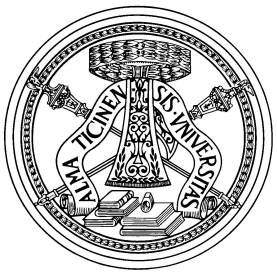


Università degli Studi di Pavia

Facoltà di Ingegneria

Dipartimento di Ingegneria Civile e Architettura

Master degree in Civil Engineering

Max-Ent approximants and applications with collocation method

Funzioni approssimanti alla massima entropia e applicazioni con il metodo di collocazione

– Master Thesis –

Supervisor: Professor
Ferdinando Auricchio
Co - supervisor:
Andrea Montanino

Author:
Alberto Cattenone
UIN **419049**

Academic year 2014/2015

HARD WORK & DEDICATION
(Floyd Mayweather Jr.)

Acknowledgments

I would like to express my very great appreciation to professor Ferdinando Auricchio and Ing. Andrea Montanino for their valuable and constructive suggestions during the planning and development of this research work. Their willingness to give their time so generously has been very much appreciated. I want to thanks all my friends Alice, Margherita, Valentina, Xi, Rodrigo and Marco for their help in time of trouble. A special thanks to my dear friends Alessio and Nicolò for having supported me in all my studies carrier. At least but not at last, i desire to express all my gratitude to professors Gianni Sorato and Caterina Armao for their help every time i need it.

This work is dedicated to all my parents in particular to my dear grandmother.

Abstract

In this work we present a new family of approximation schemes based on the maximum entropy, which are generated by a compromise between the statistic inference of the nodes of the domain and the necessity to have local shape functions. During this research we outline the principal mathematical and features, with particular attention to computational codes. In the first part of the work we focus on the Janeys' maximum entropy principle and on the Shannon's theorem; from a physicist viewpoint, instead, we sketch out the thermodynamic background which define the formal aspect of the shape functions. Then we move to a more practical contest, by introducing the first kind of approximants we want to investigate: the Local *max-ent* approximation schemes (LME). We focus on the optimization program which allows to build the shape functions. In particular we follow step by step the minimization of the associated Lagrangian problem with Newton-Raphson method, which is the core of the program. We have also tested the approximants with simple approximating problems. After the basis functions, we give the necessary environment to build up the first and second derivatives of the approximants; as usual we start from the analytic form to derive a practical code. This is the first big section of our work. In the second, we start from LMEs with purpose to impose a second order consistency condition (SME). Also for this case we start from a pure mathematical viewpoint to delineate the construction of the basis functions and associated first and second derivatives, by following analogous steps we approached to in LME. In appendix of both the methods we draw attention to the computational aspects that involve the optimization of the program in order to reduce the machine time processing, in specific during the minimization program and the memory usage in storing the results. In the last part of the work, we purpose to test both the approximation schemes in some elastic problem, to prove the efficiency of the approximants. We adopt first regular grids of nodes and then irregular ones, in order to really test the program in mesh-less conditions and we use the collocation method in order to find the solutions. First we face a convex problem, as recommended by the mathematical base, then we purpose to test LMEs for a problem fixed on a concave domain to see the response of the approximants also in this conditions. In conclusion we give our critique on the studied methods, illustrating the virtues and the limits of the approximants, focus on the possible future applications.

Sommario

In questo lavoro viene presentata una famiglia di approssimanti *meshfree* basati sulla massima entropia che traggono la loro base da un compromesso computazionale, nel senso ottimale inteso da Pareto, tra l'inferenza statistica dei nodi del dominio e la necessità di avere funzioni di forma quanto più possibile locali. Nella presente ricerca si illustrano le fondamenta matematico - fisiche che sorreggono il metodo. Nello specifico, ci si sofferma sul principio di massima entropia di Jaynes e sulla misura dell'entropia fornita dal teorema di Shannon; dal punto di vista fisico si punta lo sguardo sul background termodinamico che costituisce il punto di partenza dell'aspetto formale di definizione delle funzioni di forma di seguito illustrate. Partendo dalle necessarie condizioni preliminari tipiche di tutti i metodi senza mesh, vengono delineate le caratteristiche dei metodi di approssimazione aventi ordine di consistenza lineare (LME) e quadratico (SME), focalizzando l'attenzione sulla costruzione delle funzioni di forma e delle loro derivate prime e seconde. In particolare viene seguito passo passo tutto l'iter di costruzione delle funzioni, dalla delineazione del problema di minimizzazione Lagrangiano al metodo iterativo di Newton-Raphson utilizzato per trovare i minimi del problema. Nel primo grande blocco della trattazione ci soffermiamo sul metodo LME e nel secondo sulla sua naturale estensione all' SME. Per entrambe i metodi è stato inserito un capitolo riguardante la pura parte di implementazione con l'intento di suggerire delle soluzioni che permettano di ridurre i tempi di calcolo e salvare spazio di memoria. Infine, si vogliono testare gli approssimanti su problemi pratici; d'apprima si sono affrontati dei problemi elastici definiti su domini concavi sia per griglie di punti regolari che non: questo per verificare la risposta di entrambe i metodi in un contesto generale. In seconda istanza si è voluto testare il metodo LME per un dominio concavo al fine di verificare se il metodo è in grado di fornire risultati soddisfacenti anche in queste condizioni matematicamente sfavoreli. Dato che, grazie alla derivazione matematica, si è in grado di calcolare analiticamente le derivate seconde, in ambo i casi i problemi elastici sono stati risolti grazie all'uso del metodo di collocazione. In coda alla trattazione riportiamo un giudizio critico sulla effettiva praticità del metodo evidenziandone le virtù e i limiti. All'interno del lavoro ci si è serviti altresì di diverse immagini al fine di illustrare meglio forma degli approssimanti e risultati ottenuti.

Contents

List of Tables

List of Figures

Chapter 1

Introduction to *meshfree* methods

Mesh-free methods are a particular field of numerical simulation that do not require any data connection between the nodes of the domain. The only thing the methods need are the nodes and their coordinates. In this way the shape functions are built only from the node set. Mesh-free methods have been introduced in the 70's in order to face border-less problem. Then those methods have been approached to different study areas including engineering and in particular civil engineering. The motivation of the usage of those methods is that in others methods like finite elements or finite volumes, every node has a predefined number of fixed neighbor nodes; this could be a problem when we have a material that can move around or that can have excursion in high deformation field (in cracking problem for example). In this case, in fact, the neighbors of a single node can change during the simulation and this causes a consistent error in the final results. With this methods, instead, since there isn't any connection grid between the nodes, the error of the method is caused principally by the way the shape functions are constructed. For example in fluid dynamics or in structural analysis with large strain and deformations, those methods have been performed successfully. The most important mesh-free methods are [12]:

- Mesh-less Local Petrov Galerkin (MLPG) [4];
- Moving least squares (MLS) [26];
- Generalized finite difference method (GFDM) [12];
- Smoothed particle hydrodynamics (SPH) [12];

All those methods have been approached to different ways the philosophical principle of mesh-free, but there are some features that never change. In particular the shape functions of every method must have some common proprieties:

- Partition unity (compulsory condition)

$$\sum_{a=1}^{n} \phi_i(x) = 1$$

  which means that in every evaluation point the summary of the shape functions in that point must be unitary;

- Linear fields reproduction (preferable condition)

$$\sum_{a=1}^{n} \phi_i(x) x_i = x$$

that guarantees the approximants are able to exactly reproduce any linear field;

- Delta-Kroneker propriety (preferable condition)

$$\phi_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

this is a typical condition of interpolating shape functions we can recover also in classical finite element schemes.

In addition to this proprieties a good mesh-free method should have also this requirements:

- arbitrary nodal distribution;
- stability;
- consistency;
- compact support;
- efficiency;
- delta function property;
- compatibility.

The principal limitation of mesh-free approximants, in comparison to the classical finite element methods is that in general it's more heavy, in a computational way, to compute the shape functions and their derivatives. We list below, for some methods, the way to approach to the shape functions construction:

- finite integral representation methods (ex. SPH) :

$$f(x) = \int_{x_1}^{x_2} f(\zeta) W(x - \zeta) d\zeta$$

- finite series representation methods (ex. MLS) or points interpolation methods (ex. PIMs) :

$$f(x) = a_0 + a_1 p_1(x) + ... a_n p_n(x)$$

- finite differential representation methods :

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2$$

It's possible to appreciate how the basis functions haven't a predefined form, like in simple finite element methods, but we need to calculate integrals or derivatives to find them out.

Generally the shape functions have the typical form of a radial function.

1.1 Application of meshfree methods

In a practical view, those methods have been used in geodesy, geophysics, mapping, or meteorology. Later, applications were found in many areas such as in the numerical solution of PDEs, artificial intelligence, learning theory, neural networks, signal processing, sampling theory, statistics

(kriging), finance and optimization. In particular their usage in approximating PDEs is the most important application in math fields. For example the usage of the RBF (radial basis functions) in order to approximating partial differential equations has been accelerated by Kansa's experience (1990); in this way Kansa gave an enormous contribute to the diffusion of mesh-free usage in approximating problems. PDEs are the mathematical expression of a lot of physic problems, for example the equations that operates on meteorology, in which mesh-free methods are essential to have a good approximations of natural phenomenon and so have trustworthy weather forecast. In our study area we can find application of mesh-free approximation scheme in those problems:

- Simulations where creating a useful mesh from the geometry of a complex 3D object may be especially difficult;
- Simulations where nodes may be created or destroyed, such as in cracking simulations;
- Simulations where the problem geometry may move out of alignment with a fixed mesh, such as in bending simulations;
- Simulations containing nonlinear material behavior, discontinuities or singularities.

Local max-ent approximation schemes and *Second order max-ent approximation schemes*, we are going to investigate are fully assimilated in mesh-free methods, in fact they have all the characteristics over described. The newness of this methods consists in the total generality way the shape functions are constructed with. The name, in fact, reveals the basis of the methods: the maximum entropy principle. The method's esprit is to have the most adaptive and general shape functions possible. The classical entropy definition is well known in thermodynamics and refers to the measure of disorder of a system. This approach is very indicated for mesh-free in fact with a random set of nodes, as it will be possible to appreciate, the generic shape function is constructed only depending from the relative positions of all the other nodes.

1.2 Work pourposes

In the following chapters we purpose to delineate the features of both *Local max-ent approximation schemes* and *Second order max-ent approximation schemes*. First of all we'll give the mathematical basis to the construction of the shape functions, with particular attention to the constraints which rule over the problem. We want also to give a consistent representation of those approximants so we'll propose some figures to make easier the comprehension of certain features of our schemes. Then we'll point out that which we consider to be the most relevant part of our program: the construction of the first and second derivatives of the basis functions; this aspect is very important cause their correct definition allows us to solve directly PDEs. At least, after having tested the approximation schemes for simple cases, we'll face some elastic problems with different type of grids, in order to being able to make a concrete critical analysis of the method. In the last part of the work we'll test both the approximation schemes to solve elastic problems with regular and random grids of nodes, for convex and concave domain.

Chapter 2

Local *max-ent* approximation schemes (LME)

In this chapter we purpose to outline the principal features of the first method we want to illustrate. First we give a general review of which are the mathematical characteristics of this kind of approximants, by starting from the simple domain constraints, to the more specific analytic conditions. Then we move from the *Global max-ent approximation schemes*, which are the first empiric attempt of building consistent approximants, to *Local max-ent approximation schemes* which are the first type of schemes we want to investigate. We also focus on some mathematical and physicist peculiarity which are very important for a complete comprehension of the background of the method.

2.1 Convex approximation schemes

We have, first of all, to define the type of set of nodes we have to work with. Before starting the introduction of *max-ent* approximants it's important to remember the majority of approximation schemes belong to the class of the *convex approximation schemes*; also our LMEs be part of this large family. In particular the convex approximation schemes are characterized by two important features:

- the positivity of all the shape functions;
- being exact on affine function.

In specific those conditions [1] do not allow to define an unique approximation scheme, but are essential, we others ancillary conditions to obtain our goal.

We define a set of nodes $\mathbf{X} = \{\mathbf{x}_a, a = 1, ..., N\} \subset \mathbb{R}^d$ a convex hull of $\mathbf{X}$ if and only if:

$$conv\mathbf{X} = \mathbf{x} \in \mathbb{R}^d | \ \mathbf{x} = \mathbf{X}\boldsymbol{\lambda}, \quad \boldsymbol{\lambda} \in \mathbb{R}^N_+, \quad \mathbf{1} \cdot \boldsymbol{\lambda} = 1 \tag{2.1}$$

Let $u : conv\mathbf{X} \to \mathbb{R}$ be a function whose values $\{u_a; a = 1, ..., N\}$ are known on the node set, we wish to construct approximations to u of the form

$$u^h(\mathbf{x}) = \sum_{a=1}^{n} p_a(\mathbf{x}) u_a \tag{2.2}$$

where $p_a : conv\mathbf{X} \to \mathbb{R}$ can be considered *shape functions*, in fact they are the coefficients of a linear combination of u_a. We want also to construct these shape functions, in way they have three specific proprieties [1]:

$$p_a(\mathbf{x}) \geq 0, \quad \forall \mathbf{x} \in conv\mathbf{X}, a = 1, ..., N \tag{2.3}$$

$$\sum_{a=1}^{n} p_a(\mathbf{x}) = 1, \quad \forall \mathbf{x} \in conv\mathbf{X} \tag{2.4}$$

$$\sum_{a=1}^{n} p_a(\mathbf{x})\mathbf{x}_a = \mathbf{x}, \quad \forall \mathbf{x} \in conv\mathbf{X} \tag{2.5}$$

These three conditions guarantee that affine functions are exactly reproduced by the convex approximants. In particular, with (2.4), in every single point of $conv\mathbf{X}$ we want the sum of the values of the shape functions being equal to 1 and with with (2.5) we impose that a linear situation must be reproduced exactly by a simple linear combination of $\mathbf{x}_a$ whose coefficients are precisely the values of the shape functions in the considered point of the convex hull.

However the newest implication of this 3 proprieties, is that if we consider both the propriety (2.3) and the propriety (2.4) we can interpret the *shape functions* as *probability distributions*. Finally we can also write a vector: $p(x)$ whose entires are: $\{p_1(\mathbf{x}), ..., p_N(\mathbf{x})\}$ to contain all the components. By virtue of (2.4) and (2.5), this can be defined in this way:

$$P_x(X) = \{\mathbf{p} \in \mathbb{R}^N_+ \mid \mathbf{Xp} = \mathbf{x}, \mathbf{1} \cdot \mathbf{p} = 1\} \tag{2.6}$$

So we are sure this is a convex hull. Now the natural question is: if $P_x(X)$ is not empty, it's possible to define shape functions consistent with the constraints? To answer this question we refers to Charatheodory's theorem [10]:

Theorem 1 (CHARATHEODORY'S THEOREM)**.** *Let* C *denote a convex N-dimension cone and let* $z=\sum_{j=1}^{b} x_j$ *with* $x_j \subset C$*. Exist* $\{y_1, \dots y_n\} \subset C \mid z = \sum_{i=1}^{b} y_i$ *so that* $\forall i$ *it's possible to write:*

$$y_i = \epsilon_j x_j \quad with \quad 0 \leq \epsilon_j \leq 1$$

According to this theorem, if $b < N$ it's possible to demonstrate that at least N-d-1 points are not necessary in order to express $\mathbf{x} \in \text{conv}\mathbf{X}$ as convex combination of points. This allow us to consider single domains which are subsets of $\text{conv}\mathbf{X}$, in which it's always possible to associate ϵ_j to the probability distribution. For simplicity, in this work we consider always convex domains.

Fixed this important frameworks, we have now all the instruments to be able to start constructing the maximum entropy approximants.

2.2 Global *max-ent* approximants

Given that we have already established the affinity between the shape functions and probability distributions, we can now introduce the concept of entropy, which is preparatory to the construction of our approximation schemes. In particular in this section we want to build the bridge between the mathematical background and a consistent physical expression of the approximants. The right definition of entropy can be researched by introducing a mono-dimensional variable which is not a statistic one, but a probabilistic one. We call *Entropy* of this variable, a measure of the associated uncertainty. In a very first time, the term *entropy* was introduced by Clausius

linked to thermodynamic studies; then the same concept was adopted also in other scientific fields. In our area of interest, the uncertainty associated with a finite scheme can also be interpreted as the amount of information gained by realizing the random variable. To measure the uncertainty associated to the scheme (or the information entropy) we must refer to *Shannon's theorem* [24]. Starting from the concept that the uncertainty of a variable is maximum when the probability associated with it's 50%, we can must recall this theorem to give a measure of the information entropy:

Theorem 2 (SHANNON'S THEOREM)**.** *Let* $\boldsymbol{X}$ *be the group of values that variable* $\boldsymbol{a}$ *may assume and let* $p(\boldsymbol{x}) = P_r(\boldsymbol{a} = \boldsymbol{x})$ *be the probability that variable* $\boldsymbol{a}$ *affects exactly* $\boldsymbol{x}$*, then it's possible to define the* ***information entropy*** *of varible* $\boldsymbol{a}$*:*

$$H(\mathbf{x}) = E_{\mathbf{x}}[I'(\mathbf{x})]$$

where $I'(\mathbf{x})$ *is the* ***auto-information****, namely the contribution of every single bit to the total entropy [15]. Applying this concept and introducing the expression of the probability function, we can express the entropy of any finite convex scheme in this way:*

$$H(p_1, ..., p_n) = -\sum_{a=1}^{n} p_a \log p_a \tag{2.7}$$

this theorem has also an important corollary:

Corollary 1 (SHANNON'S THEOREM)**.** *Let* $H(\boldsymbol{x})$ *be the information entropy of the variable* $\boldsymbol{a}$*, this functions has the following properties:*

$H(\boldsymbol{x})$ *is:*

- *non-negative;*
- *symmetric;*
- *continuous;*
- ***strictly concave*** *in the domain;*

and owns all the classical proprieties in general the entropy has.

Clearly, it's possible to recover the sense of the auto-information, explained by Shannon's theorem, in the expression adopted in (2.7): the auto-information is the contribution to the total system entropy given by any single bit (node) according to its occurrence probability.
Finally we can regard to every single point of a finite convex scheme as a complete system of events, where $\{p_1(\mathbf{x}), ..., p_n(\mathbf{x})\}$ are the corresponding probabilities and $H(p_1(\mathbf{x}), ..., p_n(\mathbf{x}))$ are the entropy linked to the finite scheme. From this point of view, to reproduce a function from scattered points can be regarded as a statistical inference problem, in fact, following this approach, Eq.(2.2) expresses the expected value of $u^h(\mathbf{x})$ as determined by probabilities $(p_1(\mathbf{x}), ..., p_n(\mathbf{x}))$.
Our goal is to create a process of inference which could be *unbiased* and depending just from the knowledge of the function, without any other assumption. To reach our purposes we have to refer to the *"Principle of maximum entropy"* by *Edwin T. Jaynes*; according to Jaynes [13] the least biased probability distribution is that which maximize entropy.

Theorem 3 (JAYNES' PRINCIPLE OF MAXIMUM ENTROPY)**.** *The maximum entropy distribution is uniquely determined as the one which is maximally non committal with regard to missing information, in that it agrees with what is known, but expresses the maximum uncertainty with respect to all other matters.*

So, following Jaynes' principle and looking to the problem with an informational-theoretical viewpoint, it's possible to find the optimal convex approximation schemes by solving the problem:

$$\text{(MaxEnt)} \quad \text{maximize} \quad H(\mathbf{p}) = -\sum_{a=1}^{n} p_a \log p_a \tag{2.8}$$

subject to Eq: (2.3), (2.4), (2.5).

Jaynes looks at the entropy like a measure of the amount of the information of the initial distribution, in this way by maximizing the entropy, it's possible to find a distribution which contains the minimum amount of consistent information. Finally Jaynes proved that the normal (standard) distribution ($\sigma = 1, \mu = 0$) is the only one able to maximize the entropy in a continuous space.

Now, looking carefully at the problem with a mathematical viewpoint, we notice that (2.7) is strictly concave in its domain (see corollary of Shannon's theorem), so, due to the strictly convexity of the set of point $\mathbf{X}$ the solution of the *MaxEnt* problem is unique. This proposition is very simple to prove by only applying *Weierstrass* extreme value theorem and considering the convexity of the domain. This method can provide to find the shape functions and to build up the basis for constructing conforming elements in a convex set of point, however, it conceals some problems. By looking to figure 2.1

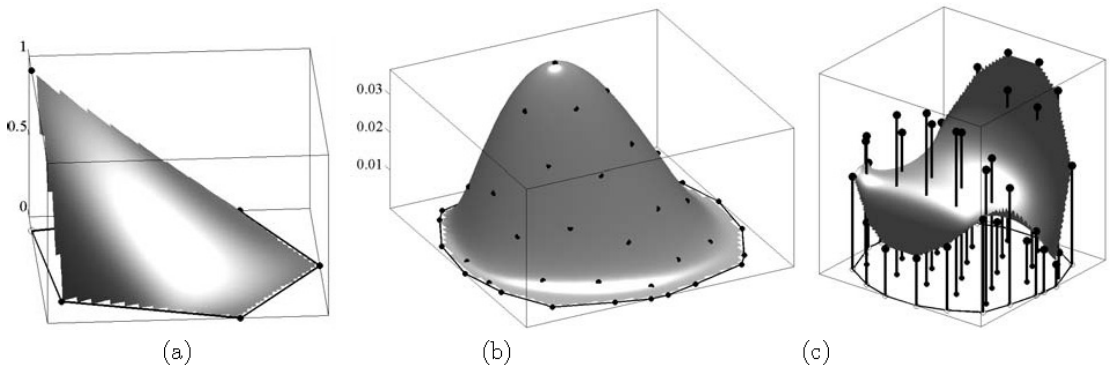


Figure 2.1: Examples of *max-ent* approximation schemes. **a)** Vertex shape function illustrating *Delta-Kroneker* propriety. **b)** Interior node shape function showing the character of *max-ent* program. **c)** Example of *max-ent* approximation illustrating the very poor interpolation character of those kind of approximation schemes.

we can appreciate how, in a pentagon domain, with only border nodes, **[a)]** the *Delta-Kroneker* propriety is satisfied and the restriction of *max-ent* shape functions on the edge is correctly linear; however, in example **[c)]** it's clear that the *max-ent* program has a very poor interpolation of initial data even if the general shape function of an interior node **[b)]** has in general the correct behavior of vanishing at the boundary (but is highly non local).

