs which refer to:

- u1: monomer concentration
- u2: change of monomer concentration
- u3: monomer flow rate

- u4 and u5: local temperature inside the plant.

  And an output y which is the setpoint for monomer flow rate.

The problem is described in detail in [68].

TABLE A.2: DATA SET FOR THE CHEMICAL PLANT PROBLEM

u1	u2	u3	u4	u5	y
6.8000	-0.0500	401	-0.2000	-0.1000	500
6.5900	-0.2100	464	-0.1000	0.1000	700
6.5900	0	703	-0.1000	0.1000	900
6.5000	-0.0900	797	0.1000	0.1000	700
6.4800	-0.0200	717	-0.1000	0.1000	700
6.5400	0.0600	706	-0.2000	0.1000	800
6.4500	-0.0900	784	0	0.1000	800
6.4500	0	794	-0.2000	0.1000	800
6.2000	-0.2500	792	0	0	1000
6.0200	-0.1800	1211	0	0.1000	1400
5.8000	-0.2200	1557	-0.2000	0	1600
5.5100	-0.2900	1782	-0.1000	0	1900
5.4300	-0.0800	2206	-0.1000	0.1000	2300
5.4400	0.0100	2404	-0.1000	-0.1000	2500
5.5100	0.0700	2685	0.1000	0	2800
5.6200	0.1100	3562	-0.4000	0.1000	3700
5.7700	0.1500	3629	-0.1000	0	3800
5.9400	0.1700	3701	-0.2000	0.1000	3800
5.9700	0.0300	3775	-0.1000	0	3800
6.0200	0.0500	3829	-0.1000	-0.1000	3900
5.9900	-0.0300	3896	0.2000	-0.1000	3900
5.8200	-0.1700	3920	0.2000	-0.1000	3900
5.7900	-0.0300	3895	0.2000	-0.1000	3900
5.6500	-0.1400	3887	-0.1000	0	3900
5.4800	-0.1700	3930	0.2000	0	4000
5.2400	-0.2400	4048	0.1000	0	4400
5.0400	-0.2000	4448	0	0	4700
4.8100	-0.2300	4462	0	0.1000	4900
4.6200	-0.1900	5078	-0.3000	0.3000	5200
4.6100	-0.0100	5284	-0.1000	0.2000	5400
4.5400	-0.0700	5225	-0.3000	0.1000	5600
4.7100	0.1700	5391	-0.1000	0	6000
4.7200	0.0100	5668	0	-0.1000	6000
4.5800	-0.1400	5844	-0.2000	0.1000	6100
4.5500	-0.0300	6068	-0.2000	0	6400
4.5900	0.0400	6250	-0.2000	-0.1000	6400
4.6500	0.0600	6358	-0.1000	-0.1000	6400

TABLE A.2: FOR THE CHEMICAL PLANT PROBLEM (CONT)

4.7000	**0.0500**	**6368**	**-0.1000**	**0**	**6400**
4.8100	0.1100	6379	-0.3000	0	6400
4.8400	0.0300	6412	-0.1000	-0.1000	6400
4.8300	-0.0100	6416	0.1000	-0.1000	6500
4.7600	-0.0700	6514	0	0	6600
4.7700	0.0100	6587	-0.1000	0.1000	6600
4.7700	0	6569	0	-0.1000	6600
4.7700	0	6559	0	0	6700
4.7300	-0.0400	6672	0	0	6700
4.7300	0	6844	-0.1000	0	6800
4.7400	0.0100	6775	-0.2000	0	6800
4.7700	0.0300	6779	0	-0.1000	6800
4.7100	-0.0600	6783	0	0	6800
4.6600	-0.0500	6816	0	0	6800
4.7000	0.0400	6812	0	0	6800
4.6300	-0.0700	6849	0	0	6800
4.6100	-0.0200	6803	0	0	6800
4.5700	-0.0400	6832	0	0.1000	6800
4.5600	-0.0100	6832	-0.1000	0.1000	6900
4.5400	-0.0200	6862	-0.1000	-0.1000	7000
4.5100	-0.0300	6958	0.1000	-0.1000	7000
4.4700	-0.0400	6998	0	0.1000	7000
4.4700	0	6986	-0.1000	0.1000	7000
4.4800	0.0100	6975	0	0	7000
4.4800	0	6973	0	0	7000
4.5000	0.0200	7006	0	0.1000	7000
4.5000	0	7027	0	0	7000
4.4800	-0.0200	7032	0	0	7000
4.5400	0.0600	6995	0	0	7000
4.5700	0.0300	6986	0.1000	-0.1000	7000
4.5600	-0.0100	7009	-0.1000	0.1000	7000
4.5600	0	7022	0	0	7000
4.5700	0.0100	6998	-0.1000	0	7000

A.6 The Mackey–Glass chaotic time series

The Mackey-Glass dynamic system has been introduced in [201] as a model of white blood cell production and is described by the following differential equation.

$$\dot{x}(t) = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \tag{A.4}$$

The time series values were obtained applying the fourth-order Runge–Kutta method to find the numerical solution for the above equation. Integration was carried on with fixed time steps of 0.1. The three parameters of the equation determine the series behavior. When the parameters *a* and *b* are taken as a=0.2 and b=0.1, the value τ =17 and the initial value is 1.2, this leads to a chaotic time series concentrated around a strange attractor of fractal dimension.

From the generated time series, 1000 input output pairs with the following format are extracted.

$$\left[x(t-18), x(t-12), x(t-6), x(t), x(t+6)\right] \tag{A.5}$$

where t=118…1117. The first 500 pairs are used as training data and the remaining are used as test data.

The next figure illustrates the time series sequence.

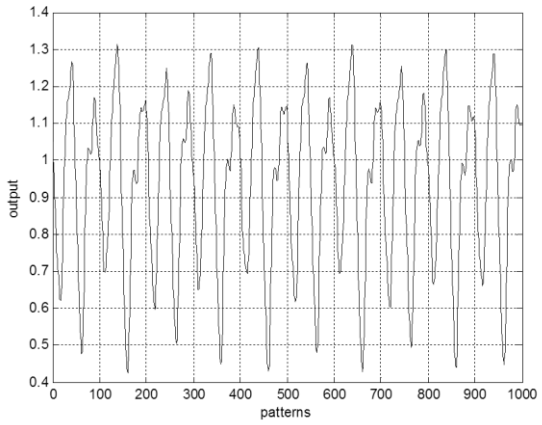


Fig.A.3. Mackey-Glass time series for 1000 time instants

A.7 Identification of a nonlinear process

This problem involves determining the relation between input and output of a process described. Information on this problem was retrieved from [202].

The output of this process is given by the difference equations:

$$y(k) = u(k)^3 + \frac{y(k-1)}{1+y(k-1)^2} \tag{A.6}$$

where y(k) and u(k) are the current output and input of the process, and y(k-1) is the delayed output.

The training data patterns were obtained as follows.

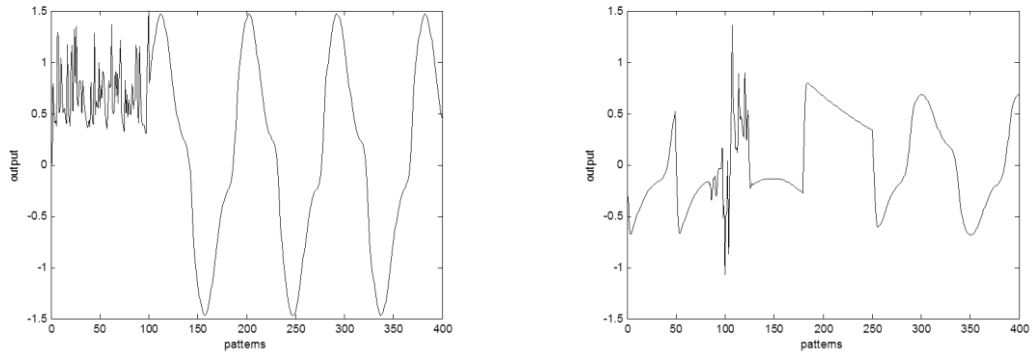
The training data consist of 400 input samples. Using an *independent and identically-distributed* uniform sequence over the range [-1,1] for about a ¼ of the period (100 samples) and the remaining time using the sinusoid function $\sin\left(\frac{\pi k}{45}\right)$.

The test data set is given from the following equation:

$$u(k) = \begin{cases} -0.7 + \dfrac{\text{mod}(k,50)}{40}, & k \le 80 \\ \text{rands}(1,1), & 80 < k \le 130 \\ 0.7 - \dfrac{\text{mod}(k,180)}{180}, & 130 < k \le 250 \\ 0.6\cos(\pi k / 50), & 250 < k \end{cases} \tag{A.7}$$

From which, 400 input samples were extracted.

The following figures illustrate the target for the training and test data.



a) b)

Fig.A.4. Desired output for the a) training data and b) test data sets.

Appendix B

RELATIONS BETWEEN LINEAR WEIGHTS: QUADRATIC AND CUBIC SPLINES

This Appendix extends the relations between the linear weights, using the input domain decomposition proposed in chapter 6, for the case of quadratic and cubic splines. The notation used in section 6.2.2 is employed.

B.1 Quadratic splines

With quadratic splines, C^2 continuity must be ensured. Consider again a univariate model which can be seen as the sum of two sub-models, in the same input variable, x:

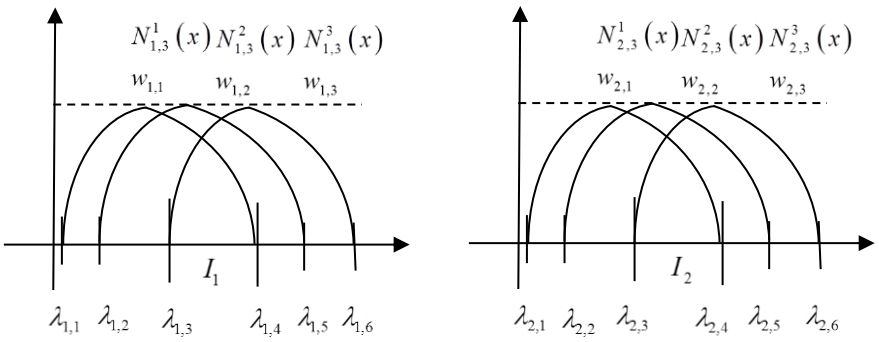


Fig. B.1. Quadratic splines representation for a univariate model composed of two univariate submodels, in the same input variable ($I_1 \cap I_2 = \phi$).

B.1.1 Domain contiguity

Just like triangular splines, one starts by setting a requirement on the interior knots from the two submodels, $\lambda_{1,4} = \lambda_{2,3}$, providing an intersection between input domains I_1 and I_2.

Under these circumstances the output of the sum of these two sub-models is given by:

$$y = y_1 + y_2 = \begin{bmatrix} N_{1,3}^1 & N_{1,3}^2 & N_{1,3}^3 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{1,3} \end{bmatrix} + \begin{bmatrix} N_{2,3}^1 & N_{2,3}^2 & N_{2,3}^3 \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \end{bmatrix} \tag{B.1}$$

At $\lambda_{1,4}$: $N_{1,3}^3(\lambda_{1,4}) = 1 - N_{1,3}^2(\lambda_{1,4})$ and $N_{2,3}^2(\lambda_{1,4}) = 1 - N_{2,3}^1(\lambda_{1,4})$.

It is also known that,

$$\begin{aligned} N_{1,3}^2(\lambda_{1,4}) &= \frac{\lambda_{1,4} - \lambda_{1,5}}{\lambda_{1,3} - \lambda_{1,5}} \\ N_{2,3}^1(\lambda_{1,4}) &= \frac{\lambda_{2,4} - \lambda_{1,4}}{\lambda_{2,4} - \lambda_{2,2}} \end{aligned} \tag{B.2}$$

Continuity on the first derivative for interior knot $\lambda_{1,4}$ is also required. The derivatives of the submodels are shown in the next figure.

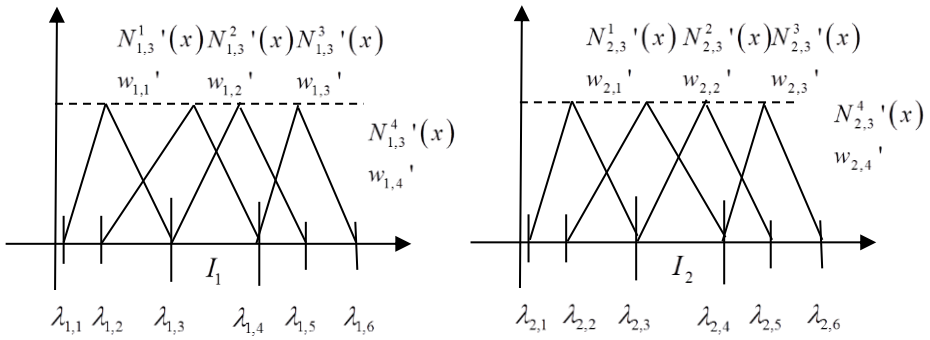


Fig. B.2. 1st derivative of the submodels shown in Fig. B.1.

Note that the triangular splines in Fig. B.2 are derived from differentiating a quadratic B-spline along the input domain using equation (2.23) whereas the weights will be given from (2.25).

In the same manner as before, to ensure continuity at $\lambda_{1,4}$, one must have:

$$w_{1,3}{}' = w_{2,2}{}' \tag{B.3}$$

Now, since:

$$\begin{aligned} w_{1,3}{}' &= 2\frac{w_{1,2} - w_{1,3}}{\lambda_{1,3} - \lambda_{1,5}} \\ w_{2,2}{}' &= 2\frac{w_{2,1} - w_{2,2}}{\lambda_{2,4} - \lambda_{2,2}} \end{aligned} \tag{B.4}$$

Using equations (B.2) to (B.4):

$$\begin{aligned} & w_{1,2}N_{1,3}^{2}(\lambda_{1,4})+w_{1,3}\left(1-N_{1,3}^{2}(\lambda_{1,4})\right)= \\ & =w_{2,1}N_{2,3}^{1}(\lambda_{1,4})+w_{2,2}\left(1-N_{2,3}^{1}(\lambda_{1,4})\right)\Leftrightarrow N_{1,3}^{2}(\lambda_{1,4})\left(w_{1,2}-w_{1,3}\right)+w_{1,3}= \\ & =N_{2,3}^{1}(\lambda_{1,4})\left(w_{2,1}-w_{2,2}\right)+w_{2,2}\Leftrightarrow \\ & \frac{\lambda_{1,5}-\lambda_{1,4}}{\lambda_{1,5}-\lambda_{1,3}}\left(w_{1,2}-w_{1,3}\right)+w_{1,3}=\frac{\lambda_{2,4}-\lambda_{1,4}}{\lambda_{2,4}-\lambda_{2,2}}\left(w_{2,1}-w_{2,2}\right)+w_{2,2} \end{aligned} \tag{B.5}$$

Since,

$$\frac{w_{1,2}-w_{1,3}}{\lambda_{1,3}-\lambda_{1,5}}=\frac{w_{2,1}-w_{2,2}}{\lambda_{2,2}-\lambda_{2,4}}\Rightarrow w_{2,1}=\frac{\lambda_{2,2}-\lambda_{2,4}}{\lambda_{1,3}-\lambda_{1,5}}\left(w_{1,2}-w_{1,3}\right)+w_{2,2} \tag{B.6}$$

Replacing (B.6) in (B.5):

$$\begin{aligned} w_{2,2} &= \frac{w_{1,2}\left(\lambda_{1,5}-\lambda_{2,4}\right)+w_{1,3}\left(\lambda_{2,4}-\lambda_{1,3}\right)}{\lambda_{1,5}-\lambda_{1,3}} \\ w_{2,1} &= \frac{w_{1,2}\left(\lambda_{1,5}-\lambda_{2,2}\right)+w_{1,3}\left(\lambda_{2,2}-\lambda_{1,3}\right)}{\lambda_{1,5}-\lambda_{1,3}} \end{aligned} \tag{B.7}$$

And so, the output is given by:

$$\begin{aligned} y=y_1+y_2= & \begin{bmatrix} N_{1,3}^{1} & N_{1,3}^{2}+N_{2,3}^{1}\frac{\lambda_{1,5}-\lambda_{2,2}}{\lambda_{1,5}-\lambda_{1,3}} & +N_{2,3}^{2}\frac{\lambda_{1,5}-\lambda_{2,4}}{\lambda_{1,5}-\lambda_{1,3}} \end{bmatrix}\begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix}+ \\ & \begin{bmatrix} N_{1,3}^{3}+N_{2,3}^{1}\frac{\lambda_{2,2}-\lambda_{1,3}}{\lambda_{1,5}-\lambda_{1,3}} & +N_{2,3}^{2}\frac{\lambda_{2,4}-\lambda_{1,3}}{\lambda_{1,5}-\lambda_{1,3}} & N_{2,3}^{3} \end{bmatrix}\begin{bmatrix} w_{1,3} \\ w_{2,3} \end{bmatrix} \end{aligned} \tag{B.8}$$

A general expression for the weights, assuming r_1 interior knots in submodel 1 is:

$$\begin{aligned} w_{2,2} &= \frac{w_{1,2+r_1}\left(\lambda_{1,5+r_1}-\lambda_{2,4}\right)+w_{1,3+r_1}\left(\lambda_{2,4}-\lambda_{1,3+r_1}\right)}{\lambda_{1,5+r_1}-\lambda_{1,3+r_1}} \\ w_{2,1} &= \frac{w_{1,2+r_1}\left(\lambda_{1,5+r_1}-\lambda_{2,2}\right)+w_{1,3+r_1}\left(\lambda_{2,2}-\lambda_{1,3+r_1}\right)}{\lambda_{1,5+r_1}-\lambda_{1,3+r_1}} \end{aligned} \tag{B.9}$$

B.1.2 Input merging

Assume two sub-models as illustrated in the next figure:

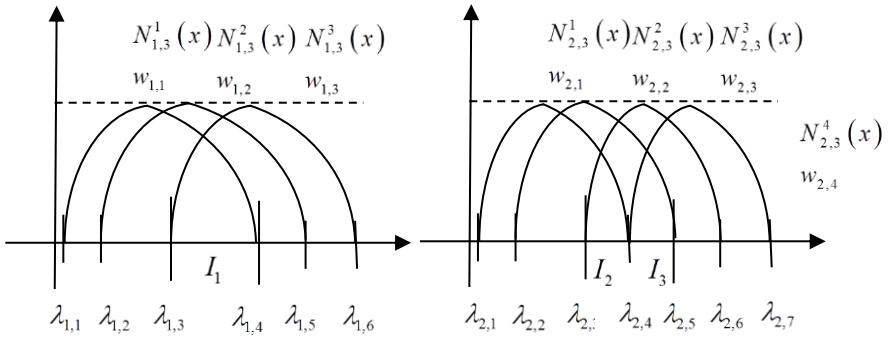


Fig. B.3. Quadratic splines representation of two univariate submodels, in the same input variable $I_1 = I_2 \cup I_3$.

Since

$$y_1 = \begin{bmatrix} N_{1,3}^1 & N_{1,3}^2 & N_{1,3}^3 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{1,3} \end{bmatrix}$$
$$y_2 = \begin{bmatrix} N_{2,3}^1 & N_{2,3}^2 & N_{2,3}^3 & N_{2,3}^4 \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \\ w_{2,4} \end{bmatrix} \tag{B.10}$$

Assuming $\lambda_{1,3} = \lambda_{2,3}$ and $\lambda_{1,4} = \lambda_{2,5}$: $I_1 = I_2 \cup I_3$.

As the splines are of order *k*=3, one needs to evaluate both output continuity and 1st derivative continuity. This is accomplished at two points:

- $x = \lambda_{1,3}$:

$$y_1(\lambda_{1,3}) = y_2(\lambda_{1,3}) \Leftrightarrow N_{1,3}^1 w_{1,1} + N_{1,3}^2 w_{1,2} = N_{2,3}^1 w_{2,1} + N_{2,3}^2 w_{2,2}$$
$$w_{2,1} = \left[\frac{\lambda_{1,4} - \lambda_{1,3}}{\lambda_{1,4} - \lambda_{1,2}} w_{1,1} + \frac{\lambda_{1,3} - \lambda_{1,2}}{\lambda_{1,4} - \lambda_{1,2}} w_{1,2} - \frac{\lambda_{1,3} - \lambda_{2,2}}{\lambda_{2,4} - \lambda_{2,2}} w_{2,2} \right] \frac{\lambda_{2,4} - \lambda_{2,2}}{\lambda_{2,4} - \lambda_{1,3}} \tag{B.11}$$

$$\frac{\partial y_1(x)}{\partial \lambda_{1,3}} = \frac{\partial y_2(x)}{\partial \lambda_{1,3}} \Leftrightarrow \frac{w_{1,1} - w_{1,2}}{\lambda_{1,2} - \lambda_{1,4}} = \frac{w_{2,1} - w_{2,2}}{\lambda_{2,2} - \lambda_{2,4}}$$
$$\Rightarrow w_{2,1} = w_{2,2} + \frac{\lambda_{2,2} - \lambda_{2,4}}{\lambda_{1,2} - \lambda_{1,4}} \left(w_{1,1} - w_{1,2} \right) \tag{B.12}$$

Replacing (B.12) in (B.11), then:

$$w_{2,1} = \frac{\lambda_{2,2} - \lambda_{1,4}}{\lambda_{1,2} - \lambda_{1,4}} w_{1,1} + \frac{\lambda_{1,2} - \lambda_{2,2}}{\lambda_{1,2} - \lambda_{1,4}} w_{1,2}$$
$$w_{2,2} = \frac{\lambda_{2,4} - \lambda_{1,4}}{\lambda_{1,2} - \lambda_{1,4}} w_{1,1} + \frac{\lambda_{1,2} - \lambda_{2,4}}{\lambda_{1,2} - \lambda_{1,4}} w_{1,2} \quad \text{(B.13)}$$

- $x = \lambda_{1,4}$:

$$y_1(\lambda_{1,4}) = y_2(\lambda_{1,4}) \Leftrightarrow N_{1,3}^2 w_{1,2} + N_{1,3}^3 w_{1,3} = N_{2,3}^3 w_{2,3} + N_{2,3}^4 w_{2,4} \Leftrightarrow$$
$$w_{2,4} = \left[\frac{\lambda_{1,5} - \lambda_{1,4}}{\lambda_{1,5} - \lambda_{1,3}} w_{1,2} + \frac{\lambda_{1,4} - \lambda_{1,3}}{\lambda_{1,5} - \lambda_{1,3}} w_{1,3} - \frac{\lambda_{2,6} - \lambda_{1,4}}{\lambda_{2,5} - \lambda_{2,4}} w_{2,3} \right] \frac{\lambda_{2,6} - \lambda_{2,4}}{\lambda_{1,4} - \lambda_{2,4}} \quad \text{(B.14)}$$

$$\frac{\partial y_1(x)}{\partial \lambda_{1,4}} = \frac{\partial y_2(x)}{\partial \lambda_{1,4}} \Leftrightarrow \frac{w_{1,2} - w_{1,3}}{\lambda_{1,3} - \lambda_{1,5}} = \frac{w_{2,3} - w_{2,4}}{\lambda_{2,4} - \lambda_{2,6}}$$
$$\Rightarrow w_{2,4} = w_{2,3} + \frac{\lambda_{2,6} - \lambda_{2,4}}{\lambda_{1,3} - \lambda_{1,5}} \left(w_{1,2} - w_{1,3} \right) \quad \text{(B.15)}$$

Replacing (B.15) in (B.14):

$$w_{2,3} = \frac{\lambda_{2,4} - \lambda_{1,5}}{\lambda_{1,3} - \lambda_{1,5}} w_{1,2} + \frac{\lambda_{1,3} - \lambda_{2,4}}{\lambda_{1,3} - \lambda_{1,5}} w_{1,3}$$
$$w_{2,4} = \frac{\lambda_{2,6} - \lambda_{1,5}}{\lambda_{1,3} - \lambda_{1,5}} w_{1,2} + \frac{\lambda_{1,3} - \lambda_{2,6}}{\lambda_{1,3} - \lambda_{1,5}} w_{1,3} \quad \text{(B.16)}$$

This example requires no more conditions. However, if the difference in the number of interior knots from both sub-models exceeds the value (*k*-1), another expression must be formulated.

Considering $x = \lambda_{2,4}$ one requires that:

$$y_1(\lambda_{2,4}) = y_2(\lambda_{2,4}) \Leftrightarrow$$
$$N_{1,3}^1 w_{1,1} + N_{1,3}^2 w_{1,2} + N_{1,3}^3 w_{1,3} = N_{2,3}^2 w_{2,2} + N_{2,3}^3 w_{2,3}$$
$$w_{2,3} = \left[\begin{array}{l} \dfrac{\left(\lambda_{1,4} - \lambda_{2,4}\right)^2}{\left(\lambda_{1,4} - \lambda_{1,2}\right)\left(\lambda_{1,4} - \lambda_{1,3}\right)} w_{1,1} + \left[\dfrac{\lambda_{2,4} - \lambda_{1,2}}{\lambda_{1,4} - \lambda_{1,2}} \dfrac{\lambda_{1,4} - \lambda_{2,4}}{\lambda_{1,4} - \lambda_{1,3}} + \dfrac{\lambda_{1,5} - \lambda_{2,4}}{\lambda_{1,5} - \lambda_{1,3}} \dfrac{\lambda_{2,4} - \lambda_{1,3}}{\lambda_{1,4} - \lambda_{1,3}} \right] w_{1,2} + \\ + \dfrac{\left(\lambda_{2,4} - \lambda_{1,3}\right)^2}{\left(\lambda_{1,5} - \lambda_{1,3}\right)\left(\lambda_{1,4} - \lambda_{1,3}\right)} w_{1,3} - \dfrac{\lambda_{2,5} - \lambda_{2,4}}{\lambda_{2,5} - \lambda_{2,3}} w_{2,2} \end{array} \right] \times$$
$$\times \frac{\lambda_{2,5} - \lambda_{2,3}}{\lambda_{2,4} - \lambda_{2,3}}$$

(B.17)

If there are r_1 interior knots in sub-model 1, and r_2 interior knots in sub-model 2, the general expression for the weights relation can be computed as follows:

$$\begin{cases} w_{2,1+i} = \dfrac{\lambda_{2,2+i} - \lambda_{1,4+j}}{\lambda_{1,2+j} - \lambda_{1,4+j}} w_{1,1+j} + \dfrac{\lambda_{1,2+j} - \lambda_{2,2+i}}{\lambda_{1,2+j} - \lambda_{1,4+j}} w_{1,2+j} \\ w_{2,2+i} = \dfrac{\lambda_{2,4+i} - \lambda_{1,4+j}}{\lambda_{1,2+j} - \lambda_{1,4+j}} w_{1,1+j} + \dfrac{\lambda_{1,2+j} - \lambda_{2,4+i}}{\lambda_{1,2+j} - \lambda_{1,4+j}} w_{1,2+j} \end{cases}, if\ \lambda_{1,3+j} = \lambda_{2,3+i}$$

$$w_{2,3+m} = \left[\begin{array}{l} \dfrac{\left(\lambda_{1,3+\text{int}1} - \lambda_{2,4+m}\right)^2}{\left(\lambda_{1,3+\text{int}1} - \lambda_{1,1+\text{int}1}\right)\left(\lambda_{1,3+\text{int}1} - \lambda_{1,2+\text{int}1}\right)} w_{1,\text{int}1} + \left[\begin{array}{l} \dfrac{\lambda_{2,4+m} - \lambda_{1,1+\text{int}1}}{\lambda_{1,3+\text{int}1} - \lambda_{1,1+\text{int}1}} \dfrac{\lambda_{1,3+\text{int}1} - \lambda_{2,4+m}}{\lambda_{1,3+\text{int}1} - \lambda_{1,2+\text{int}1}} + \\ + \dfrac{\lambda_{1,4+\text{int}1} - \lambda_{2,4+m}}{\lambda_{1,4+\text{int}1} - \lambda_{1,2+\text{int}1}} \dfrac{\lambda_{2,4+m} - \lambda_{1,2+\text{int}1}}{\lambda_{1,3+\text{int}1} - \lambda_{1,2+\text{int}1}} \end{array} \right] w_{1,1+\text{int}1} + \\ + \dfrac{\left(\lambda_{2,4+m} - \lambda_{1,2+\text{int}1}\right)^2}{\left(\lambda_{1,4+\text{int}1} - \lambda_{1,2+\text{int}1}\right)\left(\lambda_{1,3+\text{int}1} - \lambda_{1,2+\text{int}1}\right)} w_{1,2+\text{int}1} - \dfrac{\lambda_{2,5+m} - \lambda_{2,4+m}}{\lambda_{2,5+m} - \lambda_{2,3+m}} w_{2,2+m} \end{array} \right] \times$$

$$\times \frac{\lambda_{2,5+m} - \lambda_{2,3+m}}{\lambda_{2,4+m} - \lambda_{2,3+m}}, \quad \left(i = 0...r_2\right) \wedge \left(j = 0...r_1\right) \wedge \left(m = 0...r_2 - 2\right)$$

(B.18)

The expressions in (B.18) can be summarized as:

$$w_{2,3+m} = \left[\begin{array}{l} \dfrac{\left(\lambda_{1,3+\text{int}1} - \lambda_{2,4+m}\right)^2}{\left(\lambda_{1,3+\text{int}1} - \lambda_{1,1+\text{int}1}\right)\left(\lambda_{1,3+\text{int}1} - \lambda_{1,2+\text{int}1}\right)} w_{1,\text{int}1} + \left[\begin{array}{l} \dfrac{\lambda_{2,4+m} - \lambda_{1,1+\text{int}1}}{\lambda_{1,3+\text{int}1} - \lambda_{1,1+\text{int}1}} \dfrac{\lambda_{1,3+\text{int}1} - \lambda_{2,4+m}}{\lambda_{1,3+\text{int}1} - \lambda_{1,2+\text{int}1}} + \\ + \dfrac{\lambda_{1,4+\text{int}1} - \lambda_{2,4+m}}{\lambda_{1,4+\text{int}1} - \lambda_{1,2+\text{int}1}} \dfrac{\lambda_{2,4+m} - \lambda_{1,2+\text{int}1}}{\lambda_{1,3+\text{int}1} - \lambda_{1,2+\text{int}1}} \end{array} \right] w_{1,1+\text{int}1} + \\ + \dfrac{\left(\lambda_{2,4+m} - \lambda_{1,2+\text{int}1}\right)^2}{\left(\lambda_{1,4+\text{int}1} - \lambda_{1,2+\text{int}1}\right)\left(\lambda_{1,3+\text{int}1} - \lambda_{1,2+\text{int}1}\right)} w_{1,2+\text{int}1} - \dfrac{\lambda_{2,5+m} - \lambda_{2,4+m}}{\lambda_{2,5+m} - \lambda_{2,3+m}} w_{2,2+m} \end{array} \right] \times$$

$$\times \frac{\lambda_{2,5+m} - \lambda_{2,3+m}}{\lambda_{2,4+m} - \lambda_{2,3+m}}$$

$$w_{2,2+m} = \frac{\lambda_{2,3+m} - \lambda_{1,3+\text{int}1}}{\lambda_{1,1+\text{int}1} - \lambda_{1,3+\text{int}1}} w_{1,\text{int}1} + \frac{\lambda_{1,1+\text{int}1} - \lambda_{2,3+m}}{\lambda_{1,1+\text{int}1} - \lambda_{1,3+\text{int}1}} w_{1,1+\text{int}1}, \quad if\ \lambda_{2,4+m} = \lambda_{1,2+\text{int}1} \tag{B.19}$$

where $m = -1...r_2$.

Notice that int_1 is the interval number in submodel 1 to which $\lambda_{2,4+i}$ belongs to.

B.2 Cubic splines

Cubic splines are splines of order k=4. Therefore C^3 continuity is required.

B.2.1 Domain contiguity

Consider that the model is composed of two sub-models, where univariate cubic splines are used.

The knots from the two sub-models are distinct except for $\lambda_{1,5} = \lambda_{2,4}$. This would yield the following model:

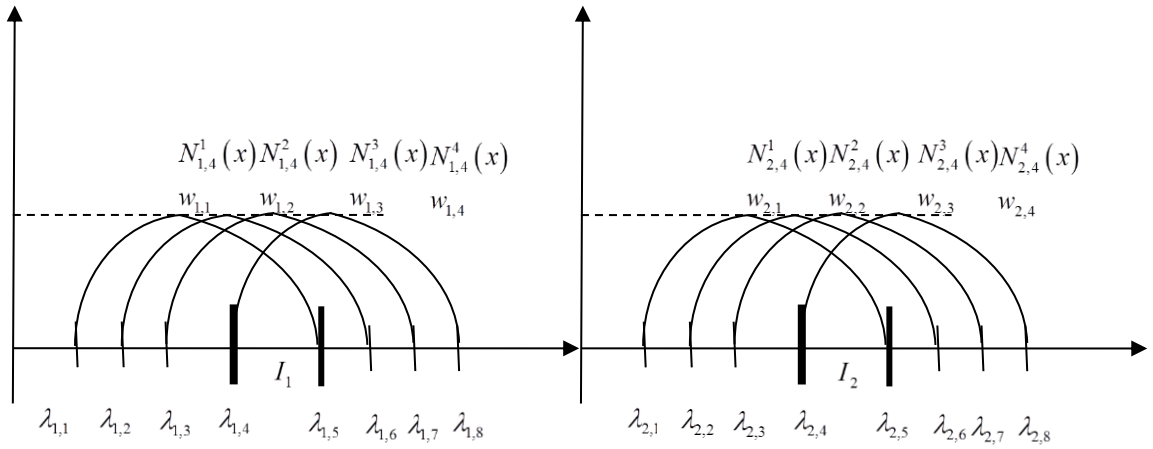


Fig. B.4. Cubic splines representation of two univariate sub-models, in the same input variable

For the model sketched above, the output is given as

$$y = y_1 + y_2 = \begin{bmatrix} N_{1,4}^1 & N_{1,4}^2 & N_{1,4}^3 & N_{1,4}^4 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{1,3} \\ w_{1,4} \end{bmatrix} + \begin{bmatrix} N_{2,4}^1 & N_{2,4}^2 & N_{2,4}^3 & N_{2,4}^4 \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \\ w_{2,4} \end{bmatrix} \tag{B.20}$$

Because there are cubic splines, 4 splines are active simultaneously per input interval, and one needs not only to ensure function output continuity, but also first and second derivative continuities. All these conditions must be employed at the input domain intersection for any two sub-models. In this case, to ensure continuity of the function at $\lambda_{1,5}$, one must have:

$$y_1(\lambda_{1,5}) = y_2(\lambda_{1,5}) \Leftrightarrow \begin{bmatrix} 0 & N_{1,4}^2 & N_{1,4}^3 & N_{1,4}^4 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{1,3} \\ w_{1,4} \end{bmatrix} = \begin{bmatrix} N_{2,4}^1 & N_{2,4}^2 & N_{2,4}^3 & 0 \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \\ w_{2,4} \end{bmatrix} \tag{B.21}$$

Replacing the output of the splines by their definition, and solving for $w_{2,1}$:

$$w_{2,1} = \frac{N_{1,4}^2 w_{1,2} + N_{1,4}^3 w_{1,4} + N_{1,4}^4 w_{1,4} - N_{2,4}^2 w_{2,2} - N_{2,4}^3 w_{2,3}}{N_{2,4}^1} \Leftrightarrow$$

$$w_{2,1} = \left[\begin{array}{l} \dfrac{\left(\lambda_{1,6}-\lambda_{1,5}\right)^2}{\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)} w_{1,2} + \left[\dfrac{\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,6}-\lambda_{1,5}\right)}{\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)} + \dfrac{\left(\lambda_{1,7}-\lambda_{1,5}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}{\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)} \right] w_{1,3} + \\ + \dfrac{\left(\lambda_{1,5}-\lambda_{1,4}\right)^2}{\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)} w_{1,4} - \left[\dfrac{\left(\lambda_{1,5}-\lambda_{2,2}\right)\left(\lambda_{2,5}-\lambda_{1,5}\right)}{\left(\lambda_{2,5}-\lambda_{2,2}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)} + \dfrac{\left(\lambda_{2,6}-\lambda_{1,5}\right)\left(\lambda_{1,5}-\lambda_{2,3}\right)}{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)} \right] w_{2,2} - \\ - \dfrac{\left(\lambda_{1,5}-\lambda_{2,3}\right)^2}{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)} w_{2,3} \end{array} \right] \times$$

$$\times \frac{\left(\lambda_{2,5}-\lambda_{2,2}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)}{\left(\lambda_{2,5}-\lambda_{1,5}\right)^2}$$

Now, for the first derivative to be continuous at $\lambda_{1,5}$:

$$N_{1,4}^3{}'\left(\lambda_{1,5}\right) w_{1,3}{}' + N_{1,4}^4{}'\left(\lambda_{1,5}\right) w_{1,4}{}' = N_{2,4}^2{}'\left(\lambda_{1,5}\right) w_{2,2}{}' + N_{2,4}^3{}'\left(\lambda_{1,5}\right) w_{2,3}{}' \Leftrightarrow$$

$$3\frac{\left(\lambda_{1,6}-\lambda_{1,5}\right)\left(w_{1,3}-w_{1,2}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)} + 3\frac{\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(w_{1,4}-w_{1,3}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)} =$$

$$3\frac{\left(\lambda_{2,5}-\lambda_{1,5}\right)\left(w_{2,2}-w_{2,1}\right)}{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)} + 3\frac{\left(\lambda_{1,5}-\lambda_{2,3}\right)\left(w_{2,3}-w_{2,2}\right)}{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)} \Leftrightarrow$$

$$\frac{w_{1,2}\left(\lambda_{1,5}-\lambda_{1,6}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right) + w_{1,3}\left(\left(\lambda_{1,6}-\lambda_{1,5}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right) - \left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\right) + w_{1,4}\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)}$$

$$= \frac{w_{2,1}\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right) + w_{2,2}\left(\left(\lambda_{2,5}-\lambda_{1,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right) - \left(\lambda_{1,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right) + w_{2,3}\left(\lambda_{1,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)}$$

If one solves for $\lambda_{2,2}$:

$$\begin{aligned} w_{2,2} = {} & \frac{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)\left[\left(\lambda_{2,5}-\lambda_{1,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right) - \left(\lambda_{1,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right]} \times \\ & \times \left[w_{1,2}\left(\lambda_{1,5}-\lambda_{1,6}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right) + w_{1,3}\left[\left(\lambda_{1,6}-\lambda_{1,5}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right) - \left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\right] + w_{1,4}\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\right] + \\ & + \frac{1}{\left[\left(\lambda_{2,5}-\lambda_{1,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right) - \left(\lambda_{1,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right]} \times \left[-w_{2,1}\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right) - w_{2,3}\left(\lambda_{1,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right] \end{aligned} \quad \text{(B.22)}$$

In (B.22) the first term depends on a subtraction term on the denominator. Under certain conditions the denominator can be zero. If one explicits the equation in terms of $w_{2,1}$ instead of $w_{2,2}$, then:

$$w_{2,1}=\frac{-\left[\left(\lambda_{2,5}-\lambda_{2,4}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)-\left(\lambda_{2,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right]w_{2,2}-w_{2,3}\left(\lambda_{2,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}{\left(\lambda_{2,4}-\lambda_{2,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)}+$$
$$+\frac{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}{\left(\lambda_{1,6+r_1}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)\left(\lambda_{2,4}-\lambda_{2,5}\right)}\times\begin{bmatrix}w_{1,2+r_1}\left(\lambda_{2,4}-\lambda_{1,6+r_1}\right)\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)+\\+w_{1,3+r_1}\begin{bmatrix}\left(\lambda_{1,6+r_1}-\lambda_{2,4}\right)\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)-\\-\left(\lambda_{2,4}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\end{bmatrix}+\\+w_{1,4+r_1}\left(\lambda_{2,4}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\end{bmatrix} \quad \text{(B.23)}$$

which gives a finite solution in any case. The second derivative continuity condition at $x=\lambda_{1,5}$ is expressed as:

$$N_{1,4}^{4}"\left(\lambda_{1,5}\right)w_{1,4}"=N_{2,4}^{3}"\left(\lambda_{1,5}\right)w_{2,3}"\Leftrightarrow$$
$$w_{1,4}"=w_{2,3}"\Leftrightarrow$$
$$\frac{6}{\left(\lambda_{1,6}-\lambda_{1,4}\right)}\left[\frac{\left(w_{1,4}-w_{1,3}\right)}{\left(\lambda_{1,7}-\lambda_{1,4}\right)}-\frac{\left(w_{1,3}-w_{1,2}\right)}{\left(\lambda_{1,6}-\lambda_{1,3}\right)}\right]=\frac{6}{\left(\lambda_{2,5}-\lambda_{2,3}\right)}\left[\frac{\left(w_{2,3}-w_{2,2}\right)}{\left(\lambda_{2,6}-\lambda_{2,3}\right)}-\frac{\left(w_{2,2}-w_{2,1}\right)}{\left(\lambda_{2,5}-\lambda_{2,2}\right)}\right]\Leftrightarrow$$
$$\left[\frac{w_{1,4}\left(\lambda_{1,6}-\lambda_{1,3}\right)-w_{1,3}\left(\lambda_{1,7}-\lambda_{1,4}+\lambda_{1,6}-\lambda_{1,3}\right)+w_{1,2}\left(\lambda_{1,7}-\lambda_{1,4}\right)}{\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)}\right]=$$
$$=\frac{\left(\lambda_{1,6}-\lambda_{1,4}\right)}{\left(\lambda_{2,5}-\lambda_{2,3}\right)}\left[\frac{w_{2,3}\left(\lambda_{2,5}-\lambda_{2,2}\right)-w_{2,2}\left(\lambda_{2,5}-\lambda_{2,2}+\lambda_{2,6}-\lambda_{2,3}\right)+w_{2,1}\left(\lambda_{2,6}-\lambda_{2,3}\right)}{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}\right]$$

(B.24)

And, solving for $w_{2,3}$:

$$w_{2,3}=\frac{1}{\left(\lambda_{2,5}-\lambda_{2,2}\right)}\left[\begin{bmatrix}w_{2,2}\left(\lambda_{2,5}-\lambda_{2,2}+\lambda_{2,6}-\lambda_{2,3}\right)-w_{2,1}\left(\lambda_{2,6}-\lambda_{2,3}\right)+\frac{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)}\times\\\times\left[w_{1,2}\left(\lambda_{1,7}-\lambda_{1,4}\right)-w_{1,3}\left(\lambda_{1,7}-\lambda_{1,4}+\lambda_{1,6}-\lambda_{1,3}\right)+w_{1,4}\left(\lambda_{1,6}-\lambda_{1,3}\right)\right]\end{bmatrix}\right] \quad \text{(B.25)}$$

B.2.1.1 General Expressions

Assume that sub-model 1 has r_1 interior knots and that the intersection is found at $\lambda_{2,4}$. Please note that for domain intersection the number of interior knots in sub-model 2 is irrelevant. Then, the weights relations are given by:

$$w_{2,1}=\left[\begin{array}{l}\dfrac{\left(\lambda_{1,6+r_1}-\lambda_{2,4}\right)^2}{\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,4+r_1}\right)}w_{1,2+r_1}+\left[\dfrac{\left(\lambda_{2,4}-\lambda_{1,3+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{2,4}\right)}{\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,4+r_1}\right)}+\dfrac{\left(\lambda_{1,7+r_1}-\lambda_{2,4}\right)\left(\lambda_{2,4}-\lambda_{1,4+r_1}\right)}{\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,4+r_1}\right)}\right]w_{1,3+r_1}+\\ +\dfrac{\left(\lambda_{2,4}-\lambda_{1,4+r_1}\right)^2}{\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,4+r_1}\right)}w_{1,4+r_1}-\left[\dfrac{\left(\lambda_{2,4}-\lambda_{2,2}\right)\left(\lambda_{2,5}-\lambda_{2,4}\right)}{\left(\lambda_{2,5}-\lambda_{2,2}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)}+\dfrac{\left(\lambda_{2,6}-\lambda_{2,4}\right)\left(\lambda_{2,4}-\lambda_{2,3}\right)}{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)}\right]w_{2,2}-\\ -\dfrac{\left(\lambda_{2,4}-\lambda_{2,3}\right)^2}{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)}w_{2,3}\end{array}\right]\times$$

$$\times\frac{\left(\lambda_{2,5}-\lambda_{2,2}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)}{\left(\lambda_{2,5}-\lambda_{2,4}\right)^2}$$

$$w_{2,2}=\frac{-\left[\left(\lambda_{2,5}-\lambda_{2,4}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)-\left(\lambda_{2,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right]w_{2,2}-w_{2,3}\left(\lambda_{2,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}{\left(\lambda_{2,4}-\lambda_{2,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)}+$$

$$+\frac{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}{\left(\lambda_{1,6+r_1}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)\left(\lambda_{2,4}-\lambda_{2,5}\right)}\times\left[\begin{array}{l}w_{1,2+r_1}\left(\lambda_{2,4}-\lambda_{1,6+r_1}\right)\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)+\\ +w_{1,3+r_1}\left[\begin{array}{l}\left(\lambda_{1,6+r_1}-\lambda_{2,4}\right)\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)-\\ -\left(\lambda_{2,4}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\end{array}\right]+\\ +w_{1,4+r_1}\left(\lambda_{2,4}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\end{array}\right]$$

$$w_{2,3}=\frac{1}{\left(\lambda_{2,5}-\lambda_{2,2}\right)}\left[\begin{array}{l}w_{2,2}\left(\lambda_{2,5}-\lambda_{2,2}+\lambda_{2,6}-\lambda_{2,3}\right)-w_{2,1}\left(\lambda_{2,6}-\lambda_{2,3}\right)+\dfrac{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}{\left(\lambda_{1,6+r_1}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)}\times\\ \times\left[w_{1,2+r_1}\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}\right)-w_{1,3+r_1}\left(\lambda_{1,7+r_1}-\lambda_{1,4+r_1}+\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)+w_{1,4+r_1}\left(\lambda_{1,6+r_1}-\lambda_{1,3+r_1}\right)\right]\end{array}\right]$$

B.2.2 Input merging

Assume the following model:

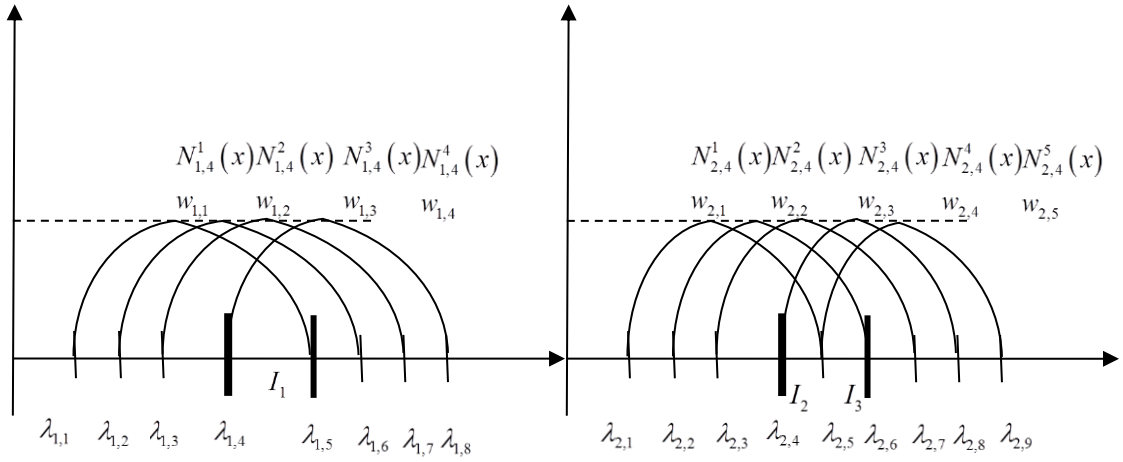


Fig. B.5. Cubic splines representation of two univariate submodels, in the same input variable $I_1=I_2\cup I_3$

If $\lambda_{1,4}=\lambda_{2,4}, \lambda_{1,5}=\lambda_{2,6}$ then $I_1=I_2$. This way, both sub-models span the same input domain.

It is required to ensure that both submodels' outputs (and also its derivatives) are equal at some particular input points, allowing the model to be expressed in a simpler way. In other words,

$$y_1=y_2 \Leftrightarrow \begin{bmatrix} N_{1,4}^1 & N_{1,4}^2 & N_{1,4}^3 & N_{1,4}^4 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ w_{1,3} \\ w_{1,4} \end{bmatrix} = \begin{bmatrix} N_{2,4}^1 & N_{2,4}^2 & N_{2,4}^3 & N_{2,4}^4 & N_{2,4}^5 \end{bmatrix} \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ w_{2,3} \\ w_{2,4} \\ w_{2,5} \end{bmatrix} \tag{B.26}$$

Consider 3 points:

- $x=\lambda_{1,4}$: To ensure the output continuity for this point:

$$w_{2,1}=\frac{N_{1,4}^1 w_{1,1}+N_{1,4}^2 w_{1,2}-N_{1,4}^3 w_{1,3}-N_{2,4}^2 w_{2,2}-N_{2,4}^3 w_{2,3}}{N_{2,4}^1} \Leftrightarrow$$

$$w_{2,1}=\left[\begin{array}{l} \dfrac{(\lambda_{1,5}-\lambda_{1,4})^2}{(\lambda_{1,5}-\lambda_{1,2})(\lambda_{1,5}-\lambda_{1,3})} w_{1,1}+\left[\dfrac{(\lambda_{1,4}-\lambda_{1,2})(\lambda_{1,5}-\lambda_{1,4})}{(\lambda_{1,5}-\lambda_{1,2})(\lambda_{1,5}-\lambda_{1,3})}+\dfrac{(\lambda_{1,6}-\lambda_{1,4})(\lambda_{1,4}-\lambda_{1,3})}{(\lambda_{1,6}-\lambda_{1,3})(\lambda_{1,5}-\lambda_{1,3})}\right] w_{1,2}+\dfrac{(\lambda_{1,4}-\lambda_{1,3})^2}{(\lambda_{1,6}-\lambda_{1,3})(\lambda_{1,5}-\lambda_{1,3})} w_{1,3}- \\ -\left[\dfrac{(\lambda_{1,4}-\lambda_{2,2})(\lambda_{2,5}-\lambda_{1,4})}{(\lambda_{2,5}-\lambda_{2,2})(\lambda_{2,5}-\lambda_{2,3})}+\dfrac{(\lambda_{2,6}-\lambda_{1,4})(\lambda_{1,4}-\lambda_{2,3})}{(\lambda_{2,6}-\lambda_{2,3})(\lambda_{2,5}-\lambda_{2,3})}\right] w_{2,2}-\dfrac{(\lambda_{1,4}-\lambda_{2,3})^2}{(\lambda_{2,6}-\lambda_{2,3})(\lambda_{2,5}-\lambda_{2,3})} w_{2,3} \end{array}\right] \cdot \tag{B.27}$$

$$\cdot\frac{(\lambda_{2,5}-\lambda_{2,2})(\lambda_{2,5}-\lambda_{2,3})}{(\lambda_{2,5}-\lambda_{1,4})^2}$$

To ensure the first derivative continuity for this point:

$$N_{1,4}^2{}'(\lambda_{1,4}) w_{1,2}'+N_{1,4}^3{}'(\lambda_{1,4}) w_{1,3}'=N_{2,4}^2{}'(\lambda_{1,4}) w_{2,2}'+N_{2,4}^3{}'(\lambda_{1,5}) w_{2,3}' \Leftrightarrow$$

$$3\frac{(\lambda_{1,5}-\lambda_{1,4})(w_{1,2}-w_{1,1})}{(\lambda_{1,5}-\lambda_{1,3})(\lambda_{1,5}-\lambda_{1,2})}+3\frac{(\lambda_{1,4}-\lambda_{1,3})(w_{1,3}-w_{1,2})}{(\lambda_{1,5}-\lambda_{1,3})(\lambda_{1,6}-\lambda_{1,3})}=$$

$$3\frac{(\lambda_{2,5}-\lambda_{1,4})(w_{2,2}-w_{2,1})}{(\lambda_{2,5}-\lambda_{2,3})(\lambda_{2,5}-\lambda_{2,2})}+3\frac{(\lambda_{1,4}-\lambda_{2,3})(w_{2,3}-w_{2,2})}{(\lambda_{2,5}-\lambda_{2,3})(\lambda_{2,6}-\lambda_{2,3})} \Leftrightarrow$$

$$\frac{w_{1,1}(\lambda_{1,6}-\lambda_{1,3})(\lambda_{1,4}-\lambda_{1,5})+w_{1,2}\left((\lambda_{1,6}-\lambda_{1,3})(\lambda_{1,5}-\lambda_{1,4})-(\lambda_{1,4}-\lambda_{1,3})(\lambda_{1,5}-\lambda_{1,2})\right)+w_{1,3}(\lambda_{1,4}-\lambda_{1,3})(\lambda_{1,5}-\lambda_{1,2})}{(\lambda_{1,5}-\lambda_{1,3})(\lambda_{1,5}-\lambda_{1,2})(\lambda_{1,6}-\lambda_{1,3})}$$

$$=\frac{w_{2,1}(\lambda_{1,4}-\lambda_{2,5})(\lambda_{2,6}-\lambda_{2,3})+w_{2,2}\left((\lambda_{2,5}-\lambda_{1,4})(\lambda_{2,6}-\lambda_{2,3})-(\lambda_{1,4}-\lambda_{2,3})(\lambda_{2,5}-\lambda_{2,2})\right)+w_{2,3}(\lambda_{1,4}-\lambda_{2,3})(\lambda_{2,5}-\lambda_{2,2})}{(\lambda_{2,5}-\lambda_{2,3})(\lambda_{2,5}-\lambda_{2,2})(\lambda_{2,6}-\lambda_{2,3})}$$

Solving for $w_{2,2}$:

$$w_{2,2}=\frac{\left(\lambda_{2,5}-\lambda_{2,2}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)}{\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\left[\left(\lambda_{2,5}-\lambda_{1,4}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)-\left(\lambda_{1,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right]}\times$$
$$\times\left[w_{1,1}\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,4}-\lambda_{1,5}\right)+w_{1,2}\left[\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)-\left(\lambda_{1,4}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)\right]+w_{1,3}\left(\lambda_{1,4}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)\right]+$$
$$-\frac{1}{\left[\left(\lambda_{2,5}-\lambda_{1,4}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)-\left(\lambda_{1,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right]}\times\left[w_{2,1}\left(\lambda_{1,4}-\lambda_{2,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)+w_{2,3}\left(\lambda_{1,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right]$$
(B.28)

As seen in the domain contiguity case, expression (B.28) must be rewritten, and so:

$$w_{2,1}=\frac{-\left[\left(\lambda_{2,5}-\lambda_{1,4}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)-\left(\lambda_{1,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)\right]w_{2,2}-w_{2,3}\left(\lambda_{1,4}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}{\left(\lambda_{1,4}-\lambda_{2,5}\right)\left(\lambda_{2,6}-\lambda_{2,3}\right)}+$$
$$+\frac{\left(\lambda_{2,5}-\lambda_{2,2}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)}{\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,4}-\lambda_{2,5}\right)}\times\left[\begin{array}{l}w_{1,1}\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,4}-\lambda_{1,5}\right)+\\+w_{1,2}\left[\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)-\left(\lambda_{1,4}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)\right]\\+w_{1,3}\left(\lambda_{1,4}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)\end{array}\right]$$
(B.29)

To ensure second derivative continuity at this point:

$$N_{1,4}^{3}{}''\left(\lambda_{1,4}\right)w_{1,3}''=N_{2,4}^{3}{}''\left(\lambda_{1,4}\right)w_{2,3}''\Leftrightarrow w_{1,3}''=w_{2,3}''$$
$$\frac{6}{\left(\lambda_{1,5}-\lambda_{1,3}\right)}\left[\frac{\left(w_{1,3}-w_{1,2}\right)}{\left(\lambda_{1,6}-\lambda_{1,3}\right)}-\frac{\left(w_{1,2}-w_{1,1}\right)}{\left(\lambda_{1,5}-\lambda_{1,2}\right)}\right]=\frac{6}{\left(\lambda_{2,5}-\lambda_{2,3}\right)}\left[\frac{\left(w_{2,3}-w_{2,2}\right)}{\left(\lambda_{2,6}-\lambda_{2,3}\right)}-\frac{\left(w_{2,2}-w_{2,1}\right)}{\left(\lambda_{2,5}-\lambda_{2,2}\right)}\right]\Leftrightarrow$$
$$\left[\frac{w_{1,3}\left(\lambda_{1,5}-\lambda_{1,2}\right)-w_{1,2}\left(\lambda_{1,5}-\lambda_{1,2}+\lambda_{1,6}-\lambda_{1,3}\right)+w_{1,1}\left(\lambda_{1,6}-\lambda_{1,3}\right)}{\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)}\right]=$$
$$=\frac{\left(\lambda_{1,5}-\lambda_{1,3}\right)}{\left(\lambda_{2,5}-\lambda_{2,3}\right)}\left[\frac{w_{2,3}\left(\lambda_{2,5}-\lambda_{2,2}\right)-w_{2,2}\left(\lambda_{2,5}-\lambda_{2,2}+\lambda_{2,6}-\lambda_{2,3}\right)+w_{2,1}\left(\lambda_{2,6}-\lambda_{2,3}\right)}{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,2}\right)}\right]$$

And, solving for $w_{2,3}$:

$$w_{2,3}=\frac{1}{\left(\lambda_{2,5}-\lambda_{2,2}\right)}\left[w_{2,2}\left(\lambda_{2,5}-\lambda_{2,2}+\lambda_{2,6}-\lambda_{2,3}\right)-w_{2,1}\left(\lambda_{2,6}-\lambda_{2,3}\right)\right]+$$
$$+\frac{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,5}-\lambda_{2,3}\right)}{\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)}\times\left[w_{1,3}\left(\lambda_{1,5}-\lambda_{1,2}\right)-w_{1,2}\left(\lambda_{1,5}-\lambda_{1,2}+\lambda_{1,6}-\lambda_{1,3}\right)+w_{1,1}\left(\lambda_{1,6}-\lambda_{1,3}\right)\right]$$
(B.30)