2.3 Local *max-ent* approximants

In chapter § 2.2 we have established the global *max-ent* approximation schemes are not able to satisfy the necessity of good interpolation and locality on the convex domain. So we have to introduce some changes in the program, principally in order to control the degree to which the value at $\mathbf{x}$ is correlated to nearby nodal values.
To localize the problem, we need to introduce the concept of width of shape function, which can

be defined in this way:

$$w|\mathbf{p}_a| \qquad \int_\Omega p_a(\mathbf{x})|\mathbf{x}-\mathbf{x}_a|^2 d\mathbf{x} \tag{2.9}$$

where we define $\Omega = conv\mathbf{X}$. Remembering we are reasoning in a probabilistic-inferential way, we can define Eq.(2.9) a second moment of $\mathbf{p}_a$ around $\mathbf{x}_a$. This concept can be compared to the concept of dispersion in a Probability Density Function (PDF) around the mean value:

$$E[\mathbf{x}-\mu_{\mathbf{x}}] \tag{2.10}$$

So we can define the local approximation scheme as the summary in every $\mathbf{x}$ of the width of any shape function:

$$W|\mathbf{p}| = \sum_{a=1}^{n} w|p_a| = \int_\Omega \sum_{a=1}^{n} p_a(\mathbf{x})|\mathbf{x}-\mathbf{x}_a|^2 d\mathbf{x} \tag{2.11}$$

The most local approximation scheme is that which minimize (2.11) subjected to the constraints (2.3), (2.4), (2.5). Thus the functional does not involve any derivatives its minimization can be performed pointwise by this way:

$$(RAJ) \qquad \text{For fixed } \mathbf{x} \text{ minimize} \qquad U(\mathbf{x},\mathbf{p}) \equiv \sum_{a=1}^{n} p_a(\mathbf{x})|\mathbf{x}-\mathbf{x}_a|^2 d\mathbf{x} \tag{2.12}$$

We refers to Eq. (2.12) as *Rajan convex approximation schemes* cause Rajan showed [21] that if the nodes are in general position (meaning that there are no 3 col-linear points in bi-dimensional cases or 4 cospherical points in three-dimensional cases) there is an unique solution to RAJ corresponding to the piece-wise affine shape functions supported by the unique Delanuay triangulation [21] associated with the node set $\mathbf{X}$. Clearly this solution is optimal in the sense of the width.

At this point we have introduced two principles: one about the maximum entropy and one about the maximum locality, to define a convex approximation scheme. However we need to conciliate the two criteria, because, generally, it's very hard to find a simultaneously convex approximation scheme that agrees to both the conditions.

2.3.1 Local *max-ent* approximation schemes as a Pareto set

To carry on with our presentation we need to single out an economic principle: the *"Pareto Optima"*. It was introduced by the italian economist Vilfredo Pareto. The *Pareto efficiency* is a state of allocation of resources in which it's impossible to make any one individual better off without making at least one individual worse off. Given an initial allocation of goods among a set of individuals, a change to a different allocation that makes at least one individual better off without making any other individual worse off is called a Pareto improvement. An allocation is defined as *Pareto efficient* or *Pareto optimal* when no further Pareto improvements can be made.

Theorem 4 (PARETO'S PRINCIPLE)**.** *Consider a system with function $f : \mathbb{R}^n \to \mathbb{R}^m$ where $\boldsymbol{X}$ is a compact set of feasible decisions in the metric space $\mathbb{R}^n$, and $\boldsymbol{Y}$ is the feasible set of criterion vectors in $\mathbb{R}^m$ such that:*

$$Y = \{y \in \mathbb{R}^n : y = f(x), x \in \mathbf{X}\}$$

*Assuming that the preferred directions of criteria values are known, a point $y" \in \mathbb{R}^m$ is preferred to another point $y' \in \mathbb{R}^m$, written as, $y" > y'$. The **Pareto set** is thus written as:*

$$P((Y)) = \{y \in \mathbb{R}^m : \{y" \in \boldsymbol{Y} : y" > y', y" \neq y'\} = \oslash\}$$

In our background this solution is what we need, cause we have to harmonize the competing

objectives described before. Introducing the β parameter we can define the Pareto optimal solution of the problem:

$$(LME)_\beta \qquad \text{For fixed } \mathbf{x} \text{ minimize} \qquad f_\beta(\mathbf{x},\mathbf{p}) \equiv \beta U(\mathbf{x},\mathbf{p}) - H(\mathbf{p}) \tag{2.13}$$

with the restriction imposed by Eq. (2.3), (2.4), (2.5),.

It is simple to recover the program (2.13) degenerates in (ME) if $\beta = 0$ and in (RAJ) if $\beta = +\infty$. This problem has multiple solutions in general, but the only element of (RAJ) with maximum entropy is in Pareto set. Remember that, cause $H(p)$ it's strictly concave in (RAJ) the Pareto-optimal for $\beta = +\infty$ is unique and can be described by:

$$p_\infty^{Pareto}(\mathbf{x}) = arg \quad \max \quad H(p) \tag{2.14}$$

Since for $\beta \in [0, +\infty)$ the function $f(\mathbf{x}, \cdot\,)$ is continuous and convex in $P_{\mathbf{x}}(\mathbf{X})$ the $(LME)_\beta$ has an unique solution: $p_\beta(\mathbf{x})$ if and only if $\mathbf{x} \in conv(\mathbf{X})$. We call the convex approximation scheme defined by $p_\beta(\mathbf{x})$: *local max-ent convex approximation scheme.*

2.3.2 Optimization program and exponential form of the shape functions

We purpose now to give an explicit expression of the shape functions so as to be able to calculate it in every point of the domain. To do this we have, first, to rewrite Eq. (2.5) in this way:

$$\sum_{a=1}^{N} p_a(\mathbf{x})(\mathbf{x}_a - \mathbf{x}) \equiv \mathbf{Y}\mathbf{p} = 0 \tag{2.15}$$

where $\mathbf{Y}$ is a matrix whose columns are $\mathbf{x}_a - \mathbf{x}$. Now, introducing a Lagrangian formulation of the problem, $(LME)_\beta$ we obtain:

$$L(\mathbf{p}, \lambda_{\mathbf{0}}, \boldsymbol{\lambda}) = \int_\beta (\mathbf{p}) + \lambda_0(\mathbf{1}\cdot\mathbf{p} - 1) + \boldsymbol{\lambda}\cdot\mathbf{Y}\mathbf{p} \tag{2.16}$$

in which we can recognize $\boldsymbol{\lambda} \in \mathbb{R}^d$ and $\lambda_{\mathbf{0}} \in \mathbb{R}$: the Lagrange multipliers. The problem (2.16) has an unique solution, if it's defined in a convex hull and if $\beta \in [0, +\infty)$. It's possible to prove that there is only a couple of $(, \boldsymbol{\lambda}_0^*, \boldsymbol{\lambda}^*)$ that simultaneously satisfying the "$Karush - Kuhn - Tucker$" conditions and such that $\{p_\beta, , \boldsymbol{\lambda}_0, \boldsymbol{\lambda}\}$ is a saddle point of (2.16).

In mathematical optimization, the Karush–Kuhn–Tucker (KKT) conditions are first order necessary conditions for a solution in nonlinear programming to be optimal, provided that some regularity conditions are satisfied [10]. Allowing inequality constraints, the KKT approach to nonlinear programming and generalizes the method of Lagrange multipliers, which allows only equality constraints.

Theorem 5 (KARUSH-KUHN-TUCKER CONDITIONS)**.** *Given the following non linear optimization problem:*

$$\begin{aligned} &\min f(x) \\ &g_i(x) < 0 \\ &h_j(x) = 0 \end{aligned}$$

where $f(x)$ is the function we want to minimize, $g_i(x), (i = 1, \ldots, m)$ are mono-lateral restrictions

and $h_j(x), (j = 1, \ldots, l)$ *are bilateral restrictions. Suppose that* $f : \mathbb{R}^n \to \mathbb{R}$ *and that* $g_i : \mathbb{R}^n \to \mathbb{R}$, $h_j : \mathbb{R}^n \to \mathbb{R}$ *and suppose they are continuous differenziable around* $\boldsymbol{x^*}$. *If* $\boldsymbol{x^*}$ *it's a local minimum point and it satisfies both the constraints regularity conditions, then exist multipliers* $g_i : \mathbb{R}^n \to \mathbb{R}$ *and* $h_j : \mathbb{R}^n \to \mathbb{R}$ *thus that:*

- $$\nabla(f(x*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x*) + \sum_{i=1}^{l} \mu_j \nabla h_j(x*) = 0$$
- $$g_i(x*) \leq 0, \quad \forall i = 1, \ldots, m$$
- $$h_j(x*) = 0, \quad \forall j = 1, \ldots, l$$
- $$\lambda_i \geq 0 \quad (i = 1, \ldots, m)$$
- $$\lambda_i g_i(x*) = 0 \quad \forall i = 1, \ldots, m$$

The first condition refers to annulment of the gradient connected to the Lagrangian function; the second and the third are the admissibility constraints of the point $\mathbf{x^*}$; the fourth regards the non-negativity of the multiplier associated to the inequality constraints; the fifth means that the multiplier of the inactive constraints must be 0.

In order that the necessary KKT conditions allow to find a minimum point we must refer to the corollary of the theorem:

Corollary 2 (KARUSH-KUHN-TUCKER CONDITIONS)**.** *To find a solution of the KKT minimizing problem its' sufficient to have regularity on point* $\boldsymbol{x^*}$. *The regularity of* $\boldsymbol{x^*}$ *is warranted by onl one of this conditions:*

- *Linear independence constraint qualification (LICQ);*
- *Mangasarian–Fromovitz constraint qualification (MFCQ);*
- *Constant rank constraint qualification (CRCQ);*
- *Constant positive linear dependence constraint qualification (CPLD);*
- *Quite-normality constraint qualification (QNCQ);*
- ***Slater's conditions**;*

The last point is interesting for our pourposes. Slater's conditions state that if we have a convex optimization problem there is surely an interior point of $conv(\mathbf{X})$ where the equality contraints are satisfied and the inequality constraints are strictly <0.

We can note that every interior point: $\mathbf{x} \in int(conv(\mathbf{X}))$ satisfy, in LME_β, the Slater's conditions. So we can derive the expression of the *Local Max-ent approximants* by finding the unique solution of the Lagrangian problem (2.16):

$$p_{\beta a}(\mathbf{x}) = \frac{1}{Z(\mathbf{x}, \boldsymbol{\lambda}^*(\mathbf{x}))} \exp[-\beta|\mathbf{x} - \mathbf{x}_a|^2 + \boldsymbol{\lambda}^*(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_a)] \qquad a = 1, ..., N \tag{2.17}$$

where:

$$\boldsymbol{\lambda}^*(\boldsymbol{x}) = \arg\min \log Z(\mathbf{x}, \boldsymbol{\lambda}) \tag{2.18}$$

with:

$$Z(\mathbf{x}, \boldsymbol{\lambda}) \equiv \sum_{a=1}^{N} \exp[-\beta|\mathbf{x} - \mathbf{x}_a|^2 + \boldsymbol{\lambda}^*(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_a)] \tag{2.19}$$

In fact the shape function has an exponential form, so to calculate the solutions of the minimizing program (2.18) we can avoid to introduce: $\log(Z)$. Considering Slater's conditions, we know there is a solution of Eq. (2.18), now we want to prove the solution is unique. Like for any other function, we write first the gradient and then the hessian of the objective function:

$$\mathbf{r}(\mathbf{x}, \boldsymbol{\lambda}) \equiv \partial_{\boldsymbol{\lambda}} \log Z(\mathbf{x}, \boldsymbol{\lambda}) = \sum_{a=1}^{N} p_a(\mathbf{x}, \boldsymbol{\lambda})(\mathbf{x}_a - \mathbf{x}) \tag{2.20}$$

$$\mathbf{J}(\mathbf{x}, \boldsymbol{\lambda}) \equiv \partial_{\boldsymbol{\lambda}} \partial_{\boldsymbol{\lambda}} \log Z(\mathbf{x}, \boldsymbol{\lambda}) = \sum_{a=1}^{N} p_a(\mathbf{x}, \boldsymbol{\lambda})(\mathbf{x}_a - \mathbf{x}) \otimes (\mathbf{x}_a - \mathbf{x}) - \mathbf{r}(\mathbf{x}, \boldsymbol{\lambda}) \otimes \mathbf{r}(\mathbf{x}, \boldsymbol{\lambda}) \tag{2.21}$$

Both in the gradient Eq: (2.20) and in the hessian function Eq: (2.21), $p_a(\mathbf{x}, \boldsymbol{\lambda})$ denotes the evaluation of (2.17) at an arbitrary value $\boldsymbol{\lambda}$ of the Lagrange multiplier; and as we expected we can found the 1^{st} order consistency condition.

Now if we consider a non-zero vector $\mathbf{u} \in \mathbb{R}^d$ and let $u_a = \mathbf{u} \cdot (\mathbf{x} - \mathbf{x_a})$, since by assumption $aff(\mathbf{X} \in \mathbb{R}$, it follows that not all u_a are identical. We also, already, know from (2.17) that $p_a(\mathbf{x}, \boldsymbol{\lambda}) > 0$, then we can write the square function associated and by the strictly convexity propriety we have:

$$\mathbf{u} \cdot \mathbf{J}(\mathbf{x}, \boldsymbol{\lambda}) \cdot \mathbf{u} = \sum_{a=1}^{N} p_a(\mathbf{x}, \boldsymbol{\lambda}) u_a^2 - \left(\sum_{a=1}^{N} p_a(\mathbf{x}, \boldsymbol{\lambda}) u_a \right)^2 > 0 \tag{2.22}$$

So $\mathbf{J}(\mathbf{x}, \boldsymbol{\lambda})$ is positive define and therefore $\log(Z)$ is strictly convex and the minimizer is unique since we are working in a convex domain.

2.3.3 Phisicist interpretation of the shape functions

In the abstract of our work, we have already sketched the importance of the physicist basis which have inspired the construction of those approximants. Now we want to underline the importance of β parameter in *Pareto optimization problem* and try to give it not just a mathematical significance, but also a physicist one.
First we must introduce the meaning of *Partition function* as we usually call Eq. (2.19). In physics, a partition function describes the statistical properties of a system in thermodynamic equilibrium (which in fact is a Pareto optima of the problem). We can define the Partition function in this way:

$$Z = \sum_{s} e^{-\beta \cdot E_s}$$

where:

- $\mathbf{s}$ is the index for the micro-states of the system;
- $\boldsymbol{\beta}$ is the thermodynamic beta;
- $\mathbf{E_s}$ is the total energy of the system in the respective micro-state.;

According to the second law of thermodynamics, a system assumes a configuration of maximum entropy at thermodynamic equilibrium , which recalls (2.8), in fact for a generic system the entropy can be expressed in this way:

$$H(p) = -k_\beta \sum_i p_i \ln p_i$$

subjected to:

- unity propriety;

$$\sum_i p_i = 1 \quad \text{which is similar to} \quad \sum_{a=1}^{n} p_a(\mathbf{x}) = 1,$$

- fixed average energy;

$$\sum_i p_i E_i = U \quad \text{which is similar to} \quad \sum_{a=1}^{n} p_a(\mathbf{x})\mathbf{x}_a = \mathbf{x}$$

in fact, in a probabilistic view, the energy of the generic configuration i can be expressed as $E_a = |\mathbf{x_a} - \mathbf{x}|^2$. Now if we solve the associated Lagrange problem, p_i fields are:

$$p_i = \frac{1}{Z} e^{\frac{1}{k_\beta}(-1+\lambda_1)}$$

in which we can easily recover the analytic form of the approximants described by the LME problem (2.17). At last we can define:

$$\beta = \frac{1}{k_\beta T}$$

where k_β is Boltzmann's constant and T is the absolute temperature. By this standpoint we can regard to the Rajan problem as the zero-limit temperature of the thermalization problem, in fact:

$$T = 0 \qquad \rightarrow \qquad \beta = +\infty$$

instead the generic thermalization problem may be regarded as the LME_β problem in Eq. (2.13)

2.4 Monodimensional domain shape functions

The 1D domain is the simplest but also the one which allow us to have a better comprehension of the behavior of our approximants. For what we have exposed in chapter §(2.3.3) it's clear the parameter that states on the shape functions is β. Here we want to represent the 1D Local *max-ent* approximants for different values of these parameter so that it's clear the influence of β. In figure 2.2, 2.3, 2.4, 2.5, we use the same grid with 11 nodes and 1000 points to make easier the comparison. In this work we distinguish between *nodes* and *points*. We call:

- NODES: the grid *points* in which we construct the shape functions. Every node has its shape function.
- POINTS: the grid *points* in which we compute the value of every single shape function. The increasing of the number of points allows to have a better representation of the shape functions.

Clearly to have a good comparison between different cases, we necessitate to use few nodes and many points.

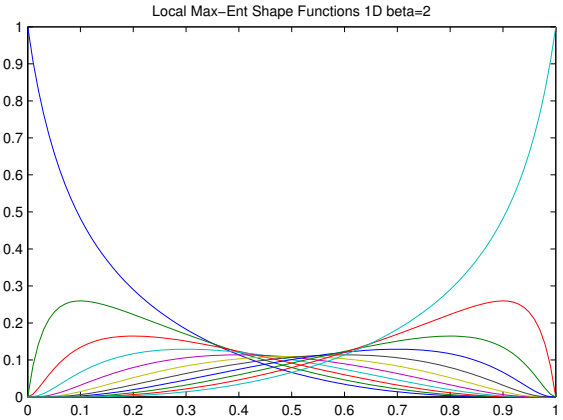


Figure 2.2: 1D shape functions represented with 11 nodes and 1000 points; β=2

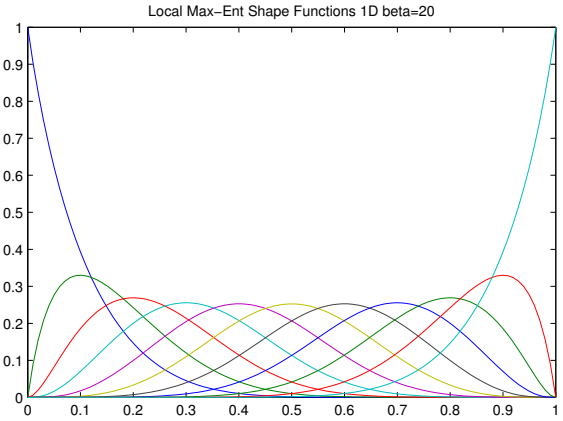


Figure 2.3: 1D shape functions represented with 11 nodes and 1000 points; β=20

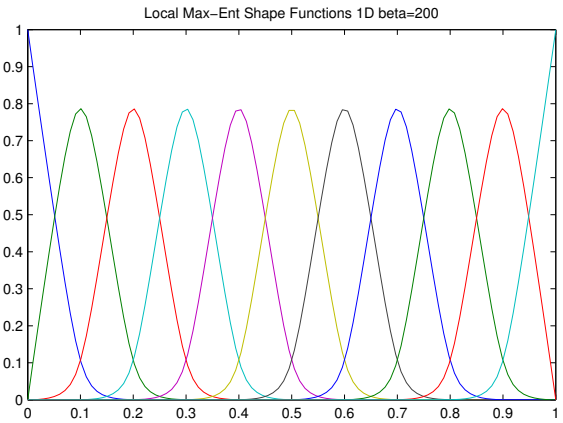


Figure 2.4: 1D shape functions represented with 11 nodes and 1000 points; β=200

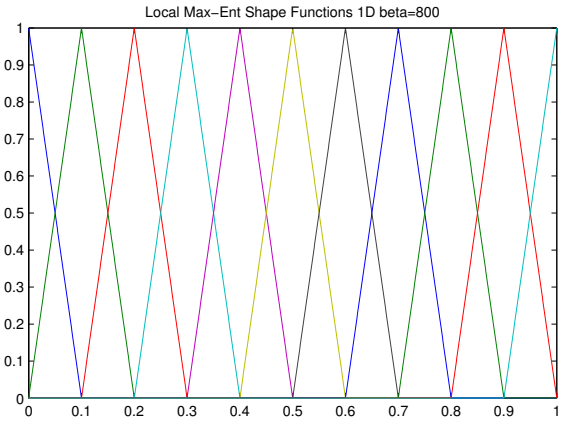


Figure 2.5: 1D shape functions represented with 11 nodes and 1000 points; β=800

We can appreciate how, when β has low values, the $Max - Ent$ program is found, in fact the shape functions extend on all the domain. Instead, when we increase the value of β we recover the $Rajan$ program, so the shape functions have the maximum degree of locality.

Now we extend the treatise also to a random grid of nodes, in fact in our purpose, those approximants must be more general as possible so we want to see if the Eq. (2.3), (2.4), (2.5), which are the principal constraints of the method, are respected. In figure 2.6 we adopt a random grid for the nodes and also a random grid for the points, so as to be more general as possible, we also adopt a middle value of β. It's possible to note that all the proprieties are completely satisfied.