-For the input $\lambda_{1,5}$: The output continuity for this point is guaranteed if:

$$w_{2,3}=\frac{N_{1,4}^{2}w_{1,2}+N_{1,4}^{3}w_{1,3}+N_{1,4}^{4}w_{1,4}-N_{2,4}^{4}w_{2,4}-N_{2,4}^{5}w_{2,5}}{N_{2,4}^{1}}\Leftrightarrow$$

$$w_{2,3}=\left[\begin{array}{l}\dfrac{\left(\lambda_{1,6}-\lambda_{1,5}\right)^{2}}{\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)}w_{1,2}+\left[\dfrac{\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,6}-\lambda_{1,5}\right)}{\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)}+\dfrac{\left(\lambda_{1,7}-\lambda_{1,5}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}{\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)}\right]w_{1,3}+\dfrac{\left(\lambda_{1,5}-\lambda_{1,4}\right)^{2}}{\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)}w_{1,4}-\\ -\left[\dfrac{\left(\lambda_{1,5}-\lambda_{2,4}\right)\left(\lambda_{2,7}-\lambda_{1,5}\right)}{\left(\lambda_{2,7}-\lambda_{2,4}\right)\left(\lambda_{2,7}-\lambda_{2,5}\right)}+\dfrac{\left(\lambda_{2,8}-\lambda_{1,5}\right)\left(\lambda_{1,5}-\lambda_{2,5}\right)}{\left(\lambda_{2,8}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,5}\right)}\right]w_{2,4}-\dfrac{\left(\lambda_{1,5}-\lambda_{2,5}\right)^{2}}{\left(\lambda_{2,8}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,5}\right)}w_{2,5}\end{array}\right]\times$$

$$\times\frac{\left(\lambda_{2,7}-\lambda_{2,4}\right)\left(\lambda_{2,7}-\lambda_{2,5}\right)}{\left(\lambda_{2,7}-\lambda_{1,5}\right)^{2}} \qquad \text{(B.31)}$$

To ensure the first derivative continuity for this point one has:

$$N_{1,4}^{3}{}'\left(\lambda_{1,5}\right)w_{1,3}{}'+N_{1,4}^{4}{}'\left(\lambda_{1,5}\right)w_{1,4}{}'=N_{2,4}^{4}{}'\left(\lambda_{1,5}\right)w_{2,4}{}'+N_{2,4}^{5}{}'\left(\lambda_{1,5}\right)w_{2,5}{}'\Leftrightarrow$$

$$3\frac{\left(\lambda_{1,6}-\lambda_{1,5}\right)\left(w_{1,3}-w_{1,2}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)}+3\frac{\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(w_{1,4}-w_{1,3}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)}=$$

$$3\frac{\left(\lambda_{2,7}-\lambda_{1,5}\right)\left(w_{2,4}-w_{2,3}\right)}{\left(\lambda_{2,7}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)}+3\frac{\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(w_{2,5}-w_{2,4}\right)}{\left(\lambda_{2,7}-\lambda_{2,5}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)}\Leftrightarrow$$

$$\frac{w_{1,2}\left(\lambda_{1,5}-\lambda_{1,6}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)+w_{1,3}\left(\left(\lambda_{1,6}-\lambda_{1,5}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)-\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\right)+w_{1,3}\left(\lambda_{1,4}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,2}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)}$$

$$=\frac{w_{2,3}\left(\lambda_{1,5}-\lambda_{2,7}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)+w_{2,4}\left(\left(\lambda_{2,7}-\lambda_{1,5}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)-\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)\right)+w_{2,5}\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)}{\left(\lambda_{2,7}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)}$$

Solving for $w_{2,4}$:

$$w_{2,4}=\frac{\left(\lambda_{2,7}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)}\times$$

$$\times\left[\frac{w_{1,2}\left(\lambda_{1,5}-\lambda_{1,6}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)+w_{1,3}\left[\left(\lambda_{1,6}-\lambda_{1,5}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)-\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\right]+w_{1,4}\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)}{\left(\lambda_{2,7}-\lambda_{1,5}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)-\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)}\right]-$$

$$-\frac{1}{\left(\lambda_{2,7}-\lambda_{1,5}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)-\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)}\left[w_{2,3}\left(\lambda_{1,5}-\lambda_{2,7}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)+w_{2,5}\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)\right]$$

(B.32)

Once again, a subtraction between two terms appears on the denominator in equation (B.32) which requires a recast of the expression. Solving for $w_{2,3}$ instead of $w_{2,4}$:

$$w_{2,3}=\frac{-\left[\left(\lambda_{2,7}-\lambda_{1,5}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)-\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)\right]w_{2,4}-w_{2,5}\left(\lambda_{1,5}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)}{\left(\lambda_{1,5}-\lambda_{2,7}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)}+$$

$$+\frac{\left(\lambda_{2,7}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,5}-\lambda_{2,7}\right)}\times$$

$$\times\left[\begin{array}{l}w_{1,2}\left(\lambda_{1,5}-\lambda_{1,6}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)+\\ w_{1,3}\left[\left(\lambda_{1,6}-\lambda_{1,5}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)-\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\right]+\\ w_{1,4}\left(\lambda_{1,5}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)\end{array}\right]$$

(B.33)

And, to ensure the second derivative continuity at this point one has:

$$N_{1,4}^{4}\text{"}\left(\lambda_{1,5}\right)w_{1,4}\text{"}=N_{2,4}^{5}\text{"}\left(\lambda_{1,5}\right)w_{2,5}\text{"}\Leftrightarrow w_{1,4}\text{"}=w_{2,5}\text{"}$$

$$\frac{6}{\left(\lambda_{1,6}-\lambda_{1,4}\right)}\left[\frac{\left(w_{1,4}-w_{1,3}\right)}{\left(\lambda_{1,7}-\lambda_{1,4}\right)}-\frac{\left(w_{1,3}-w_{1,2}\right)}{\left(\lambda_{1,6}-\lambda_{1,3}\right)}\right]=\frac{6}{\left(\lambda_{2,7}-\lambda_{2,5}\right)}\left[\frac{\left(w_{2,5}-w_{2,4}\right)}{\left(\lambda_{2,8}-\lambda_{2,5}\right)}-\frac{\left(w_{2,4}-w_{2,3}\right)}{\left(\lambda_{2,7}-\lambda_{2,4}\right)}\right]\Leftrightarrow$$

$$\left[\frac{w_{1,4}\left(\lambda_{1,6}-\lambda_{1,3}\right)-w_{1,3}\left(\lambda_{1,6}-\lambda_{1,3}+\lambda_{1,7}-\lambda_{1,4}\right)+w_{1,2}\left(\lambda_{1,7}-\lambda_{1,4}\right)}{\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)}\right]= \tag{B.34}$$

$$=\frac{\left(\lambda_{1,6}-\lambda_{1,4}\right)}{\left(\lambda_{2,7}-\lambda_{2,5}\right)}\left[\frac{w_{2,5}\left(\lambda_{2,7}-\lambda_{2,4}\right)-w_{2,4}\left(\lambda_{2,7}-\lambda_{2,4}+\lambda_{2,8}-\lambda_{2,5}\right)+w_{2,3}\left(\lambda_{2,8}-\lambda_{2,5}\right)}{\left(\lambda_{2,8}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)}\right]$$

And, solving for $w_{2,5}$, which is the last weight:

$$w_{2,5}=\frac{1}{\left(\lambda_{2,7}-\lambda_{2,4}\right)}\left[w_{2,4}\left(\lambda_{2,7}-\lambda_{2,4}+\lambda_{2,8}-\lambda_{2,5}\right)-w_{2,3}\left(\lambda_{2,8}-\lambda_{2,5}\right)\right]+$$

$$+\frac{\left(\lambda_{2,7}-\lambda_{2,5}\right)\left(\lambda_{2,8}-\lambda_{2,5}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{2,7}-\lambda_{2,4}\right)\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,3}\right)}\times$$

$$\left[w_{1,4}\left(\lambda_{1,6}-\lambda_{1,3}\right)-w_{1,3}\left(\lambda_{1,6}-\lambda_{1,3}+\lambda_{1,7}-\lambda_{1,4}\right)+w_{1,2}\left(\lambda_{1,7}-\lambda_{1,4}\right)\right]$$

(B.35)

- $x=\lambda_{2,5}$: Likewise quadratic splines, this point will not add additional information on the output continuity condition, as long as sufficient equations are drawn from the remaining two other points. Notice, however, that this equation is needed if and only if this interior knot is at least the 3rd in the knot vector (because order k=3).

$$w_{2,3}=\frac{N_{1,4}^{4}w_{1,1}+N_{1,4}^{2}w_{1,2}+N_{1,4}^{3}w_{1,3}+N_{1,4}^{4}w_{1,4}-N_{2,4}^{2}w_{2,2}-N_{2,4}^{4}w_{2,4}}{N_{2,4}^{3}}\Leftrightarrow$$

$$\left[\begin{array}{l}\dfrac{\left(\lambda_{1,5}-\lambda_{2,5}\right)^{3}}{\left(\lambda_{1,5}-\lambda_{1,2}\right)\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}w_{1,1}+\left[\begin{array}{l}\dfrac{\left(\lambda_{2,5}-\lambda_{1,2}\right)\left(\lambda_{1,5}-\lambda_{2,5}\right)^{2}}{\left(\lambda_{1,5}-\lambda_{1,2}\right)\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}+\\ \dfrac{\left(\lambda_{1,6}-\lambda_{2,5}\right)}{\left(\lambda_{1,6}-\lambda_{1,3}\right)}\left[\dfrac{\left(\lambda_{2,5}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{2,5}\right)}{\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}+\dfrac{\left(\lambda_{1,6}-\lambda_{2,5}\right)\left(\lambda_{2,5}-\lambda_{1,4}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}\right]\end{array}\right]w_{1,2}+\\ +\left[\dfrac{\left(\lambda_{2,5}-\lambda_{1,3}\right)}{\left(\lambda_{1,6}-\lambda_{1,3}\right)}\left[\dfrac{\left(\lambda_{2,5}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{2,5}\right)}{\left(\lambda_{1,5}-\lambda_{1,3}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}+\dfrac{\left(\lambda_{1,6}-\lambda_{2,5}\right)\left(\lambda_{2,5}-\lambda_{1,4}\right)}{\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}\right]+\dfrac{\left(\lambda_{1,7}-\lambda_{2,5}\right)\left(\lambda_{2,5}-\lambda_{1,4}\right)^{2}}{\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}\right]w_{1,3}+\\ +\dfrac{\left(\lambda_{2,5}-\lambda_{1,4}\right)^{3}}{\left(\lambda_{1,7}-\lambda_{1,4}\right)\left(\lambda_{1,6}-\lambda_{1,4}\right)\left(\lambda_{1,5}-\lambda_{1,4}\right)}w_{1,4}-\dfrac{\left(\lambda_{2,6}-\lambda_{2,5}\right)^{2}}{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,6}-\lambda_{2,4}\right)}w_{2,2}-\dfrac{\left(\lambda_{2,5}-\lambda_{2,4}\right)^{2}}{\left(\lambda_{2,7}-\lambda_{2,4}\right)\left(\lambda_{2,6}-\lambda_{2,4}\right)}w_{2,4}\end{array}\right]/$$

$$/\left[\frac{\left(\lambda_{2,5}-\lambda_{2,3}\right)\left(\lambda_{2,6}-\lambda_{2,5}\right)}{\left(\lambda_{2,6}-\lambda_{2,3}\right)\left(\lambda_{2,6}-\lambda_{2,4}\right)}+\frac{\left(\lambda_{2,7}-\lambda_{2,5}\right)\left(\lambda_{2,5}-\lambda_{2,4}\right)}{\left(\lambda_{2,7}-\lambda_{2,4}\right)\left(\lambda_{2,6}-\lambda_{2,4}\right)}\right]$$

(B.36)

B.2.2.1 General Expressions

Assume r_1 and r_2 interior knots in sub-models 1 and 2, respectively.

Consider that $\left(i=0...r_2+1\right)\wedge\left(j=0...r_1+1\right)\wedge\lambda_{2,4+i}=\lambda_{1,4+j}\wedge m=5...r_2+2$. Then:

$$w_{2,1+i}=\left[\begin{array}{l}\dfrac{\left(\lambda_{1,5+j}-\lambda_{2,4+i}\right)^2}{\left(\lambda_{1,5+j}-\lambda_{1,2+j}\right)\left(\lambda_{1,5+j}-\lambda_{1,3+j}\right)}w_{1,1+j}+\left[\dfrac{\left(\lambda_{2,4+i}-\lambda_{1,2+j}\right)\left(\lambda_{1,5+j}-\lambda_{2,4+i}\right)}{\left(\lambda_{1,5+j}-\lambda_{1,2+j}\right)\left(\lambda_{1,5+j}-\lambda_{1,3+j}\right)}+\dfrac{\left(\lambda_{1,6+j}-\lambda_{2,4+i}\right)\left(\lambda_{2,4+i}-\lambda_{1,3+j}\right)}{\left(\lambda_{1,6+j}-\lambda_{1,3+j}\right)\left(\lambda_{1,5+j}-\lambda_{1,3+j}\right)}\right]w_{1,2+j}+\\ +\dfrac{\left(\lambda_{2,4+i}-\lambda_{1,3+j}\right)^2}{\left(\lambda_{1,6+j}-\lambda_{1,3+j}\right)\left(\lambda_{1,5+j}-\lambda_{1,3+j}\right)}w_{1,3+j}-\left[\dfrac{\left(\lambda_{2,4+i}-\lambda_{2,2+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,4+i}\right)}{\left(\lambda_{2,5+i}-\lambda_{2,2+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,3+i}\right)}+\dfrac{\left(\lambda_{2,6+i}-\lambda_{2,4+i}\right)\left(\lambda_{2,4+i}-\lambda_{2,3+i}\right)}{\left(\lambda_{2,6+i}-\lambda_{2,3+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,3+i}\right)}\right]w_{2,2+i}-\\ -\dfrac{\left(\lambda_{2,4+i}-\lambda_{2,3+i}\right)^2}{\left(\lambda_{2,6+i}-\lambda_{2,3+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,3+i}\right)}w_{2,3+i}\end{array}\right]\times$$
$$\times\frac{\left(\lambda_{2,5+i}-\lambda_{2,2+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,3+i}\right)}{\left(\lambda_{2,5+i}-\lambda_{2,4+i}\right)^2}$$

$$w_{2,2+i}=\frac{\left(\lambda_{2,5+i}-\lambda_{2,2+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,3+i}\right)\left(\lambda_{2,6+i}-\lambda_{2,3+i}\right)}{\left(\lambda_{1,5+j}-\lambda_{1,3+j}\right)\left(\lambda_{1,5+j}-\lambda_{1,2+j}\right)\left(\lambda_{1,6+j}-\lambda_{1,3+j}\right)\left[\left(\lambda_{2,5+i}-\lambda_{2,4+i}\right)\left(\lambda_{2,6+i}-\lambda_{2,3+i}\right)-\left(\lambda_{2,4+i}-\lambda_{2,3+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,2+i}\right)\right]}\times$$
$$\times\left[\begin{array}{l}w_{1,1+j}\left(\lambda_{1,6+j}-\lambda_{1,3+j}\right)\left(\lambda_{2,4+i}-\lambda_{1,5+j}\right)+w_{1,2+j}\left[\left(\lambda_{1,6+j}-\lambda_{1,3+j}\right)\left(\lambda_{1,5+j}-\lambda_{2,4+i}\right)-\left(\lambda_{2,4+i}-\lambda_{1,3+j}\right)\left(\lambda_{1,5+j}-\lambda_{1,2+j}\right)\right]+\\ +w_{1,3+j}\left(\lambda_{2,4+i}-\lambda_{1,3+j}\right)\left(\lambda_{1,5+j}-\lambda_{1,2+j}\right)\end{array}\right]+$$
$$-\frac{1}{\left[\left(\lambda_{2,5+i}-\lambda_{2,4+i}\right)\left(\lambda_{2,6+i}-\lambda_{2,3+i}\right)-\left(\lambda_{2,4+i}-\lambda_{2,3+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,2+i}\right)\right]}\times\left[\begin{array}{l}w_{2,1+i}\left(\lambda_{2,4+i}-\lambda_{2,5+i}\right)\left(\lambda_{2,6+i}-\lambda_{2,3+i}\right)+\\ +w_{2,3+i}\left(\lambda_{2,4+i}-\lambda_{2,3+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,2+i}\right)\end{array}\right]$$

$$w_{2,3+i}=\frac{1}{\left(\lambda_{2,5+i}-\lambda_{2,2+i}\right)}\left[-w_{2,1+i}\left(\lambda_{2,6+i}-\lambda_{2,3+i}\right)+w_{2,2+i}\left(\lambda_{2,5+i}-\lambda_{2,2+i}+\lambda_{2,6+i}-\lambda_{2,3+i}\right)\right]+$$
$$+\frac{\left(\lambda_{2,6+i}-\lambda_{2,3+i}\right)\left(\lambda_{2,5+i}-\lambda_{2,3+i}\right)}{\left(\lambda_{1,5+j}-\lambda_{1,3+j}\right)\left(\lambda_{1,6+j}-\lambda_{1,3+j}\right)\left(\lambda_{1,5+j}-\lambda_{1,2+j}\right)}\times$$
$$\times\left[w_{1,3+j}\left(\lambda_{1,5+j}-\lambda_{1,2+j}\right)-w_{1,2+j}\left(\lambda_{1,5+j}-\lambda_{1,2+j}+\lambda_{1,6+j}-\lambda_{1,3+j}\right)+w_{1,1+j}\left(\lambda_{1,6+j}-\lambda_{1,3+j}\right)\right]$$

$$
w_{2,m} = \left[\begin{array}{l}
\dfrac{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{2,m+2}\right)^3}{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,1+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} w_{1,\mathrm{int}1} + \\
+ \left[\begin{array}{l}
\dfrac{\left(\lambda_{2,m+2} - \lambda_{1,1+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{2,m+2}\right)^2}{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,1+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} + \\
\dfrac{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{2,m+2}\right)}{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)} \left[\dfrac{\left(\lambda_{2,m+2} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{2,m+2}\right)}{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} + \dfrac{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{2,m+2}\right)\left(\lambda_{2,m+2} - \lambda_{1,3+\mathrm{int}1}\right)}{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} \right]
\end{array} \right] w_{1,1+\mathrm{int}1} + \\
+ \left[\begin{array}{l}
\dfrac{\left(\lambda_{2,m+2} - \lambda_{1,2+\mathrm{int}1}\right)}{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)} \left[\dfrac{\left(\lambda_{2,m+2} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{2,m+2}\right)}{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} + \dfrac{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{2,m+2}\right)\left(\lambda_{2,m+2} - \lambda_{1,3+\mathrm{int}1}\right)}{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} \right] + \\
+ \dfrac{\left(\lambda_{1,6+\mathrm{int}1} - \lambda_{2,m+2}\right)\left(\lambda_{2,m+2} - \lambda_{1,3+\mathrm{int}1}\right)^2}{\left(\lambda_{1,6+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)}
\end{array} \right] w_{1,2+\mathrm{int}1} + \\
+ \dfrac{\left(\lambda_{2,m+2} - \lambda_{1,3+\mathrm{int}1}\right)^3}{\left(\lambda_{1,6+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} w_{1,3+\mathrm{int}1} - \dfrac{\left(\lambda_{2,m+3} - \lambda_{2,m+2}\right)^2}{\left(\lambda_{2,m+3} - \lambda_{2,m}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+1}\right)} w_{2,m-1} - \\
- \dfrac{\left(\lambda_{2,m+2} - \lambda_{2,m+1}\right)^2}{\left(\lambda_{2,m+4} - \lambda_{2,m+1}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+1}\right)} w_{2,m+1}
\end{array} \right] /
$$

$$
/ \left[\frac{\left(\lambda_{2,m+2} - \lambda_{2,m}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+2}\right)}{\left(\lambda_{2,m+3} - \lambda_{2,m}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+1}\right)} + \frac{\left(\lambda_{2,m+4} - \lambda_{2,m+2}\right)\left(\lambda_{2,m+2} - \lambda_{2,m+1}\right)}{\left(\lambda_{2,m+4} - \lambda_{2,m+1}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+1}\right)} \right]
$$

In the last expression, int_1 refers to the interval number to which $\lambda_{2,m+2}$ belongs to.

$$
w_{2,m} = \left[\begin{array}{l}
\dfrac{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{2,m+2}\right)^3}{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,1+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} w_{1,\mathrm{int}1} + \\
+ \left[\begin{array}{l}
\dfrac{\left(\lambda_{2,m+2} - \lambda_{1,1+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{2,m+2}\right)^2}{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,1+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} + \\
\dfrac{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{2,m+2}\right)}{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)} \left[\dfrac{\left(\lambda_{2,m+2} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{2,m+2}\right)}{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} + \dfrac{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{2,m+2}\right)\left(\lambda_{2,m+2} - \lambda_{1,3+\mathrm{int}1}\right)}{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} \right]
\end{array} \right] w_{1,1+\mathrm{int}1} + \\
+ \left[\begin{array}{l}
\dfrac{\left(\lambda_{2,m+2} - \lambda_{1,2+\mathrm{int}1}\right)}{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)} \left[\dfrac{\left(\lambda_{2,m+2} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{2,m+2}\right)}{\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,2+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} + \dfrac{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{2,m+2}\right)\left(\lambda_{2,m+2} - \lambda_{1,3+\mathrm{int}1}\right)}{\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} \right] + \\
+ \dfrac{\left(\lambda_{1,6+\mathrm{int}1} - \lambda_{2,m+2}\right)\left(\lambda_{2,m+2} - \lambda_{1,3+\mathrm{int}1}\right)^2}{\left(\lambda_{1,6+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)}
\end{array} \right] w_{1,2+\mathrm{int}1} + \\
+ \dfrac{\left(\lambda_{2,m+2} - \lambda_{1,3+\mathrm{int}1}\right)^3}{\left(\lambda_{1,6+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,5+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)\left(\lambda_{1,4+\mathrm{int}1} - \lambda_{1,3+\mathrm{int}1}\right)} w_{1,3+\mathrm{int}1} - \dfrac{\left(\lambda_{2,m+3} - \lambda_{2,m+2}\right)^2}{\left(\lambda_{2,m+3} - \lambda_{2,m}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+1}\right)} w_{2,m-1} - \\
- \dfrac{\left(\lambda_{2,m+2} - \lambda_{2,m+1}\right)^2}{\left(\lambda_{2,m+4} - \lambda_{2,m+1}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+1}\right)} w_{2,m+1}
\end{array} \right] /
$$

$$
/ \left[\frac{\left(\lambda_{2,m+2} - \lambda_{2,m}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+2}\right)}{\left(\lambda_{2,m+3} - \lambda_{2,m}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+1}\right)} + \frac{\left(\lambda_{2,m+4} - \lambda_{2,m+2}\right)\left(\lambda_{2,m+2} - \lambda_{2,m+1}\right)}{\left(\lambda_{2,m+4} - \lambda_{2,m+1}\right)\left(\lambda_{2,m+3} - \lambda_{2,m+1}\right)} \right]
$$

Appendix C

MATHEMATICAL FORMULATION FOR THE FUNCTIONAL APPROACH

As noted in chapter 8 employing the functional approach requires computation of some terms. These are:

$$\left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \right]^{-1} \tag{C.1}$$

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx}{2\partial \mathbf{v}^T} \tag{C.2}$$

$$\int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} dx \tag{C.3}$$

$$\int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T dx \tag{C.4}$$

Terms (C.1) and (C.2) are necessary to compute the functional version of the gradient vector and terms (C.3) and (C.4) are needed for the LM algorithm (see chapter 2, section 2.3.4).

In this Appendix a detailed description on how to calculate the former terms is given for B-Spline neural networks and Radial Basis Function networks.

C.1 B-Spline neural networks

C.1.1 Terms for a sub-model

C.1.1.1 Computation of $\left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}$

One will consider the inverse of $\left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}$.

Therefore define:

$$\boldsymbol{\Phi} = \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \tag{C.5}$$

C.1.1.1.1 Univariate case

Constant splines (k=1)

$$\boldsymbol{\Phi}_{i,j} = \begin{cases} 0, & i \neq j \\ \lambda_i - \lambda_{i-1}, & i = j \end{cases} \tag{C.6}$$

Triangular splines (k=2)

$$\boldsymbol{\Phi}_{i,j} = \begin{cases} \dfrac{\lambda_i - \lambda_{i-2}}{3}, & i = j \\ \dfrac{\lambda_i - \lambda_{i-1}}{6}, & i = j+1 \vee j = i+1 \\ 0, & otherwise \end{cases} \tag{C.7}$$

Quadratic splines (k=3)

$$\mathbf{\Phi}_{i,j}=\begin{cases}\dfrac{(\lambda_{i-3}-\lambda_i)\left(\lambda_{i-2}^2+\lambda_{i-2}\lambda_{i-1}+\lambda_{i-1}(\lambda_{i-1}-3\lambda_i)+3\lambda_{i-3}(\lambda_i-\lambda_{i-2})\right)}{15(\lambda_i-\lambda_{i-2})(\lambda_{i-1}-\lambda_{i-3})}, & i=j\\ \dfrac{1}{30(\lambda_{i-2}-\lambda_i)^2}\cdot & \\ \left[\begin{array}{l}\dfrac{1}{\lambda_{i-3}-\lambda_{i-1}}\left((\lambda_{i-2}-\lambda_{i-1})^2(\lambda_{i-2}{}^2+3\lambda_{i-2}\lambda_{i-1}+6\lambda_{i-1}{}^2-\right.\\ \left.2\lambda_{i-3}(2\lambda_{i-2}+3\lambda_{i-1}-5\lambda_i)-(\lambda_{i-2}+9\lambda_{i-1})\lambda_i\right)+\\ \dfrac{1}{\lambda_{i-1}-\lambda_{i+1}}(\lambda_{i-1}-\lambda_i)^2\left(6\lambda_{i-1}{}^2-\lambda_{i-2}(9\lambda_{i-1}+\lambda_i-10\lambda_{i+1})+\right.\\ \left.\lambda_i(\lambda_i-4\lambda_{i+1})+3\lambda_{i-1}(\lambda_i-2\lambda_{i+1})\right)\end{array}\right], & i=j+1\vee j=i+1\\ \dfrac{(\lambda_{i-1}-\lambda_i)^3}{30(\lambda_i-\lambda_{i-2})(\lambda_{i-1}-\lambda_{i+1})}, & i=j+2\vee j=i+2\\ 0, \quad \textit{otherwise} & \end{cases}$$

(C.8)

C.1.1.1.2 Multivariate case

Consider that $\mathbf{\Phi}^{(i)}=\int_{x_{i_{\min}}}^{x_{i_{MAX}}}\boldsymbol{\varphi}^{(i)}\boldsymbol{\varphi}^{(i)T}dx_i$, i.e., is the integral for the i^{th} dimension, computed with the univariate basis functions for this dimension. If $\otimes$ denotes the tensor product operation, then:

$$\mathbf{\Phi}=\int_{\mathbf{x}_{\min}}^{\mathbf{x}_{MAX}}\boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathbf{x}=\mathbf{\Phi}^{(1)}\otimes\cdots\otimes\mathbf{\Phi}^{(i)}\otimes\cdots\mathbf{\Phi}^{(n)} \tag{C.9}$$

C.1.1.2 Computation of $\dfrac{\partial\boldsymbol{\varphi}^T}{\partial\mathbf{v}^T}$

C.1.1.2.1 Univariate case

Using the recurrence formulae, one has:

$$\frac{\partial N_k^j(x)}{\partial\lambda_i}=\frac{\partial}{\partial\lambda_i}\left(\frac{x-\lambda_{j-k}}{\lambda_{j-1}-\lambda_{j-k}}\right)N_{k-1}^{j-1}(x)+\left(\frac{x-\lambda_{j-k}}{\lambda_{j-1}-\lambda_{j-k}}\right)\frac{\partial N_{k-1}^{j-1}(x)}{\partial\lambda_i}+$$
$$+\frac{\partial}{\partial\lambda_i}\left(\frac{\lambda_j-x}{\lambda_j-\lambda_{j-k+1}}\right)N_{k-1}^{j}(x)+\left(\frac{\lambda_j-x}{\lambda_j-\lambda_{j-k+1}}\right)\frac{\partial N_{k-1}^{j}(x)}{\partial\lambda_i},\qquad \frac{\partial}{\partial\lambda_i}N_1^j(x)=0 \tag{C.10}$$

C.1.1.2.2 Multivariate case

Considering that $\frac{\partial N_k^j(x_i)^T}{\partial \lambda_{i,m}} = \frac{\partial N_k^j(x_i)^T}{\partial v_{i,m}}$, i.e., is the derivative of the basis function for the i^{th} dimension in respect to the m^{th} interior knot, computed with the univariate basis functions for this dimension. If $\otimes$ denotes the tensor product operation, then

$$\frac{\partial}{\partial \lambda_{i,j}} \boldsymbol{\varphi}_m(\mathrm{x}) = N(x_1) \otimes \cdots \otimes N(x_{k-1}) \otimes \frac{\partial N(x_k)}{\partial \lambda_{k,j}} \otimes N(x_{k+1}) \cdots \otimes N(x_n) \tag{C.11}$$