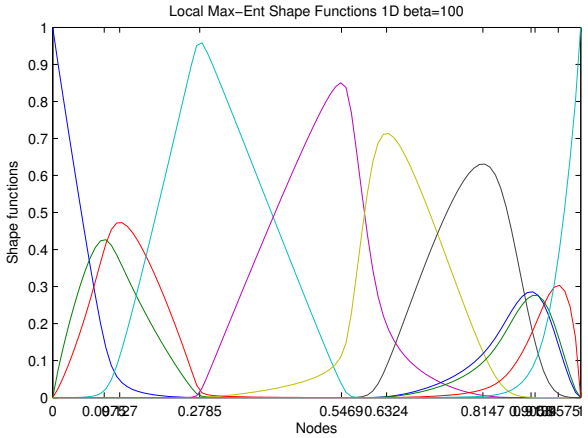


Figure 2.6: 1D shape functions represented with 11 nodes and 1000 points with a random grid; β=100

2.4.1 Example of a function approximation

According to Eq. (2.5), the shape functions, so as they are constructed, must guarantee a perfect approximation of any type of linear function and, if the method is good, they have to be able to reproduce any kind of function. In this section we want to test the effectively capability of the approximants to reproduce a simple analytic function. In the next examples we adopt the following strategy: we use an increasing number of nodes and a fixed number of evaluation points.

This is a useful strategy in order to have a good discretization of the analytic function and have an increasing better interpolation of the approximants. In this case, although, we have to use $Moore - Penrose$ pseudo-inverse method to invert a rectangular matrix. We want to underline that if we use same number of nodes, points and definition points for the analytic function, we obtain non significant results, in fact we have an error close to zero ($\simeq 10^{-17}$) as we expect due to propriety (2.5).

We use the analytic function: $f(x) = \sin(x)$ and we assume β increasing with the decreasing of the nodal spacing: in particular form the practical experience we can derive this formulation [1]:

$$\gamma = \beta h^2$$

where h is the medium nodal spacing between the nodes of the approximation scheme. In particular our purpose is to have a concrete comparison between a regular and a random grid of nodes. This is very important in order to find out if the approximants are well working also in a worse condition.

To prove the goodness of the results, it's possible to estimate the absolute error between the analytic function and the max-ent approximation. Since the functions are defined point-wise we need to calculate in every point of the domain the difference between the two functions and so the absolute error can be written in this way:

$$Err = \frac{\sqrt{\sum_{i=1}^{N} \|y_{i_{an}} - u_{i_{lme}}\|^2}}{\sqrt{\sum_{i=1}^{N} \|y_{i_{an}}\|^2}}$$

In figure 2.7 and 2.8 we show the decreasing of the error with the increasing of the number of the shape functions used to approximate the analytic one. The first figure refers to a regular condition, second to an irregular one.

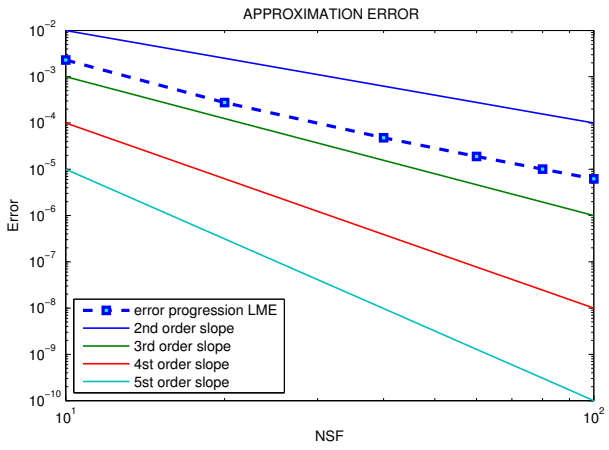


Figure 2.7: Approximation error with trend lines, regular grid

where NSF means the number of shape functions.

As we expected $Err \to 0$ when we increase the numbers of the shape functions we reproduce the analytic function with. In particular we note, that both for the regular grid then for the irregular one, the decreasing of the errors of LME program follow a tendency line with a slope

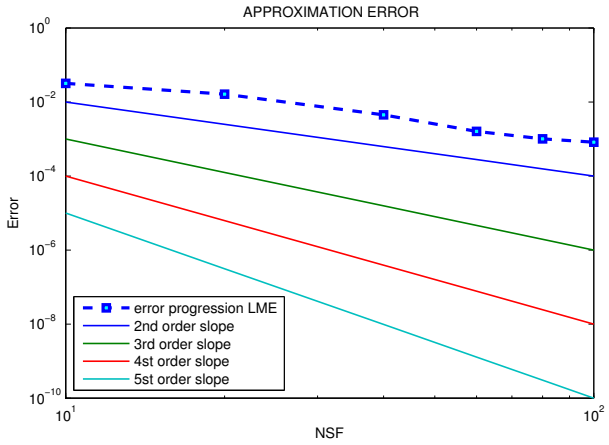


Figure 2.8: Approximation error with trend lines, irregular grid

quite equal to 2. This is very comfortable cause the method has the same behavior in both cases.

2.5 Bidimensional domain shape functions

We extend, now, our study, to the bi-dimensional case. This in general involve a more computational burden, but the main program does not change. The only thing we introduce is a new parameter: γ, which controls the degrees of locality of the approximants, according to Ortiz experience [1]. γ can be such defined:

$$\gamma = \beta h^2$$

where β is the *thermalization parameter* and h is a measure of the nodal spacing. Since the nodal spacing can be different along different directions, we can take as $"h"$ the lower between the values of the side of the domain (conservative choice). To facilitate the comparison between different values of γ, we have taken a regular distribution of nodes. In figure 2.9, 2.10, 2.11, 2.12 we have represented the shape functions for different values of parameter γ:
As in the mono-dimensional case, we can notice that if $\beta \to 0$ shape functions are similar to (ME) solution; if $\beta \to \infty$ the linear program (RAJ) can be recovered.

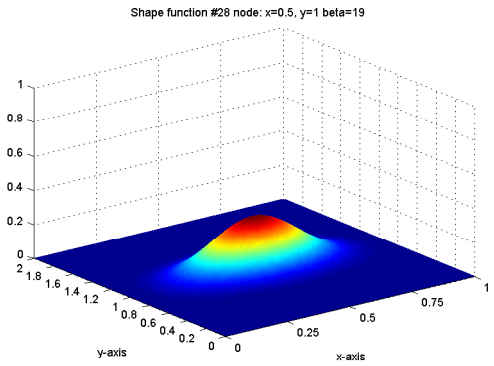


Figure 2.9: 2D shape function in rectangular domain with 55 nodes and 5500 points; γ=0.8

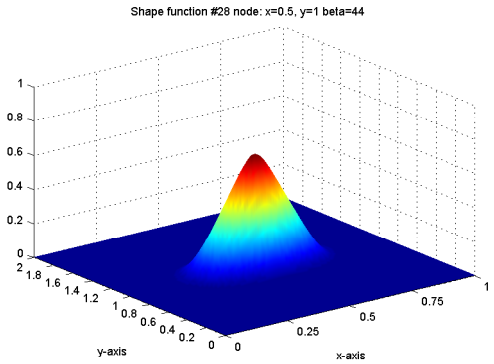


Figure 2.10: 2D shape function in rectangular domain with 55 nodes and 5500 points; γ=1.8

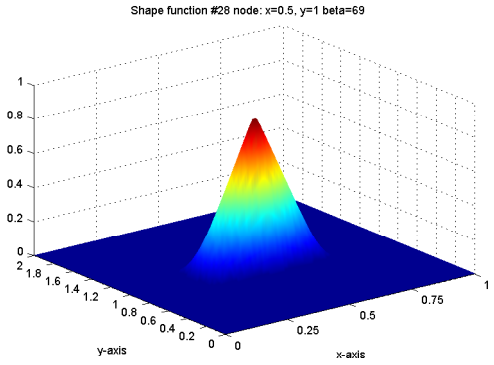


Figure 2.11: 2D shape function in rectangular domain with 55 nodes and 5500 points; γ=2.8

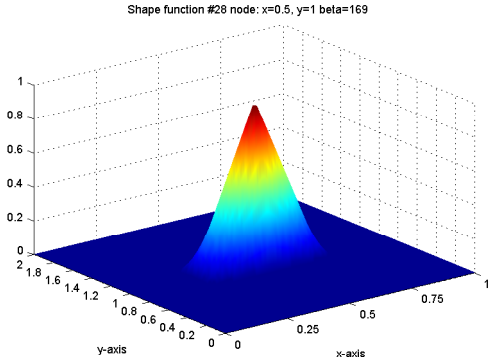


Figure 2.12: 2D shape function in rectangular domain with 55 nodes and 5500 points; γ=6.8

For the same reason expressed in chapter § 2.4 we want to show the generality of method also with a scattered nodes distribution. We take, for example, the random set of nodes defined in a square domain (L=1), represented in figure 2.13: in this example we adopted the strategy to keep

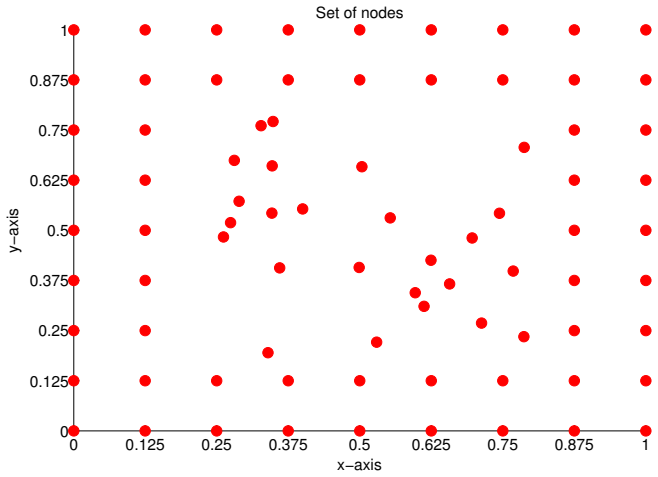


Figure 2.13: Random 2D set of nodes

constant the distribution of the nodes on the boundary and next to the boundary. This choice is useful in order to have a simple comparison with the second order consistency approximants, as suggested by M.Ortiz and M.Arroyo [2].
By taking, for example, node 51, we obtain the associated shape function (see fig. 2.14)

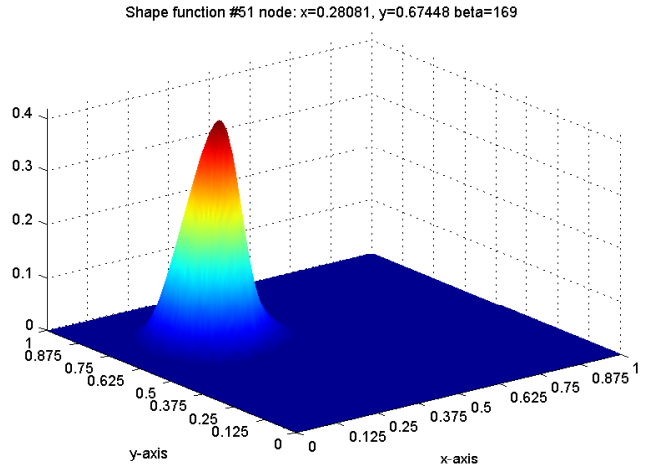


Figure 2.14: Generic shape function of a node of the random set; γ=1.8

With this example it's possible to better appreciate the locality propriety also with scattered nodes.

2.5.1 Exemple of a function approximation

For the same reasons exposed in section § 2.4.1, we are going to test the LME_β program in a bi-dimensional domain. this doesn't involve any significant variation from the 1D solution, but in this case we use a quite more difficult function. Of course we use more points then nodes, in fact if use the same number of nodes and definition points, due to (2.5), we have an error close to 0 ($\simeq 10^{-16}$).
Analytic function:

$$f(x,y) = \sin\Big(3(-x^2 - y^5)\Big) \cdot \cos\Big(3(-x^5 - y^2)\Big)$$

As in 1D case, we want to make a comparison between regular grid and irregular one to test if the program is really able to reproduce the function in any condition. To prove the goodness of the results, it's possible to estimate the absolute error between the analytic function and the max-ent approximation. Since the functions are defined point-wise we need to calculate in every point of the domain the difference between the two functions and so the absolute error can be written in this way:

$$Err = \frac{\sqrt{\sum_{i=1}^{N} \|z_{i_{an}} - z_{i_{lme}}\|^2}}{\sqrt{\sum_{i=1}^{N} \|z_{i_{an}}\|^2}}$$

In figure 2.15 and 2.16 we show the decreasing of the error with the increasing of the number of the shape functions used to approximate the analytic one. We use parameter γ=1.

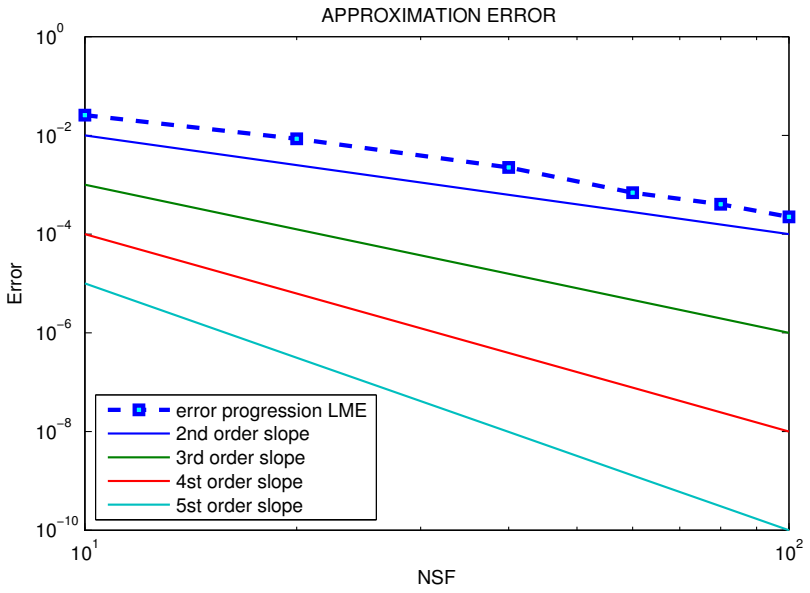


Figure 2.15: Approximation error with associated tendency lines; regular grid

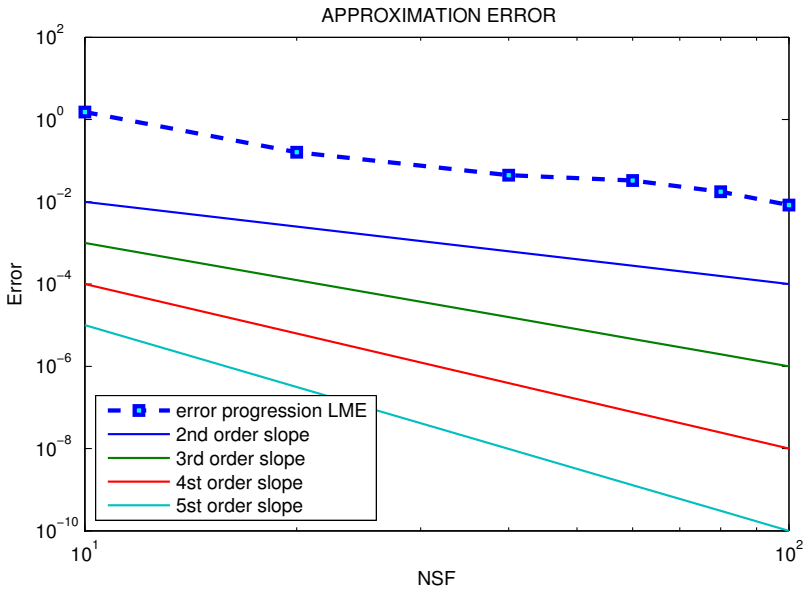


Figure 2.16: Approximation error with associated tendency lines; irregular grid

Like in 1D case, we can recognize $Err \rightarrow 0$ with the increasing of the number of the shape functions. For the regular case we have a decreasing line with slope $\simeq 2$. It's interesting to note for the irregular grid e obtain quite the same slope, even if we have bigger errors. In general the approximation errors are bigger then 1D case, this due to the more complexity of the function we have investigated.

Chapter 3

First derivatives of the shape functions

In this chapter our goal is to compute a close mathematical form of the first derivative of the shape functions. The gradient of LME_β is very important in order to have at our disposition the instruments to solve PDEs. To be able to evaluate the derivatives in an analytic way could be an advantage since we can avoid to implement any finite difference methods, which could be more onerous in computational terms.

We have remarked in chapter § 1, the shape functions has an exponential form, but the smoothness of the local max-ent approximants is not guaranteed *a priori* since our program is characterized point-wise so we are not sure they are derivable [2].

Proposition 1 (CONTINUITY CLASS)**.** *Consider aff*$\mathbf{X} = \mathbb{R}^d$ *and let* β*: conv*$\mathbf{X}$ *be* C^r *in int conv*$\mathbf{X}$*. Then the local max-ent shape functions are of class* C^r *in int conv*$\mathbf{X}$*.*

This proposition is readily checked by considering (2.15), in fact due to (2.5) it's clear that:

$$\sum_{a=1}^{N} p_a(\mathbf{x})(\mathbf{x}_a - \mathbf{x}) = \mathbf{r}(\mathbf{x}, \boldsymbol{\lambda}^*) = 0 \tag{3.1}$$

and for (2.21) we also sure that:

$$\det \partial_\lambda \mathbf{r}(\mathbf{x}, \boldsymbol{\lambda}^*) \neq 0$$

$$\boldsymbol{\lambda}^* \in C^r$$

and so the shape functions are of class C^r.
We are now allowed to calculate ∇p_a^*. First we introduce f_a which is the argument of the exponential function in (2.17) :

$$f_a(\mathbf{x}, \boldsymbol{\lambda}, \beta) = -\beta|\mathbf{x} - \mathbf{x}_a|^2 + \boldsymbol{\lambda}^*(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_a); \tag{3.2}$$

so we can rewrite the expression of the generic shape function in this way:

$$p_a(\mathbf{x}, \boldsymbol{\lambda}, \beta) = \frac{\exp[f_a(\mathbf{x}, \boldsymbol{\lambda}, \beta)]}{\sum_b \exp[f_b(\mathbf{x}, \boldsymbol{\lambda}, \beta)]} \tag{3.3}$$

Introducing now the Lagrangian problem:

$$g(\boldsymbol{\lambda}) = -\log\Big\{ \sum_a \exp[f_a(\mathbf{x}, \boldsymbol{\lambda}, \beta)] \Big\} \tag{3.4}$$

we have to underline 2 important aspects which controls the first derivatives:

- $\boldsymbol{\lambda}(\mathbf{x})$: which means the derivatives of Lagrange multiplier is involved in the calculation;
- β could also be considered non-constant, so that $\beta(\mathbf{x})$. In this case we introduce another unknown in the minimizing problem and in the derivation one. According to Ortiz [1] is not useful to consider $\beta(\mathbf{x})$, in fact in practical cases β is assumed constant.

it' simply to recover:

$$\nabla p_a^* = p_a^* \left(\nabla f_a^* - \sum_b p_b^* \nabla p_b^* \right) \tag{3.5}$$

we can apply the chain rule to compute every single part of ∇f_a^*:

$$\nabla f_a^* = \left(\frac{\partial f_a}{\partial \mathbf{x}} \right)^* + \left(\frac{\partial f_a}{\partial \boldsymbol{\lambda}} \right)^* \cdot D\boldsymbol{\lambda}^* + \left(\frac{\partial f_a}{\partial \beta} \right)^* \cdot \nabla \beta \tag{3.6}$$

and we do non consider the last addend since β is constant. It's now easy to check:

$$\left(\frac{\partial f_a}{\partial \mathbf{x}} \right)^* = -2\beta(\mathbf{x} - \mathbf{x}_a) + \boldsymbol{\lambda}^*(\mathbf{x}), \qquad \left(\frac{\partial f_a}{\partial \boldsymbol{\lambda}} \right)^* = (\mathbf{x} - \mathbf{x}_a)$$

the only term which is unknown is $D\boldsymbol{\lambda}^*$. To compute it we must refer to (3):

$$D\mathbf{r}^* = \left(\frac{\partial \mathbf{r}}{\partial \mathbf{x}} \right)^* + \left(\frac{\partial \mathbf{r}}{\partial \boldsymbol{\lambda}} \right)^* \cdot D\boldsymbol{\lambda}^* = 0$$

by considering that:

$$\left(\frac{\partial \mathbf{r}}{\partial \mathbf{x}} \right)^* = \mathbf{J}^*, \qquad \left(\frac{\partial f_a}{\partial \boldsymbol{\lambda}} \right)^* = -2\beta \mathbf{J}^* \mathbf{I_d}$$

where $\mathbf{I_d}$ is the identity matrix in the dimensions of the problem. It follows that:

$$D\boldsymbol{\lambda}^* = 2\beta \mathbf{J}^* \mathbf{I_d} - \mathbf{J}^{*-1}$$

now by substituting the last expression in(3.6) we obtain the final expression of the first derivatives of the approximants:

$$\nabla p_{\beta a}(\mathbf{x}) = -p_{\beta a}(\mathbf{x}) \mathbf{J}(\mathbf{x}, \boldsymbol{\lambda}^*(\mathbf{x}))^{-1} (\mathbf{x} - \mathbf{x}_a) \tag{3.7}$$

where $\mathbf{J}^*$ is the Hessian of the objective function described in (2.21).

3.1 Continuity and differentiability of the shape functions

The LME_β first derivatives have some important proprieties $\forall \mathbf{x} \in conv(\mathbf{x})$ we have to fix to better know the behavior of the approximants.

Proposition 2 (Continuity Propriety)**.** *: Let $\mathbf{x} \in conv(\mathbf{X})$. If $p_\beta(\mathbf{x})$ is the unique solution of LME_β then $p_\beta(\mathbf{x})$ is a continuous function of β in $[0; +\infty)$.*

From this propriety derives an immediate corollary:

Corollary 3.

$$\lim_{\beta \to 0} p_\beta(\mathbf{x}) = p_0(\mathbf{x})$$

which means that it's possible to recover the *max-ent* program, from the more general LME_β when $\beta \to 0$. This allow us to retain general the program 2.17.

Proposition 3 (Smoothness Propriety)**.** *: Let* $\mathbf{x} \in conv(\mathbf{X})$*, then* $p_\beta(\mathbf{x})$ *is a* C^r *function of* β *in* $(0, +\infty)$*.*

This propriety is very important in order to be licensed to follow a mathematical way to compute the derivatives of the approximants, cause they are continuous in every point of $conv(\mathbf{X})$.

Those two proprieties are both linked to the situation: $\beta = [0, +\infty)$, were, rigorously, the local max-ent approximants should not have any problem. Now we propose other more significant proprieties regarding the approximants when $\beta \to +\infty$

Proposition 4 (Uniqueness of Rajan solution)**.** *: Let* $\mathbf{x} \in conv(\mathbf{X})$*. Consider a sequence of non-negative reals* $\{\beta_k\}$ *(*$k \in N$*) diverging to* $+\infty$ *as* $k \to +\infty$*. Then every convergent subsequence of* $\{p_{\beta_k}(\mathbf{x})\}$ *converges to a solution of (RAJ), Furthermore, if the nodes are in general position, then* $p_\beta(\mathbf{x})$ *converges to the unique solution of (RAJ), as* $\beta \to +\infty$*.*

This is very important in order to recover the RAJAN solution when nodes are not co-spherical, so to have a general method.

Proposition 5 (Uniqueness of Pareto Optima)**.** *: Let* $\mathbf{x} \in conv(\mathbf{X})$*. Consider the solution fo (RAJ) with maximum entropy:*

$$p_\infty^{Pareto}(\mathbf{x}) = arg \quad \max \quad H(p)$$

which is unique by the virtue of the strict concavity of the entropy in the convex set of nodes, then it's possible to demonstrate that:

$$\lim_{\beta \to +\infty} p_\beta(\mathbf{x}) = p_\infty^{Pareto}(\mathbf{x})$$

This last propriety states that, regardless to the uniqueness of the minimizers of RAJAN problem, the limit of LME_β exist as $\beta \to +\infty$, in fact the $max - ent$ regularization program of RAJ extracts from the set of the Rajan's solutions the only one which is optimal in Pareto sense and then is unique by the strict concavity of the entropy in the convex domain.

3.2 Monodimensional first derivatives of the shape functions

We purpose now to displace the first derivatives of the shape functions presented in chapter § 2.4. In figure 3.1, 3.2, 3.3, 3.4, we apply (3.7) progam to reach our goal.

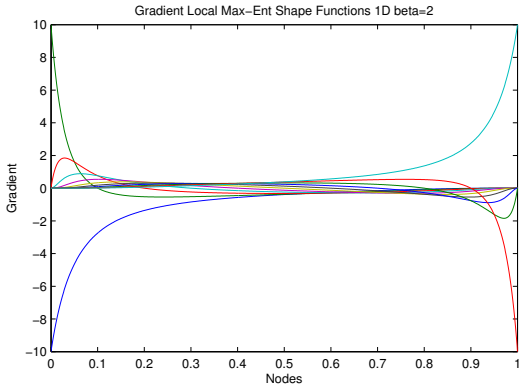


Figure 3.1: 1D first derivatives of the shape functions represented with 11 nodes and 1000 definition points; β=2

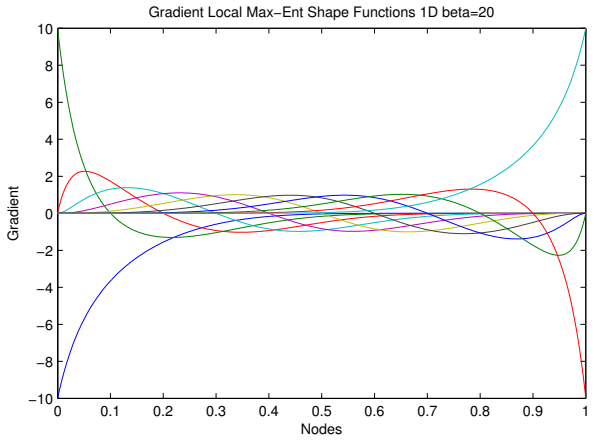


Figure 3.2: 1D first derivatives of the shape functions represented with 11 nodes and 1000 definition points; β=20

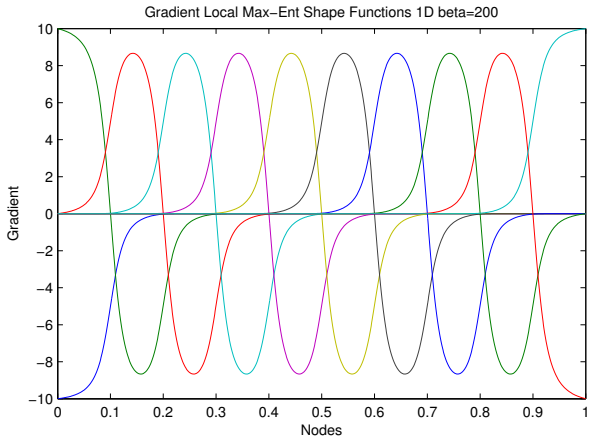


Figure 3.3: 1D first derivatives of the shape functions represented with 11 nodes and 1000 definition points; β=200

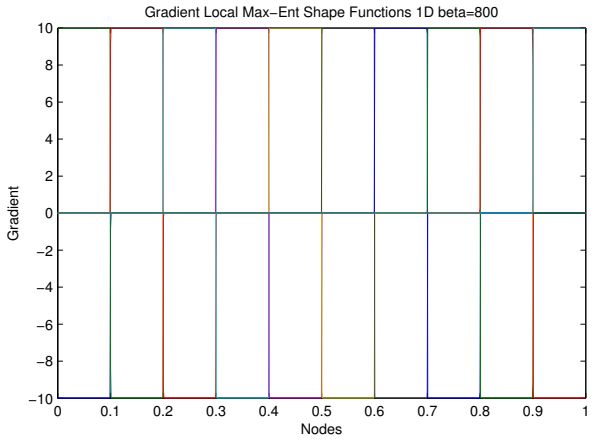


Figure 3.4: 1D first derivatives of the shape functions represented with 11 nodes and 1000 definition points; β=800

It' s possible to appreciate how the first derivatives have the same tendency shown by the shape functions: in fact we note for $\beta = 2$ the derivatives involve all the domain, while for $\beta = 800$ they are full local.

Now we want to test the program for a non regular grid, so to prove the goodness of the approximants derivatives for a random set of nodes. With reference to figure 2.6 we propose in figure 3.5 the associated derivatives:

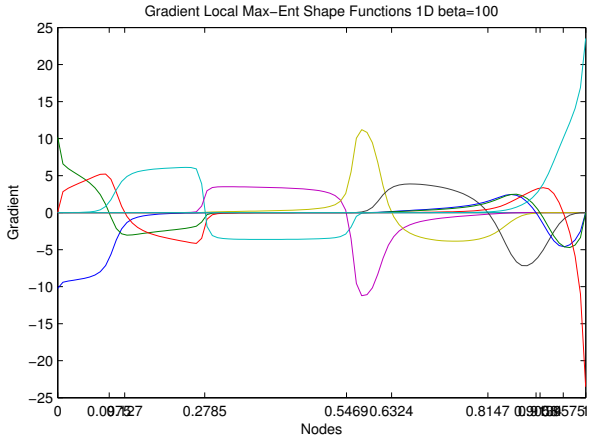


Figure 3.5: 1D first derivatives of the shape functions represented with 11 nodes and 1000 definition points with a random grid; β=100

3.2.1 Program Correction

We give now a suggestion to avoid an implicit error involved in the basis program (3.7). Looking to the original program it is possible to appreciate how the derivatives of the first and the last shape function, in the first and the last point in which they are evaluated, vanishes. In fact, in those points we obtain:

$$(\mathbf{x} - \mathbf{x}_a) = 0;$$

this condition is analytically correct, but inconsistent in a physicist vision, cause it involves:

$$\nabla p_{\beta a}(\mathbf{x}) = 0$$

which is not possible cause in mono-dimensional case the first derivative of the function represent the slope of the tangent line; this could mean that the tangent line is parallel to x-axis, in this way it's inconsistent this solution.

To solve this problem we can follow 2 different ways:

- FINITE DIFFERENCE METHOD: so to ignore the value of the derivatives in the critical points obtained with the basis program and compute it by applying this method by using the ordinates of the derivatives of the nearest nodes. This way, as we will see, is not efficient to our final target;
- "HOMEMADE" METHOD: the way we apply, which consists, simply, in modifying the value of the critical points.

In figure we represent for example the β=20 case, without any correction to original program in order to give a complete vision of the bug.

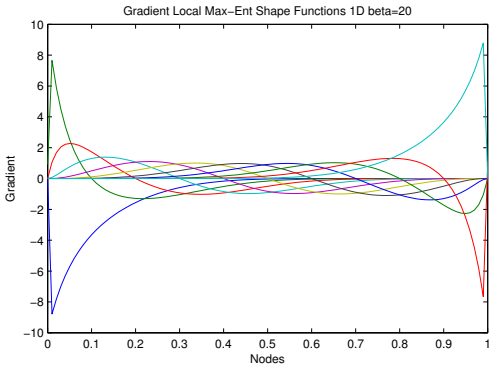


Figure 3.6: 1D first derivatives of the shape functions represented with 11 nodes and 100 definition points. Original program; β=20

In this case we have use a 100 points grid (instead 1000 points grid used before) in order to make easier to see the computational error.
Here we illustrate now the *homemade* method we use to solve problem bug:

- the value of every node $\mathbf{x_a}$ must never change;
- assume first $\mathbf{x}$ point:

  $$\mathbf{x}_1 = \mathbf{x}_1 + \epsilon$$

  where ϵ is a small value of your choosing, with the only purpose to make:

  $$(\mathbf{x} - \mathbf{x}_a) \neq 0$$

- assume last $\mathbf{x}$ point:

  $$\mathbf{x}_N = \mathbf{x}_N - \epsilon$$

  for the same reason over exposed. In practical cases ϵ must be assumed in relation to the nodal spacing of the problem.

In our illustrated cases we have imposed:

$$x_1 = 10^{-8} \qquad x_N = L - 10^{-8}$$

Clearly in this way we obtain not the real value of the derivative we want to compute, but the value in a point very close to the investigated one, the error between the true value and the approximated one is normally very small and this does not create any problem in practical applications.

3.3 Bi-dimensional first derivatives of the shape functions

The derivatives of 2D shape functions are the natural extension of the mono-dimensional case. In following figures we represent the first derivatives, of the approximants exposed in chapter §2.5, in both directions of the plane:

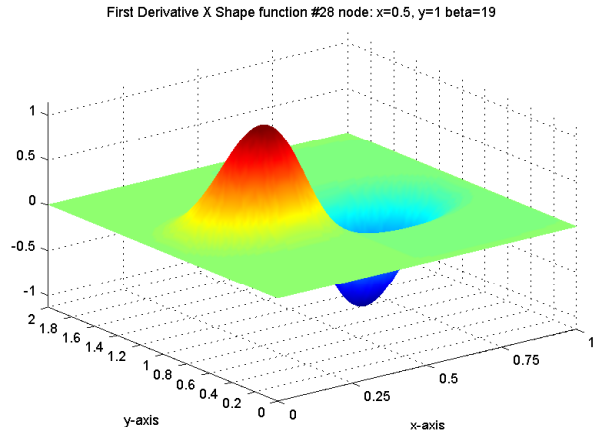


Figure 3.7: 2D X derivative of the shape function represented with 55 nodes and 5500 definition points; γ=0.8

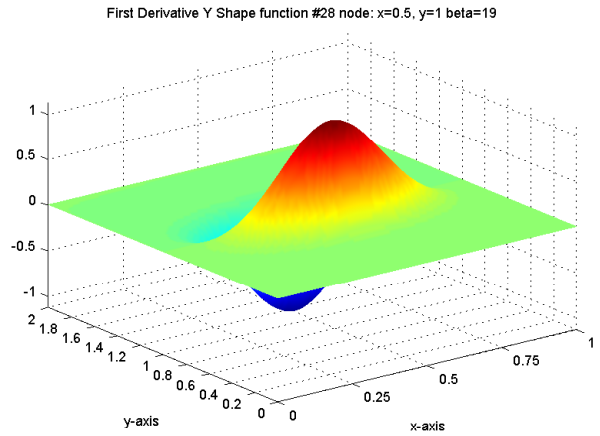


Figure 3.8: 2D Y derivative of the shape function represented with 55 nodes and 5500 definition points; γ=0.8

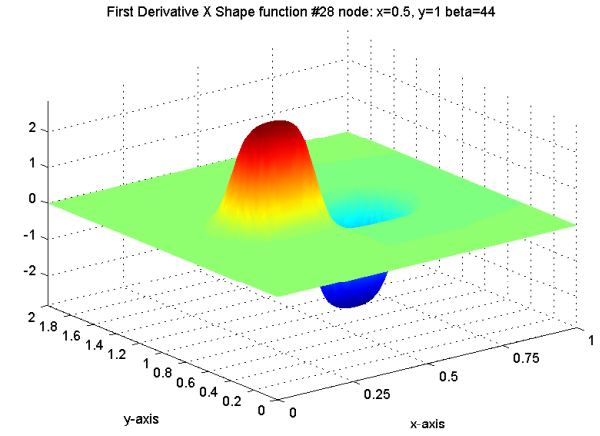


Figure 3.9: 2D X derivative of the shape function represented with 55 nodes and 5500 definition points; γ=1.8

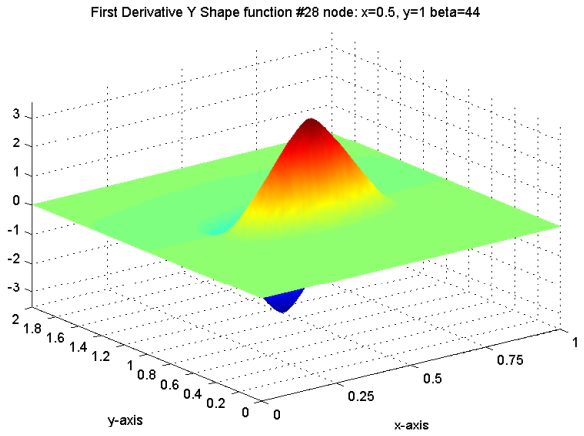


Figure 3.10: 2D Y derivative of the shape function represented with 55 nodes and 5500 definition points; γ=1.8

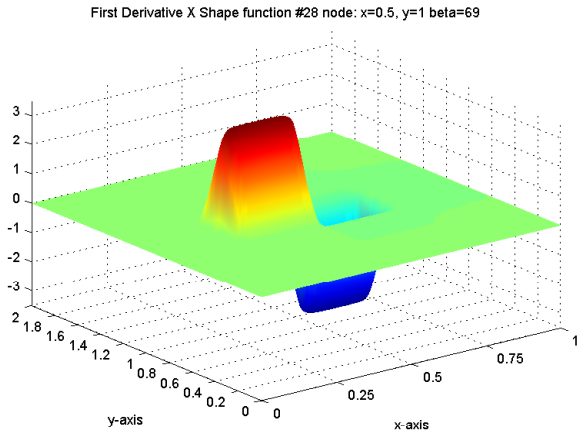


Figure 3.11: 2D X derivative of the shape function represented with 55 nodes and 5500 definition points; γ=2.8

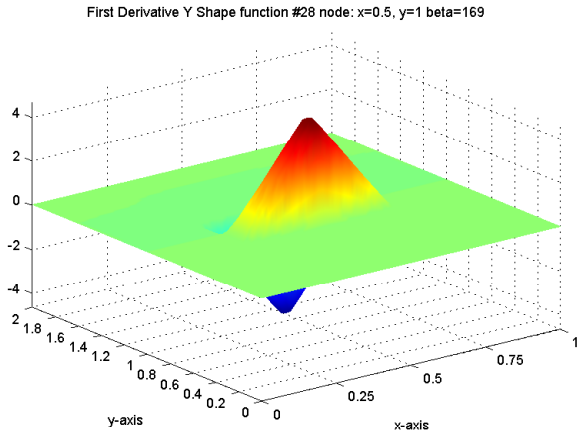


Figure 3.12: 2D Y derivative of the shape function represented with 55 nodes and 5500 definition points; γ=2.8

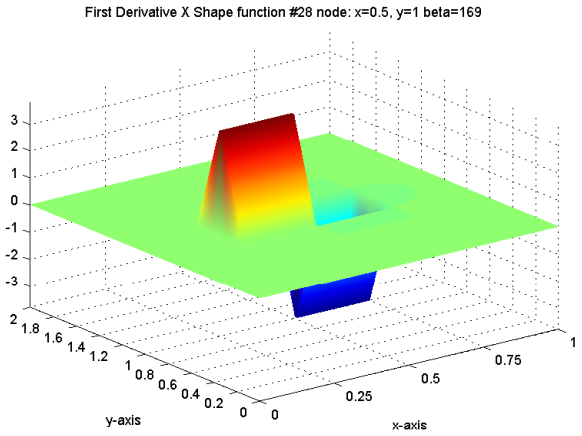


Figure 3.13: 2D X derivative of the shape function represented with 55 nodes and 5500 definition points; γ=6.8

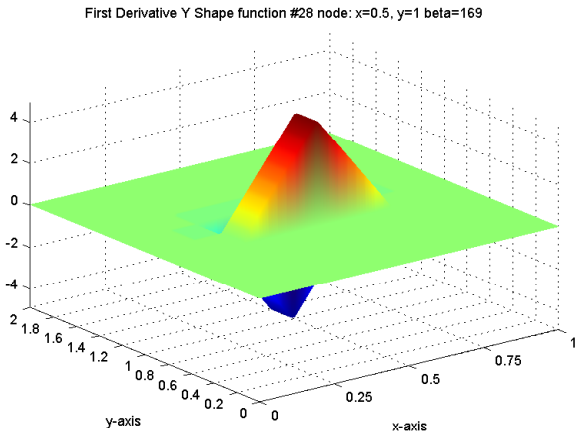


Figure 3.14: 2D Y derivative of the shape function represented with 55 nodes and 5500 definition points; γ=6.8

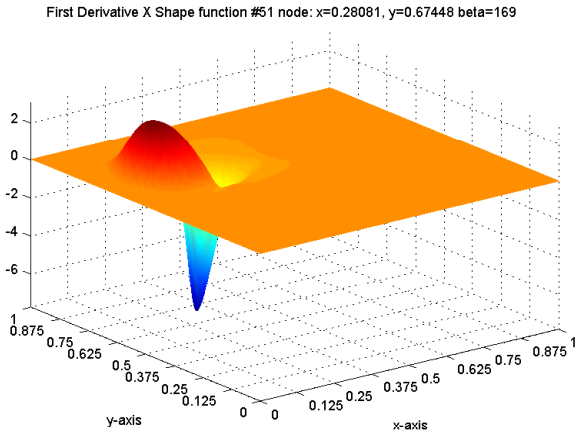


Figure 3.15: 2D X derivative of the shape function represented with 55 nodes and 5500 definition points; random grid; γ=1.8

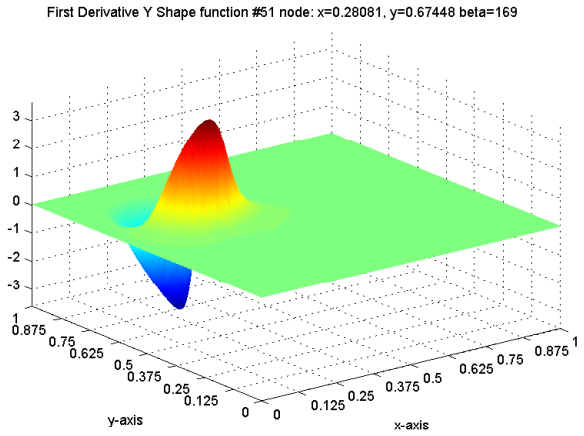


Figure 3.16: 2D X derivative of the shape function represented with 55 nodes and 5500 definition points; random grid; γ=1.8

where 3.15 and 3.16, refers to the shape function of the random grid.

3.3.1 Program correction

It is possible to recover also in the bi-dimensional case the same problem illustrated for the mono-dimensional one. The greatest difficult we have to face it's the fact we can find the problem bug in every border of the domain. In figure 3.17 we displace the gradient functions obtained with the original program, near the angle and on the border of a simple rectangular domain:

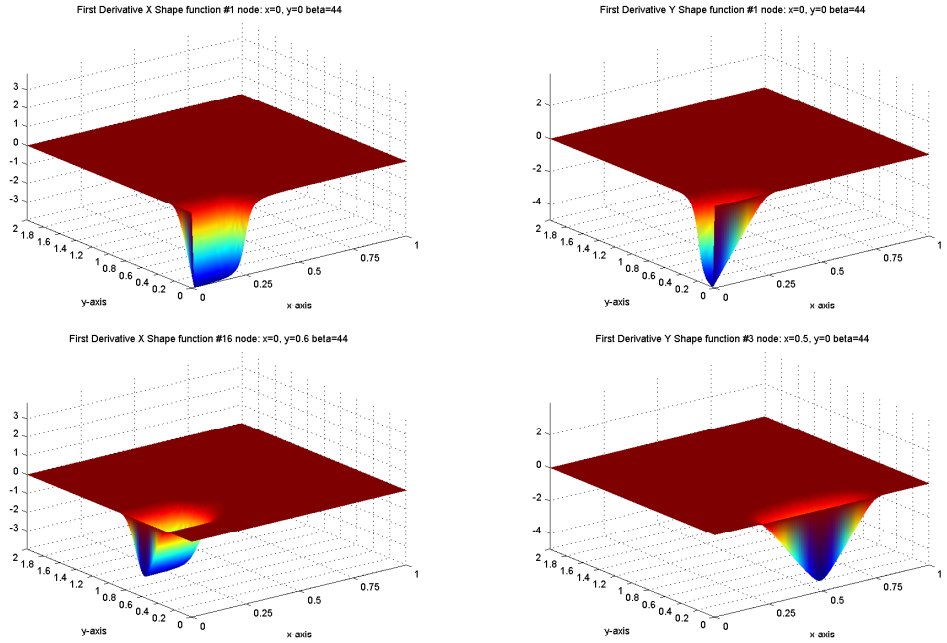


Figure 3.17: 2D shape functions X and Y gradient represented without any program correction

It's possible to appreciate how each component of the gradient function near the border of the

angle of the domain presents the extended version of the problem we had for the 1D first derivatives. The reasons of these errors are analogous to ones exposed in chapter §3.2.1. In this case, clearly, the gradient represent the unitary vector normal to tangent plane.