C.1.1.3 Computation of $\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathrm{x}}{\partial \mathbf{v}^T}$

C.1.1.3.1 Univariate case

Constant splines (k=1)

$$\frac{\partial}{\partial \lambda_k} \boldsymbol{\Phi}_{i,j} = \begin{cases} 0, & i \neq j \\ 1, & i = j = k \\ -1, & i = j = k-1 \end{cases} \tag{C.12}$$

Triangular splines (k=2)

$$\frac{\partial}{\partial \lambda_k} \boldsymbol{\Phi}_{i,j} = \begin{cases} 1/3, & i = j = k \\ -1/3, & i = j = k+2 \\ 1/6, & i = j+1 \vee j = i+1, i = k \\ -1/6, & i = j+1 \vee j = i+1, i = k+1 \\ 0, & \textit{otherwise} \end{cases} \tag{C.13}$$

Quadratic splines (k=3)

$$\frac{\partial}{\partial\lambda_k}\mathbf{\Phi}_{i,i}=\begin{cases}\dfrac{-4\lambda_{i-3}\lambda_{i-2}{}^2+\lambda_{i-3}{}^3+2\lambda_{i-3}\lambda_{i-2}\lambda_{i-1}+\lambda_{i-2}{}^2\lambda_{i-1}-\lambda_{i-3}\lambda_{i-1}{}^2+\lambda_{i-2}\lambda_{i-1}{}^2+6\lambda_{i-3}\lambda_{i-2}\lambda_i-6\lambda_{i-2}\lambda_{i-1}\lambda_i-3\lambda_{i-3}\lambda_i{}^2+3\lambda_{i-1}\lambda_i{}^2}{15\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2}, & i=k\\ \dfrac{\left(2\lambda_{i-3}-\lambda_{i-2}-\lambda_{i-1}\right)\left(\lambda_{i-1}-\lambda_{i-2}\right)\left(\lambda_{i-3}-\lambda_i\right)}{15\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)}, & i=k+1\\ \dfrac{\left(\lambda_{i-2}-\lambda_{i-1}\right)\left(\lambda_{i-2}+\lambda_{i-1}-2\lambda_i\right)\left(\lambda_{i-3}-\lambda_i\right)}{15\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2}, & i=k+2\\ -\dfrac{\left(\begin{matrix}\lambda_{i-2}^2\left(\lambda_{i-1}-4\lambda_i\right)+3\lambda_{i-2}^2\left(\lambda_{i-2}-\lambda_i\right)+\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_i\right)+\\ 6\lambda_{i-3}\lambda_{i-1}\left(-\lambda_{i-2}+\lambda_i\right)+\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}+2\lambda_i\right)\end{matrix}\right)}{15\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)}, & i=k+3\\ 0, \ otherwise\end{cases} \tag{C.14}$$

$$\left.\frac{\partial}{\partial\lambda_k}\mathbf{\Phi}_{i,j}\right|_{i=j+1\vee j=i+1}=$$

$$\begin{cases}\dfrac{\left(-\lambda_{i-2}+\lambda_{i-1}\right)^3}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)}, & i=j+1\vee j=i+1, i=k+3\\ \dfrac{\begin{matrix}2\lambda_{i-2}^3\left(\lambda_{i-1}-\lambda_{i+1}\right)+\lambda_{i-2}^2\left(\lambda_{i-1}-3\lambda_i\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)+2\lambda_{i-2}\lambda_{i-1}\lambda_i\left(-\lambda_{i-1}+\lambda_{i+1}\right)-\\ -\lambda_{i-1}\left(\begin{matrix}\lambda_{i-1}^2\left(2\lambda_i-\lambda_{i+1}\right)+\\ \lambda_{i-1}\lambda_i\left(-5\lambda_i+\lambda_{i+1}\right)+\lambda_i^3\left(\lambda_i+2\lambda_{i+1}\right)\end{matrix}\right)+\\ +\lambda_{i-3}\left(\lambda_{i-1}^3-4\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_{i+1}\right)+8\lambda_{i-2}\lambda_i\left(\lambda_{i-1}-\lambda_{i+1}\right)-\lambda_{i-1}^2\left(\lambda_i+2\lambda_{i+1}\right)+\lambda_i^2\left(\lambda_i+2\lambda_{i+1}\right)+\lambda_{i-1}\lambda_i\left(-5\lambda_i+4\lambda_{i+1}\right)\right)\end{matrix}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & i=k+2\\ \dfrac{\begin{matrix}\lambda_{i-2}^3\left(\lambda_{i-1}-\lambda_{i+1}\right)^2-\lambda_{i-2}\lambda_{i-1}^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2-\lambda_{i-1}^2\left(\lambda_{i-1}-\lambda_i\right)\left(\lambda_i-\lambda_{i+1}\right)\left(\lambda_{i-1}+\lambda_i-2\lambda_{i+1}\right)+\\ +\lambda_{i-3}^2\left(\begin{matrix}-2\lambda_{i-1}^3-\lambda_{i-1}^2\left(\lambda_i-5\lambda_{i+1}\right)+2\lambda_{i-2}\left(\lambda_{i-1}-\lambda_{i+1}\right)^2+2\lambda_{i-1}\left(\lambda_i-2\lambda_{i+1}\right)\lambda_{i+1}+\\ \lambda_i\left(\lambda_i^3-3\lambda_i\lambda_{i+1}+2\lambda_{i+1}^2\right)\end{matrix}\right)+\\ \lambda_{i-3}\left(\begin{matrix}-3\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2+2\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}-\lambda_{i+1}\right)^2+\\ \lambda_{i-1}\left(\begin{matrix}\lambda_{i-1}^3+2\lambda_{i-1}^2\left(\lambda_i-2\lambda_{i+1}\right)+\lambda_{i-1}\lambda_{i+1}\left(-4\lambda_i+5\lambda_{i+1}\right)-\\ 2\lambda_i\left(\lambda_i^3-3\lambda_i\lambda_{i+1}+2\lambda_{i+1}^2\right)\end{matrix}\right)\end{matrix}\right)\end{matrix}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & i=k+1\\ \dfrac{\begin{matrix}\left(\lambda_{i-2}^3-5\lambda_{i-2}^2\lambda_{i-1}\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)-\lambda_{i-1}\left(\lambda_{i-1}+2\lambda_i\right)\left(\lambda_i^3+\lambda_{i-1}\lambda_{i+1}-2\lambda_i\lambda_{i+1}\right)\\ +\lambda_{i-1}\lambda_{i-2}\left(2\lambda_{i-1}^2+\lambda_i\left(3\lambda_i-8\lambda_{i+1}\right)+\lambda_{i-1}\left(2\lambda_i+\lambda_{i+1}\right)\right)+\\ \lambda_{i-3}\left(\begin{matrix}-\lambda_{i-1}^2+\lambda_{i-1}\lambda_i^2+2\lambda_i^2\left(\lambda_i-2\lambda_{i+1}\right)+2\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_{i+1}\right)+2\lambda_{i-1}^2\lambda_{i+1}+\\ \lambda_{i-2}\left(\lambda_{i-1}^2-2\lambda_{i-1}\left(\lambda_i+2\lambda_{i+1}\right)+\lambda_i\left(-3\lambda_i+8\lambda_{i+1}\right)\right)\end{matrix}\right)\end{matrix}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & i=k\\ \dfrac{\left(\lambda_{i-1}-\lambda_i\right)^3}{30\left(\lambda_{i-1}-\lambda_{i+1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)}, & i=k-1\\ 0, \ otherwise\end{cases} \tag{C.15}$$

C.1.1.3.2 Multivariate case

As every knot is associated with only one dimension, one has:

$$\frac{\partial}{\partial \lambda_{i,j}}\mathbf{\Phi} = \mathbf{\Phi}^{(1)} \otimes \cdots \otimes \mathbf{\Phi}^{(k-1)} \otimes \frac{\partial \mathbf{\Phi}^{(k)}}{\partial \lambda_{i,j}} \otimes \mathbf{\Phi}^{(k+1)} \cdots \otimes \mathbf{\Phi}^{(n)} \tag{C.16}$$

C.1.1.4 Computation of $\int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T}\frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} dx$

Assume $\int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T}\frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} dx = \mathbf{\Phi}'$ and $\int_{x_{min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial k}\frac{\partial \boldsymbol{\varphi}^T}{\partial m} dx = \left[\mathbf{\Phi}_{km}\right]'$

C.1.1.4.1 Univariate case

Constant splines (k=1)

$$\mathbf{\Phi}' = 0 \tag{C.17}$$

Triangular splines (k=2)

$$\left[\mathbf{\Phi}_{km}\right]'_{ij} = \left[\mathbf{\Phi}_{mk}\right]'_{ji} =$$

$$= \begin{cases} -\dfrac{1}{3\left(\lambda_{i-2} - \lambda_{i-1}\right)}, & i = j = k+2, m = k \\ \dfrac{-\lambda_{i-2} + \lambda_i}{3\left(\lambda_{i-2} - \lambda_{i-1}\right)\left(\lambda_{i-1} - \lambda_i\right)}, & i = j = k+1, m = k \\ -\dfrac{1}{3\left(\lambda_{i-1} - \lambda_i\right)}, & i = j = k = m \\ \dfrac{1}{6\left(\lambda_k - \lambda_{k+1}\right)}, & i = j+1, m = j = k+1 \\ \dfrac{1}{3\left(\lambda_{i-1} - \lambda_i\right)}, & i = j-1, m = k,\ j > k > i-2 \\ 0, & \textit{otherwise} \end{cases} \tag{C.18}$$

Quadratic splines (k=2)

$$[\mathbf{\Phi}_{km}]'_{ii} = [\mathbf{\Phi}_{mk}]'_{ii} =$$

$$= \begin{cases} -\dfrac{(2\lambda_{i-3}+\lambda_{i-2}-3\lambda_{i-1})}{15(\lambda_{i-3}-\lambda_{i-1})^2}, & k=i-3, m=k \\ \dfrac{\left(-3\lambda_{i-3}(\lambda_{i-2}-\lambda_i)^2+\lambda_{i-2}^2(\lambda_{i-1}+2\lambda_i)+\lambda_{i-2}(\lambda_{i-1}^2-4\lambda_{i-1}\lambda_i-3\lambda_i^2)+\lambda_{i-1}(\lambda_{i-1}^2-4\lambda_{i-1}\lambda_i+6\lambda_i^2)\right)}{30(\lambda_{i-3}-\lambda_{i-1})^2(\lambda_{i-2}-\lambda_i^2)}, & k=i-3, m=k+1 \\ \dfrac{\lambda_{i-2}^2+\lambda_{i-2}\lambda_{i-1}+\lambda_{i-1}(\lambda_{i-1}-3\lambda_i)-3\lambda_{i-3}(\lambda_{i-2}-\lambda_i)}{30(\lambda_{i-3}-\lambda_{i-1})^2(\lambda_{i-2}-\lambda_i)}, & k=i-3, m=k+2 \\ \dfrac{(\lambda_{i-2}-\lambda_{i-1})^3}{30(\lambda_{i-3}-\lambda_{i-1})^2(\lambda_{i-2}-\lambda_i)^2}, & k=i-3, m=k+3 \\ \dfrac{(2\lambda_{i-2}+\lambda_{i-1}-3\lambda_i)(\lambda_i-\lambda_{i-3})}{15(\lambda_{i-3}-\lambda_{i-1})(\lambda_{i-2}-\lambda_i)^2}, & k=i-2, m=k \\ -\dfrac{(3\lambda_{i-3}-\lambda_{i-2}-2\lambda_{i-1})(\lambda_{i-3}-\lambda_i)}{15(\lambda_{i-3}-\lambda_{i-1})^2(\lambda_{i-2}-\lambda_i)}, & k=i-1, m=k \\ \dfrac{-3\lambda_{i-2}+\lambda_{i-1}+2\lambda_i}{15(\lambda_{i-2}-\lambda_i)^2}, & k=i=m \\ 0, & \textit{otherwise} \end{cases}$$

$$
\left[\boldsymbol{\Phi}_{km}\right]'_{i,i+1} = \left[\boldsymbol{\Phi}_{mk}\right]'_{i+1,i} =
$$

$$
=\begin{cases}
-\dfrac{\left(\lambda_{i-2}-\lambda_{i-1}\right)^2\left(3\lambda_{i-2}+\lambda_{i-1}-4\lambda_i\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)^2}, & k=i-3, m=k+1 \\
-\dfrac{\left(\lambda_{i-2}-\lambda_{i-1}\right)^2}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)^2}, & k=i-3, m=k+2 \\
\dfrac{\left(-\lambda_{i-2}+\lambda_{i-1}\right)^3}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)^2}, & k=i-3, m=k+3 \\
-\dfrac{\left(3\lambda_{i-2}^2-2\lambda_{i-3}\left(2\lambda_{i-2}+\lambda_{i-1}-3\lambda_i\right)+2\lambda_{i-2}\left(\lambda_{i-1}-2\lambda_i\right)+\lambda_{i-1}\left(\lambda_{i-1}-2\lambda_i\right)\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2}, & k=i-2, m=k \\
\dfrac{\left(\begin{array}{l}-\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2-\lambda_{i-2}\left(2\lambda_{i-1}-\lambda_i\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)^2+\\+\lambda_{i-1}\left(2\lambda_{i-1\,i+1}^2+\lambda_{i-1}\left(\lambda_i^2-2\lambda_i\lambda_{i+1}-3\lambda_{i+1}^2\right)+\lambda_i\left(3\lambda_i^2-4\lambda_i\lambda_{i+1}+5\lambda_{i+1}^2\right)\right)+\\+\lambda_{i-3}\left(3\lambda_{i-2}\left(\lambda_{i-2}-\lambda_{i+1}\right)^2-\lambda_{i-1}^2\left(\lambda_i+2\lambda_{i+1}\right)+\lambda_{i-1}\left(-\lambda_i^2+4\lambda_i\lambda_{i+1}+3\lambda_{i+1}^2\right)-\lambda_i\left(\lambda_i^2-4\lambda_i\lambda_{i+1}+6\lambda_{i+1}^2\right)\right)\end{array}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i-2, m=k+1 \\
\dfrac{\left(\begin{array}{l}\lambda_{i-2}^2\left(-\lambda_{i-1}+\lambda_{i+1}\right)+\lambda_{i-2}\lambda_{i-1}\left(-\lambda_{i-1}+\lambda_{i+1}\right)+\lambda_{i-3}\left(3\lambda_{i-2}\lambda_{i-1}-\lambda_{i-1}^2-\lambda_{i-1}\lambda_{i-1}-\lambda_i^2-3\lambda_{i-2}\lambda_{i+1}+3\lambda_i\lambda_{i+1}\right)+\\+\lambda_{i-1}\left(\lambda_i\left(\lambda_i-3\lambda_{i+1}\right)+\lambda_{i-1}\left(\lambda_i+\lambda_{i+1}\right)\right)\end{array}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & k=i-2, m=k+2 \\
\dfrac{\left(-\lambda_{i-1}+\lambda_i\right)^3}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i-2, m=k+3 \\
\dfrac{\begin{array}{l}3\lambda_{i-2}^3+\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}-3\lambda_i\right)+2\lambda_{i-2}^2\left(\lambda_{i-1}-2\lambda_i\right)+3\lambda_{i-3}^2\left(\lambda_{i-2}+\lambda_{i-1}-2\lambda_i\right)+\lambda_i\lambda_{i-1}^2+\\+\lambda_{i-3}\left(-7\lambda_{i-2}^2+\lambda_{i-1}\left(-2\lambda_{i-1}+\lambda_i\right)+\lambda_{i-2}\left(-3\lambda_{i-1}+11\lambda_i\right)\right)\end{array}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)^2}, & k=i-1, m=k-1 \\
\dfrac{\begin{array}{l}-2\lambda_{i-3}\left(\lambda_{i-1}\left(\lambda_{i-1}+\lambda_i-2\lambda_{i+1}\right)^2+2\lambda_{i-2}\left(\lambda_{i-1}-\lambda_{i+1}\right)^2\right)+\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2+2\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}-\lambda_{i+1}\right)^2+\\+\lambda_{i-1}^2\left(2\lambda_{i-1}+\lambda_i-3\lambda_{i+1}\right)\left(\lambda_i-\lambda_{i+1}\right)+\lambda_{i-3}^2\left(3\lambda_{i-1}^2+\lambda_i^2+2\lambda_{i-1}\left(\lambda_i-4\lambda_{i+1}\right)-4\lambda_i\lambda_{i+1}+6\lambda_{i+1}^2\right)\end{array}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i-1, m=k \\
\dfrac{\begin{array}{l}-\lambda_{i-1}^2\lambda_i\left(2\lambda_{i-1}+\lambda_i-3\lambda_{i+1}\right)+\lambda_{i-2}^3\left(\lambda_{i-1}-\lambda_{i+1}\right)+\lambda_{i-2}^2\lambda_{i-1}\left(\lambda_{i-1}-\lambda_{i+1}\right)+\\+\lambda_{i-2}\lambda_{i-1}^2\left(\lambda_i-\lambda_{i+1}\right)+\lambda_{i-3}^2\left(-3\lambda_{i-1}^2-2\lambda_{i-1}\lambda_i-\lambda_i^2+\lambda_{i-2}\left(5\lambda_{i-1}+\lambda_i-6\lambda_{i+1}\right)+3\lambda_{i-1}\lambda_{i+1}+3\lambda_i\lambda_{i+1}\right)-\\-2\lambda_{i-3}\left(\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}+\lambda_i-2\lambda_{i+1}\right)+2\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_{i+1}\right)-\lambda_{i-1}\left(\lambda_{i-1}^2+2\lambda_{i-1}\lambda_i+\lambda_i^2-\lambda_{i-1}\lambda_{i+1}-3\lambda_i\lambda_{i+1}\right)\right)\end{array}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & k=i-1, m=k+1 \\
-\dfrac{\left(\lambda_{i-1}-\lambda_i\right)^2}{30\left(\lambda_{i-2}-\lambda_i\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i-1, m=k+2 \\
\dfrac{1}{10\left(\lambda_{i-2}-\lambda_i\right)}, & k=i, m=k-2 \\
\dfrac{\begin{array}{l}-\lambda_{i-1}^2\left(\lambda_i-2\lambda_{i+1}\right)+\lambda_{i-1}\left(-2\lambda_i^2+3\lambda_i\lambda_{i+1}-3\lambda_{i+1}^2\right)+\lambda_i\left(-3\lambda_i^2+7\lambda_i\lambda_{i+1}-3\lambda_{i+1}^2\right)-\\-\lambda_{i-2}\left(\lambda_{i-1}^2-4\lambda_i^2+11\lambda_i\lambda_{i+1}-6\lambda_{i+1}^2+\lambda_{i-1}\left(-3\lambda_i+\lambda_{i+1}\right)\right)\end{array}}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i, m=k-1 \\
-\dfrac{\lambda_{i-1}^2+\lambda_i\left(3\lambda_i-4\lambda_{i+1}\right)-2\lambda_{i-2}\left(\lambda_{i-1}+2\lambda_i-3\lambda_{i+1}\right)+2\lambda_{i-1}\left(\lambda_i-\lambda_{i+1}\right)}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & m=k=i \\
-\dfrac{\left(4\lambda_{i-2}-\lambda_{i-1}-3\lambda_i\right)\left(\lambda_{i-1}-\lambda_i\right)^2}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i, m=k+1 \\
0, & \textit{otherwise}
\end{cases}
$$

$$\left[\boldsymbol{\Phi}_{km}\right]'_{i,i+2} = \left[\boldsymbol{\Phi}_{mk}\right]'_{i+2,i} =$$

$$= \begin{cases} \dfrac{\left(\lambda_{i-1}-\lambda_i\right)^3}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i-2, m=k+3 \\ \dfrac{\left(\lambda_{i-1}-\lambda_i\right)^2}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & k=i-2, m=k+2 \\ \dfrac{\left(\lambda_{i-1}-\lambda_i\right)^2\left(3\lambda_{i-1}+\lambda_i-4\lambda_{i+1}\right)}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i-2, m=k+1 \\ \dfrac{\left(\lambda_{i-1}-\lambda_i\right)\left(3\lambda_{i-1}+\lambda_i-4\lambda_{i+1}\right)}{30\left(\lambda_{i-2}-\lambda_i\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i-1, m=k \\ \dfrac{\left(\lambda_{i-1}-\lambda_i\right)}{30\left(\lambda_{i-2}-\lambda_i\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & k=i-1, m=k+1 \\ \dfrac{\left(\lambda_{i-1}-\lambda_i\right)^2}{30\left(\lambda_{i-2}-\lambda_i\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i-1, m=k+2 \\ \dfrac{\left(4\lambda_{i-2}-\lambda_{i-1}-3\lambda_i\right)\left(\lambda_{i-1}-\lambda_i\right)^2}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i, m=k+1 \\ \dfrac{\left(4\lambda_{i-2}-\lambda_{i-1}-3\lambda_i\right)\left(\lambda_{i-1}-\lambda_i\right)}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & k=i, m=k \\ \dfrac{-\left(\lambda_{i-1}-\lambda_i\right)\left(3\lambda_{i-1}^2+\lambda_i\left(3\lambda_i-7\lambda_{i+1}\right)+\lambda_{i-1}\left(5\lambda_i-4\lambda_{i+1}\right)+\lambda_{i-2}\left(-7\lambda_{i-1}-4\lambda_i+11\lambda_{i+1}\right)\right)}{30\left(\lambda_{i-2}-\lambda_i\right)^2\left(\lambda_{i-1}-\lambda_{i+1}\right)^2}, & k=i, m=k-1 \\ 0, \quad \textit{otherwise} \end{cases}$$

(C.19)

C.1.1.4.2 Multivariate case

Assume that $\mathbf{v}^{(i)} = \left\{\lambda_{i,1},\ldots,\lambda_{i,r_i}\right\}$, are the interior knots for the i^{th} dimension, and that $\boldsymbol{\Phi}'^{(l)} = \int_{x_{i\min}}^{x_{iMAX}} \frac{\partial \boldsymbol{\varphi}^{(l)}}{\partial \mathbf{v}^{(l)T}} \frac{\partial \boldsymbol{\varphi}^{(l)T}}{\partial \mathbf{v}^{(l)T}} dx_l$.

If $\otimes$ denotes the tensor product operation, then

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^{(l)T}} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^{(l)T}} dx = \boldsymbol{\Phi}^{(1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(l-1)} \otimes \left[\boldsymbol{\Phi}\right]'^{(l)} \otimes \boldsymbol{\Phi}^{(l+1)} \cdots \otimes \boldsymbol{\Phi}^{(n)} \tag{C.20}$$

And if the parameters belong to distinct input variables,

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^{(l)T}} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^{(k)T}} dx = \boldsymbol{\Phi}^{(1)} \otimes \cdots \otimes \boldsymbol{\Phi}^{(l-1)} \otimes \left[\boldsymbol{\Phi}\right]'^{(l\otimes k)} \otimes \boldsymbol{\Phi}^{(l+1)} \otimes \boldsymbol{\Phi}^{(k-1)} \otimes \boldsymbol{\Phi}^{(k+1)} \cdots \otimes \boldsymbol{\Phi}^{(n)} \tag{C.21}$$

In the last expression,

$$\left[\boldsymbol{\Phi}\right]^{'(l\otimes k)} = \int_{x_{l\min}}^{x_{lMAX}} \frac{\partial \boldsymbol{\varphi}^{(l)}}{\partial l} \boldsymbol{\varphi}^{(l)^T} dx_l \otimes \int_{x_{k\min}}^{x_{kMAX}} \left[\frac{\partial \boldsymbol{\varphi}^{(k)}}{\partial k} \boldsymbol{\varphi}^{(k)^T} \right]^T dx_k \tag{C.22}$$

C.1.1.5 Computation of $\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial k} \boldsymbol{\varphi}^T dx$

Using similar denotations as before, one seeks to compute $\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial k} \boldsymbol{\varphi}^T dx = \left[\boldsymbol{\Phi}_k\right]''$.

C.1.1.5.1 Univariate case

Constant splines (k=1)

$$\left[\boldsymbol{\Phi}\right]'' = 0 \tag{C.23}$$

Triangular splines (k=2)

$$\left[\boldsymbol{\Phi}_k\right]''_{ij} == \begin{cases} -\frac{1}{6}, & i = j+1 = k+1 \vee i = j = k+2 \\ \frac{1}{6}, & j = i+1 = k+2 \vee i = j = k \\ \frac{1}{3}, & j = i+1 = k+1 \\ -\frac{1}{3}, & i = j+1 = k+2 \\ 0, & otherwise \end{cases} \tag{C.24}$$

Quadratic splines (k=3)

$$\left[\boldsymbol{\Phi}_{i-3}\right]''_{ij} = \begin{cases} -\frac{\left(\lambda_{i-3}-\lambda_{i-2}\right)\left(3\lambda_{i-3}+\lambda_{i-2}-4\lambda_{i-1}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2}, & j = i-2 \\ -\frac{\left(2\lambda_{i-3}+\lambda_{i-2}-3\lambda_{i-1}\right)}{15\left(\lambda_{i-3}-\lambda_{i-1}\right)}, & j = i-1 \\ -\frac{\left(\begin{array}{l}\lambda_{i-1}^2\left(\lambda_{i-1}-4\lambda_i\right)+3\lambda_{i-3}^2\left(\lambda_{i-2}-\lambda_i\right)+\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_i\right)+ \\ +6\lambda_{i-3}\lambda_{i-1}\left(-\lambda_{i-2}+\lambda_i\right)+\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}+2\lambda_i\right)\end{array}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)}, & j = i \\ \frac{\left(-\lambda_{i-2}+\lambda_{i-1}\right)^3}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^2\left(\lambda_{i-2}-\lambda_i\right)}, & j = i+1 \\ 0, & otherwise \end{cases}$$

$$
\left[\boldsymbol{\Phi}_{i-2}\right]''_{ij}=\begin{cases}
\dfrac{\left(-\lambda_{i-3}+\lambda_{i-2}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)}, & j=i-2\\
\dfrac{-3\lambda_{i-3}\left(\lambda_{i-2}-\lambda_{i}\right)^{2}+\lambda_{i-2}^{2}\left(\lambda_{i-1}+2\lambda_{i}\right)+\lambda_{i-1}\left(\lambda_{i-1}^{2}-4\lambda_{i-1}\lambda_{i}+6\lambda_{i}^{2}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_{i}\right)^{2}}, & j=i-1\\
\dfrac{\left(\lambda_{i-2}-\lambda_{i-1}\right)\left(\lambda_{i-2}+\lambda_{i-1}-2\lambda_{i}\right)\left(\lambda_{i-3}-\lambda_{i}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_{i}\right)^{2}}, & j=i\\
\dfrac{\begin{pmatrix}-\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}-4\lambda_{i}\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)-\lambda_{i-2}^{2}\left(\lambda_{i-1}-\lambda_{i}\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)+\lambda_{i-2}^{3}\left(-\lambda_{i-1}+\lambda_{i+1}\right)+\\
+\lambda_{i-3}\begin{pmatrix}\lambda_{i}^{2}\left(\lambda_{i}-4\lambda_{i+1}\right)+3\lambda_{i-2}^{2}\left(\lambda_{i-1}-\lambda_{i+1}\right)+\\
+\lambda_{i-1}^{2}\left(\lambda_{i}-\lambda_{i+1}\right)+6\lambda_{i-2}\lambda_{i}\left(-\lambda_{i-1}+\lambda_{i+1}\right)+\lambda_{i-1}\lambda_{i}\left(\lambda_{i}+2\lambda_{i+1}\right)\end{pmatrix}+\\
+\lambda_{i-1}\left(-\lambda_{i}^{2}\left(\lambda_{i}-4\lambda_{i+1}\right)+\lambda_{i-1}^{2}\lambda_{i+1}-\lambda_{i-1}\lambda_{i}\left(\lambda_{i}+3\lambda_{i+1}\right)\right)\end{pmatrix}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_{i}\right)^{2}\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & j=i+1\\
\dfrac{\left(\lambda_{i-1}-\lambda_{i}\right)^{3}}{30\left(\lambda_{i-2}-\lambda_{i}\right)^{2}\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & j=i+2\\
0, \quad \textit{otherwise}
\end{cases}
$$

$$
\left[\boldsymbol{\Phi}_{i-1}\right]''_{ij}=\begin{cases}
-\dfrac{\left(\lambda_{i-3}-\lambda_{i-2}\right)^{2}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^{2}}, & j=i-2\\
\dfrac{\lambda_{i-2}^{2}+\lambda_{i-2}\lambda_{i-1}+\lambda_{i-1}\left(\lambda_{i-1}-3\lambda_{i}\right)-3\lambda_{i-3}\left(\lambda_{i-2}-\lambda_{i}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_{i}\right)}, & j=i-1\\
\dfrac{\left(2\lambda_{i-3}-\lambda_{i-2}-\lambda_{i-1}\right)\left(-\lambda_{i-2}+\lambda_{i-1}\right)\left(\lambda_{i-3}-\lambda_{i}\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^{2}\left(\lambda_{i-2}-\lambda_{i}\right)}, & j=i\\
\dfrac{\begin{pmatrix}2\lambda_{i-3}\left(\lambda_{i-1}\left(\lambda_{i-1}^{2}+2\lambda_{i-1}\lambda_{i}+\lambda_{i}\left(\lambda_{i}-4\lambda_{i+1}\right)\right)-2\lambda_{i-2}^{2}\left(\lambda_{i-1}-\lambda_{i+1}\right)-2\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}-\lambda_{i+1}\right)\right)-\\
-\lambda_{i-3}^{2}\left(3\lambda_{i-1}^{2}+\lambda_{i}\left(3\lambda_{i}-4\lambda_{i+1}\right)-6\lambda_{i-2}\left(\lambda_{i-1}-\lambda_{i+1}\right)+2\lambda_{i-1}\left(\lambda_{i}-\lambda_{i+1}\right)\right)+\lambda_{i-2}^{3}\left(\lambda_{i-1}-\lambda_{i+1}\right)+\\
+\lambda_{i-2}^{2}\lambda_{i-1}\left(\lambda_{i-1}-\lambda_{i+1}\right)+\lambda_{i-2}\lambda_{i-1}^{2}\left(\lambda_{i-1}-\lambda_{i+1}\right)-\lambda_{i-1}^{2}\left(\lambda_{i}\left(\lambda_{i}-4\lambda_{i+1}\right)+\lambda_{i-1}\left(2\lambda_{i}+\lambda_{i+1}\right)\right)\end{pmatrix}}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)^{2}\left(\lambda_{i-2}-\lambda_{i}\right)\left(\lambda_{i-1}-\lambda_{i+1}\right)}, & j=i+1\\
\dfrac{\left(\lambda_{i-1}-\lambda_{i}\right)^{2}}{30\left(\lambda_{i-2}-\lambda_{i}\right)}, & j=i+2\\
0, \quad \textit{otherwise}
\end{cases}
$$

$$[\mathbf{\Phi}_i]''_{ij} = \begin{cases} \dfrac{\left(\lambda_{i-2}-\lambda_{i-1}\right)^3}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2}, & j=i-1 \\ -\dfrac{\lambda_{i-2}^3+\lambda_{i-2}^2\lambda_{i-1}+\lambda_{i-2}\lambda_{i-1}\left(\lambda_{i-1}-6\lambda_i\right)+3\lambda_{i-1}\lambda_i^2-\lambda_{i-3}\left(4\lambda_{i-2}^2+\lambda_{i-1}^2+3\lambda_i^2-2\lambda_{i-2}\left(4\lambda_{i-1}+3\lambda_i\right)\right)}{30\left(\lambda_{i-3}-\lambda_{i-1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2}, & j=i \\ \dfrac{\lambda_{i-1}\lambda_i^2+\lambda_i^2\left(3\lambda_i-7\lambda_{i+1}\right)+6\lambda_{i-2}^2\left(\lambda_{i-1}-\lambda_{i+1}\right)+\lambda_{i-2}^2\left(\lambda_i+\lambda_{i+1}\right)-}{30\left(\lambda_{i-1}-\lambda_{i+1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2}, & j=i+1 \\ \qquad -2\lambda_{i-2}\left(\lambda_{i-1}^2+\lambda_i\left(2\lambda_i-7\lambda_{i+1}+\lambda_{i-1}\left(3\lambda_i+\lambda_{i+1}\right)\right)\right) & \\ \dfrac{\left(4\lambda_{i-2}-\lambda_{i-1}-3\lambda_i\right)\left(\lambda_{i-1}-\lambda_i\right)^2}{30\left(\lambda_{i-1}-\lambda_{i+1}\right)\left(\lambda_{i-2}-\lambda_i\right)^2}, & j=i+2 \\ 0, \quad \textit{otherwise} & \end{cases}$$