To avoid this problem, we can follow this method:

- the value of every node $\mathbf{x_a}$ must never change;
- for angle nodes:

  we can impose those conditions:

  $$\mathbf{x}_{angle} = \mathbf{x}_{angle} \pm \epsilon \qquad \mathbf{y}_{angle} = \mathbf{y}_{angle} \pm \epsilon$$

  where sign "+" or "-" is introduced with the purpose to move to interior points the previous value of $\vec{\mathbf{x}}$.
- For border nodes:

  in general we can adopt this way:

  $$\begin{bmatrix}\mathbf{x}_{border}\\ \mathbf{y}_{border}\end{bmatrix} = \begin{bmatrix}\mathbf{x}_{border}\\ \mathbf{y}_{border}\end{bmatrix} \pm \begin{bmatrix}\epsilon\\ 0\end{bmatrix} \; or \pm \begin{bmatrix}0\\ \epsilon\end{bmatrix}$$

  where we use the first or the second addend in order to move the border points to the line of the "next to boundary nodes", and with the same criteria we choose sign "+" or "-".
- For nodes belonging to not orthogonal lines:

  In this section we consider only bi-dimensional elements composed of orthogonal lines. If we want to extend the problem to elements with not orthogonal borderlines, we have to move every border point to interior ones on the direction described by the normal to the considered line. In general:

  $$\begin{bmatrix}\mathbf{x}_{border}\\ \mathbf{y}_{border}\end{bmatrix} = \begin{bmatrix}\mathbf{x}_{border}\\ \mathbf{y}_{border}\end{bmatrix} \cdot \mathbf{n} \pm \begin{bmatrix}\epsilon\\ \epsilon\end{bmatrix} \cdot \mathbf{n}$$

  where $\vec{n}$ is a unit vector normal to the considered line. The sign "+" or "-" is chosen in order to move the considered point to interior ones.

In all the cases the value of ϵ, must be chosen in relation to the nodal spacing between the points. In general we adopt the reasonable choice of $\epsilon = 10^{-8}$.

Chapter 4

Second derivatives of the shape functions

In this chapter we propose to evaluate the second derivatives of the shape functions. This operation is not exactly conventional in fact, even if the approximants have an exponential form, the derivatives are consistent only to the consistency order we have imposed with (2.5), so to the first order. In this work, although we want to evaluate the second order derivatives and test them in practical problems to check if the consistency conditions are really essential to have good approximating schemes.

By considering the SMOOTHNESS PROPRIETY, we know the shape functions are C^r in $conv\mathbf{X}$, and we can try to evaluate the second derivatives in an analytic way, like we did for the first derivatives. It's necessary always to consider that the shape functions are defined point-wise, by considering this we can evaluate the second derivatives by following the *generalized chain rule for multi-variable functions.*

The general math expression of our goal is:

$$Hp_a^* = \frac{1}{p_a^*}\nabla p_a^* \otimes \nabla p_a^* + p_a^* H f_a^* - p_a^* \sum_{b=1}^{N} \frac{1}{p_b^*}\nabla_b^* \otimes \nabla p_b^* - p_a^* \sum_{b=1}^{N} p_b^* H f_b^* \tag{4.1}$$

now it's clear the only term we need to compute is Hf_a^*. Here we introduce the *generalized chain rule for multi-variable functions*:

$$\frac{\partial^2 f}{\partial x^2} = Hf_a^* = \frac{\partial}{\partial x}\left[\frac{\partial f}{\partial \lambda}\frac{\partial \lambda}{\partial x}\right] + \frac{\partial}{\partial x}\left[\frac{\partial f}{\partial x}\frac{\partial x}{\partial x}\right]$$

and so by splitting the formula:

$$\begin{aligned} Hf_a^* = &\left(\frac{\partial^2 f}{\partial x^2}\frac{\partial x}{\partial x} + \frac{\partial^2 f}{\partial \lambda \partial x}\frac{\partial \lambda}{\partial x}\right)\frac{\partial x}{\partial x} + \frac{\partial f}{\partial x}\frac{\partial^2 x}{\partial x^2} + \\ &\left(\frac{\partial^2 f}{\partial x^2}\frac{\partial \lambda}{\partial x} + \frac{\partial^2 f}{\partial x \partial \lambda}\frac{\partial x}{\partial x}\right)\frac{\partial \lambda}{\partial x} + \frac{\partial f}{\partial \lambda}\frac{\partial^2 \lambda}{\partial x^2} \end{aligned}$$

by applying the chain rule to (3.6) we derive the general mathematical expression:

$$Hf_a^* = -2\beta + D\boldsymbol{\lambda}^* + D\boldsymbol{\lambda}^{*^T} + \sum_{k=1}^{d}(D_x^2\boldsymbol{\lambda}_k)^*(\partial_{\boldsymbol{\lambda}_k} f_a)^* \tag{4.2}$$

now, by substituting the previous expression in (30), we obtain:

$$Hp_a^* = \frac{1}{p_a^*}\nabla p_a^* \otimes \nabla p_a^* - p_a^* \sum_{b=1}^{N} \frac{1}{p_b^*}\nabla_b^* \otimes \nabla p_b^* + p_a^* \sum_{k=1}^{d}(\mathbf{x}_a - \mathbf{x}_{ak})(D_{\mathbf{x}}^2\boldsymbol{\lambda}_k)^* \tag{4.3}$$

The most complicated part of the program is the one which goes down to the evaluate the other two unknowns. We start from the analytic condition:

$$D_{\mathbf{x}}^2\mathbf{r}^* = \mathbf{0}$$

from here we can derive the following condition:

$$D_{\mathbf{x}}^2\boldsymbol{\lambda}^* = \mathbf{0}$$

where we evaluate:

$$D_{\mathbf{x}}^2 g_{\boldsymbol{\lambda}_i^*} = (\partial_{\mathbf{x}}\partial_{\mathbf{x}} g_{\boldsymbol{\lambda}_i})^* + (\partial_{\mathbf{x}}\partial_{\boldsymbol{\lambda}} g_{\boldsymbol{\lambda}_i})^*(D\boldsymbol{\lambda}^*)^T + (D\boldsymbol{\lambda}^*)(\partial_{\mathbf{x}}\partial_{\boldsymbol{\lambda}} g_{\boldsymbol{\lambda}_i})^{*T} + \sum_{k=1}^{d} g^*_{\boldsymbol{\lambda}_i\boldsymbol{\lambda}_k}(D_{\mathbf{x}}^2\boldsymbol{\lambda}_k)^* \tag{4.4}$$

it's simple to recover that, in a 2D problem, since $\lambda = 1 \rightarrow 2$ from the previous expression we have a system of 4 equations in 4 unknowns. From the 1^{st} derivatives study, we know that:

$$D\boldsymbol{\lambda}^* = 2\beta\mathbf{I} - (\mathbf{J})^{-1}$$

then the other unknowns can be calculated in this way:

$$(\partial_{\mathbf{x}}\partial_{\mathbf{x}} g_{\boldsymbol{\lambda}_i})^* = 4\sum_{b=1}^{N} p_b^*(\mathbf{x}_i - \mathbf{x}_{b_i})[\beta^*(\mathbf{x} - \mathbf{x}_b)] \otimes [\beta^*(\mathbf{x} - \mathbf{x}_b)]$$

$$(\partial_{\mathbf{x}}\partial_{\boldsymbol{\lambda}} g_{\boldsymbol{\lambda}_i}) = -2\beta^* \sum_{b=1}^{N} p_b^*(\mathbf{x}_i - \mathbf{x}_{b_i})(\mathbf{x} - \mathbf{x}_b) \otimes (\mathbf{x} - \mathbf{x}_b)$$

$$(\partial_{\boldsymbol{\lambda}}\partial_{\boldsymbol{\lambda}} g_{\boldsymbol{\lambda}_i})^* = -\sum_{b=1}^{N} p_b^*(\mathbf{x}_i - \mathbf{x}_{b_i})(\mathbf{x} - \mathbf{x}_b) \otimes (\mathbf{x} - \mathbf{x}_b)$$

Now we have all the instruments to compute the second derivatives.

4.1 Mono-dimensional second derivatives of the shape functions

We want now to give a concrete representation of the second derivatives of the approximants. With reference to the shape functions represented in chapter § 2.4, in figure 4.1, 4.2, 4.3, 4.4 we displace the associated second derivatives.

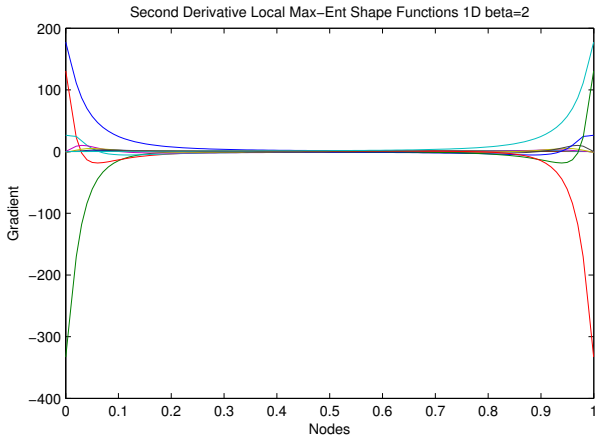


Figure 4.1: 1D second derivatives of the shape functions represented with 11 nodes and 1000 definition points; β=2

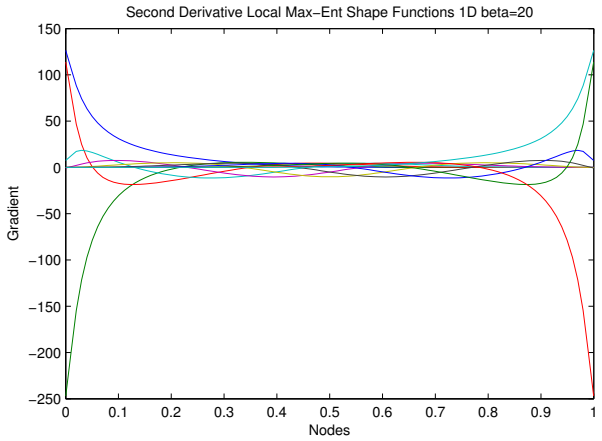


Figure 4.2: 1D second derivatives of the shape functions represented with 11 nodes and 1000 definition points; β=20

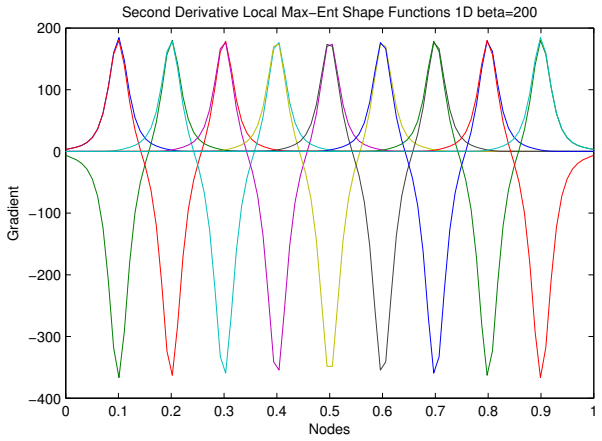


Figure 4.3: 1D second derivatives of the shape functions represented with 11 nodes and 1000 definition points; β=200

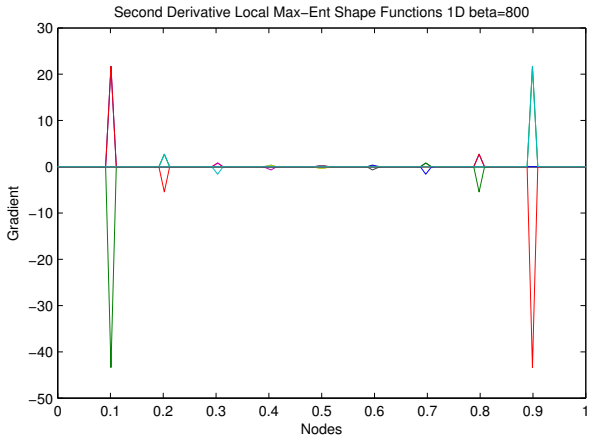


Figure 4.4: 1D second derivatives of the shape functions represented with 11 nodes and 1000 definition points; β=800

In particular we note the second derivatives are all correct according the formulation, except the last ones for $\beta = 800$, where we obtain a wrong results in some points, according to the analytic formulation. Although this bug is not of much importance to our final purposes, in fact the value of β=800 is very high in relation to the selected nodal spacing.

In figure 4.5, we displace the derivatives of the random grid approximants:

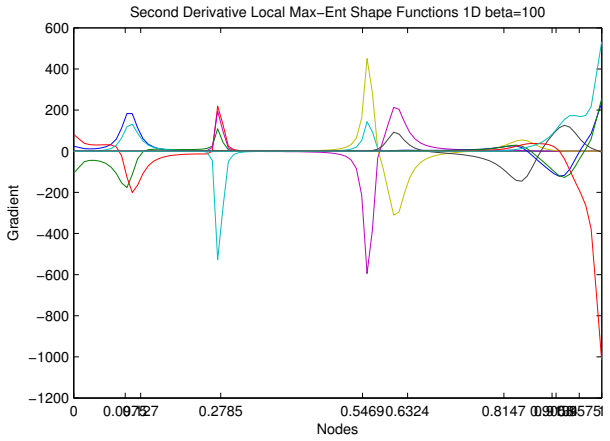


Figure 4.5: 1D second derivatives of the shape functions represented with 11 nodes and 1000 definition points with a random grid; β=100

As we can appreciate, the program is well working also in this situation.

4.2 Bi-dimensional second derivatives of the shape functions

We now extend the treatise to the bi-dimensional case and with reference to the shape functions of chapter §2.5, we displace in the next figures the XX, XY and YY second derivatives for different values of parameter γ:

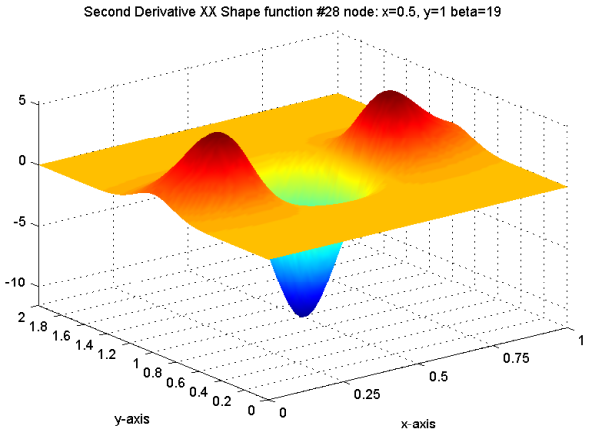


Figure 4.6: 2D XX second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=0.8

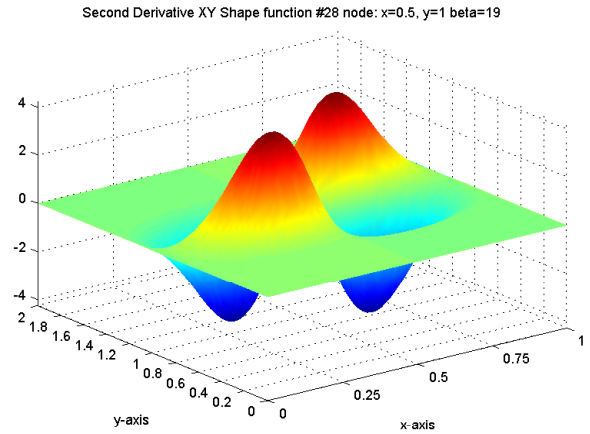


Figure 4.7: 2D XY second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=0.8

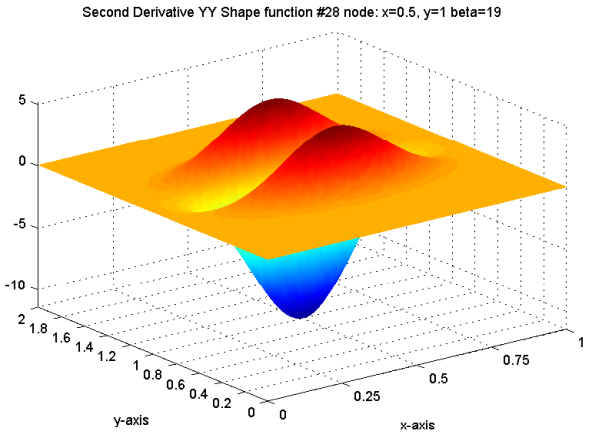


Figure 4.8: 2D YY second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=0.8

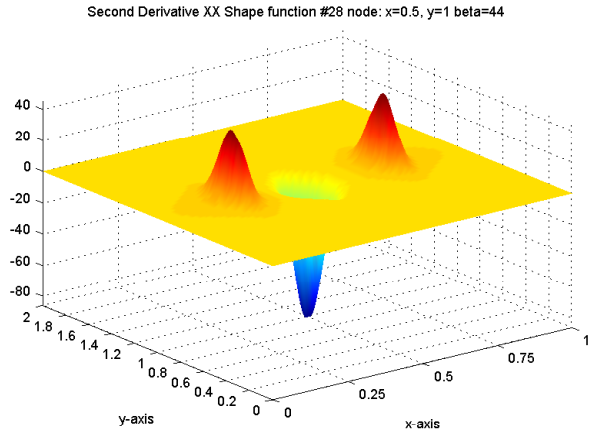


Figure 4.9: 2D XX second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=1.8

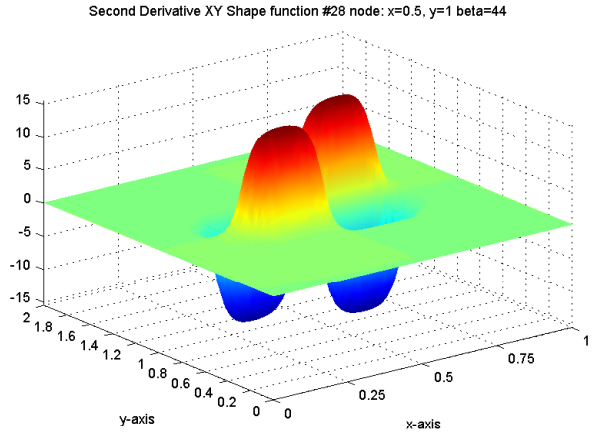


Figure 4.10: 2D XY second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=1.8

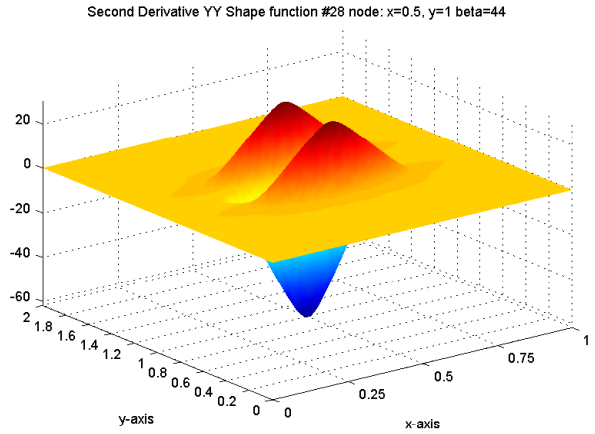


Figure 4.11: 2D YY second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=1.8

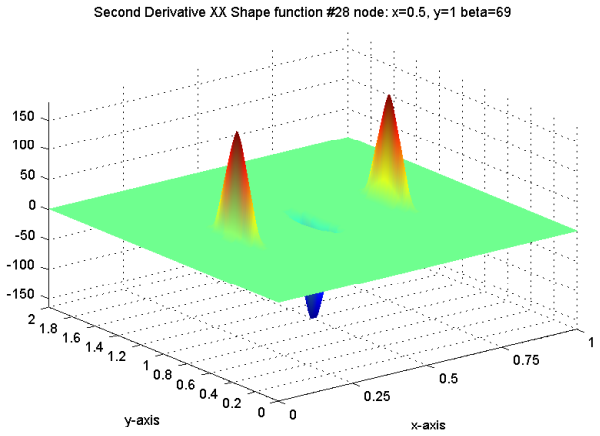


Figure 4.12: 2D XX second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=2.8

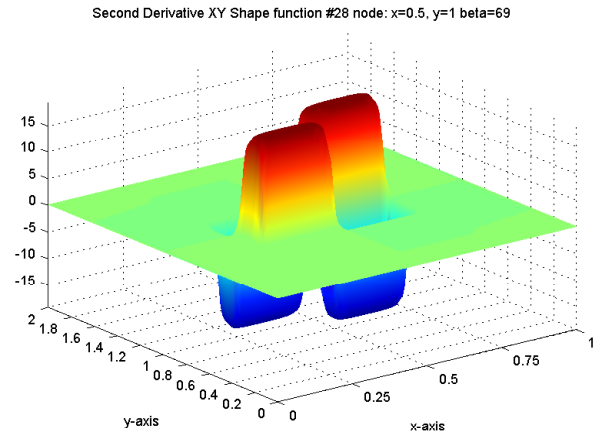


Figure 4.13: 2D XY second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=2.8

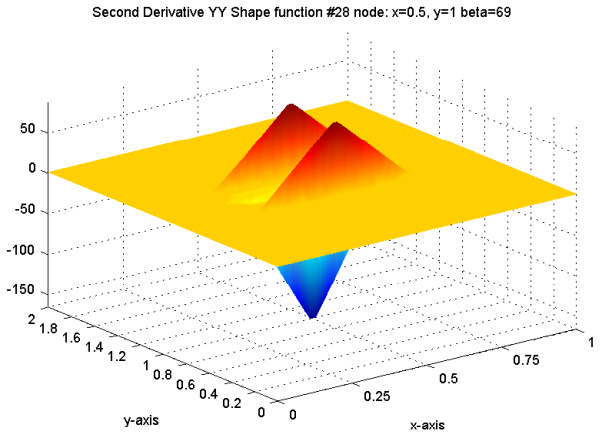


Figure 4.14: 2D YY second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=2.8