(C.25)

C.1.1.5.2 Multivariate case

As every knot is associated with only one dimension, one has:

$$\mathbf{\Phi}' = \mathbf{\Phi}^{(1)} \otimes \cdots \otimes \mathbf{\Phi}^{(k-1)} \otimes [\mathbf{\Phi}]''^{(k)} \otimes \mathbf{\Phi}^{(k+1)} \cdots \otimes \mathbf{\Phi}^{(n)} \tag{C.26}$$

C.1.2 Computing the terms for multiple sub-models

All calculations carried out in this section refer to the integral of a matrix seen in block form as:

$$\boldsymbol{\varphi}\boldsymbol{\varphi}^T = \begin{bmatrix} \boldsymbol{\varphi}_{\underline{1}}\boldsymbol{\varphi}_{\underline{1}}^T & \boldsymbol{\varphi}_{\underline{1}}\boldsymbol{\varphi}_{\underline{2}}^T & \cdots & \boldsymbol{\varphi}_{\underline{1}}\boldsymbol{\varphi}_{\underline{m}}^T \\ \boldsymbol{\varphi}_{\underline{2}}\boldsymbol{\varphi}_{\underline{1}}^T & \boldsymbol{\varphi}_{\underline{2}}\boldsymbol{\varphi}_{\underline{2}}^T & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\varphi}_{\underline{m}}\boldsymbol{\varphi}_{\underline{1}}^T & \cdots & \cdots & \boldsymbol{\varphi}_{\underline{m}}\boldsymbol{\varphi}_{\underline{m}}^T \end{bmatrix} \tag{C.27}$$

where $\boldsymbol{\varphi}_{\underline{i}}$ denotes the set of spline functions in the i^{th} sub-model.

The subsequent subsections show how to compute the terms corresponding to the elements other than those in the diagonal of the matrix; the equations for the terms corresponding to the elements in the diagonal of the matrix have already been given in the previous sections.

In the following, $\lambda_{i,v,k}$ denotes the k^{th} interior knot from the v^{th} input in the i^{th} sub-model.

Also, assume $\varphi^{(v)}_{i,j}$ to be the j^{th} basis function from the v^{th} input dimension in the i^{th} sub-model.

C.1.2.1 Computation of $\left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}$

As before, one will consider the inverse of $\left[\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}$. Therefore define:

$$\boldsymbol{\Phi} = \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx \tag{C.28}$$

Assume there are m sub-models where $\boldsymbol{\varphi}_{\underline{i}}$ denotes the set of spline functions in the i^{th} submodel. This way, one has:

$$\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx = \begin{bmatrix} \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}_{\underline{1}}\boldsymbol{\varphi}_{\underline{1}}{}^T dx & \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}_{\underline{1}}\boldsymbol{\varphi}_{\underline{2}}{}^T dx & \cdots & \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}_{\underline{1}}\boldsymbol{\varphi}_{\underline{m}}{}^T dx \\ \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}_{\underline{2}}\boldsymbol{\varphi}_{\underline{1}}{}^T dx & \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}_{\underline{2}}\boldsymbol{\varphi}_{\underline{2}}{}^T dx & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}_{\underline{m}}\boldsymbol{\varphi}_{\underline{1}}{}^T dx & \cdots & \cdots & \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}_{\underline{m}}\boldsymbol{\varphi}_{\underline{m}}{}^T dx \end{bmatrix} \tag{C.29}$$

From (C.29), one notices the need to compute the integral of the combination of basis functions within the i^{th} sub-model and the integral of the combination between basis functions from different sub-models. When the latter calculation is required, it is solely the product between basis functions depending on different input variables, hence the computation of the term, $\int_{x_{\underline{i}\,min}}^{x_{\underline{i}MAX}} \boldsymbol{\varphi}_{\underline{i}}\, dx_{\underline{i}}$ in the input variables $x_{\underline{i}}$ is required.

C.1.2.2 Computation of $\int_{x_{\underline{i}\,min}}^{x_{\underline{i}MAX}} \boldsymbol{\varphi}_i dx_i$

C.1.2.2.1 Univariate case

Constant splines (k=1)

$$\int_{x_{v\,min}}^{x_{vMAX}} \varphi^{(v)}{}_{i,j} dx_v = \lambda_{i,v,j} - \lambda_{i,v,j-1} \tag{C.30}$$

Linear splines (k=2)

$$\int_{x_{v\min}}^{x_{vMAX}} \varphi^{(v)}{}_{i,j}\, dx_v = \frac{\lambda_{i,v,j} - \lambda_{i,v,j-2}}{2} \tag{C.31}$$

Quadratic splines (k=3)

$$\int_{x_{v\min}}^{x_{vMAX}} \varphi^{(v)}{}_{i,j}\, dx_v = \frac{\lambda_{i,v,j} - \lambda_{i,v,j-3}}{3} \tag{C.32}$$

C.1.2.2.2 Multivariate case

Consider that if $\otimes$ denotes the tensor product operation, then

$$\int_{\underline{x}_{i\min}}^{\underline{x}_{iMAX}} \boldsymbol{\varphi}_i\, d\underline{x}_i = \int_{x_{i\min}}^{x_{iMAX}} \boldsymbol{\varphi}^{(1)}{}_i dx_1 \otimes \cdots \otimes \int_{x_{v\min}}^{x_{vMAX}} \boldsymbol{\varphi}^{(v)}{}_i dx_v \otimes \cdots \int_{x_{n\min}}^{x_{nMAX}} \boldsymbol{\varphi}^{(n)}{}_i dx_{n_i} \tag{C.33}$$

Assuming n_i input dimensions in the i^{th} submodel.

C.1.2.3 Computation of $\frac{\partial \int_{\underline{x}_{\min}}^{\underline{x}_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\underline{x}}{\partial \mathbf{v}^T}$

The calculus presented in C.1.1.3 is applicable here, except for the terms which depend on the product between basis functions from different sub-models.

In that case, assume:

$$\Phi_{ij} = \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}_{\underline{i}} \boldsymbol{\varphi}_{\underline{j}}{}^T dx \tag{C.34}$$

Then,

$$\frac{\partial \Phi_{ij}}{\partial \lambda_{i,v,k}} = \int_{\underline{x}_{i\min}}^{\underline{x}_{iMAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} d\underline{x}_{\underline{i}} \int_{\underline{x}_{j\min}}^{\underline{x}_{jMAX}} \boldsymbol{\varphi}_{\underline{j}}{}^T d\underline{x}_{\underline{j}} \tag{C.35}$$

Thus, one requires computing the integral of the first term in (C.35), since the second is already given in the previous subsection.

C.1.2.4 Computation of $\int_{\underline{x}_{i\min}}^{\underline{x}_{iMAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} d\underline{x}_{\underline{i}}$

C.1.2.4.1 Univariate case

Constant splines (k=1)

$$\int_{x_{v\min}}^{x_{vMAX}} \frac{\partial \varphi^{(v)}{}_{i,j}}{\partial \lambda_{i,v,k}} dx_v = 0 \tag{C.36}$$

Linear splines (k=2)

$$\int_{x_{v\min}}^{x_{vMAX}} \frac{\partial \varphi^{(v)}{}_{i,j}}{\partial \lambda_{i,v,k}} dx_v = \begin{cases} \frac{1}{2}, k = j \\ 0, \text{ otherwise} \\ -\frac{1}{2}, k = j-2 \end{cases} \tag{C.37}$$

Quadratic splines (k=3)

$$\int_{x_{v\min}}^{x_{vMAX}} \frac{\partial \varphi^{(v)}{}_{i,j}}{\partial \lambda_{i,v,k}} dx_v = \begin{cases} \frac{1}{3}, k = j \\ 0, \text{ otherwise} \\ -\frac{1}{3}, k = j-3 \end{cases} \tag{C.38}$$

C.1.2.4.2 Multivariate case

Consider that If $\otimes$ denotes the tensor product operation, then

$$\int_{x_{\underline{i}\min}}^{x_{\underline{i}MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} dx_{\underline{i}} = \int_{x_{i\min}}^{x_{iMAX}} \boldsymbol{\varphi}^{(1)}{}_i dx_1 \otimes \cdots \otimes \int_{x_{v\min}}^{x_{vMAX}} \frac{\partial \boldsymbol{\varphi}^{(v)}{}_i}{\partial \lambda_{i,v,k}} dx_v \otimes \cdots \int_{x_{n\min}}^{x_{nMAX}} \boldsymbol{\varphi}^{(n)}{}_i dx_{n_i} \tag{C.39}$$

Assuming that the k^{th} interior knot belongs to the v^{th} input dimension. Also, consider n_i input dimensions in the i^{th} submodel.

C.1.2.5 Computation of $\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} dx$

Define

$$\left[\boldsymbol{\Phi}_{km}\right]' = \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} \frac{\partial \boldsymbol{\varphi}_{\underline{j}}{}^T}{\partial \lambda_{j,v,m}} dx \tag{C.40}$$

And because the input variables from the i^{th} and j^{th} sub-models are distinct,

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} \frac{\partial \boldsymbol{\varphi}_{\underline{j}}{}^T}{\partial \lambda_{j,v,m}} dx = \int_{x_{\underline{i}\min}}^{x_{\underline{i}MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} dx_{\underline{i}} \int_{x_{\underline{j}\min}}^{x_{\underline{j}MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{j}}{}^T}{\partial \lambda_{j,v,m}} dx_{\underline{j}} . \tag{C.41}$$

Thus, one requires computing the integral for the term $\int_{x_{\underline{i}\min}}^{x_{\underline{i}MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} dx_{\underline{i}}$ which is given in section C.1.2.4.

C.1.2.6 Computation of $\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T dx$

Define

$$\left[\boldsymbol{\Phi}_k\right]'' = \int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} \boldsymbol{\varphi}_{\underline{j}}^T dx \tag{C.42}$$

As before, since the input variables from the i^{th} and j^{th} sub-models are distinct,

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} \frac{\partial \boldsymbol{\varphi}_{\underline{j}}^T}{\partial \lambda_{j,v,m}} dx = \int_{x_{\underline{i}\min}}^{x_{\underline{i}MAX}} \frac{\partial \boldsymbol{\varphi}_{\underline{i}}}{\partial \lambda_{i,v,k}} dx_{\underline{i}} \int_{x_{\underline{j}\min}}^{x_{\underline{j}MAX}} \boldsymbol{\varphi}_{\underline{j}}^T dx_{\underline{j}} \,. \tag{C.43}$$

Notice that $\boldsymbol{\Phi}''$ is a sparse matrix.

The mathematical expressions required for (C.43) are already computed in sections C.1.2.2 and C.1.2.5.

C.2 Radial Basis Function neural networks

C.2.1 Computation of $\left[\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1}$

Consider

$$\left[\int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T dx\right]^{-1} = \Phi^{-1} \tag{C.44}$$

where Φ (with dimensions $n_u * n_u$) is the integral of the basis functions along the n^{th} dimensional input domain. This is a symmetric matrix with its $[i,j]$ element defined as

$$\Phi_{ij} = \int_{x_{\min}}^{x_{MAX}} \boldsymbol{\varphi}_i \boldsymbol{\varphi}_j dx \tag{C.45}$$

Since the RBF network structure includes a bias term in the output layer, computation of (C.45) requires analysis of three different situations, i.e.:

- if $i, j \neq p$:

In the one dimensional input case,

$$\int_{x_{\min}}^{x_{MAX}} \varphi_i\left(x,\mathbf{v}\right)\varphi_j\left(x,\mathbf{v}\right)dx = \sqrt{\frac{\pi v_i v_j}{2\left(v_i+v_j\right)}}e^{-\frac{\left(c_i-c_j\right)^2}{2\left(v_1+v_2\right)}}\left[\begin{array}{l} erf\left(\frac{x_{MAX}\left(v_i+v_j\right)-c_i v_j-c_j v_i}{\sqrt{2v_i v_j\left(v_i+v_j\right)}}\right)- \\ erf\left(\frac{x_{\min}\left(v_i+v_j\right)-c_i v_j-c_j v_i}{\sqrt{2v_i v_j\left(v_i+v_j\right)}}\right) \end{array}\right] \tag{C.46}$$

Extending to the n^{th} dimensional input case gives

$$\int_{x_{1_{\min}}}^{x_{1_{MAX}}} \varphi_i\left(\mathrm{x},\mathbf{v}_i\right)\varphi_j\left(\mathrm{x},\mathbf{v}_j\right)d\mathrm{x} = \sqrt[n]{\frac{\pi v_i v_j}{2\left(v_i+v_j\right)}}e^{-\frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)}}\left[\begin{array}{l} erf\left(\frac{x_{1_{MAX}}\left(v_i+v_j\right)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_i v_j\left(v_i+v_j\right)}}\right)- \\ erf\left(\frac{x_{1_{\min}}\left(v_i+v_j\right)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_i v_j\left(v_i+v_j\right)}}\right) \end{array}\right]\cdots$$

$$\cdots\left[\begin{array}{l} erf\left(\frac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_i v_j\left(v_i+v_j\right)}}\right)- \\ erf\left(\frac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_i v_j\left(v_i+v_j\right)}}\right) \end{array}\right]\cdots\left[\begin{array}{l} erf\left(\frac{x_{n_{MAX}}\left(v_i+v_j\right)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_i v_j\left(v_i+v_j\right)}}\right)- \\ erf\left(\frac{x_{n_{\min}}\left(v_i+v_j\right)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_i v_j\left(v_i+v_j\right)}}\right) \end{array}\right] \tag{C.47}$$

In the last equation, *erf(.)* represents the error function:

$$erf\left(x\right) = \sqrt{\frac{2}{\pi}}\int_0^x e^{-t^2}dt \tag{C.48}$$

- if $\left(i=p\right)\vee\ \left(j=p\right)$:

In the one dimensional input case,

$$\int_{x_{\min}}^{x_{MAX}} \varphi_i\left(x,\mathbf{v}_i\right)dx = \sqrt{\frac{\pi v_i}{2}}\left[erf\left(\frac{x_{MAX}-c_i}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{\min}-c_i}{\sqrt{2v_i}}\right)\right] \tag{C.49}$$

Extending to the n^{th} dimensional input case

$$\int_{x_{1_{min}}}^{x_{1_{MAX}}} \varphi_i(\mathbf{x},\mathbf{v}_i)\,d\mathbf{x} = \sqrt[n]{\frac{\pi v_i}{2}}\left[erf\left(\frac{x_{1_{MAX}}-c_{i1}}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{1_{min}}-c_{i1}}{\sqrt{2v_i}}\right)\right]\cdots$$
$$\cdots\left[erf\left(\frac{x_{k_{MAX}}-c_{ik}}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{k_{min}}-c_{ik}}{\sqrt{2v_i}}\right)\right]\cdots\left[erf\left(\frac{x_{n_{MAX}}-c_{in}}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{n_{min}}-c_{in}}{\sqrt{2v_i}}\right)\right]$$

(C.50)

- if $i = j = p$:

This case is straightforward, i.e.,

$$\int_{x_{1_{min}}}^{x_{1_{MAX}}} d\mathbf{x} = \left(x_{1_{MAX}}-x_{1_{min}}\right)\ldots\left(x_{k_{MAX}}-x_{k_{min}}\right)\ldots\left(x_{n_{MAX}}-x_{n_{min}}\right) \quad \text{(C.51)}$$

At this point it should be stressed out that with help of no more than the basic properties of multivariate function integration theory, it becomes extremely easy to understand the steps associated with the extension from the *1*th to the *n*th dimensional input (for all previous equations).

C.2.2 Computation of $\frac{\partial \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathbf{x}}{\partial \mathbf{v}^T}$

Applying the same reasoning as in the last subsection:

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}\boldsymbol{\varphi}^T d\mathbf{x}}{2\partial \mathbf{v}^T} = \frac{1}{2}\frac{\partial \Phi}{\partial \mathbf{v}^T} = \frac{1}{2}\Phi' \quad \text{(C.52)}$$

where Φ' is a hyper-matrix of dimensions n_u*n_u*n_v. Its [i,j,k] element is

$$\Phi_{ijk}' = \partial \frac{\int_{x_{min}}^{x_{MAX}} \boldsymbol{\varphi}_i\boldsymbol{\varphi}_j d\mathbf{x}}{\partial \mathbf{v}_k^T} \quad \text{(C.53)}$$

To compute (C.53), the derivatives concerning Eqs. (C.47) and (C.50) are required.

Using the fact that,

$$\frac{\partial\left(erf\left(f(x)\right)\right)}{\partial x} = f'(x)\frac{2}{\sqrt{\pi}}e^{-f^2(x)} \quad \text{(C.54)}$$

and observing the terms in (C.47) and (C.50), the following derivatives are also required:

$$\frac{\partial\left(erf\left(\frac{x_{1_{MAX}}-c_i}{\sqrt{2v_i}}\right)\right)}{\partial c_i}=-\sqrt{\frac{2}{\pi v_i}}e^{-\frac{\left(x_{1_{MAX}}-c_i\right)^2}{2v_i}}$$

$$\frac{\partial\left(erf\left(\frac{x_{1_{MAX}}-c_i}{\sqrt{2v_i}}\right)\right)}{\partial v_i}=\frac{-\left(x_{1_{MAX}}-c_i\right)e^{-\frac{\left(x_{1_{MAX}}-c_i\right)^2}{2v_i}}}{v_i^{3/2}\sqrt{2\pi}}$$

$$\frac{\partial\left(erf\left(\frac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right)}{\partial c_{ik}}=-v_j\frac{\sqrt{\frac{2}{\pi}}e^{-\frac{\left(x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j\left(v_i+v_j\right)}}}{\sqrt{v_iv_j\left(v_i+v_j\right)}}$$

$$\frac{\partial\left(erf\left(\frac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right)}{\partial c_{jk}}=-v_i\frac{\sqrt{\frac{2}{\pi}}e^{-\frac{\left(x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j\left(v_i+v_j\right)}}}{\sqrt{v_iv_j\left(v_i+v_j\right)}}$$

$$\frac{\partial\left(erf\left(\frac{x_{1_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right)}{\partial v_i}=$$

$$=\frac{-v_j^2\left(x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}\left(2v_i+v_j\right)+c_{jk}v_i\right)e^{-\frac{\left(x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j\left(v_i+v_j\right)}}}{\left(v_iv_j\left(v_i+v_j\right)\right)^{3/2}\sqrt{2\pi}}$$

$$\frac{\partial\left(erf\left(\frac{x_{1_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right)}{\partial v_j}=$$

$$=\frac{-v_i^2\left(x_{k_{MAX}}\left(v_i+v_j\right)+c_{ik}v_j-c_{jk}\left(v_i+2v_j\right)\right)e^{-\frac{\left(x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j\left(v_i+v_j\right)}}}{\left(v_iv_j\left(v_i+v_j\right)\right)^{3/2}\sqrt{2\pi}}$$

$$\frac{\partial \sqrt[n]{\frac{\pi v_i}{2}}}{\partial v_i} = 2^{-\left(\frac{n}{2}+1\right)} n\pi^{\frac{n}{2}} v_i^{\frac{n}{2}-1} = \left[\sqrt[n]{\frac{\pi v_i}{2}}\right]\frac{n}{2v_i}$$

$$\frac{\partial \sqrt[n]{\frac{\pi v_i v_j}{2\left(v_i+v_j\right)}}}{\partial v_i} = 2^{-\left(\frac{n}{2}+1\right)} n\pi^{\frac{n}{2}} \left(\frac{v_i v_j}{v_i+v_j}\right)^{\frac{n}{2}+1} = \left[\sqrt[n]{\frac{\pi v_i v_j}{2\left(v_i+v_j\right)}}\right]\frac{nv_j}{2v_i\left(v_i+v_j\right)}$$

$$\frac{\partial \sqrt[n]{\frac{\pi v_i v_j}{2\left(v_i+v_j\right)}}}{\partial v_j} = 2^{-\left(\frac{n}{2}+1\right)} n\pi^{\frac{n}{2}} \left(\frac{v_i v_j}{v_i+v_j}\right)^{\frac{n}{2}+1} = \left[\sqrt[n]{\frac{\pi v_i v_j}{2\left(v_i+v_j\right)}}\right]\frac{nv_i}{2v_j\left(v_i+v_j\right)}$$

$$\begin{aligned}
\frac{\partial e^{-\frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)}}}{\partial c_{ik}} &= -\frac{\left[e^{-\frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)}}\right]\left(c_{ik}-c_{jk}\right)}{v_i+v_j} \\
\frac{\partial e^{-\frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)}}}{\partial c_{jk}} &= \frac{\left[e^{-\frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)}}\right]\left(c_{ik}-c_{jk}\right)}{v_i+v_j} \\
\frac{\partial e^{-\frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)}}}{\partial v_i} &= \frac{\partial e^{-\frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)}}}{\partial v_j} = \frac{\left[e^{-\frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)}}\right]\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2\left(v_i+v_j\right)^2}
\end{aligned} \tag{C.55}$$

Again, the computation of (C.53) requires analysis of three different situations, i.e.,

- if $i, j \neq p$:

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx}{\partial c_{ik}}$$

$$= \sqrt[n]{\frac{\pi v_i v_j}{2(v_i+v_j)}} \frac{\partial e^{-\frac{\|\mathbf{c}_i-\mathbf{c}_j\|_2^2}{2(v_i+v_j)}}}{\partial c_{ik}} \left[erf\left(\frac{x_{1_{MAX}}(v_i+v_j)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{1_{min}}(v_i+v_j)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots$$

$$\cdots \left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots \left[erf\left(\frac{x_{n_{MAX}}(v_i+v_j)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{n_{min}}(v_i+v_j)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] +$$

$$\sqrt[n]{\frac{\pi v_i v_j}{2(v_i+v_j)}} e^{-\frac{\|\mathbf{c}_i-\mathbf{c}_j\|_2^2}{2(v_i+v_j)}} \left[erf\left(\frac{x_{1_{MAX}}(v_i+v_j)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{1_{min}}(v_i+v_j)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots$$

$$\cdots \frac{\partial \left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right]}{\partial c_{ik}} \cdots \left[erf\left(\frac{x_{n_{MAX}}(v_i+v_j)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{n_{min}}(v_i+v_j)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \quad \text{(C.56)}$$

Replacing the derivatives,

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx}{\partial c_{ik}}$$

$$= -\frac{(c_{ik}-c_{jk})}{v_i+v_j}\int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx +$$

$$+\frac{\int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx}{\left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}}\right)-erf\left(\frac{x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}}\right)\right]}\times$$

$$\times\left[\frac{v_j\sqrt{\frac{2}{\pi}}}{\sqrt{v_iv_j(v_i+v_j)}}\left(e^{-\frac{\left(x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}}-e^{-\frac{\left(x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}}\right)\right]=$$

$$=\int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx\times$$

$$\times\left[-\frac{(c_{ik}-c_{jk})}{v_i+v_j}+\right.$$

$$\left.+\frac{v_j\sqrt{\frac{2}{\pi}}}{\sqrt{v_iv_j(v_i+v_j)}}\frac{e^{-\frac{\left(x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}}-e^{-\frac{\left(x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}}}{erf\left(\frac{x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}}\right)-erf\left(\frac{x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}}\right)}\right]$$

(C.57)

Now, for

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \varphi_i(\mathrm{x},\mathbf{v}_i)\varphi_j(\mathrm{x},\mathbf{v}_j)d\mathrm{x}}{\partial c_{jk}}$$

$$= \sqrt[n]{\frac{\pi v_i v_j}{2(v_i+v_j)}} \frac{\partial e^{-\frac{\|\mathbf{c}_i-\mathbf{c}_j\|_2^2}{2(v_i+v_j)}}}{\partial c_{jk}} \left[erf\left(\frac{x_{1_{MAX}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{1_{min}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots$$

$$\cdots \left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{k_{min}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots \left[erf\left(\frac{x_{n_{MAX}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{n_{min}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] +$$

$$\sqrt[n]{\frac{\pi v_i v_j}{2(v_i+v_j)}} e^{-\frac{\|\mathbf{c}_i-\mathbf{c}_j\|_2^2}{2(v_i+v_j)}} \left[erf\left(\frac{x_{1_{MAX}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{1_{min}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots$$

$$\cdots \frac{\partial \left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{k_{min}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right]}{\partial c_{jk}} \cdots \left[erf\left(\frac{x_{n_{MAX}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{n_{min}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right]$$

(C.58)

Resulting in,

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx}{\partial c_{jk}}$$

$$= \frac{(c_{ik}-c_{jk})}{v_i+v_j} \int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx +$$

$$+ \frac{\int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx}{\left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}}\right)-erf\left(\frac{x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}}\right)\right]} \times$$

$$\times\left[\frac{v_i\sqrt{\frac{2}{\pi}}}{\sqrt{v_iv_j(v_i+v_j)}}\left(e^{-\frac{\left(x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}}-e^{-\frac{\left(x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}}\right)\right]=$$

$$= \int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx \times$$

$$\times\left[\frac{(c_{ik}-c_{jk})}{v_i+v_j}+\frac{v_i\sqrt{\frac{2}{\pi}}}{\sqrt{v_iv_j(v_i+v_j)}}\frac{e^{-\frac{\left(x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}}-e^{-\frac{\left(x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}}}{erf\left(\frac{x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}}\right)-erf\left(\frac{x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}}\right)}\right]$$

(C.59)