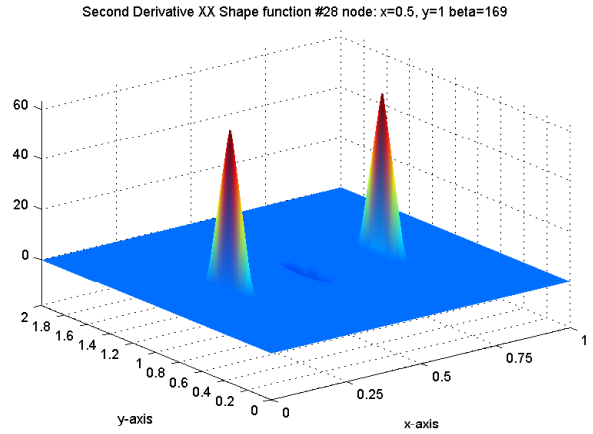


Figure 4.15: 2D XX second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=6.8

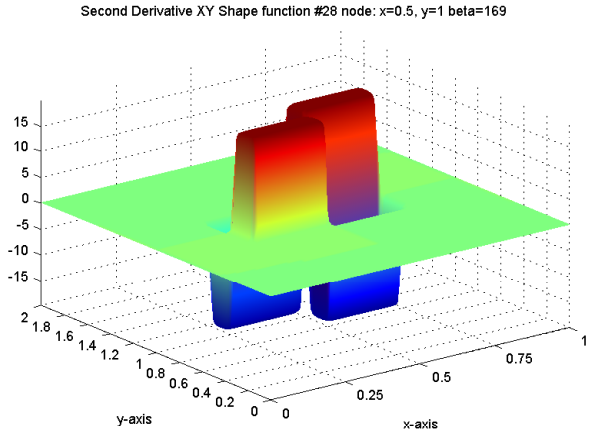


Figure 4.16: 2D XY second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=6.8

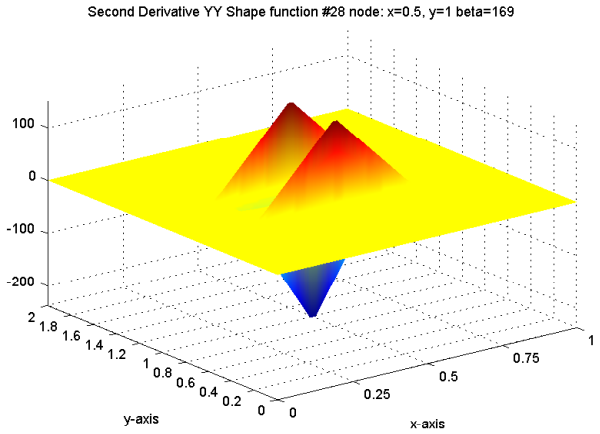


Figure 4.17: 2D YY second derivative of the shape function represented with 55 nodes and 5500 definition points; γ=6.8

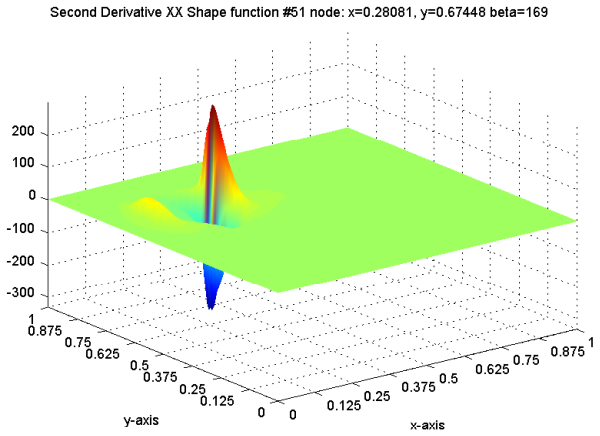


Figure 4.18: 2D XX second derivative of the shape function represented with 55 nodes and 5500 definition points; random grid; γ=1.8

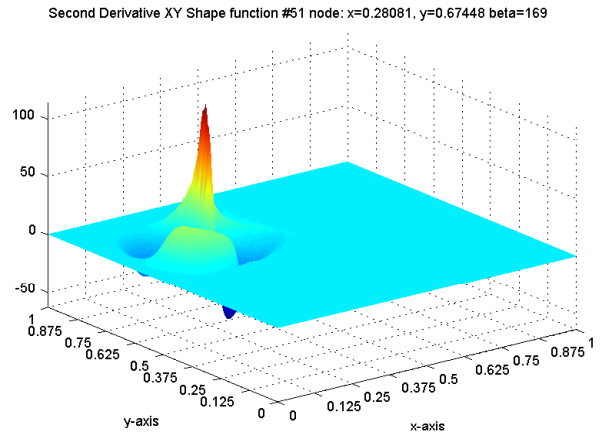


Figure 4.19: 2D XY second derivative of the shape function represented with 55 nodes and 5500 definition points; random grid; γ=1.8

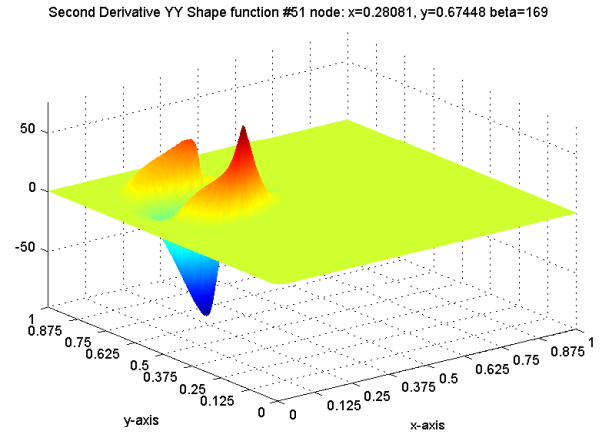


Figure 4.20: 2D YY second derivative of the shape function represented with 55 nodes and 5500 definition points; random grid; γ=1.8

where in figure 4.18, 4.19, 4.20, we refer to the random grid.

Chapter 5

Computer implementation of Local *Max-ent* program

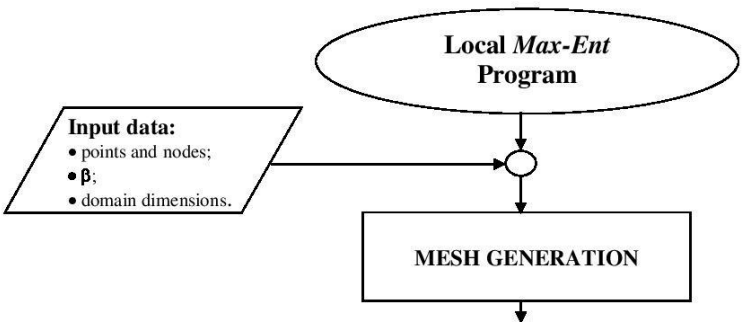
In this chapter we purpose to outline the principal features of the computer implementation of our approximation schemes. In particular, we focus on the optimization program which is the heart of the code, then we show some important aspects of pure programming which allow us to create more efficient programs in terms of time and memory usage.

First, although, we want to give a general view of the Local $max - ent$ program, so to make easier the comprehension every step of the program we evaluate shape functions and associated derivatives with.

5.1 General features of *Max-ent* program

To create a code which is able to reproduce the program we have illustrated in chapter § 2.3, we must consider that the most important and onerous part of the problem is the optimization one, made with the Newton-Raphson iterative method. Our goal is to create a program which is faster and less expensive (in memory terms) as possible. To made our program we have used two languages: MATLAB®and FORTRAN 77®. The first language has been used for the preliminary part of the code, in order to create the necessary files for the core of the program, and to displace the final results. FORTRAN 77®, instead, has been used for the computational part in which we evaluate the shape functions and their derivatives. In particular the usage of FORTRAN 77®is very useful to reduce the computational time and the memory usage during the processing; in fact the program involves many *for* and *while* cycles and MATLAB®is quite fifty times slower than FORTRAN 77®, furthermore with this language we can use more RAM memory sources.

In figure 9.1 we present a flow-chart to illustrate the operative scheme of the program.



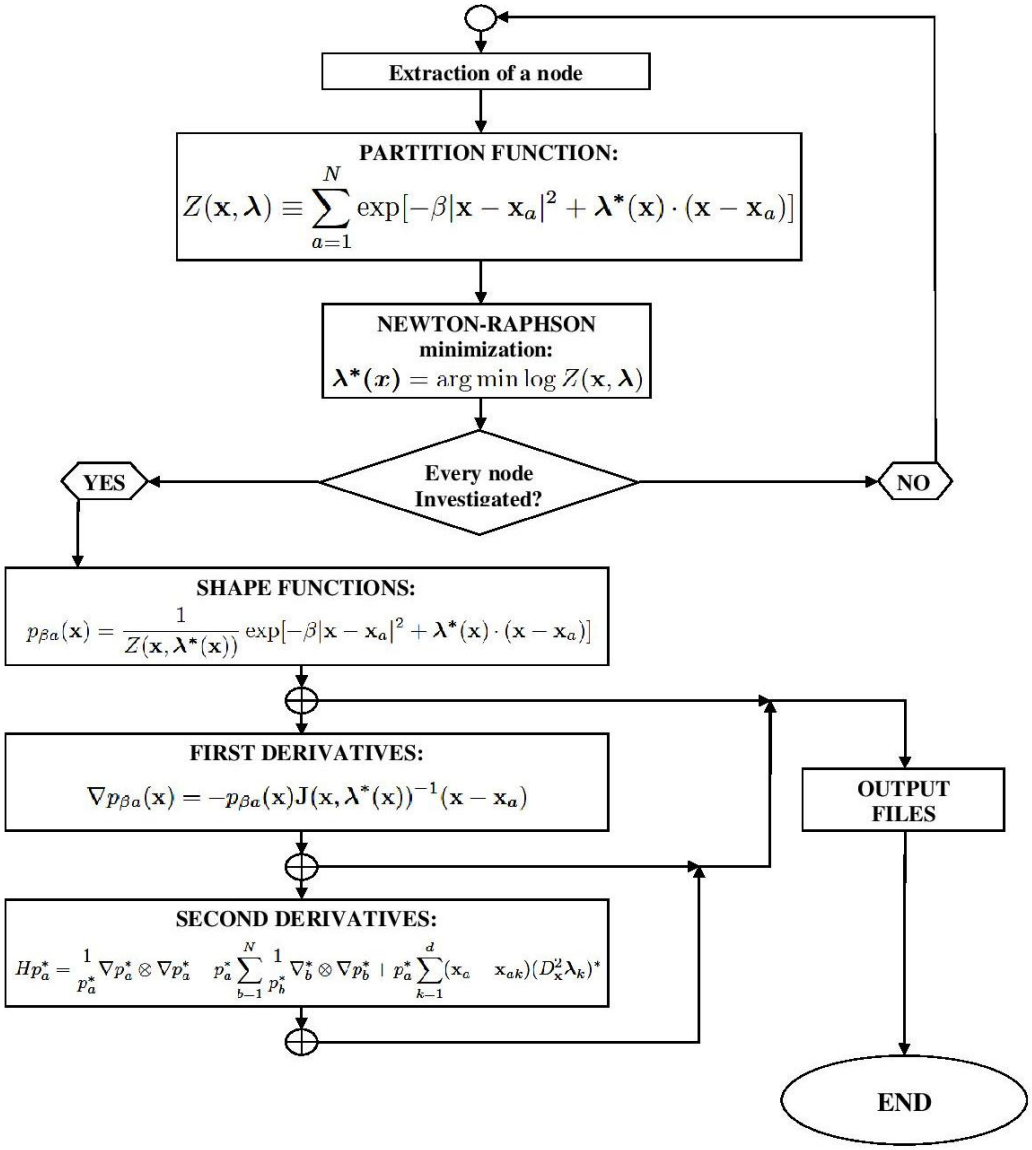


Figure 5.1: Local *Max-Ent* program flowchart

where we have used MATLAB®for the "Mesh Generation" and FORTRAN 77®for the rest of the program. In flow-chart 9.1 we have omitted two possibly part of the program that are not essential:

- Program controls: we can control if the minimazing program has worked right by checking proprieties:
  - positivity propriety: 2.3;
  - unity propriety: 2.4;
  - linear propriety: 2.5;
  - equality of the mixed second partial derivatives.
- Results displacements: we can realize some plots in which we displace the shape functions with associated derivatives and the program controls; in this way we have a better vision of the final results.

it's possible to insert this part of the program after the output files. In this case is simplest to work with MATLAB®.

5.2 Newton-Raphson method

In this section we purpose to focus on the Newton-Raphson iterative method which is the core of Local *Max-ent* program, in fact allow us to find the solution of Eq. 2.13. We must remind we have already demonstrated the solution is unique due to the concavity of the entropy on the convex domain, in this way this non-linear approximation method is optimal to reach our goal.
We now to illustrate this iteration method, with a particular attention to convergence conditions.

First of all, we consider a function: $f(x)$. we can write its Taylor series about the generic point x_0:

$$f(x_0 + \epsilon) = f(x_0) + f'(x_0)\epsilon + \frac{1}{2}f''(x_0)\epsilon^2$$

now, by taking only the first order terms:

$$f(x_0 + \epsilon) = f(x_0) + f'(x_0)\epsilon \qquad (*)$$

we can recognize in this scripture the equation of the tangent line (in a mono-dimensional case) or of the tangent plane (in a bi-dimensional case) in the point $(x_0; f(x_0))$. This expression above can be used to estimate the amount of offset epsilon needed to land closer to the root starting from an initial guess x_0. Setting $f(x_0 + \epsilon) = 0$ and solving (*) for $\epsilon = \epsilon_0$ gives:

$$\epsilon_0 = -\frac{f(x_0)}{f'(x_0)}$$

which is the first-order adjustment to the root's position. By letting $x_1 = x_0 + \epsilon_0$, calculating a new ϵ_1, and so on, the process can be repeated until it converges to a fixed point (which is precisely a root) using:

$$\epsilon_n = -\frac{f(x_n)}{f'(x_n)}$$

Unfortunately, this procedure can be unstable near a horizontal asymptotic or a local extreme. However, with a good initial choice of the root's position, the algorithm can be applied iterative to obtain :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

with $\{n = 1, 2, 3, \dots\}$. An initial point x_0 that provides safe convergence of Newton's method is called an approximate zero.
Thus, for what we have explained in precedent chapters, we know there are no problems in finding a zero in our problem, since, as the shape functions has been constructed, they have none asymptote, and there is only one local extreme which is also an absolute extreme.
For the bi-dimensional case, the analytic approach is the same, but have to compute the two partial derivatives of the function:

$$\frac{\partial f(x,y)}{\partial x}$$
$$\frac{\partial f(x,y)}{\partial y}$$

which constructs the gradient of the function. We have to find an unique solution of the equation:

$$\nabla(x,y) = 0$$

In particular, since we are working to find an extreme point, we must fix our attention upon the derivative of the function, so in this case, the expression: $f'(x) = 0$ is replaced by the Jacobian matrix of the two first derivative, so by the Hessian Matrix:

$$H = \begin{bmatrix} \frac{\partial^2 f(x,y)}{\partial x^2} & \frac{\partial^2 f(x,y)}{\partial xy} \\ \frac{\partial^2 f(x,y)}{\partial xy} & \frac{\partial^2 f(x,y)}{\partial y^2} \end{bmatrix}$$

5.3 Program optimization

The problem of the storage of the results it's very critical, in fact, during the computer processing we need high RAM package in order to store results and temporary variables needed to compute next unknowns of the program. To give an example of RAM usage, we can consider a square grid with 50 nodes for side and we assume the number of points like the nodes for simplicity; we have totally 2500 nodes that generate a shape function's matrix with 2500^2 elements which are the values in every point of any nodal shape function. If we consider the classical representation of the numbers in double precision we need 64 bit for number and so we must have about 200 Mb of memory; if we want to compute also the first and the second derivatives we need about 1.2 Gb of RAM. It's now clear that if we have an higher number of nodes the necessity of memory exponentially increases, for example with 6400 nodes we need 7.1 Gb of RAM. This simple explanation let we understand how expensive those approximants are in terms of memory usage and so in computing time.
Our goal is to reduce the usage of memory in order to be able to compute the LME approximants for higher grids. The approach we use aim at evaluate the value of any shape functions only in the points next to the investigated node, where we know the shape function has significant ordinates different to 0. It's possible to evaluate in this way the radius from the node were we want to compute the shape functions:

$$R_a = \max\left(\sqrt{-\ln(Tol_0)} \cdot h_a; 4h_a\right)$$

where: Tol_0 is the fixed tolerance below that we consider 0 the value of the shape function, h_a is the value of the maximum nodal spacing around the investigated node. In general it's reasonable to assume $Tol_0 = 10^{-9}$.

We can organize the shape functions and derivatives matrices in this way:

$$\begin{bmatrix} i & j & value \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

where i is the number of the nodal shape function (the row of the entire matrix) and j is the point in which we evaluate the associated nodal shape function. Of course we organize in the same way all the matrices.
It's important to note that this method allow us also to reduce considerably the computing time, in particular cause we avoid to evaluate the shape functions and their derivatives in points where we know the results are close to zero.

Such we have illustrated, is very important in order to reduce the memory usage and, considerably also the computational time, but if we want to make more performing the program, we have to consider the most expansive part, in terms of time, is the Newton-Raphson optimization of 2.13. In fact to reduce the numbers of the iterations is very important in order to make it better. To do this the easier way is to choose the right point $\boldsymbol{\lambda}$ form where we start the iterations. In our program we have selected as first attempt solution, the optimal solution $\boldsymbol{\lambda}^*$ of the closest node. In this way we are able to have very good results in terms of number of iterations saved.

In table 5.1 and 5.2 we report some interesting results about the original and optimized program:

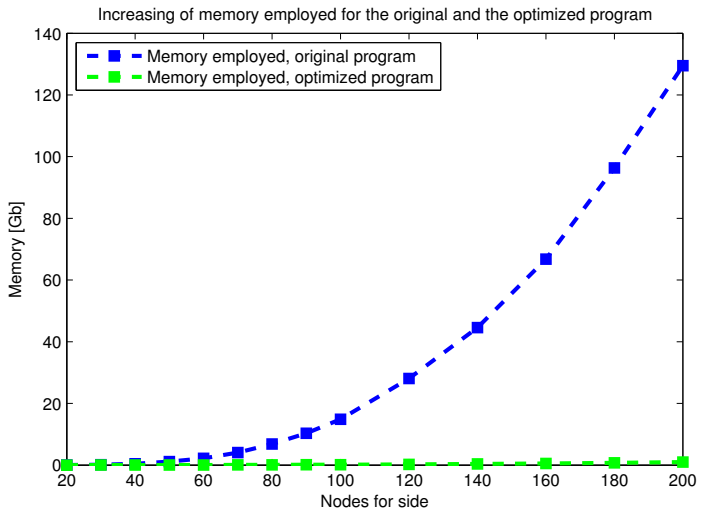
Nodes for side	Memory employed	Machine time	Iterations
	[Mb]	[s]	
20	28.8	1.56	4796
30	147	6.23	11747
40	465	19.23	23127
50	1134	49.45	36422
60	2235	116.95	54819
70	4153	265.84	74050
80	6987	502.35	102122
90	10562	812.35	128125
100	15236	1256.32	162897
120	28756	2286.57	245982
140	45620	3562.13	335157
160	68354	4986.78	438625
180	98625	6785.45	549355
200	132564	8956.26	665321

Table 5.1: Computational values for the ORIGINAL PROGRAM

Nodes for side	Memory employed	Machine time	Iterations
	[Mb]	[s]	
20	4.00	0.35	2735
30	11.76	1.21	5087
40	20.34	3.30	8364
50	34.36	7.86	11503
60	50.00	19.96	15743
70	68.95	47.13	19616
80	96.32	98.12	25453
90	114.49	173.58	30464
100	156.88	279.43	37311
120	233.88	491.72	51428
140	364.03	837.55	64083
160	520.79	1416.68	82582
180	742.32	2049.91	102352
200	1017.77	2834.24	121856

Table 5.2: Computational values for the OPTIMIZED PROGRAM

To make easier the comprehension of the importance of the optimization, we displace the results in figure 5.2:



It's very remarkable how, the trend of iteration line and the trend of the machine time line is quite the same, as expected, in fact the most expensive part of the program is the optimization one and so by decreasing the waste of time in this part, we can reduce in general the total computational time.
The little difference in favor of the iteration line, in terms of decreasing, is due to the second derivatives calculation, in fact even if we have approached with *sparse matrices*, this part of the program involves point-wise the solution of a linear system [2x2] which causes the increasing of the total machine elaboration time.
With a normal computer with 8Gb of Ram it is possible to elaborate mesh with 160000 nodes and points, but the program needs a lot of time, this is the most significant problem of the approximants calculation.

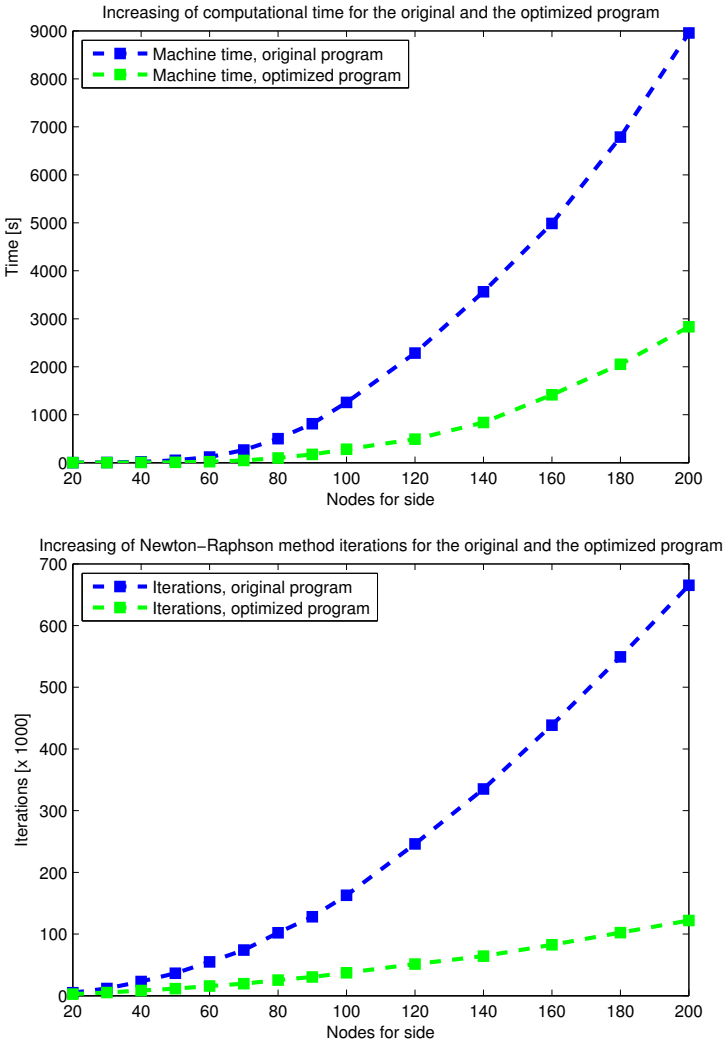


Figure 5.2: Comparison between the original and the optimized program, in terms of memory, time and iterations

Chapter 6

Second order *max-ent* approximation schemes (SME)

In this chapter we purpose to extend the previous method to a second order consistency condition. In Local *max-ent* approximation schemes we have imposed the linear consistency condition (Eq. 2.5), which involves the capability of perfect reproduction of a linear function. We want, now, to impose a condition which allows the approximants to perfectly reproduce a parabolic function. In the following sections, we outline the principal features of the method, focusing our attention on the differences between this method and the previous. In particular we propose some comparisons of function approximation to test the efficiency of the method.

6.1 Feasibility conditions for second order convex approximants

First of all we recall the feasibility conditions introduced for the *Local Max-Ent approximation schemes* [8]:

$$s_a(\mathbf{x}) \geq 0, \quad \forall \mathbf{x} \in conv\mathbf{X}, a = 1, ..., N \tag{6.1}$$

$$\sum_{a=1}^{n} s_a(\mathbf{x}) = 1, \quad \forall \mathbf{x} \in conv\mathbf{X} \tag{6.2}$$

$$\sum_{a=1}^{n} s_a(\mathbf{x})\mathbf{x}_a = \mathbf{x}, \quad \forall \mathbf{x} \in conv\mathbf{X} \tag{6.3}$$

where $s_a(\mathbf{x})$ is the new calling of the shape functions of node $\mathbf{x}_a$.

Those conditions are still working also for this method extension. Now if we want to extend the approximants to a second order consistency condition it is natural to consider the following formulation:

$$\sum_{a=1}^{n} s_a(\mathbf{x})\mathbf{x}_a^2 = \mathbf{x}^2, \quad \forall \mathbf{x} \in conv\mathbf{X} \tag{6.4}$$

where $\mathbf{x}_a^2$ is chosen as coefficient of the combination. This is not the only possible solution, but it's the more convenient one to not have an implicit problem, since s_a is unknown.
Although, by considering Eq. (6.3) and (6.1) it's evident that to satisfy (6.4) it's necessary that, for $\mathbf{x}$ away from the boundary, all the shape functions vanish at this point. This solution, even mathematically correct, hasn't any consistent physicist validation. So to bypass this problem, we

can move away from the canonical formulation and replace (6.4) with a relaxed form:

$$\sum_{a=1}^{n} s_a(\mathbf{x})\mathbf{x}_a^2 = \mathbf{x}^2 + g(\mathbf{x}), \quad \forall \mathbf{x} \in conv\mathbf{X} \tag{6.5}$$

where $g(\mathbf{x})$ is called *gap function* and must satisfy two constrain:

$$g(\mathbf{x}) \geq 0; \tag{6.6}$$

$$g(\mathbf{x}) = 0, \quad \forall \mathbf{x} \in bd(conv\mathbf{X}) \tag{6.7}$$

In this way in a mono-dimensional domain, $x^2 + g(x)$ lies within the convex hull of points (x_a, x_a^2) and the second order consistency condition is not violated.
It's very important to remark the second condition requires $x^2 + g(x)$ to be piece-wise quadratic and smooth, since the smoothness of the shape functions is that of the *gap function*. The main limitation of the gap function method is that it's difficult to define appropriate gap functions for unstructured node sets and so it's difficult also to control the aspect ratio of the basis functions and their smoothness. To evade the problem, we can rewrite (6.5) in this way [8]:

$$\sum_{a=1}^{n} s_a(\mathbf{x})(\mathbf{x}_a^2 - \mathbf{d}_a) = \mathbf{x}^2, \quad \longrightarrow \quad \sum_{a=1}^{n} s_a(\mathbf{x})(\mathbf{x} - \mathbf{x}_a)^2 = \sum_{a=1}^{n} s_a \mathbf{d}_a \tag{6.8}$$

for some non-negative parameters $\mathbf{d}_a$. Clearly $\mathbf{d}_a$ is subjected to same restriction of the gap function $g(\mathbf{x})$.
We can also observe, the right side of the second expression of (6.8) is greater than zero. This viewpoint allows us to define the feasible constraints only by setting the offsets $\mathbf{d}_a$. This make easier to define the gap function upon a sparse set of nodes with variable density. In a more general formulation, in multiple dimensions:

$$\sum_{a=1}^{n} s_a(\mathbf{x})(\mathbf{x} - \mathbf{x}_a) \otimes (\mathbf{x} - \mathbf{x}_a) = \sum_{a=1}^{n} s_a \mathbf{d}_a \tag{6.9}$$

where, now, $\mathbf{d}_a$ is a positive-define symmetric matrix depending by the nodal gap in the problem dimensions.

6.2 Design of feasible constraints

6.2.1 Mono-dimensional case

The one dimensional case is the simplest one cause we have to work only with scalar objects. We have to make a distinction between different type of nodes. Consider a set of nodes:

$$x_1, x_2 \ldots, x_{N-1}, x_N$$

we can define d_a in 3 different way:

- BOUNDARY NODES

  According to (6.8):

  $$d_1 = d_N = 0$$

- NEXT TO BOUNDARY NODES

$$d_2 = \max\left\{\beta h_1^2, \frac{\alpha}{4} h_2^2\right\} \qquad d_{N-1} = \max\left\{\beta h_{N-1}^2, \frac{\alpha}{4} h_{N-2}^2\right\}$$

where: α is a non dimensional parameter we assume ≥ 1; β is a slack parameter in general ≥ 1. We adopt β=1 in most cases.

- INTERIOR NODES

$$d_a = \max\left\{\beta h_{a-1}^2, \frac{\alpha}{4} h_a^2\right\} \quad for \quad \alpha = 3, \ldots, N-2;$$

where $h_{a-1} = x_a - x_{a-1}$.

6.2.2 Bidimensional case

We center our attention to a bi-dimensional case. Consider the simple square domain in figure 6.1:

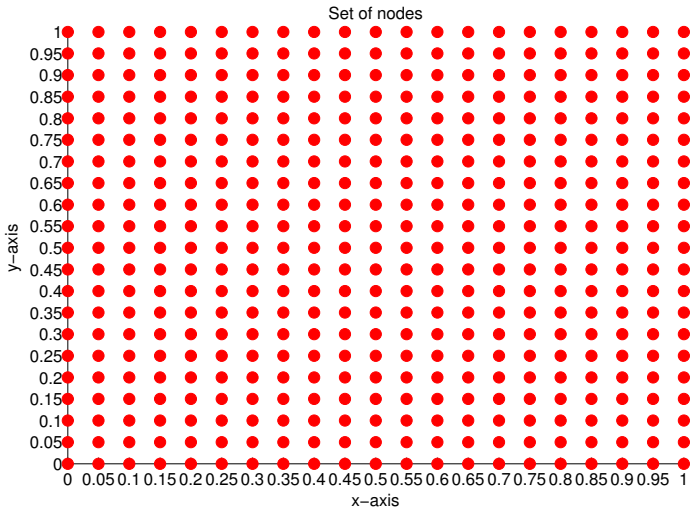


Figure 6.1: General set of nodes

we can now analyze any single case.

- BOUNDARY NODES

For the boundary nodes we have to consider individually any line which the boundary is composed of. For example we can refer to segment which has zero ordinate 6.2. We call: A node (0;0) and B node (1;0). We have so 3 different cases in face $\bar{AB}$:

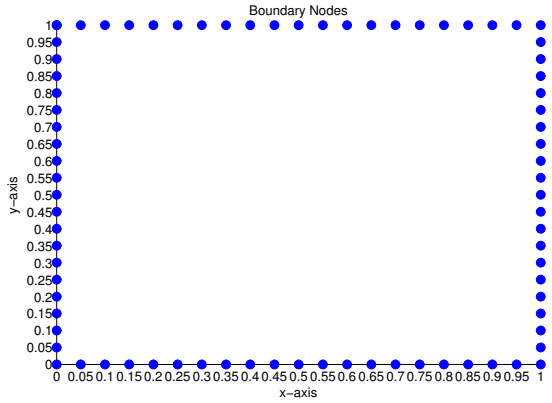


Figure 6.2: Boundary nodes

– BOUNDARY NODES

$$\mathbf{d}_A = \mathbf{d}_B = 0$$

– NEXT TO BOUNDARY NODES

$$\mathbf{d}_a = \beta h_a^2 \mathbf{t} \otimes \mathbf{t}$$

– INTERIOR NODES

$$\mathbf{d}_a = \frac{\alpha}{4} h_a^2 \mathbf{t} \otimes \mathbf{t}$$

where $\mathbf{t}$ is the unit tangent vector to face $\bar{AB}$. In this case, since face $\bar{AB}$ is straight, $\mathbf{t}$ it's constant, in different context it could be variable and defined by coordinates of node a and node $a+1$.

- NEXT TO BOUNDARY NODES

  Now we look to figure 6.3, in particular to the face parallel to $\bar{AB}$; in this case we discern two different situation:

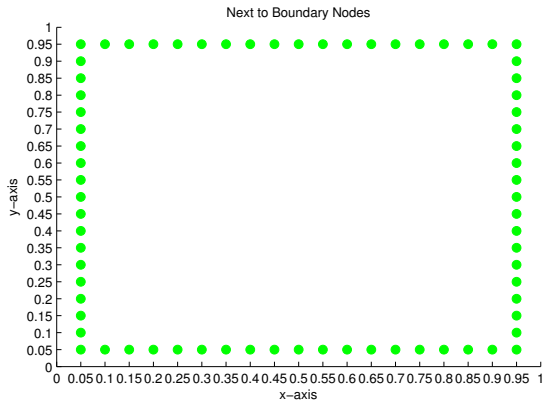


Figure 6.3: Next to boundary nodes

– "ANGLE" NODES:
we call "angle" nodes whose which are in the junction between 2 different faces. In fact they are the boundary nodes of this face.

$$\mathbf{d}_a = \beta h_a^2 (\mathbf{n} \otimes \mathbf{n} + \mathbf{n}' \otimes \mathbf{n}')$$

where $\mathbf{n}$ and $\mathbf{n}'$ are the normal to the two different faces the node refers to.

– INTERIOR NODES

$$\mathbf{d}_a = \beta h_a^2 \mathbf{n} \otimes \mathbf{n} + \frac{\alpha}{4} h_a^2 \mathbf{t} \otimes \mathbf{t}$$

where $\mathbf{n}$ and $\mathbf{t}$ are the normal unit vector and the tangent unit vector of the node face, respectively.

- INTERIOR NODES

In this case we have to identify three different situations by starting from the simplest:

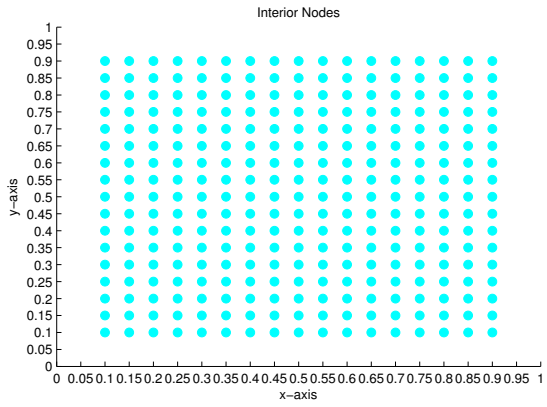


Figure 6.4: Interior nodes

– ISOTROPIC SPACING AMONG TWO ORTHOGONAL DIRECTIONS
which is the example presented in figure 6.4;

$$\mathbf{d}_a = \frac{\alpha}{4} h_a^2 \mathbf{Id}$$

where $\mathbf{Id}$ is the identity matrix whose dimensions are $\mathbf{d}$, in our example [2x2].

– NON-ISOTROPIC SPACING AMONG TWO ORTHOGONAL DIRECTIONS
which is the natural extension of the previous case.

$$\mathbf{d}_a = \frac{\alpha}{4} \begin{bmatrix} h_a^2 & 0 \\ 0 & h_b^2 \end{bmatrix}$$

where h_a^2 and h_b^2 are the spacing among the two different orthogonal directions.

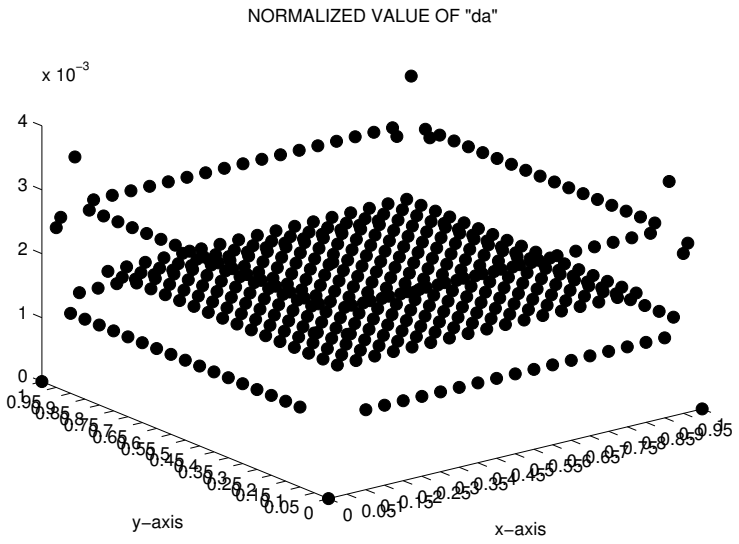
– NON-ISOTROPIC SPACING AMONG TWO GENERAL DIRECTIONS
In this case, which is by far the more difficult one, we have to introduce the general metric tensor h_a. It must be symmetric and positive define. It characterizes the nodal

spacing on each direction. We define:

$$\mathbf{d}_a = \frac{\alpha}{4} \sum_{i=1}^{d} (h_a^i)^2 \mathbf{v}_a^i \otimes \mathbf{v}_a^i$$

where h_a^i and $\mathbf{v}_a^i$ are the eigenvalues and the eigenvectors of the metric tensor, respectively.

For example, by considering an orthogonal coordinates system, the matrix $\mathbf{d}_a$ has only two non-zero component (on the principle diagonal), we can display a "normalized" value of $\mathbf{d}_a$ in every node:



We must remark α can assume any value grater then 1; although, for highly non-uniforms grids, it's better to use values between the range: $1.2 \leq \alpha \leq 3$.

6.3 Optimization program for the second order max-ent approximation schemes

We introduce now the non-linear program to find the basis functions s_a. We still use maximum entropy as a selection principle and practical computation method to choose the optimal approximants between the set of second order consistent convex approximants satisfying feasible constraints. We obtain:

$$(SME) \text{ for fixed } \mathbf{x} \text{ maximize} - \sum_{a=1}^{N} s_a(\mathbf{x}) \ln s_a$$

which is subjected to 6.1, 6.2, 6.3 and also to second consistency condition, we can rewrite in this way:

$$\sum_{a=1}^{N} s_a \mathbf{D}_a(\mathbf{x}, \mathbf{x}_a, \mathbf{d}_a) = 0 \tag{6.10}$$

where:

$$\mathbf{D}_a(\mathbf{x}, \mathbf{x}_a, \mathbf{d}_a) = (\mathbf{x} - \mathbf{x}_a) \otimes (\mathbf{x} - \mathbf{x}_a) - \mathbf{d}_a \tag{6.11}$$

where $\mathbf{D}_a$ is a symmetric matrix which has the dimensions of the problem.
To solve the minimizing problem, we can follow the same approach we used for *Local max-ent approximants* and we recall the partition function:

$$Z(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \sum_{a=1}^{N} \exp\left[\boldsymbol{\lambda} \cdot (\mathbf{x} - \mathbf{x}_a) - \boldsymbol{\mu} : \mathbf{D}_a\right] \tag{6.12}$$

and so the optimal solution can be written as:

$$s_a = \frac{\exp\left[\boldsymbol{\lambda}^*(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_a) - \boldsymbol{\mu}^*(\mathbf{x}) : \mathbf{D}_a\right]}{Z(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)} \tag{6.13}$$

where $\boldsymbol{\lambda}^*(\mathbf{x})$ and $\boldsymbol{\mu}^*(\mathbf{x})$ are the Lagrangian multipliers obtained by minimizing the dual Lagrangian function:

$$g(\boldsymbol{\lambda}(\mathbf{x}), \boldsymbol{\mu}(\mathbf{x})) = \ln Z(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \tag{6.14}$$

The minimizing problem is subjected to Eq. (6.1), Eq. (6.2), Eq.(6.3) and (6.5), which we can rewrite in this more consistent way:

$$\sum_{a=1}^{N} \mathbf{D}_a(\mathbf{x}, \mathbf{x}_a, \mathbf{d}_a) = \mathbf{0} \tag{6.15}$$

In particular we remark that $\boldsymbol{\lambda}^*(\mathbf{x})$ has the same dimensions of LME program $\boldsymbol{\mu}^*(\mathbf{x})$, instead, is a new term which is symmetric and in general positive define. Since also $\mathbf{D}_a$ is positive define and symmetric, we can spit the scalar product in Voigt notation in this way:

$$\boldsymbol{\mu}^*(\mathbf{x}) : \mathbf{D}_a = \begin{bmatrix} \boldsymbol{\mu}_{11} \cdot \mathbf{D}_{a11} \\ \boldsymbol{\mu}_{22} \cdot \mathbf{D}_{a22} \\ 2\boldsymbol{\mu}_{12} \cdot \mathbf{D}_{a12} \end{bmatrix}$$

where in this case we have considered a bi-dimensional space.

6.4 Mono-dimensional domain shape functions

We give, now, a representation of the shape functions in a mono-dimensional domain. We assume β=1 and we change the value of α, which is the parameter that controls the second order feasibility condition.
In figure 6.5, 6.6, 6.7 and 6.8 in particular we can notice how for lower values of α the shape functions span about less nodes. For example with α=1.5 the basis function spans about four node spacing, instead for α=4 the function spans about a six nodal spacing. In figure 6.5, specifically, it is possible to recognize the trend of the *gap functions*.

This is very important in computational problems, in fact playing with this parameter we can reduce numbers of non-zero elements of the matrices and so reduce the machine processing time and the memory usage.