In the case of the variance for the i^{th} neuron,

$$
\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i(\mathrm{x},\mathbf{v}_i)\varphi_j(\mathrm{x},\mathbf{v}_j)d\mathrm{x}}{\partial v_i}
$$

$$
= \frac{\partial \sqrt[n]{\frac{\pi v_i v_j}{2(v_i+v_j)}}}{\partial v_i} e^{-\frac{\|\mathbf{c}_i-\mathbf{c}_j\|_2^2}{2(v_i+v_j)}} \left[erf\left(\frac{x_{1_{MAX}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{1_{\min}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots
$$

$$
\cdots \left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{k_{\min}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots \left[erf\left(\frac{x_{n_{MAX}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{n_{\min}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] +
$$

$$
+ \sqrt[n]{\frac{\pi v_i v_j}{2(v_i+v_j)}} \frac{\partial e^{-\frac{\|\mathbf{c}_i-\mathbf{c}_j\|_2^2}{2(v_i+v_j)}}}{\partial v_i} \left[erf\left(\frac{x_{1_{MAX}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{1_{\min}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots
$$

$$
\cdots \left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{k_{\min}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots \left[erf\left(\frac{x_{n_{MAX}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{n_{\min}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] +
$$

$$
\sqrt[n]{\frac{\pi v_i v_j}{2(v_i+v_j)}} e^{-\frac{\|\mathbf{c}_i-\mathbf{c}_j\|_2^2}{2(v_i+v_j)}} \frac{\partial \left\{ \left[erf\left(\frac{x_{1_{MAX}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{1_{\min}}(v_i+v_j) - c_{i1}v_j - c_{j1}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots \left[\cdots \left[erf\left(\frac{x_{k_{MAX}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{k_{\min}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \cdots \left[erf\left(\frac{x_{n_{MAX}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - erf\left(\frac{x_{n_{\min}}(v_i+v_j) - c_{in}v_j - c_{jn}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \right] \right\}}{\partial v_i}
$$

(C.60)

and, as before,

$$\frac{\partial\left\{\left[erf\left(\frac{x_{1_{MAX}}\left(v_i+v_j\right)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)-erf\left(\frac{x_{1_{\min}}\left(v_i+v_j\right)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right]\cdots\left[erf\left(\frac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)-erf\left(\frac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right]\cdots\left[erf\left(\frac{x_{n_{MAX}}\left(v_i+v_j\right)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)-erf\left(\frac{x_{n_{\min}}\left(v_i+v_j\right)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right]\right\}}{\partial v_i}=$$

$$= \left\{ \left[\begin{array}{l} erf\left(\dfrac{x_{1_{MAX}}\left(v_i+v_j\right)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{1_{\min}}\left(v_i+v_j\right)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right] \cdots \left[\begin{array}{l} erf\left(\dfrac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right] \cdots \left[\begin{array}{l} erf\left(\dfrac{x_{n_{MAX}}\left(v_i+v_j\right)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{n_{\min}}\left(v_i+v_j\right)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right] \right\} \times$$

$$\sum_{k=1}^{n} \frac{\partial \left[\begin{array}{l} erf\left(\dfrac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right]}{\partial v_i} =$$

$$\left[\begin{array}{l} erf\left(\dfrac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right]$$

$$= \left\{ \left[\begin{array}{l} erf\left(\dfrac{x_{1_{MAX}}\left(v_i+v_j\right)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{1_{\min}}\left(v_i+v_j\right)-c_{i1}v_j-c_{j1}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right] \cdots \left[\begin{array}{l} erf\left(\dfrac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right] \cdots \left[\begin{array}{l} erf\left(\dfrac{x_{n_{MAX}}\left(v_i+v_j\right)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{n_{\min}}\left(v_i+v_j\right)-c_{in}v_j-c_{jn}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right] \right\} \times$$

$$\times \sum_{k=1}^{n} \frac{v_j^2 \left(\begin{array}{l} \left(x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}\left(2v_i+v_j\right)+c_{jk}v_i \right) e^{-\frac{\left(x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j\left(v_i+v_j\right)}} \\ -\left(x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}\left(2v_i+v_j\right)+c_{jk}v_i \right) e^{-\frac{\left(x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j\left(v_i+v_j\right)}} \end{array} \right)}{\left(v_iv_j\left(v_i+v_j\right)\right)^{3/2}\sqrt{2\pi}}}{\left[\begin{array}{l} erf\left(\dfrac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - \\ erf\left(\dfrac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \end{array} \right]}$$

(C.61)

Computing the derivatives,

$$\frac{\partial \int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx}{\partial v_i}$$

$$= \int_{x_{min}}^{x_{MAX}} \varphi_i(x,\mathbf{v}_i)\varphi_j(x,\mathbf{v}_j)dx \times$$

$$\times \left[\frac{nv_j}{2v_i(v_i+v_j)} + \frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2(v_i+v_j)^2} + \right.$$

$$\left. + \sum_{k=1}^{n} \frac{v_j^2 \left(\begin{array}{l} \left(x_{k_{min}}(v_i+v_j) - c_{ik}(2v_i+v_j) + c_{jk}v_i\right) e^{-\frac{\left(x_{k_{min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}} \\ -\left(x_{k_{MAX}}(v_i+v_j) - c_{ik}(2v_i+v_j) + c_{jk}v_i\right) e^{-\frac{\left(x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}} \end{array} \right)}{\left(v_iv_j(v_i+v_j)\right)^{3/2}\sqrt{2\pi}} \right.$$

$$\left. \frac{}{\left[\begin{array}{l} erf\left(\frac{x_{k_{MAX}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - \\ erf\left(\frac{x_{k_{min}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \end{array} \right]} \right] \quad \text{(C.62)}$$

Using the same reasoning for v_j,

$$\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i(\mathrm{x},\mathbf{v}_i)\varphi_j(\mathrm{x},\mathbf{v}_j)d\mathrm{x}}{\partial v_j} = \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i(\mathrm{x},\mathbf{v}_i)\varphi_j(\mathrm{x},\mathbf{v}_j)d\mathrm{x} \times$$

$$\times \left[\frac{nv_i}{2v_j(v_i+v_j)} + \frac{\left\|\mathbf{c}_i-\mathbf{c}_j\right\|_2^2}{2(v_i+v_j)^2} + \right.$$

$$\left. + \sum_{k=1}^{n} \frac{v_i^2 \left(\begin{array}{l} \left(x_{k_{\min}}(v_i+v_j) - c_{jk}(2v_j+v_i) + c_{ik}v_j\right) e^{-\frac{\left(x_{k_{\min}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}} \\ -\left(x_{k_{MAX}}(v_i+v_j) - c_{jk}(2v_j+v_i) + c_{ik}v_j\right) e^{-\frac{\left(x_{k_{MAX}}(v_i+v_j)-c_{ik}v_j-c_{jk}v_i\right)^2}{2v_iv_j(v_i+v_j)}} \end{array} \right)}{\left(v_iv_j(v_i+v_j)\right)^{3/2}\sqrt{2\pi}} \right.$$

$$\left. \frac{}{\left[\begin{array}{l} erf\left(\frac{x_{k_{MAX}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) - \\ erf\left(\frac{x_{k_{\min}}(v_i+v_j) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_iv_j(v_i+v_j)}} \right) \end{array} \right]} \right] \qquad \text{(C.63)}$$

Remark:

Notice that in all previous cases:

$$\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i^2(\mathrm{x},\mathbf{v}_i)d\mathrm{x}}{\partial \mathbf{v}} = 2\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i(\mathrm{x},\mathbf{v}_i)\varphi_j(\mathrm{x},\mathbf{v}_j)d\mathrm{x}}{\partial \mathbf{v}}\Bigg|_{i\neq j}$$

- if $(i=p)\vee\ (j=p)$:

$$\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i\left(\mathrm{x}, \mathbf{v}_i\right) d\mathrm{x}}{\partial c_{ik}}$$

$$= \sqrt[n]{\frac{\pi v_i}{2}} \left[erf\left(\frac{x_{1_{MAX}} - c_{i1}}{\sqrt{2v_i}} \right) - erf\left(\frac{x_{1_{\min}} - c_{i1}}{\sqrt{2v_i}} \right) \right] \cdots \frac{\partial \left[erf\left(\frac{x_{k_{MAX}} - c_{ik}}{\sqrt{2v_i}} \right) - erf\left(\frac{x_{k_{\min}} - c_{ik}}{\sqrt{2v_i}} \right) \right]}{\partial c_{i,k}} \cdots$$

$$\cdots \left[erf\left(\frac{x_{n_{MAX}} - c_{in}}{\sqrt{2v_i}} \right) - erf\left(\frac{x_{n_{\min}} - c_{in}}{\sqrt{2v_i}} \right) \right] = \tag{C.64}$$

$$= \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i\left(\mathrm{x}, \mathbf{v}_i\right) d\mathrm{x}}{\left[erf\left(\frac{x_{k_{MAX}} - c_{ik}}{\sqrt{2v_i}} \right) - erf\left(\frac{x_{k_{\min}} - c_{ik}}{\sqrt{2v_i}} \right) \right]} \left[\sqrt{\frac{2}{\pi v_i}} \left(e^{-\frac{\left(x_{k_{\min}} - c_{ik}\right)^2}{2v_i}} - e^{-\frac{\left(x_{k_{MAX}} - c_{ik}\right)^2}{2v_i}} \right) \right]$$

$$
\begin{aligned}
&\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i(\mathrm{x},\mathbf{v}_i)\,d\mathrm{x}}{\partial v_i} \\
&= \frac{\partial^n \sqrt{\frac{\pi v_i}{2}}}{\partial v_i}\left[erf\left(\frac{x_{1_{MAX}}-c_{i1}}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{1_{\min}}-c_{i1}}{\sqrt{2v_i}}\right)\right]\cdots\left[erf\left(\frac{x_{k_{MAX}}-c_{ik}}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{k_{\min}}-c_{ik}}{\sqrt{2v_i}}\right)\right]\cdots \\
&\cdots\left[erf\left(\frac{x_{n_{MAX}}-c_{in}}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{n_{\min}}-c_{in}}{\sqrt{2v_i}}\right)\right]+\sum_{j=1}^{n}\left\{\frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\varphi_i(\mathrm{x},\mathbf{v}_i)\,d\mathrm{x}}{\left[erf\left(\frac{x_{j_{MAX}}-c_{ij}}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{j_{\min}}-c_{ij}}{\sqrt{2v_i}}\right)\right]}\times\right. \\
&\left.\times\frac{\partial\left[erf\left(\frac{x_{j_{MAX}}-c_{ij}}{\sqrt{2v_i}}\right)-erf\left(\frac{x_{j_{\min}}-c_{ij}}{\sqrt{2v_i}}\right)\right]}{\partial v_i}\right\} \\
&=\frac{n}{2v_i}\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\varphi_i(\mathrm{x},\mathbf{v}_i)\,d\mathrm{x}+ \\
&+\sum_{j=1}^{n}\left\{\frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\varphi_i(\mathrm{x},\mathbf{v}_i)\,d\mathrm{x}}{\left[erf\left(\frac{x_{j_{MAX}}-c_{jk}}{\sqrt{2v_j}}\right)-erf\left(\frac{x_{j_{\min}}-c_{jk}}{\sqrt{2v_j}}\right)\right]}\left[\frac{\left(x_{j_{\min}}-c_{ij}\right)e^{-\frac{\left(x_{j\min}-c_{ij}\right)^2}{2v_i}}-\left(x_{j_{MAX}}-c_{ij}\right)e^{-\frac{\left(x_{jMAX}-c_{ij}\right)^2}{2v_i}}}{v_i^{3/2}\sqrt{2\pi}}\right]\right\}= \\
&=\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}}\varphi_i(\mathrm{x},\mathbf{v}_i)\,d\mathrm{x}\left[\begin{array}{l}\frac{n}{2v_i}+ \\ +\frac{1}{v_i^{3/2}\sqrt{2\pi}}\sum_{j=1}^{n}\left\{\frac{\left(x_{j_{\min}}-c_{ij}\right)e^{-\frac{\left(x_{j\min}-c_{ij}\right)^2}{2v_i}}-\left(x_{j_{MAX}}-c_{ij}\right)e^{-\frac{\left(x_{jMAX}-c_{ij}\right)^2}{2v_i}}}{\left[erf\left(\frac{x_{j_{MAX}}-c_{jk}}{\sqrt{2v_j}}\right)-erf\left(\frac{x_{j_{\min}}-c_{jk}}{\sqrt{2v_j}}\right)\right]}\right\}\end{array}\right]
\end{aligned}
\tag{C.65}
$$

Obviously,

$$
\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i(\mathrm{x},\mathbf{v}_i)\,d\mathrm{x}}{\partial \beta}=0,
$$

$$
\forall k, \beta=\left\{\mathbf{v}_j,\mathbf{c}_{jk}\right\}, i\neq j
$$

- if $i=j=p$:

$$\frac{\partial \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i(\mathrm{x}, \mathbf{v}_i)\, d\mathrm{x}}{\partial \beta} = 0, \forall k, \forall i, \beta = \{\mathbf{v}_i, \mathbf{c}_{ik}\} \tag{C.66}$$

C.2.3 Computation of $\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} d\mathrm{x}$

To solve Eq. (C.3) start by setting that

$$\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \frac{\partial \boldsymbol{\varphi}^T}{\partial \mathbf{v}^T} d\mathrm{x} = \Phi'' \tag{C.67}$$

where Φ'' is a sparse matrix, with size $n_v * n_u * n_u * n_v$, where only their [*a,i,j,b*] elements,, $\boldsymbol{\Phi}''_{aijb}$, are non-zero provided that

$$\mathbf{v}_i \in \{c_{ik}, v_i\}, \mathbf{v}_j \in \{c_{jk}, v_j\}, k = 1 \ldots n \tag{C.68}$$

Considering the RBF nonlinear parameters, Eq. (C.67) requires the computation of:

$$\begin{cases} \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial \varphi_i(\mathrm{x}, \mathbf{v}_i)}{\partial c_{ik}} . \frac{\partial \varphi_j(\mathrm{x}, \mathbf{v}_j)}{\partial c_{jm}} d\mathrm{x} \\ \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial \varphi_i(\mathrm{x}, \mathbf{v}_i)}{\partial c_{ik}} . \frac{\partial \varphi_j(\mathrm{x}, \mathbf{v}_j)}{\partial v_j} d\mathrm{x} \\ \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial \varphi_i(\mathrm{x}, \mathbf{v}_i)}{\partial v_i} . \frac{\partial \varphi_j(\mathrm{x}, \mathbf{v}_j)}{\partial v_j} d\mathrm{x} \end{cases} \tag{C.69}$$

To move on with the computation, first compute the derivatives of basis function *i* in respect to parameters c_{ik} and v_i,

$$\begin{aligned} \frac{\partial \varphi_i(\mathrm{x}, \mathbf{v}_i)}{\partial c_{ik}} &= \frac{x_k - c_{ik}}{v_i} . \varphi_i(\mathrm{x}, \mathbf{v}_i) \\ \frac{\partial \varphi_i(\mathrm{x}, \mathbf{v}_i)}{\partial v_i} &= \sum_{j=1}^{n} \frac{(x_j - c_{ij})^2}{2 v_i^2} . \varphi_i(\mathrm{x}, \mathbf{v}_i) \end{aligned} \tag{C.70}$$

Basically, replacing (C.70) into (C.69) provides the necessary terms to solve (C.67), i.e.,

$$\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial \varphi_i(\mathrm{x}, \mathbf{v}_i)}{\partial c_{ik}} \frac{\partial \varphi_j(\mathrm{x}, \mathbf{v}_j)}{\partial c_{jm}} d\mathrm{x} = \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \left(\frac{x_k - c_{ik}}{v_i} \frac{x_m - c_{jm}}{v_j} \right) . \varphi_i(\mathrm{x}, \mathbf{v}_i) \varphi_j(\mathrm{x}, \mathbf{v}_j) d\mathrm{x} \tag{C.71}$$

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \varphi_i(\mathrm{x},\mathbf{v}_i)}{\partial v_i}\frac{\partial \varphi_j(\mathrm{x},\mathbf{v}_j)}{\partial v_j}d\mathrm{x} = \int_{x_{\min}}^{x_{MAX}} \sum_{k=1}^{n}\frac{(x_k - c_{ik})^2}{2v_i^{\,2}}\sum_{m=1}^{n}\frac{(x_m - c_{jm})^2}{2v_j^{\,2}}\varphi_i(\mathrm{x},\mathbf{v}_i)\varphi_j(\mathrm{x},\mathbf{v}_j)d\mathrm{x} \tag{C.72}$$

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \varphi_i(\mathrm{x},\mathbf{v}_i)}{\partial c_{ik}}\frac{\partial \varphi_j(\mathrm{x},\mathbf{v}_j)}{\partial v_j}d\mathrm{x} = \int_{x_{\min}}^{x_{MAX}} \left(\frac{x_k - c_{ik}}{v_i}\sum_{m=1}^{n}\frac{(x_m - c_{jm})^2}{2v_j^{\,2}}\right).\varphi_i(\mathrm{x},\mathbf{v}_i)\varphi_j(\mathrm{x},\mathbf{v}_j)d\mathrm{x} \tag{C.73}$$

The computation of the integrals in equations (C.71)-(C.73) relies on two approaches, respectively if k=m and k≠m.

C.2.3.1 Integral of equation (C.71)

- $k \neq m$:

Integration of (C.71) can be explicitly expressed as a function of the k^{th} and m^{th} dimensions, i.e.,

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \varphi_i(\mathrm{x},\mathbf{v}_j)}{\partial c_{ik}}\frac{\partial \varphi_j(\mathrm{x},\mathbf{v}_j)}{\partial c_{jm}}d\mathrm{x} = \int_{x_{k\min}}^{x_{kMAX}} \left(\frac{x_k - c_{ik}}{v_i}\right) e^{-\frac{(x_k - \mathbf{c}_{ik})^2}{2v_i} - \frac{(x_k - \mathbf{c}_{jk})^2}{2v_j}} dx_k \int_{x_{m\min}}^{x_{mMAX}} \left(\frac{x_m - c_{jm}}{v_j}\right) e^{-\frac{(x_m - \mathbf{c}_{im})^2}{2v_i} - \frac{(x_m - \mathbf{c}_{jm})^2}{2v_j}} dx_m \times \tag{C.74}$$

$$\times \int_{x_{\min}}^{x_{MAX}} \varphi_i\varphi_j(x_1,..,x_{k-1},x_{k+1},..,x_{m-1},x_{m+1},..,x_n,\mathbf{v}_i)dx$$

The integral of the product between φ_i and φ_j in(C.74), can be expressed as a function of the integral in equation (C.47) if the corresponding expansion is absent of the terms related to the k^{th} and m^{th} dimensions, as

$$\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial \varphi_i\left(\mathrm{x},\mathbf{v}_j\right)}{\partial c_{ik}} \frac{\partial \varphi_j\left(\mathrm{x},\mathbf{v}_j\right)}{\partial c_{jm}} d\mathrm{x}$$

$$= \int_{x_{k\min}}^{x_{kMAX}} \left(\frac{x_k - c_{ik}}{v_i}\right) e^{-\frac{\left(x_k-\mathbf{c}_{ik}\right)^2}{2v_i}-\frac{\left(x_k-\mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \int_{x_{m\min}}^{x_{mMAX}} \left(\frac{x_m - c_{jm}}{v_j}\right) e^{-\frac{\left(x_m-\mathbf{c}_{im}\right)^2}{2v_i}-\frac{\left(x_m-\mathbf{c}_{jm}\right)^2}{2v_j}} dx_m \times$$

$$\times \frac{2\left(v_i+v_j\right)}{\pi v_i v_j} \times e^{\frac{\left(\mathbf{c}_{ik}-\mathbf{c}_{jk}\right)^2+\left(\mathbf{c}_{im}-\mathbf{c}_{jm}\right)^2}{2\left(v_i+v_j\right)}} \times$$

$$\times \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i\varphi_j\left(\mathrm{x},\mathbf{v}_i\right) d\mathrm{x}}{\left[erf\left(\frac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)-erf\left(\frac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right]\left[erf\left(\frac{x_{m_{MAX}}\left(v_i+v_j\right)-c_{im}v_j-c_{jm}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)-erf\left(\frac{x_{m_{\min}}\left(v_i+v_j\right)-c_{im}v_j-c_{jm}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right]}$$

(C.75)

where,

$$\int_{x_{k\min}}^{x_{kMAX}} \left(\frac{x_k - c_{ik}}{v_i}\right) e^{-\frac{\left(x_k-\mathbf{c}_{ik}\right)^2}{2v_i}-\frac{\left(x_k-\mathbf{c}_{jk}\right)^2}{2v_j}} dx_k = \frac{v_j}{v_i+v_j}\left(e^{-\frac{\mathbf{c}_{jk}{}^2v_i+\mathbf{c}_{ik}{}^2v_j-2\left(\mathbf{c}_{jk}v_i+\mathbf{c}_{ik}v_j\right)x_{k\min}+\left(v_i+v_j\right)x_{k\min}{}^2}{2v_iv_j}}-e^{-\frac{\mathbf{c}_{jk}{}^2v_i+\mathbf{c}_{ik}{}^2v_j-2\left(\mathbf{c}_{jk}v_i+\mathbf{c}_{ik}v_j\right)x_{kMAX}+\left(v_i+v_j\right)x_{kMAX}{}^2}{2v_iv_j}}\right)-$$

$$-\frac{\left(c_{ik}-c_{jk}\right)\sqrt{\pi v_iv_j}}{\sqrt{2}\left(v_i+v_j\right)^{3/2}} e^{-\frac{\left(x_{ik}-\mathbf{c}_{jk}\right)^2}{2\left(v_i+v_j\right)}}\left[erf\left(\frac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)-erf\left(\frac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right]$$

(C.76)

and

$$\int_{x_{k\min}}^{x_{kMAX}} \left(\frac{x_m - c_{im}}{v_j} \right) e^{-\frac{(x_m - \mathbf{c}_{im})^2}{2v_i} - \frac{(x_m - \mathbf{c}_{jm})^2}{2v_j}} dx_m = \frac{v_i}{v_i + v_j} \left(e^{-\frac{\mathbf{c}_{im}{}^2 v_j + \mathbf{c}_{jm}{}^2 v_i - 2(\mathbf{c}_{im} v_k + \mathbf{c}_{jm} v_i) x_{m\min} + (v_i + v_j) x_{\min}{}^2}{2 v_i v_j}} - e^{-\frac{\mathbf{c}_{im}{}^2 v_j + \mathbf{c}_{jm}{}^2 v_i - 2(\mathbf{c}_{im} v_k + \mathbf{c}_{jm} v_i) x_{mMAX} + (v_i + v_j) x_{mMAX}{}^2}{2 v_i v_j}} \right) -$$

$$- \frac{(c_{jm} - c_{im}) \sqrt{\pi v_i v_j}}{\sqrt{2} (v_i + v_j)^{3/2}} e^{-\frac{(x_{jm} - \mathbf{c}_{im})^2}{2(v_i + v_j)}} \left[erf\left(\frac{x_{m_{MAX}} (v_i + v_j) - c_{jm} v_i - c_{im} v_j}{\sqrt{2 v_i v_j (v_i + v_j)}} \right) - erf\left(\frac{x_{m_{\min}} (v_i + v_j) - c_{jm} v_i - c_{im} v_j}{\sqrt{2 v_i v_j (v_i + v_j)}} \right) \right]$$

(C.77)

- $m = k$:

Now, the integration in (C.71) is explicitly expressed as a function of the k^{th} dimension only, i.e.,

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \varphi_i(\mathrm{x}, \mathbf{v}_j)}{\partial c_{ik}} \frac{\partial \varphi_j(\mathrm{x}, \mathbf{v}_j)}{\partial c_{jm}} d\mathrm{x}$$

$$= \int_{x_{k\min}}^{x_{kMAX}} \left(\frac{x_k - c_{ik}}{v_i} \right) \left(\frac{x_k - c_{jk}}{v_j} \right) e^{-\frac{(x_k - \mathbf{c}_{ik})^2}{2v_i} - \frac{(x_k - \mathbf{c}_{jk})^2}{2v_j}} dx_k \times \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j (x_1, .., x_{k-1}, x_{k+1}, .., x_n, \mathbf{v}_i) dx \quad . \quad \text{(C.78)}$$

Applying similar reasoning as before, this integral can be expressed as a function of the integral in equation (C.47) if the corresponding expansion is absent of the terms related to the k^{th} dimension,

$$\int_{x_{\min}}^{x_{MAX}} \frac{\partial \varphi_i(\mathrm{x}, \mathbf{v}_j)}{\partial c_{ik}} \frac{\partial \varphi_j(\mathrm{x}, \mathbf{v}_j)}{\partial c_{jm}} d\mathrm{x}$$

$$= \int_{x_{k\min}}^{x_{kMAX}} \left(\frac{x_k - c_{ik}}{v_i} \right) \left(\frac{x_k - c_{jk}}{v_j} \right) e^{-\frac{(x_k - \mathbf{c}_{ik})^2}{2v_i} - \frac{(x_k - \mathbf{c}_{jk})^2}{2v_j}} dx_k \times \sqrt{\frac{2(v_i + v_j)}{\pi v_i v_j}} \times e^{\frac{(\mathbf{c}_{ik} - \mathbf{c}_{jk})^2}{2(v_i + v_j)}} \times \quad \text{(C.79)}$$

$$\times \frac{\int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j (\mathrm{x}, \mathbf{v}_i) d\mathrm{x}}{\left[erf\left(\frac{x_{k_{MAX}} (v_i + v_j) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j (v_i + v_j)}} \right) - erf\left(\frac{x_{k_{\min}} (v_i + v_j) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j (v_i + v_j)}} \right) \right]}$$

where,

$$\int_{x_{k\min}}^{x_{kMAX}} \left(\frac{x_k - c_{ik}}{v_i}\right)\left(\frac{x_k - c_{jk}}{v_j}\right) e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k$$

$$= \frac{\left(v_i c_{ik} + v_j c_{jk} - \left(v_i + v_j\right) x_{kMAX}\right) e^{-\frac{v_i {c_{jk}}^2 + v_j {c_{ik}}^2 - 2\left(v_i c_{jk} + v_j c_{ik}\right) x_{kMAX} + \left(v_i + v_j\right) {x_{kMAX}}^2}{2v_i v_j}} - \left(v_i c_{ik} + v_j c_{jk} - \left(v_i + v_j\right) x_{k\min}\right) e^{-\frac{v_i {c_{jk}}^2 + v_j {c_{ik}}^2 - 2\left(v_i c_{jk} + v_j c_{ik}\right) x_{k\min} + \left(v_i + v_j\right) {x_{k\min}}^2}{2v_i v_j}}}{\left(v_i + v_j\right)^2} -$$

$$- \frac{\sqrt{\pi v_i v_j}}{\sqrt{2}\left(v_i + v_j\right)^{5/2}} \left({c_{ik}}^2 - 2c_{ik}c_{jk} + {c_{jk}}^2 - \left(v_i + v_j\right)\right) e^{-\frac{\left(c_{ik} - c_{jk}\right)^2}{2\left(v_i + v_j\right)}} \times$$

$$\times \left[erf\left(\frac{x_{k_{MAX}}\left(v_i + v_j\right) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}}\right) - erf\left(\frac{x_{k_{\min}}\left(v_i + v_j\right) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}}\right)\right] \tag{C.80}$$

C.2.3.2 Integral of equation (C.72)

A rather more complicated situation arises due to the product between two sums. However a simplified solution to the integral can be expressed in terms of the product between sums in the same input variables and distinct input variables, as shown below.

$$\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial \varphi_i\left(\mathrm{x}, \mathbf{v}_j\right)}{\partial v_i} \frac{\partial \varphi_j\left(\mathrm{x}, \mathbf{v}_j\right)}{\partial v_j} d\mathrm{x}$$

$$\left[\sum_{k=1}^{n} \int_{x_{\min}}^{x_{MAX}} \frac{\left(x_k - c_{ik}\right)^2 \left(x_k - c_{jk}\right)^2}{4{v_i}^2 {v_j}^2} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j\left(x_1, .., x_{k-1}, x_{k+1}, .., x_n, \mathbf{v}_i\right) dx + \right.$$

$$+ \sum_{k=1}^{n} \int_{x_{\min}}^{x_{MAX}} \frac{\left(x_k - c_{ik}\right)^2}{2{v_i}^2} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \sum_{m=1}^{n} \int_{x_{\min}}^{x_{MAX}} \frac{\left(x_m - c_{jm}\right)^2}{2{v_j}^2} e^{-\frac{\left(x_m - \mathbf{c}_{im}\right)^2}{2v_i} - \frac{\left(x_m - \mathbf{c}_{jm}\right)^2}{2v_j}} dx_m \times$$

$$\left. \times \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j\left(., x_{k-1}, x_{k+1}, .., x_{m-1}, x_{m+1}, ., \mathbf{v}_i\right) dx \right|_{k \neq m} \Bigg]$$

(C.81)

Picking up the terms in the same input variable,

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)^2 \left(x_k - c_{jk}\right)^2}{4{v_i}^2 {v_j}^2} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j\left(x_1, .., x_{k-1}, x_{k+1}, .., x_n, \mathbf{v}_i\right) dx$$

(C.82)

$$= \int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)^2 \left(x_k - c_{jk}\right)^2}{4v_i^{\,2} v_j^{\,2}} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \times \sqrt{\frac{2\left(v_i + v_j\right)}{\pi v_i v_j}} \times e^{\frac{\left(\mathbf{c}_{ik} - \mathbf{c}_{jk}\right)^2}{2\left(v_i + v_j\right)}}$$

$$\times \frac{\int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j \left(x_1, .., x_k, .., x_n, \mathbf{v}_i\right) dx}{\left[erf\left(\frac{x_{k_{MAX}} \left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) - erf\left(\frac{x_{k_{\min}} \left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) \right]} \tag{C.83}$$

where,

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)^2 \left(x_k - c_{jk}\right)^2}{4v_i^{\,2} v_j^{\,2}} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k =$$

$$= \frac{1}{8 v_i v_j \left(v_i + v_j\right)^{9/2}} \left[\begin{array}{l} 2\left(v_i + v_j\right)^{1/2} \times \\ \times \left[\begin{array}{l} e^{-\frac{v_i c_{jk}^{\,2} + v_j c_{ik}^{\,2} - 2\left(v_i c_{jk} + v_j c_{ik}\right) x_{k\min} + \left(v_i + v_j\right) x_{k\min}^{\,2}}{2 v_i v_j}} \times \\ \times \left(\begin{array}{l} C_{ik}^{\,3} v_i^{\,2} v_j + C_{jk}^{\,3} v_j^{\,2} v_i + \left(C_{jk}^{\,2} v_j^{\,2} + C_{ik}^{\,2} v_i^{\,2}\right)\left(v_i + v_j\right) x_{k\min} - C_{jk}\left(v_i + 4v_j\right) C_{ik}^{\,2} v_i^{\,2} + \\ \left(v_i + v_j\right)^2 x_{k\min}\left(\left(v_i + v_j\right) x_{k\min}^{\,2} + 3 v_i v_j\right) + \\ + C_{jk}\left(v_i + v_j\right)\left(-2 v_j^{\,2} x_{k\min}^{\,2} + v_i^{\,2}\left(v_j - x_{k\min}^{\,2}\right) - v_i v_j\left(4v_j + 3 x_{k\min}^{\,2}\right)\right) - \\ C_{ik}\left(\begin{array}{l} v_j^{\,3}\left(C_{jk}^{\,2} - 2 x_{k\min} C_{jk} + x_{k\min}^{\,2}\right) - v_i v_j^{\,2}\left(-4 C_{jk}^{\,2} + v_j + 8 C_{jk} x_{k\min} - 4 x_{k\min}^{\,2}\right) + \\ + 2 v_i^{\,3}\left(2 v_j - C_{jk} x_{k\min} + x_{k\min}^{\,2}\right) + v_i^{\,2} v_j\left(3 v_j - 8 C_{jk} x_{k\min} + 5 x_{k\min}^{\,2}\right) \end{array} \right) \end{array} \right) - \\ \times \left[\begin{array}{l} e^{-\frac{v_i c_{jk}^{\,2} + v_j c_{ik}^{\,2} - 2\left(v_i c_{jk} + v_j c_{ik}\right) x_{kMAX} + \left(v_i + v_j\right) x_{kMAX}^{\,2}}{2 v_i v_j}} \times \\ \times \left(\begin{array}{l} C_{ik}^{\,3} v_i^{\,2} v_j + C_{jk}^{\,3} v_j^{\,2} v_i + \left(C_{jk}^{\,2} v_j^{\,2} + C_{ik}^{\,2} v_i^{\,2}\right)\left(v_i + v_j\right) x_{kMAX} - C_{jk}\left(v_i + 4v_j\right) C_{ik}^{\,2} v_i^{\,2} + \\ \left(v_i + v_j\right)^2 x_{kMAX}\left(\left(v_i + v_j\right) x_{kMAX}^{\,2} + 3 v_i v_j\right) + \\ + C_{jk}\left(v_i + v_j\right)\left(-2 v_j^{\,2} x_{kMAX}^{\,2} + v_i^{\,2}\left(v_j - x_{kMAX}^{\,2}\right) - v_i v_j\left(4v_j + 3 x_{kMAX}^{\,2}\right)\right) - \\ C_{ik}\left(\begin{array}{l} v_j^{\,3}\left(C_{jk}^{\,2} - 2 x_{kMAX} C_{jk} + x_{kMAX}^{\,2}\right) - v_i v_j^{\,2}\left(-4 C_{jk}^{\,2} + v_j + 8 C_{jk} x_{kMAX} - 4 x_{kMAX}^{\,2}\right) + \\ + 2 v_i^{\,3}\left(2 v_j - C_{jk} x_{kMAX} + x_{kMAX}^{\,2}\right) + v_i^{\,2} v_j\left(3 v_j - 8 C_{jk} x_{kMAX} + 5 x_{kMAX}^{\,2}\right) \end{array} \right) \end{array} \right) + \\ + \sqrt{2\pi v_i v_j}\, e^{-\frac{\left(C_{ik} - c_{jk}\right)^2}{2\left(v_i + v_j\right)}} \times \left[erf\left(\frac{x_{k_{\min}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) - erf\left(\frac{x_{kMAX}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) \right] \times \\ \times \left(\begin{array}{l} C_{ik}^{\,4} v_i v_j - 4 C_{ik}^{\,3} C_{jk} v_j v_i + C_{jk}^{\,4} v_i v_j + 3 v_i v_j\left(v_i + v_j\right)^2 - 2 C_{ik} C_{jk}\left(v_i^{\,3} - 3 v_i^{\,2} v_j + v_i\left(2 C_{jk}^{\,2} - 3 v_j\right) v_j + v_j^{\,3}\right) + \\ + C_{ik}^{\,2}\left(v_i^{\,3} - 3 v_i^{\,2} v_j + 3 v_i\left(2 C_{jk}^{\,2} - v_j\right) v_j + v_j^{\,3}\right) + C_{jk}^{\,2}\left(v_i^{\,3} - 3 v_i^{\,2} v_j - 3 v_i v_j^{\,2} + v_j^{\,3}\right) \end{array} \right) \end{array} \right] \tag{C.84}$$

Next, investigate the terms with different input variables. The required integral to compute is

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} e^{-\frac{\left(x_k-\mathbf{c}_{ik}\right)^2}{2v_i}-\frac{\left(x_k-\mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \int_{x_{m\min}}^{x_{mMAX}} \frac{\left(x_m - c_{jm}\right)^2}{2v_j^{\,2}} e^{-\frac{\left(x_m-\mathbf{c}_{im}\right)^2}{2v_i}-\frac{\left(x_m-\mathbf{c}_{jm}\right)^2}{2v_j}} dx_m \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i\varphi_j\left(\mathrm{x},\mathbf{v}_i\right) d\mathrm{x}$$

(C.85)

where,

$$\int_{x_{m\min}}^{x_{mMAX}} \frac{\left(x_m - c_{im}\right)^2}{2v_i^{\,2}} e^{-\frac{\left(x_m-\mathbf{c}_{im}\right)^2}{2v_i}-\frac{\left(x_m-\mathbf{c}_{jm}\right)^2}{2v_j}} dx_m$$

$$= \frac{1}{2v_i^{\,2}} \times$$

$$\times \left[\begin{array}{l} \left(\begin{array}{l} \frac{v_i v_j}{\left(v_i+v_j\right)^2}\left(c_{jm}v_i - c_{im}\left(2v_i+v_j\right)+x_{m\min}\left(v_i+v_j\right)\right) e^{-\frac{c_{jm}^{\,2}v_i+c_{im}^{\,2}v_j-2x_{m\min}\left(c_{jm}v_i+c_{im}v_j\right)+\left(v_i+v_j\right)x_{m\min}^{\,2}}{2v_iv_j}} - \\ -\frac{v_i v_j}{\left(v_i+v_j\right)^2}\left(c_{jm}v_i - c_{im}\left(2v_i+v_j\right)+x_{mMAX}\left(v_i+v_j\right)\right) e^{-\frac{c_{jm}^{\,2}v_i+c_{im}^{\,2}v_j-2x_{mMAX}\left(c_{jm}v_i+c_{im}v_j\right)+\left(v_i+v_j\right)x_{mMAX}^{\,2}}{2v_iv_j}} \end{array} \right) + \\ +\sqrt{\frac{\pi v_j}{2}}\frac{v_i^{\,3/2}}{\left(v_i+v_j\right)^{5/2}}\left(c_{im}^{\,2}v_i - 2c_{im}c_{jm}v_i + c_{jm}^{\,2}v_i + v_j\left(v_i+v_j\right)\right) e^{-\frac{\left(c_{im}-c_{jm}\right)^2}{2\left(v_i+v_j\right)}} \times \\ \left(erf\left(\frac{x_{m_{MAX}}\left(v_i+v_j\right)-c_{im}v_j-c_{jm}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) - erf\left(\frac{x_{m_{\min}}\left(v_i+v_j\right)-c_{im}v_j-c_{jm}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}} \right) \right) \end{array} \right]$$

(C.86)

C.2.3.3 Integral of equation (C.73)

This integral can be expressed as a function of the integral in equation (C.45) if the corresponding expansion is independent of the terms related to the k^{th} and m^{th} dimensions, i.e.,

$$\begin{aligned} &\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \frac{\partial\varphi_i\left(\mathrm{x},\mathbf{v}_j\right)}{\partial c_{ik}}\frac{\partial\varphi_j\left(\mathrm{x},\mathbf{v}_j\right)}{\partial v_j} d\mathrm{x} \\ &= \int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{2v_j^{\,2}v_i}\left(x_k - c_{jk}\right)^2 dx_k \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i\varphi_j\left(\mathrm{x},\mathbf{v}_i\right) d\mathrm{x} + \\ &+ \sum_{\substack{m=1\\k\neq m}}^{n} \int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{v_i} dx_k \int_{x_{m\min}}^{x_{mMAX}} \frac{\left(x_m - c_{jm}\right)^2}{2v_j^{\,2}} dx_m \int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i\varphi_j\left(\mathrm{x},\mathbf{v}_i\right) d\mathrm{x} \end{aligned} \qquad \text{(C.87)}$$

And, factoring the terms for the k^{th} and m^{th} dimensions,

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{2v_j^{\,2} v_i}\left(x_k - c_{jk}\right)^2 dx_k \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j \left(\mathrm{x}, \mathbf{v}_i\right) d\mathrm{x} +$$

$$+\sum_{\substack{m=1 \\ m\neq k}}^{n} \int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{v_i} dx_k \int_{x_{m\min}}^{x_{mMAX}} \frac{\left(x_m - c_{jm}\right)^2}{2v_j^{\,2}} dx_m \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j \left(\mathrm{x}, \mathbf{v}_i\right) d\mathrm{x}$$

$$= \int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{2v_j^{\,2} v_i}\left(x_k - c_{jk}\right)^2 e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j \left(x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n, \mathbf{v}_i\right) dx + \quad \text{(C.88)}$$

$$+\sum_{\substack{m=1 \\ m\neq k}}^{n} \int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{v_i} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \int_{x_{m\min}}^{x_{mMAX}} \frac{\left(x_m - c_{jm}\right)^2}{2v_j^{\,2}} e^{-\frac{\left(x_m - \mathbf{c}_{im}\right)^2}{2v_i} - \frac{\left(x_m - \mathbf{c}_{jm}\right)^2}{2v_j}} dx_m \times$$

$$\times \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j \left(\ldots, x_{k-1}, x_{k+1}, \ldots, x_{m-1}, x_{m+1} \ldots, \mathbf{v}_i\right) dx$$

Finally, expanding (C.88) and leaving apart the terms related to the k^{th} and m^{th} dimensions gives

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{2v_j{}^2 v_i}\left(x_k - c_{jk}\right)^2 e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \sqrt{\frac{2\left(v_i + v_j\right)}{\pi v_i v_j}} \times e^{\frac{\left(\mathbf{c}_{ik} - \mathbf{c}_{jk}\right)^2}{2\left(v_i + v_j\right)}} \times$$

$$\times \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i \varphi_j\left(\mathrm{x}, \mathbf{v}_i\right) d\mathrm{x}}{\left[erf\left(\frac{x_{k_{MAX}}\left(v_i + v_j\right) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}}\right) - erf\left(\frac{x_{k_{\min}}\left(v_i + v_j\right) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}}\right)\right]} +$$

$$+\sum_{\substack{m=1 \\ m\neq k}}^{n} \left\{ \begin{array}{l} \int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{v_i} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \int_{x_{m\min}}^{x_{mMAX}} \frac{\left(x_m - c_{jm}\right)^2}{2v_j{}^2} e^{-\frac{\left(x_m - \mathbf{c}_{im}\right)^2}{2v_i} - \frac{\left(x_m - \mathbf{c}_{jm}\right)^2}{2v_j}} dx_m \times \\ \times \frac{2\left(v_i + v_j\right)}{\pi v_i v_j} \times e^{\frac{\left(\mathbf{c}_{ik} - \mathbf{c}_{jk}\right)^2 + \left(\mathbf{c}_{im} - \mathbf{c}_{jm}\right)^2}{2\left(v_i + v_j\right)}} \times \\ \times \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i \varphi_j\left(\mathrm{x}, \mathbf{v}_i\right) d\mathrm{x}}{\left[erf\left(\frac{x_{k_{MAX}}\left(v_i + v_j\right) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}}\right) - erf\left(\frac{x_{k_{\min}}\left(v_i + v_j\right) - c_{ik}v_j - c_{jk}v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}}\right)\right]\left[erf\left(\frac{x_{m_{MAX}}\left(v_i + v_j\right) - c_{im}v_j - c_{jm}v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}}\right) - erf\left(\frac{x_{m_{\min}}\left(v_i + v_j\right) - c_{im}v_j - c_{jm}v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}}\right)\right]} \end{array} \right\}$$

(C.89)

where,

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{x_k - c_{ik}}{2v_j^{\,2}v_i}\left(x_k - c_{jk}\right)^2 e^{-\frac{\left(x_k-\mathbf{c}_{ik}\right)^2}{2v_i}-\frac{\left(x_k-\mathbf{c}_{jk}\right)^2}{2v_j}}\,dx_k$$

$$= \frac{1}{2v_i v_j^{\,2}}\left[\begin{array}{l}
\dfrac{v_i v_j}{\left(v_i+v_j\right)^3}\left(\begin{array}{l}
e^{-\frac{c_{jk}^{\,2}v_i + c_{ik}^{\,2}v_j - 2x_{k\min}\left(c_{jk}v_i + c_{ik}v_j\right)+\left(v_i+v_j\right)x_{k\min}^{\,2}}{2v_iv_j}}\times \\
\left(\begin{array}{l}
-c_{ik}^{\,2}v_iv_j + v_j^{\,2}\left(c_{jk}-x_{k\min}\right)^2 + c_{ik}v_i\left(c_{jk}\left(v_i+3v_j\right)-\left(v_i+v_j\right)x_{k\min}\right)+ \\
+v_i^{\,2}\left(2v_j - c_{jk}x_{k\min} + x_{k\min}^{\,2}\right)+v_iv_j\left(2v_j - 3c_{jk}x_{k\min}+2x_{k\min}^{\,2}\right)
\end{array}\right) \\
-e^{-\frac{c_{jk}^{\,2}v_i + c_{ik}^{\,2}v_j - 2x_{kMAX}\left(c_{jk}v_i + c_{ik}v_j\right)+\left(v_i+v_j\right)x_{kMAX}^{\,2}}{2v_iv_j}}\times \\
\left(\begin{array}{l}
-c_{ik}^{\,2}v_iv_j + v_j^{\,2}\left(c_{jk}-x_{kMAX}\right)^2 + c_{ik}v_i\left(c_{jk}\left(v_i+3v_j\right)-\left(v_i+v_j\right)x_{kMAX}\right)+ \\
+v_i^{\,2}\left(2v_j - c_{jk}x_{kMAX} + x_{kMAX}^{\,2}\right)+v_iv_j\left(2v_j - 3c_{jk}x_{kMAX}+2x_{kMAX}^{\,2}\right)
\end{array}\right)
\end{array}\right) - \\
-\sqrt{\frac{\pi}{2}}\dfrac{\left(v_jv_i\right)^{3/2}}{\left(v_i+v_j\right)^{7/2}}\left(c_{ik}-c_{jk}\right)\left(v_i^{\,2}-v_iv_j+\left(c_{ik}^{\,2}-2c_{ik}c_{jk}+c_{jk}^{\,2}-2v_j\right)v_j\right)e^{-\frac{\left(c_{ik}-c_{jk}\right)^2}{2\left(v_i+v_j\right)}}\times \\
\left(erf\left(\dfrac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)-erf\left(\dfrac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)\right)
\end{array}\right] \tag{C.90}$$

Also,

$$\int_{x_{k\min}}^{x_{kMAX}}\left(\frac{x_k-c_{ik}}{v_i}\right)e^{-\frac{\left(x_k-\mathbf{c}_{ik}\right)^2}{2v_i}-\frac{\left(x_k-\mathbf{c}_{jk}\right)^2}{2v_j}}\,dx_k$$

$$= \frac{v_j}{v_i+v_j}\left(\begin{array}{l}
e^{-\frac{\mathbf{c}_{jk}^{\,2}v_i+\mathbf{c}_{ik}^{\,2}v_j-2\left(\mathbf{c}_{jk}v_i+\mathbf{c}_{ik}v_j\right)x_{k\min}+\left(v_i+v_j\right)x_{k\min}^{\,2}}{2v_iv_j}}- \\
-e^{-\frac{\mathbf{c}_{jk}^{\,2}v_i+\mathbf{c}_{ik}^{\,2}v_j-2\left(\mathbf{c}_{jk}v_i+\mathbf{c}_{ik}v_j\right)x_{kMAX}+\left(v_i+v_j\right)x_{kMAX}^{\,2}}{2v_iv_j}}
\end{array}\right) -$$

$$-\frac{\left(c_{ik}-c_{jk}\right)\sqrt{\pi v_iv_j}}{\sqrt{2}\left(v_i+v_j\right)^{3/2}}e^{-\frac{\left(\mathbf{c}_{ik}-\mathbf{c}_{jk}\right)^2}{2\left(v_i+v_j\right)}}\left[\begin{array}{l}
erf\left(\dfrac{x_{k_{MAX}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)- \\
erf\left(\dfrac{x_{k_{\min}}\left(v_i+v_j\right)-c_{ik}v_j-c_{jk}v_i}{\sqrt{2v_iv_j\left(v_i+v_j\right)}}\right)
\end{array}\right] \tag{C.91}$$

and,

$$\int_{x_{m\min}}^{x_{mMAX}} \frac{\left(x_m - c_{jm}\right)^2}{2v_j^{\,2}} e^{-\frac{\left(x_m - \mathbf{c}_{im}\right)^2}{2v_i} - \frac{\left(x_m - \mathbf{c}_{jm}\right)^2}{2v_j}} dx_m =$$

$$= \frac{1}{2v_j^{\,2}} \left[\begin{array}{l} \left(\begin{array}{l} \frac{v_i v_j}{\left(v_i + v_j\right)^2} \left(c_{im} v_j - c_{jm}\left(v_i + 2v_j\right) + x_{m\min}\left(v_i + v_j\right) \right) e^{-\frac{c_{jm}^{\,2} v_i + c_{im}^{\,2} v_j - 2x_{m\min}\left(c_{jm} v_i + c_{im} v_j\right) + \left(v_i + v_j\right) x_{m\min}^{\,2}}{2v_i v_j}} - \\ - \frac{v_i v_j}{\left(v_i + v_j\right)^2} \left(c_{im} v_j - c_{jm}\left(v_i + 2v_j\right) + x_{mMAX}\left(v_i + v_j\right) \right) e^{-\frac{c_{jm}^{\,2} v_i + c_{im}^{\,2} v_j - 2x_{mMAX}\left(c_{jm} v_i + c_{im} v_j\right) + \left(v_i + v_j\right) x_{mMAX}^{\,2}}{2v_i v_j}} \end{array} \right) - \\ + \sqrt{\frac{\pi v_i}{2}} \frac{v_j^{\,3/2}}{\left(v_i + v_j\right)^{5/2}} \left(c_{im}^{\,2} v_j - 2c_{im} c_{jm} v_j + c_{jm}^{\,2} v_j + v_i\left(v_i + v_j\right) \right) e^{-\frac{\left(c_{im} - c_{jm}\right)^2}{2\left(v_i + v_j\right)}} \times \\ \left(erf\left(\frac{x_{m_{MAX}}\left(v_i + v_j\right) - c_{im} v_j - c_{jm} v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}} \right) - erf\left(\frac{x_{m_{\min}}\left(v_i + v_j\right) - c_{im} v_j - c_{jm} v_i}{\sqrt{2v_i v_j\left(v_i + v_j\right)}} \right) \right) \end{array} \right] \quad \text{(C.92)}$$

C.2.4 Computation of $\int_{x_{\min}}^{x_{MAX}} \frac{\partial \boldsymbol{\varphi}}{\partial \mathbf{v}^T} \boldsymbol{\varphi}^T dx$

By derivation

$$\begin{cases} \frac{\partial \varphi_i\left(x, \mathbf{v}_i\right)}{\partial c_{ik}} \varphi_j\left(x, \mathbf{v}_j\right) = \frac{\left(x_k - c_{ik}\right)}{v_i} \varphi_i\left(x, \mathbf{v}_i\right) \varphi_j\left(x, \mathbf{v}_j\right) \\ \frac{\partial \varphi_i\left(x, \mathbf{v}_i\right)}{\partial v_i} \varphi_j\left(x, \mathbf{v}_j\right) = \sum_{k=1}^{n} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} \varphi_i\left(x, \mathbf{v}_i\right) \varphi_j\left(x, \mathbf{v}_j\right) \end{cases} \quad \text{(C.93)}$$

Introducing equation (C.93) in (C.4), 4 integrals have to be defined:

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)}{v_i} \varphi_i\left(x, \mathbf{v}_i\right) \varphi_j\left(x, \mathbf{v}_j\right) dx \quad \text{(C.94)}$$

$$\int_{x_{k\min}}^{x_{kMAX}} \sum_{k=1}^{n} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} \varphi_i\left(x, \mathbf{v}_i\right) \varphi_j\left(x, \mathbf{v}_j\right) dx \quad \text{(C.95)}$$

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)}{v_i} \varphi_i\left(x, \mathbf{v}_i\right) dx \quad \text{(C.96)}$$

$$\int_{x_{k\min}}^{x_{kMAX}} \sum_{k=1}^{n} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} \varphi_i\left(x, \mathbf{v}_i\right) dx \quad \text{(C.97)}$$

C.2.4.1 Calculation of integral (C.94)

(C.94) can be regarded as

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)}{v_i} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \times \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j \left(., x_{k-1}, x_{k+1}, ..., x_n, \mathbf{v}_i\right) dx$$

(C.98)

Alternatively

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)}{v_i} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \times \sqrt{\frac{2\left(v_i + v_j\right)}{\pi v_i v_j}} \times e^{\frac{\left(\mathbf{c}_{ik} - \mathbf{c}_{jk}\right)^2}{2\left(v_i + v_j\right)}} \times$$

$$\times \frac{\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i\left(\mathrm{x}, \mathbf{v}_i\right) \varphi_j\left(\mathrm{x}, \mathbf{v}_j\right) d\mathrm{x}}{\left[erf\left(\frac{x_{k_{MAX}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) - erf\left(\frac{x_{k_{\min}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) \right]}$$

(C.99)

where,

$$\int_{x_{\min}}^{x_{MAX}} \left(\frac{x_k - c_{ik}}{v_i} \right) e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k = \frac{v_j}{v_i + v_j} \left(e^{-\frac{\mathbf{c}_{jk}^2 v_i + \mathbf{c}_{ik}^2 v_j - 2\left(\mathbf{c}_{jk} v_i + \mathbf{c}_{ik} v_j\right) x_{k\min} + \left(v_i + v_j\right) x_{k\min}^2}{2 v_i v_j}} - e^{-\frac{\mathbf{c}_{jk}^2 v_i + \mathbf{c}_{ik}^2 v_j - 2\left(\mathbf{c}_{jk} v_i + \mathbf{c}_{ik} v_j\right) x_{kMAX} + \left(v_i + v_j\right) x_{kMAX}^2}{2 v_i v_j}} \right) -$$

$$- \frac{\left(c_{ik} - c_{jk}\right) \sqrt{\pi v_i v_j}}{\sqrt{2}\left(v_i + v_j\right)^{3/2}} e^{\frac{\left(\mathbf{c}_{ik} - \mathbf{c}_{jk}\right)^2}{2\left(v_i + v_j\right)}} \left[erf\left(\frac{x_{k_{MAX}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) - erf\left(\frac{x_{k_{\min}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) \right]$$

(C.100)

C.2.4.2 Calculation of integral (C.95)

(C.95) can be regarded as

$$\int_{x_{k\min}}^{x_{kMAX}} \sum_{k=1}^{n} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \times \int_{x_{\min}}^{x_{MAX}} \varphi_i \varphi_j \left(., x_{k-1}, x_{k+1}, ., x_n, \mathbf{v}_i\right) dx$$

(C.101)

or

$$\sum_{k=1}^{n} \int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k \times \sqrt{\frac{2\left(v_i + v_j\right)}{\pi v_i v_j}} \times e^{\frac{\left(\mathbf{c}_{ik} - \mathbf{c}_{jk}\right)^2}{2\left(v_i + v_j\right)}} \times$$

$$\int_{\mathrm{x}_{\min}}^{\mathrm{x}_{MAX}} \varphi_i\left(\mathrm{x}, \mathbf{v}_i\right) \varphi_j\left(\mathrm{x}, \mathbf{v}_j\right) d\mathrm{x} \qquad \text{(C.102)}$$

$$\times \frac{}{\left[erf\left(\frac{x_{k_{MAX}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) - erf\left(\frac{x_{k_{\min}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) \right]}$$

where,

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i} - \frac{\left(x_k - \mathbf{c}_{jk}\right)^2}{2v_j}} dx_k =$$

$$= \frac{1}{2v_i^{\,2}} \left[\begin{array}{l} \left(\begin{array}{l} \frac{v_i v_j}{\left(v_i + v_j\right)^2} \left(c_{jk} v_i - c_{ik}\left(v_j + 2v_i\right) + x_{k\min}\left(v_i + v_j\right) \right) e^{-\frac{c_{jk}^{\,2} v_i + c_{ik}^{\,2} v_j - 2x_{k\min}\left(c_{jk} v_i + c_{ik} v_j\right) + \left(v_i + v_j\right) x_{k\min}^{\,2}}{2 v_i v_j}} - \\ - \frac{v_i v_j}{\left(v_i + v_j\right)^2} \left(c_{jk} v_i - c_{ik}\left(v_j + 2v_i\right) + x_{kMAX}\left(v_i + v_j\right) \right) e^{-\frac{c_{jk}^{\,2} v_i + c_{ik}^{\,2} v_j - 2x_{kMAX}\left(c_{jk} v_i + c_{ik} v_j\right) + \left(v_i + v_j\right) x_{kMAX}^{\,2}}{2 v_i v_j}} \end{array} \right) - \\ + \sqrt{\frac{\pi v_j}{2}} \frac{v_i^{\,3/2}}{\left(v_i + v_j\right)^{5/2}} \left(c_{jk}^{\,2} v_i - 2 c_{jk} c_{ik} v_i + c_{ik}^{\,2} v_i + v_j\left(v_i + v_j\right) \right) e^{-\frac{\left(c_{ik} - c_{jk}\right)^2}{2\left(v_i + v_j\right)}} \times \\ \left(erf\left(\frac{x_{k_{MAX}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) - erf\left(\frac{x_{k_{\min}}\left(v_i + v_j\right) - c_{ik} v_j - c_{jk} v_i}{\sqrt{2 v_i v_j \left(v_i + v_j\right)}} \right) \right) \end{array} \right]$$

(C.103)

C.2.4.3 Calculation of integral (C.96)

Consider the k^{th} dimension, then,

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)}{v_i} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i}} dx_k \int_{x_{\min}}^{x_{MAX}} \varphi_i\left(., x_{k-1}, x_{k+1}, ., x_n, \mathbf{v}_i\right) dx$$

$$= \int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)}{v_i} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i}} dx_k \frac{\int_{x_{\min}}^{x_{MAX}} \varphi_i\left(\mathrm{x}, \mathbf{v}_i\right) dx}{\sqrt{\frac{\pi v_i}{2}}\left[erf\left(\frac{x_{k_{MAX}} - c_{ik}}{\sqrt{2v_i}}\right) - erf\left(\frac{x_{k_{\min}} - c_{ik}}{\sqrt{2v_i}}\right)\right]} \tag{C.104}$$

where,

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)}{v_i} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i}} dx_k = e^{-\frac{\left(x_{k\min} - \mathbf{c}_{ik}\right)^2}{2v_i}} - e^{-\frac{\left(x_{kMAX} - \mathbf{c}_{ik}\right)^2}{2v_i}}$$

C.2.4.4 Calculation of integral (C.97)

Again, considering the k^{th} dimension, (C.97) becomes

$$\int_{x_{k\min}}^{x_{kMAX}} \sum_{k=1}^{n} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} dx_k \int_{x_{\min}}^{x_{MAX}} \varphi_i\left(., x_{k-1}, x_{k+1}, ., x_n, \mathbf{v}_i\right) dx \tag{C.105}$$

which is equivalent to

$$\int_{x_{k\min}}^{x_{kMAX}} \sum_{k=1}^{n} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i}} dx_k \frac{\int_{x_{\min}}^{x_{MAX}} \varphi_i\left(\mathrm{x}, \mathbf{v}_i\right) dx}{\sqrt{\frac{\pi v_i}{2}}\left[erf\left(\frac{x_{k_{MAX}} - c_{ik}}{\sqrt{2v_i}}\right) - erf\left(\frac{x_{k_{\min}} - c_{ik}}{\sqrt{2v_i}}\right)\right]} \tag{C.106}$$

where,

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{\left(x_k - c_{ik}\right)^2}{2v_i^{\,2}} e^{-\frac{\left(x_k - \mathbf{c}_{ik}\right)^2}{2v_i}} dx_k = \frac{1}{2v_i}\left[\begin{array}{l}\left(C_{ik} - x_{kMAX}\right) e^{-\frac{\left(\mathbf{c}_{ik} - x_{kMAX}\right)^2}{2v_i}} - \left(C_{ik} - x_{k\min}\right) e^{-\frac{\left(\mathbf{c}_{ik} - x_{k\min}\right)^2}{2v_i}} + \\ + \sqrt{\frac{\pi}{2} v_i}\left[erf\left(\frac{x_{k_{MAX}} - c_{ik}}{\sqrt{2v_i}}\right) - erf\left(\frac{x_{k_{\min}} - c_{ik}}{\sqrt{2v_i}}\right)\right]\end{array}\right] \tag{C.107}$$

C.2.4.5 Computation of $\int_{x_{\min}}^{x_{MAX}} \frac{\partial\varphi(x,c,v)}{\partial v}dx$.

One knows that $\int_{x_{\min}}^{x_{MAX}} \frac{\partial\varphi(x,c,v)}{\partial v}dx = \int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx$. Using the substitution $y = \frac{(x-c)^2}{2v}$, then

$$x = \begin{cases} c+\sqrt{2vy},\ x \geq c \\ c-\sqrt{2vy},\ x \leq c \end{cases}, \qquad \frac{(c-x)^2}{2v^2} = \frac{y}{v}, \qquad dx = \begin{cases} +\sqrt{\frac{v}{2}}\frac{1}{\sqrt{y}}dy,\ x \geq c \\ -\sqrt{\frac{v}{2}}\frac{1}{\sqrt{y}}dy,\ x \leq c \end{cases}, \qquad x = x_{MAX} \Rightarrow y = \frac{(x_{MAX}-c)^2}{2v},$$

$$x = x_{\min} \Rightarrow y = \frac{(x_{\min}-c)^2}{2v},\ x = c \Rightarrow y = 0$$

If $c < x_{\min} \Rightarrow x \geq c$

$$\int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx = \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{y}{v} e^{-y} \sqrt{\frac{v}{2}}\frac{1}{\sqrt{y}} dy =$$

$$= \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{\sqrt{y}}{\sqrt{2v}} e^{-y} dy = \frac{1}{\sqrt{2v}} \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \sqrt{y} e^{-y} dy =$$

$$= -\frac{1}{\sqrt{2v}} \left(\frac{\sqrt{\pi}\,erfc\left(\sqrt{y}\right)}{2} + \sqrt{y}e^{-y} \right) \Bigg|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} =$$

$$= \frac{1}{\sqrt{2v}} \left(\begin{array}{l} \dfrac{\sqrt{\pi}\,erfc\left(\sqrt{\dfrac{(x_{\min}-c)^2}{2v}}\right)}{2} + \sqrt{\dfrac{(x_{\min}-c)^2}{2v}} e^{-\frac{(x_{\min}-c)^2}{2v}} - \\ -\dfrac{\sqrt{\pi}\,erfc\left(\sqrt{\dfrac{(x_{MAX}-c)^2}{2v}}\right)}{2} - \sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} e^{-\frac{(x_{MAX}-c)^2}{2v}} \end{array} \right) \qquad \text{(C.108)}$$

If $c > x_{MAX} \Rightarrow x \leq c$

$$\int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx = \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{y}{v} e^{-y} \left(-\sqrt{\frac{v}{2}} \frac{1}{\sqrt{y}} \right) dy =$$

$$= -\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{\sqrt{y}}{\sqrt{2v}} e^{-y} dy = -\frac{1}{\sqrt{2v}} \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \sqrt{y} e^{-y} dy =$$

$$= \frac{1}{\sqrt{2v}} \left(\frac{\sqrt{\pi} erfc\left(\sqrt{y}\right)}{2} + \sqrt{y} e^{-y} \right) \Bigg|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} =$$

$$= -\frac{1}{\sqrt{2v}} \left(\begin{array}{l} \dfrac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{\min}-c)^2}{2v}} \right)}{2} + \sqrt{\dfrac{(x_{\min}-c)^2}{2v}} e^{-\frac{(x_{\min}-c)^2}{2v}} - \\ \dfrac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} \right)}{2} - \sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} e^{-\frac{(x_{MAX}-c)^2}{2v}} \end{array} \right) \tag{C.109}$$

If $x_{\min} < c < x_{MAX}$

$$\int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx = \int_{x_{\min}}^{c} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx + \int_{c}^{x_{MAX}} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx \tag{C.110}$$

$$\int_{x_{\min}}^{c} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx = -\frac{1}{\sqrt{2v}} \left(\frac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{\min}-c)^2}{2v}} \right)}{2} + \sqrt{\frac{(x_{\min}-c)^2}{2v}} e^{-\frac{(x_{\min}-c)^2}{2v}} - \frac{\sqrt{\pi}}{2} \right) \tag{C.111}$$

$$\int_{c}^{x_{MAX}} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx = \frac{1}{\sqrt{2v}} \left(\frac{\sqrt{\pi}}{2} - \frac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} \right)}{2} - \sqrt{\frac{(x_{MAX}-c)^2}{2v}} e^{-\frac{(x_{MAX}-c)^2}{2v}} \right) \tag{C.112}$$

Adding the two together

$$\int_{x_{\min}}^{0} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx = \frac{1}{\sqrt{2v}} \left(\begin{array}{l} \sqrt{\pi} - \dfrac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{\min}-c)^2}{2v}} \right)}{2} - \sqrt{\dfrac{(x_{\min}-c)^2}{2v}} e^{-\frac{(x_{\min}-c)^2}{2v}} - \\ \dfrac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} \right)}{2} - \sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} e^{-\frac{(x_{MAX}-c)^2}{2v}} \end{array} \right) \tag{C.113}$$

C.2.4.6 Computation of $\int_{x_{\min}}^{x_{MAX}} \frac{\partial \varphi^2(x,c,v)}{\partial v} dx$

Considering that $\int_{x_{\min}}^{x_{MAX}} \frac{\partial \varphi^2(x,c,v)}{\partial v} dx = \int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{v^2} e^{-\frac{(c-x)^2}{v}} dx$. If one performs the substitution $y = \frac{(x-c)^2}{v}$, then

$$x = \begin{cases} c + \sqrt{2vy}, & x \geq c \\ c - \sqrt{2vy}, & x \leq c \end{cases}, \qquad \frac{(c-x)^2}{v^2} = \frac{2y}{v}, \qquad dx = \begin{cases} +\sqrt{\frac{v}{2}} \frac{1}{\sqrt{y}} dy, & x \geq c \\ -\sqrt{\frac{v}{2}} \frac{1}{\sqrt{y}} dy, & x \leq c \end{cases}, \qquad x = x_{MAX} \Rightarrow y = \frac{(x_{MAX} - c)^2}{2v},$$

$$x = x_{\min} \Rightarrow y = \frac{(x_{\min} - c)^2}{2v}, \; x = c \Rightarrow y = 0$$

If $c < x_{\min} \Rightarrow x \geq c$

$$\int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx = \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{2y}{v} e^{-2y} \sqrt{\frac{v}{2}} \frac{1}{\sqrt{y}} dy =$$

$$= \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{\sqrt{2y}}{\sqrt{v}} e^{-2y} dy = \frac{\sqrt{2}}{\sqrt{v}} \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \sqrt{y} e^{-2y} dy =$$

$$= -\frac{1}{\sqrt{v}} \left(\frac{\sqrt{\pi} erfc\left(\sqrt{2y}\right)}{4} + \frac{\sqrt{y} e^{-2y}}{\sqrt{2}} \right) \Bigg|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} =$$

$$= -\frac{1}{\sqrt{v}} \left(\begin{array}{l} \frac{\sqrt{\pi} erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{v}}\right)}{4} + \frac{\sqrt{\frac{(x_{MAX}-c)^2}{2v}} e^{-\frac{(x_{MAX}-c)^2}{v}}}{\sqrt{2}} - \\ - \frac{\sqrt{\pi} erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{v}}\right)}{4} - \frac{\sqrt{\frac{(x_{\min}-c)^2}{2v}} e^{-\frac{(x_{\min}-c)^2}{v}}}{\sqrt{2}} \end{array} \right) \qquad \text{(C.114)}$$

If $c > x_{MAX} \Rightarrow x \leq c$

$$\int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{2v^2} e^{-\frac{(c-x)^2}{2v}} dx = \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{2y}{v} e^{-2y} \left(-\sqrt{\frac{v}{2}} \frac{1}{\sqrt{y}} \right) dy =$$

$$= -\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{\sqrt{2y}}{\sqrt{v}} e^{-2y} dy = -\frac{\sqrt{2}}{\sqrt{v}} \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \sqrt{y} e^{-2y} dy =$$

$$= \frac{1}{\sqrt{v}} \left(\frac{\sqrt{\pi} erfc\left(\sqrt{2y}\right)}{4} + \frac{\sqrt{y} e^{-2y}}{\sqrt{2}} \right) \Bigg|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} =$$

$$= \frac{1}{\sqrt{v}} \left(\begin{array}{l} \dfrac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{MAX}-c)^2}{v}} \right)}{4} + \dfrac{\sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} e^{-\frac{(x_{MAX}-c)^2}{v}}}{\sqrt{2}} - \\ - \dfrac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{\min}-c)^2}{v}} \right)}{4} - \dfrac{\sqrt{\dfrac{(x_{\min}-c)^2}{2v}} e^{-\frac{(x_{\min}-c)^2}{v}}}{\sqrt{2}} \end{array} \right) \qquad \text{(C.115)}$$

If $x_{\min} < c < x_{MAX}$

$$\int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{v^2} e^{-\frac{(c-x)^2}{v}} dx = \int_{x_{\min}}^{c} \frac{(c-x)^2}{v^2} e^{-\frac{(c-x)^2}{v}} dx + \int_{c}^{x_{MAX}} \frac{(c-x)^2}{v^2} e^{-\frac{(c-x)^2}{v}} dx$$

$$\int_{x_{\min}}^{c} \frac{(c-x)^2}{v^2} e^{-\frac{(c-x)^2}{v}} dx = \frac{1}{\sqrt{v}} \left(\frac{\sqrt{\pi}}{4} - \frac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{\min}-c)^2}{v}} \right)}{4} - \frac{\sqrt{\dfrac{(x_{\min}-c)^2}{2v}} e^{-\frac{(x_{\min}-c)^2}{v}}}{\sqrt{2}} \right)$$

$$\int_{c}^{x_{MAX}} \frac{(c-x)^2}{v^2} e^{-\frac{(c-x)^2}{v}} dx = -\frac{1}{\sqrt{v}} \left(\frac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{MAX}-c)^2}{v}} \right)}{4} + \frac{\sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} e^{-\frac{(x_{MAX}-c)^2}{v}}}{\sqrt{2}} - \frac{\sqrt{\pi}}{4} \right)$$

Adding the two together:

$$\int_{x_{\min}}^{x_{MAX}} \frac{(c-x)^2}{v^2} e^{-\frac{(c-x)^2}{v}} dx = \frac{1}{\sqrt{v}} \left(\begin{array}{l} \dfrac{\sqrt{\pi}}{2} - \dfrac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{\min}-c)^2}{v}} \right)}{4} - \dfrac{\sqrt{\dfrac{(x_{\min}-c)^2}{2v}} e^{-\frac{(x_{\min}-c)^2}{v}}}{\sqrt{2}} - \\ - \dfrac{\sqrt{\pi} erfc\left(\sqrt{\dfrac{(x_{MAX}-c)^2}{v}} \right)}{4} - \dfrac{\sqrt{\dfrac{(x_{MAX}-c)^2}{2v}} e^{-\frac{(x_{MAX}-c)^2}{v}}}{\sqrt{2}} \end{array} \right)$$

$$\int_{x_{\min}}^{x_{MAX}} x^2 e^{-\frac{(x-c)^2}{2v}}\,dx \tag{C.116}$$

Performing the substitution $y=\dfrac{(x-c)^2}{2v}$,

then $x=\begin{cases} c+\sqrt{2vy},\ x\ge c \\ c-\sqrt{2vy},\ x\le c \end{cases}$, $x^2=\begin{cases} c^2+2c\sqrt{2vy}+2vy,\ x\ge c \\ c^2-2c\sqrt{2vy}+2vy,\ x\le c \end{cases}$ $dx=\begin{cases} +\sqrt{\dfrac{v}{2}}\dfrac{1}{\sqrt{y}}dy,\ x\ge c \\ -\sqrt{\dfrac{v}{2}}\dfrac{1}{\sqrt{y}}dy,\ x\le c \end{cases}$,

$x=x_{MAX}\Rightarrow y=\dfrac{(x_{MAX}-c)^2}{2v}$, $x=x_{\min}\Rightarrow y=\dfrac{(x_{\min}-c)^2}{2v}$, $x=c\Rightarrow y=0$

If $c<x_{\min}\Rightarrow x\ge c$

$$\int_{x_{\min}}^{x_{MAX}} x^2 e^{-\frac{(x-c)^2}{2v}}dx=\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}}\left(c^2+2c\sqrt{2vy}+2vy\right)e^{-y}\left(\sqrt{\frac{v}{2}}\frac{1}{\sqrt{y}}\right)dy=$$

$$=c^2\sqrt{\frac{v}{2}}\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}}\frac{e^{-y}}{\sqrt{y}}dy+2cv\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}}e^{-y}dy+v\sqrt{2v}\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}}\sqrt{y}e^{-y}dy=$$

$$=-c^2\sqrt{\frac{v\pi}{2}}\left(erfc\left(\sqrt{y}\right)\right)\Big|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}}-2cve^{-y}\Big|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}}-v\sqrt{2v}\left(\frac{\sqrt{\pi}erfc\left(\sqrt{y}\right)}{2}+\sqrt{y}e^{-y}\right)\Bigg|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}}=$$

$$=-c^2\sqrt{\frac{v\pi}{2}}\left(erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{2v}}\right)-erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{2v}}\right)\right)-2cv\left(e^{-\frac{(x_{MAX}-c)^2}{2v}}-e^{-\frac{(x_{\min}-c)^2}{2v}}\right)-$$

$$-v\sqrt{2v}\left(\frac{\sqrt{\pi}erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{2v}}\right)}{2}-\frac{\sqrt{\pi}erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{2v}}\right)}{2}+\sqrt{\frac{(x_{MAX}-c)^2}{2v}}e^{-\frac{(x_{MAX}-c)^2}{2v}}-\sqrt{\frac{(x_{\min}-c)^2}{2v}}e^{-\frac{(x_{\min}-c)^2}{2v}}\right)$$

(C.117)

If $c>x_{MAX}\Rightarrow x\le c$

$$\int_{x_{\min}}^{x_{MAX}} x^2 e^{-\frac{(x-c)^2}{2v}}\,dx = \int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \left(c^2 - 2c\sqrt{2vy} + 2vy\right)e^{-y}\left(-\sqrt{\frac{v}{2}}\frac{1}{\sqrt{y}}\right)dy =$$

$$= -c^2\sqrt{\frac{v}{2}}\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{e^{-y}}{\sqrt{y}}\,dy + 2cv\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} e^{-y}\,dy - v\sqrt{2v}\int_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} \sqrt{y}e^{-y}\,dy =$$

$$= +c^2\sqrt{\frac{v\pi}{2}}\left(erfc\left(\sqrt{y}\right)\right)\Big|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} - 2cv\,e^{-y}\Big|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} + v\sqrt{2v}\left(\frac{\sqrt{\pi}erfc\left(\sqrt{y}\right)}{2} + \sqrt{y}e^{-y}\right)\Bigg|_{\frac{(x_{\min}-c)^2}{2v}}^{\frac{(x_{MAX}-c)^2}{2v}} =$$

$$= +c^2\sqrt{\frac{v\pi}{2}}\left(erfc\left(\sqrt{\frac{\left(x_{MAX}-c\right)^2}{2v}}\right) - erfc\left(\sqrt{\frac{\left(x_{\min}-c\right)^2}{2v}}\right)\right) - 2cv\left(e^{-\frac{(x_{MAX}-c)^2}{2v}} - e^{-\frac{(x_{\min}-c)^2}{2v}}\right) -$$

$$+v\sqrt{2v}\left(\frac{\sqrt{\pi}erfc\left(\sqrt{\frac{\left(x_{MAX}-c\right)^2}{2v}}\right)}{2} - \frac{\sqrt{\pi}erfc\left(\sqrt{\frac{\left(x_{\min}-c\right)^2}{2v}}\right)}{2} + \sqrt{\frac{\left(x_{MAX}-c\right)^2}{2v}}e^{-\frac{(x_{MAX}-c)^2}{2v}} - \sqrt{\frac{\left(x_{\min}-c\right)^2}{2v}}e^{-\frac{(x_{\min}-c)^2}{2v}}\right)$$

(C.118)

If $x_{\min} < c < x_{MAX}$

$$\int_{x_{\min}}^{x_{MAX}} x^2 e^{-\frac{(x-c)^2}{2v}}\,dx = \int_{x_{\min}}^{c} x^2 e^{-\frac{(x-c)^2}{2v}}\,dx + \int_{c}^{x_{MAX}} x^2 e^{-\frac{(x-c)^2}{2v}}\,dx$$

$$\int_{x_{\min}}^{c} x^2 e^{-\frac{(x-c)^2}{2v}}\,dx = \int_{\frac{(x_{\min}-c)^2}{2v}}^{0} \left(c^2 - 2c\sqrt{2vy} + 2vy\right) e^{-y} \left(-\sqrt{\frac{v}{2}}\frac{1}{\sqrt{y}}\right) dy =$$

$$= -c^2\sqrt{\frac{v}{2}} \int_{\frac{(x_{\min}-c)^2}{2v}}^{0} \frac{e^{-y}}{\sqrt{y}}\,dy + 2cv \int_{\frac{(x_{\min}-c)^2}{2v}}^{0} e^{-y}\,dy - v\sqrt{2v} \int_{\frac{(x_{\min}-c)^2}{2v}}^{0} \sqrt{y} e^{-y}\,dy =$$

$$= +c^2\sqrt{\frac{v\pi}{2}} \left(erfc\left(\sqrt{y}\right)\right)\Big|_{\frac{(x_{\min}-c)^2}{2v}}^{0} - 2cv\,e^{-y}\Big|_{\frac{(x_{\min}-c)^2}{2v}}^{0} + v\sqrt{2v} \left(\frac{\sqrt{\pi}erfc\left(\sqrt{y}\right)}{2} + \sqrt{y}e^{-y}\right)\Big|_{\frac{(x_{\min}-c)^2}{2v}}^{0} =$$

$$= +c^2\sqrt{\frac{v\pi}{2}} \left(erfc(0) - erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{2v}}\right)\right) - 2cv\left(e^0 - e^{-\frac{(x_{\min}-c)^2}{2v}}\right) +$$

$$+v\sqrt{2v}\left(\frac{\sqrt{\pi}erfc(0)}{2} - \frac{\sqrt{\pi}erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{2v}}\right)}{2} - \sqrt{\frac{(x_{\min}-c)^2}{2v}}\,e^{-\frac{(x_{\min}-c)^2}{2v}}\right)$$

$$\int_{c}^{x_{MAX}} x^2 e^{-\frac{(x-c)^2}{2v}}\,dx == \int_{0}^{\frac{(x_{MAX}-c)^2}{2v}} \left(c^2 + 2c\sqrt{2vy} + 2vy\right) e^{-y} \left(\sqrt{\frac{v}{2}}\frac{1}{\sqrt{y}}\right) dy =$$

$$= c^2\sqrt{\frac{v}{2}} \int_{0}^{\frac{(x_{MAX}-c)^2}{2v}} \frac{e^{-y}}{\sqrt{y}}\,dy + 2cv \int_{0}^{\frac{(x_{MAX}-c)^2}{2v}} e^{-y}\,dy + v\sqrt{2v} \int_{0}^{\frac{(x_{MAX}-c)^2}{2v}} \sqrt{y} e^{-y}\,dy =$$

$$= -c^2\sqrt{\frac{v\pi}{2}} \left(erfc\left(\sqrt{y}\right)\right)\Big|_{0}^{\frac{(x_{MAX}-c)^2}{2v}} - 2cv\,e^{-y}\Big|_{0}^{\frac{(x_{MAX}-c)^2}{2v}} - v\sqrt{2v} \left(\frac{\sqrt{\pi}erfc\left(\sqrt{y}\right)}{2} + \sqrt{y}e^{-y}\right)\Big|_{0}^{\frac{(x_{MAX}-c)^2}{2v}} =$$

$$= -c^2\sqrt{\frac{v\pi}{2}} \left(erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{2v}}\right) - erfc(0)\right) - 2cv\left(e^{-\frac{(x_{MAX}-c)^2}{2v}} - e^0\right) -$$

$$-v\sqrt{2v}\left(\frac{\sqrt{\pi}erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{2v}}\right)}{2} - \frac{\sqrt{\pi}erfc(0)}{2} + \sqrt{\frac{(x_{MAX}-c)^2}{2v}}\,e^{-\frac{(x_{MAX}-c)^2}{2v}}\right)$$

$$\int_{x_{\min}}^{x_{MAX}} x^2 e^{-\frac{(x-c)^2}{2v}}\,dx = +c^2\sqrt{\frac{v\pi}{2}}\left(erfc(0)-erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{2v}}\right)\right)-2cv\left(e^0-e^{-\frac{(x_{\min}-c)^2}{2v}}\right)+$$
$$+v\sqrt{2v}\left(\frac{\sqrt{\pi}erfc(0)}{2}-\frac{\sqrt{\pi}erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{2v}}\right)}{2}-\sqrt{\frac{(x_{\min}-c)^2}{2v}}e^{-\frac{(x_{\min}-c)^2}{2v}}\right)-$$
$$-c^2\sqrt{\frac{v\pi}{2}}\left(erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{2v}}\right)-erfc(0)\right)-2cv\left(e^{-\frac{(x_{MAX}-c)^2}{2v}}-e^0\right)-$$
$$-v\sqrt{2v}\left(\frac{\sqrt{\pi}erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{2v}}\right)}{2}-\frac{\sqrt{\pi}erfc(0)}{2}+\sqrt{\frac{(x_{MAX}-c)^2}{2v}}e^{-\frac{(x_{MAX}-c)^2}{2v}}\right)=$$
$$=+c^2\sqrt{\frac{v\pi}{2}}\left(2erfc(0)-erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{2v}}\right)-erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{2v}}\right)\right)-2cv\left(e^{-\frac{(x_{MAX}-c)^2}{2v}}-e^{-\frac{(x_{\min}-c)^2}{2v}}\right)+$$
$$+v\sqrt{2v}\left(-\frac{\sqrt{\pi}erfc\left(\sqrt{\frac{(x_{MAX}-c)^2}{2v}}\right)}{2}+\sqrt{\pi}erfc(0)-\frac{\sqrt{\pi}erfc\left(\sqrt{\frac{(x_{\min}-c)^2}{2v}}\right)}{2}-\right.$$
$$\left.-\sqrt{\frac{(x_{MAX}-c)^2}{2v}}e^{-\frac{(x_{MAX}-c)^2}{2v}}-\sqrt{\frac{(x_{\min}-c)^2}{2v}}e^{-\frac{(x_{\min}-c)^2}{2v}}\right)$$

(C.119)

C.2.4.7 Computation of $\int_{x_{k\min}}^{x_{kMAX}} \frac{(x_k-c_{ik})^2(x_k-c_{jk})^2}{4v_i^2v_j^2}e^{-\frac{(x_k-\mathbf{c}_{ik})^2}{2v_i}-\frac{(x_k-\mathbf{c}_{jk})^2}{2v_j}}dx_k$

This integral can be expanded in terms of the 3rd order polynomial:

$$\int_{x_{k\min}}^{x_{kMAX}} \frac{(x_k-c_{ik})^2(x_k-c_{jk})^2}{4v_i^2v_j^2}e^{-\frac{(x_k-\mathbf{c}_{ik})^2}{2v_i}-\frac{(x_k-\mathbf{c}_{jk})^2}{2v_j}}dx_k=\int_{x_{k\min}}^{x_{kMAX}} \frac{x_k^4+c_3x_k^3+c_2x_k^2+c_1x_k+c_0}{4v_i^2v_j^2}e^{-\frac{(x_k-\mathbf{c}_{ik})^2}{2v_i}-\frac{(x_k-\mathbf{c}_{jk})^2}{2v_j}}dx_k$$

(C.120)

Hence, the integral of the form $\int_{x_{\min}}^{x_{MAX}} x^i e^{-\frac{(x-\mathbf{c}_i)^2}{2v_i}-\frac{(x-\mathbf{c}_j)^2}{2v_j}} dx, i=0...4$ is necessary.

C.2.4.7.1 Solving integral when i=0:

If one performs the substitution $y=\frac{(x-c_i)^2}{2v_i}+\frac{(x-c_j)^2}{2v_j}$, then:

$$x=\begin{cases}\frac{C_jv_i+C_iv_j+\sqrt{-C_i^2v_iv_j+2C_iC_jv_iv_j-C_j^2v_iv_j+2v_i^2v_jy+2v_iv_j^2y}}{(v_i+v_j)}, & x\geq C_jv_i+C_iv_j\\ \frac{C_jv_i+C_iv_j-\sqrt{-C_i^2v_iv_j+2C_iC_jv_iv_j-C_j^2v_iv_j+2v_i^2v_jy+2v_iv_j^2y}}{(v_i+v_j)}, & x\leq C_jv_i+C_iv_j\end{cases},$$

$$dx=\begin{cases}+\frac{2v_i^2v_j+2v_iv_j^2}{2(v_i+v_j)\sqrt{-C_i^2v_iv_j+2C_iC_jv_iv_j-C_j^2v_iv_j+2v_i^2v_jy+2v_iv_j^2y}}dy, & x\geq C_jv_i+C_iv_j\\ -\frac{2v_i^2v_j+2v_iv_j^2}{2(v_i+v_j)\sqrt{-C_i^2v_iv_j+2C_iC_jv_iv_j-C_j^2v_iv_j+2v_i^2v_jy+2v_iv_j^2y}}dydy, & x\leq C_jv_i+C_iv_j\end{cases} \quad \text{(C.121)}$$

$$x=x_{MAX}\Rightarrow y=\frac{(x_{MAX}-c_i)^2}{2v_i}+\frac{(x_{MAX}-c_j)^2}{2v_j}, \quad x=x_{\min}\Rightarrow y=\frac{(x_{\min}-c_i)^2}{2v_i}+\frac{(x_{\min}-c_j)^2}{2v_j},$$

$$x=\frac{C_jv_i+C_iv_j+\sqrt{-C_i^2v_iv_j+2C_iC_jv_iv_j-C_j^2v_iv_j}}{(v_i+v_j)}\Rightarrow y=0$$

Now,

$$\int_{x_{\min}}^{x_{MAX}} e^{-\frac{(x-\mathbf{c}_i)^2}{2v_i}-\frac{(x-\mathbf{c}_j)^2}{2v_j}}dx=\int_{\frac{(x_{\min}-c_i)^2}{2v_i}+\frac{(x_{\min}-c_j)^2}{2v_j}}^{\frac{(x_{Max}-c_i)^2}{2v_i}+\frac{(x_{Max}-c_j)^2}{2v_j}}\frac{2v_i^2v_j+2v_iv_j^2}{2(v_i+v_j)\sqrt{-C_i^2v_iv_j+2C_iC_jv_iv_j-C_j^2v_iv_j+2v_i^2v_jy+2v_iv_j^2y}}e^{-y}dy$$

$$=\left[e^{-\frac{(C_i-C_j)^2}{2(v_i+v_j)}}\sqrt{\frac{\pi}{2}}v_iv_j\sqrt{-C_i^2+2C_iC_j-C_j^2+2(v_i+v_j)y}\frac{Erf\left(\frac{\sqrt{-C_i^2+2C_iC_j-C_j^2+2(v_i+v_j)y}}{\sqrt{2(v_i+v_j))}}\right)}{\sqrt{-(v_i+v_j)v_iv_j\left(C_i^2-2C_iC_j+C_j^2-2(v_i+v_j)y\right)}}\right]_{\frac{(x_{\min}-c_i)^2}{2v_i}+\frac{(x_{\min}-c_j)^2}{2v_j}}^{\frac{(x_{Max}-c_i)^2}{2v_i}+\frac{(x_{Max}-c_j)^2}{2v_j}}$$

(C.122)