The terms: nodes and points have the same meaning we have illustred in chapter § 2.4

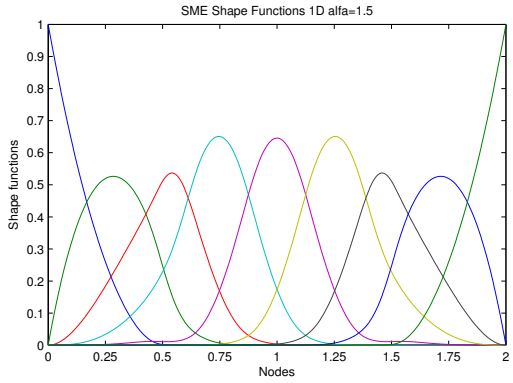


Figure 6.5: 1D shape functions represented with 9 nodes and 900 definition points; α=1.5

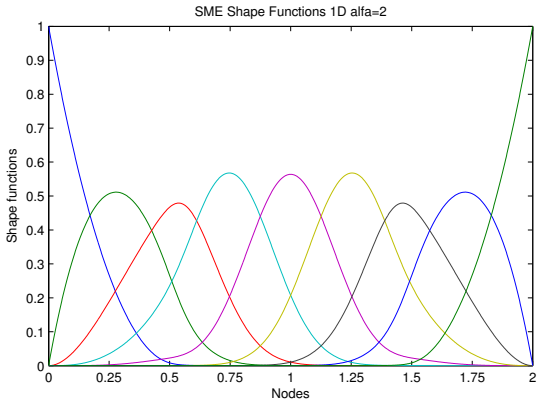


Figure 6.6: 1D shape functions represented with 9 nodes and 900 definition points; α=2.0

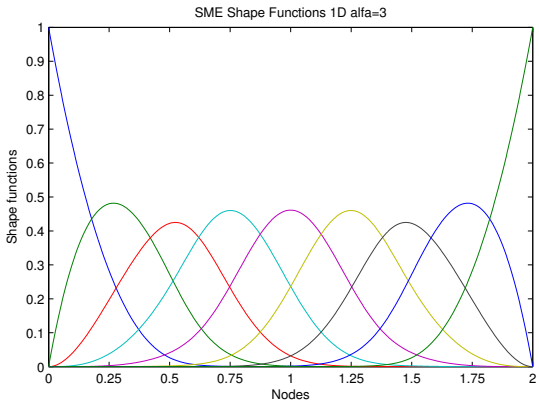


Figure 6.7: 1D shape functions represented with 9 nodes and 900 definition points; α=3.0

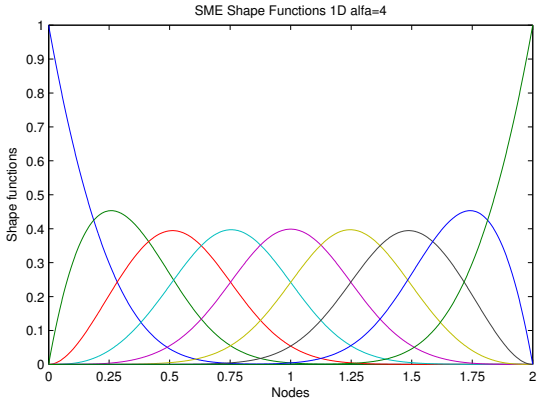


Figure 6.8: 1D shape functions represented with 9 nodes and 900 definition points; α=4.0

With the same quadratic distribution of chapter: 2.4 we test the program on a non regular grid of nodes:

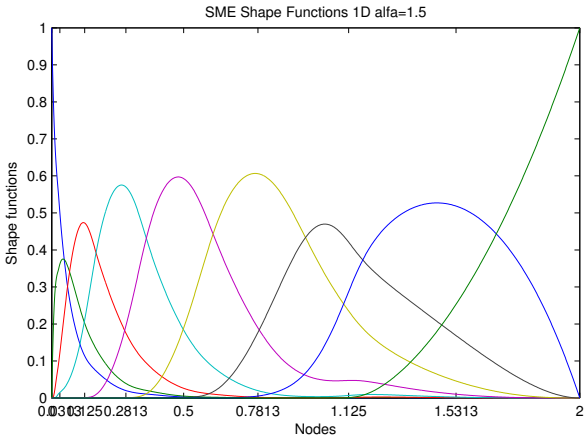


Figure 6.9: 1D shape functions represented with 11 nodes and 1000 definition points with a random grid; α=1.5

6.4.1 Example of a function approximation

Our purpose is to make a comparison between the LME and the SME method. To make this comparison we take the same function used in chapter § 2.4. Since in LME the discretization parameter is β and for SME is α we must be sure to use similar parameters, to make reasonable the comparison. We introduce this formulation that came from practical experience:

$$\alpha \approx \frac{2}{\gamma}$$

which is consistent for uniform grid in one dimensional cases. So by considering that $\gamma = \beta h^2$, we can rewrite the comparison condition in this way:

$$\beta \approx \frac{2}{\alpha h^2}$$

For the same analytic function:

$$f(x) = sin(x)$$

we want to make a consistent comparison between the two approximation schemes. To do this we must link the parameter α to a consistent analogous parameter β. Its' possible thanks to the previous formulation. To prove the goodness of the results, it's possible to estimate the absolute error between the analytic function and the SME max-ent approximation in this way:

$$Err = \frac{\sqrt{\sum_{i=1}^{N} \|y_{i_{an}} - y_{i_{sme}}\|^2}}{\sqrt{\sum_{i=1}^{N} \|y_{i_{an}}\|^2}}$$

In figure 6.10 and 6.11 we displace the comparison of the errors for the regular case and the irregular one: For the regular grid, we notice SME program has a slope $\simeq$4, and so is better then

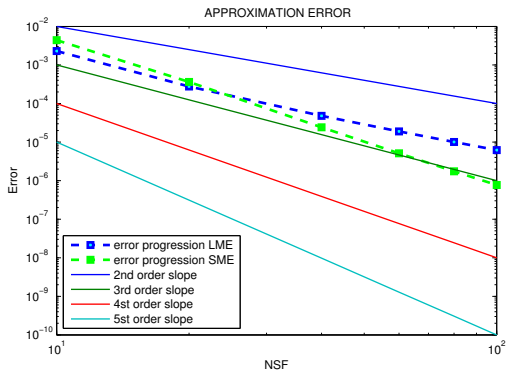


Figure 6.10: Comparison between the LME approximation and the SME approximation of the sine function

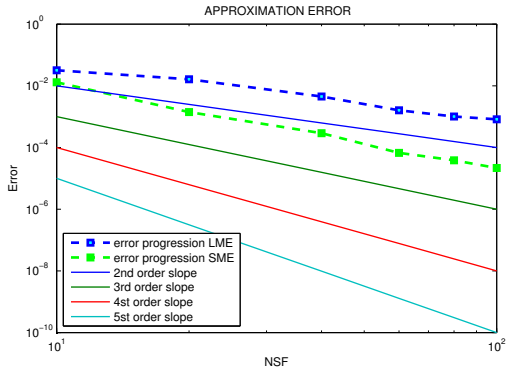


Figure 6.11: Comparison between the LME approximation and the SME approximation of the sine function

LME (slope $\simeq$2), but for the irregular case , even if the error is lower, the SME approximants have the same trend line then LME ones. This allow us to retain SMEs are good working for regular grids, but for the irregular grids we don't obtain good results. We can conclude SME method is more unstable than LME approximation schemes.

6.5 Bi-dimensional domain shape functions

In this section we want to extend the treatise to the bi-dimensional case. The natural extension of the problem involves greater computational burden. We take a different way to represent shape functions than LME problem solving. Instead displacing the same shape functions for different values of α, we displace for a usual value of α (=2) different type of basis functions. The choice has been made by taking in consideration that α differently from β in LME, has not a straight physicist value in 2D. As usual, in figure 6.12, 6.13, 6.14, 6.15, we assume β=1.

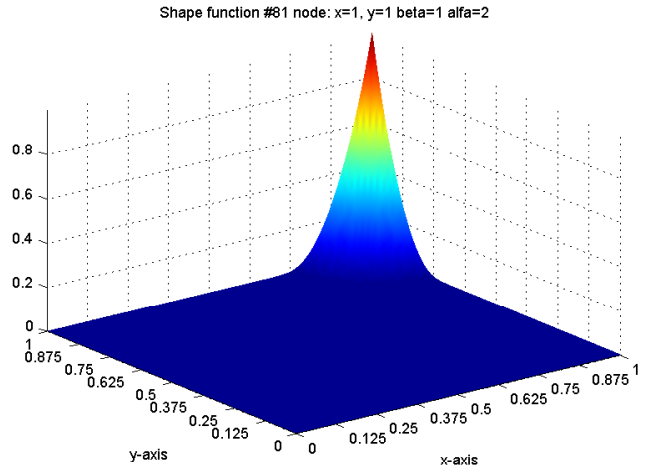


Figure 6.12: 2D *angle* shape function represented with 81 nodes and 8100 definition points; α=2

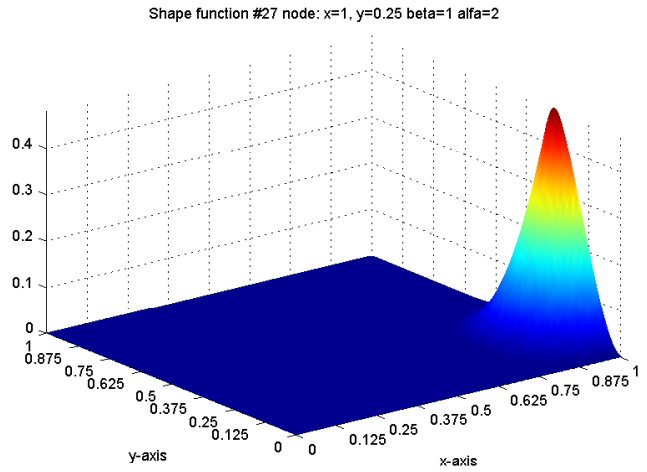


Figure 6.13: 2D *border* shape function represented with 81 nodes and 8100 definition points; α=2

It's possible to appreciate the shape functions have similar behavior in LME program.

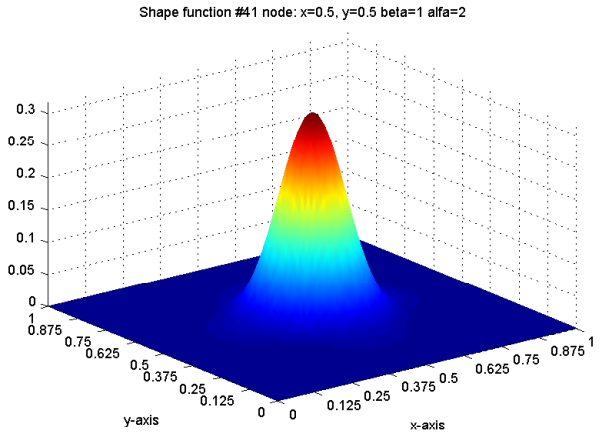


Figure 6.14: 2D *center* shape function represented with 81 nodes and 8100 definition points; α=2

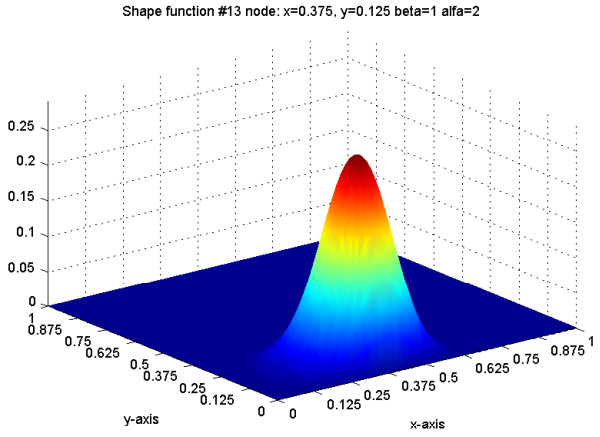


Figure 6.15: 2D *random* shape function represented with 81 nodes and 8100 definition points; α=2

6.5.1 Random Grid

Those approximants show a critical bug, in fact, form our experience, is very difficult to reproduce the shape functions for random grids of nodes. In particular the difficulties came from the definition of parameter d_a which substitutes the *gap function*. We recall the formulation of d_a for non-isotropic spacing:

$$\mathbf{d}_a = \frac{\alpha}{4} \sum_{i=1}^{d} (h_a^i)^2 \mathbf{v}_a^i \otimes \mathbf{v}_a^i$$

we take from [8]. With this approach is not clear which kind of neighbor nodes we have to consider to compute the gap. By looking to the case of an isotropic spacing along two orthogonal directions:

$$\mathbf{d}_a = \frac{\alpha}{4} \begin{bmatrix} h_a^2 & 0 \\ 0 & h_b^2 \end{bmatrix}$$

it's clear the matrix could be considered a kind of *normalized inertia tensor*, in fact if we consider a regular grid of nodes, the value of d_a for every node could be reached by considering the inertia matrix of the nodes next to the investigated one and normalizing it for the number of nodes which

are effectively contributory to the inertia in that direction. Form this consideration, we have tried 3 ways to move this problem to a random grid:

- ALL NODES METHOD:

  we consider all the nodes of the grid around the investigated one, we make the tensor of inertia and we normalize it with the number of total nodes;

- 9 NODES METHOD:

  we take just 8 nodes and the investigated one to compute the matrix of inertia and we evaluate the eigenvalues (eigenvectors) and we normalize them with number of contributory nodes

- 5 NODES METHOD:

  we take just 4 nodes and the investigated one to compute the matrix of inertia and we evaluate the eigenvalues (eigenvectors) and we normalize them with number of contributory nodes

where we call *contributory* node which one is give a non zero contribution to definition of the matrix of inertia.

In particular the second and the third ways are both consistent in a regular grid but we have noticed that for completely random grids anyway we define d_a we cant' reach a solution of the minimizing program. This means the SMEs are very sensible to this parameter and if it's not perfect the program doesn't work. Furthermore in literature there isn't any source that explains this bug or simply uses this approximants in an efficient way.

6.5.2 Example of function approximation

Considering the same function used in chapter § 2.5.1, we want now to displace the approximation error of SME program and make a comparison with the results obtained with LME. In this case we are not able to make the comparison for the regular and irregular grid of nodes, so we are going to face the results only for the regular grid of nodes. We assume $\alpha = 2$ which is a common choice and we use $\beta = 1$ as thermalization parameter.
To prove the goodness of the results, it's possible to estimate the absolute error between the analytic function and the SME max-ent approximation. Since the functions are defined point-wise we need to calculate in every point of the domain the variance between the two functions and so the absolute error can be written in this way:

$$Err = \frac{\sqrt{\sum_{i=1}^{N} \left\| z_{i_{an}} - z_{i_{sme}} \right\|^2}}{\sqrt{\sum_{i=1}^{N} \left\| z_{i_{an}} \right\|^2}}$$

where N is the number of grid points.

In figure 6.16 we displace the comparison between the LME error and the SME error.

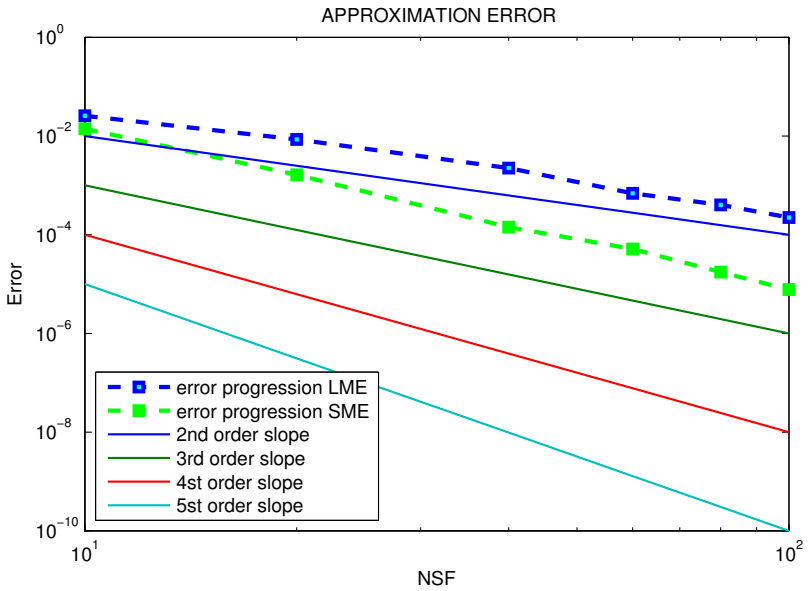


Figure 6.16: Approximation error with associated tendency lines, regular grid

As for the mono-dimensional case, we have the SMEs approximants show a decreasing tendency line with slope $\simeq 4$, instead the LMEs line has slope $\simeq 2$. This is in line with what we expected. In fact since SME has second order consistency condition the decreasing of the error must be faster.

Chapter 7

First derivatives of the shape functions

In this chapter we present the first derivatives of the shape functions. We follow the same analytic line we used for LME program. In this case the problem is more difficult due to the higher numbers of unknowns. It's also important to remember the problem is defined point-wise and the same proprieties delineated in chapter § 3 are still consistent.

We must refer to optimization problem exposed in chapter § 6.3 [8]. We can define the following functions:

$$f_a(\mathbf{x}, \lambda, \boldsymbol{\mu}) = \exp\left[\boldsymbol{\lambda}^* \cdot (\mathbf{x} - \mathbf{x}_a) - \boldsymbol{\mu}^* : \mathbf{D}_a\right] \tag{7.1}$$

and

$$\{\boldsymbol{\lambda}^*(\mathbf{x}), \boldsymbol{\mu}^*(\mathbf{x})\} = \arg\,\min\,\left[\ln Z(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})\right] \tag{7.2}$$

where $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ are the Lagrangian multipliers that minimize the problem. We notice $\boldsymbol{\lambda}^*$ is a vector and $\boldsymbol{\mu}^*$ is a symmetric second order tensor which can be written in Voigt notation to uniform the treatise. It's also very important to notice that, if we write $\boldsymbol{\mu}^*$ and $\mathbf{D}_a$ in Voigt notation, we have to multiple per 2 the 3^{rd} component of $\mathbf{D}_a$ to guarantee the correctness of the results. In this way we can write the gradient and the hessian of $\mathbf{g}$ with respect to the Lagrange multipliers found with Newton-Raphson method:

$$\mathbf{r} = \begin{bmatrix} \mathbf{g}_{\boldsymbol{\lambda}} \\ \mathbf{g}_{\boldsymbol{\mu}} \end{bmatrix} \tag{7.3}$$

$$\mathbf{J} = \mathbf{J}^{\mathbf{T}} = \begin{bmatrix} g_{\boldsymbol{\lambda}\boldsymbol{\lambda}} & g_{\boldsymbol{\lambda}\tilde{\boldsymbol{\mu}}} \\ g_{\tilde{\boldsymbol{\mu}}\boldsymbol{\lambda}} & g_{\tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}} \end{bmatrix} \tag{7.4}$$

where $\tilde{\mu}$ is the Voigt notation of tensor μ. We can define any component of the gradient and the hessian in this way:

$$\mathbf{g}_{\boldsymbol{\lambda}} = \sum_{a=1}^{N} s_a(\mathbf{x} - \mathbf{x}_a)$$

$$\mathbf{g}_{\boldsymbol{\lambda}\boldsymbol{\lambda}} = \sum_{a=1}^{N} s_a(\mathbf{x} - \mathbf{x}_a) \otimes (\mathbf{x} - \mathbf{x}_a) - \mathbf{g}_{\boldsymbol{\lambda}} \otimes \mathbf{g}_{\boldsymbol{\lambda}}$$

$$\mathbf{g}_{\tilde{\boldsymbol{\mu}}} = -\sum_{a=1}^{N} s_a \tilde{\mathbf{D}}_a$$

$$\mathbf{g}_{\tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}} = \sum_{a=1}^{N} s_a \tilde{\mathbf{D}}_a \otimes \tilde{\mathbf{D}}_a - \mathbf{g}_{\tilde{\boldsymbol{\mu}}} \otimes \mathbf{g}_{\tilde{\boldsymbol{\mu}}}$$

$$\mathbf{g}_{\tilde{\mu}\lambda} = -\sum_{a=1}^{N} s_a \tilde{\mathbf{D}}_a \otimes (\mathbf{x} - \mathbf{x}_a) - \mathbf{g}_{\tilde{\mu}} \otimes \mathbf{g}_{\lambda}$$

It's readily checked the first order spatial derivatives of the basis functions can be written as:

$$\nabla s_a^* = s_a^* \left(\nabla f_a^* - \sum_{b=1}^{N} s_b^* \nabla f_b^* \right) \tag{7.5}$$

in fact the analytic form must be the same for the LMEs approximants. The main difference is the term ∇f_a^*, since we have 3 unknowns: $\mathbf{x}$, $\boldsymbol{\lambda}^*(\mathbf{x})$, $\boldsymbol{\mu}^*(\mathbf{x})$. It's, so, possible to define ∇f_a^* in this way:

$$\nabla f_a^* = (\partial_x f_a)^* + \mathbf{D}\boldsymbol{\lambda}^* (\partial_y f_a)^* + \mathbf{D}\tilde{\boldsymbol{\mu}}^* (\partial_{\tilde{\boldsymbol{\mu}}} f_a)^*$$

where:

$$(\partial_x f_a)^* = \boldsymbol{\lambda}^* - 2\boldsymbol{\mu}(\mathbf{x} - \mathbf{x}_a), \quad (\partial_y f_a) = (\mathbf{x} - \mathbf{x}_a), \quad (\partial_{\tilde{\boldsymbol{\mu}}} f_a)^* = -\tilde{\mathbf{D}}_a$$

we notice that the terms $\mathbf{D}\boldsymbol{\lambda}^*$ and $\mathbf{D}\tilde{\boldsymbol{\mu}}^*$ are not explicitly available, but considering that $\mathbf{r}^*$ is identically zero, from the Newton-Raphson results, we can write:

$$\mathbf{0} = \begin{bmatrix} \mathbf{g}^*_{\mathbf{x}\boldsymbol{\lambda}} & \mathbf{g}^*_{\mathbf{x}\tilde{\boldsymbol{\mu}}} \end{bmatrix} + \begin{bmatrix} \mathbf{D}\boldsymbol{\lambda}^* & \mathbf{D}\tilde{\boldsymbol{\mu}}^* \end{bmatrix} \begin{bmatrix} \mathbf{g}^*_{\boldsymbol{\lambda}\boldsymbol{\lambda}} & \mathbf{g}^*_{\boldsymbol{\lambda}\tilde{\boldsymbol{\mu}}} \\ \mathbf{g}^*_{\tilde{\boldsymbol{\mu}}\boldsymbol{\lambda}} & \mathbf{g}^*_{\tilde{\boldsymbol{\mu}}\tilde{\boldsymbol{\mu}}} \end{bmatrix}$$

and so we define:

$$\mathbf{r_x}^* = \begin{bmatrix} \mathbf{g}^*_{\mathbf{x}\boldsymbol{\lambda}} & \mathbf{g}^*_{\mathbf{x}\tilde{\boldsymbol{\mu}}} \end{bmatrix}$$

where:

$$\mathbf{g}^*_{\mathbf{x}\boldsymbol{\lambda}} = (\partial_{\mathbf{x}}\partial_{\boldsymbol{\lambda}}\mathbf{g})^* = -2\boldsymbol{\mu}^* \sum_{a=1}^{N} s_a (\mathbf{x} - \mathbf{x}_a) \otimes (\mathbf{x} - \mathbf{x}_a) + \mathbf{Id} = -2\boldsymbol{\mu}^* \mathbf{g}^*_{\boldsymbol{\lambda}\boldsymbol{\lambda}} + \mathbf{Id}$$

$$\mathbf{g}^*_{\mathbf{x}\tilde{\mu}} = (\partial_{\mathbf{x}}\partial_{\boldsymbol{\lambda}}\mathbf{g})^* = -2\boldsymbol{\mu}^* \sum_{a=1}^{N} s_a (\mathbf{x} - \mathbf{x}_a) \otimes \tilde{\mathbf{D}}_a = -2\boldsymbol{\mu}^* \mathbf{g}^*_{\tilde{\boldsymbol{\mu}}\boldsymbol{\lambda}}$$

with this expression, we can define exactly $\nabla f_a(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ and so we are able to evaluate the gradient of the basis function:

$$\nabla s_a^* = -s_a^* \left\{ 2\boldsymbol{\mu}^* (\mathbf{x} - \mathbf{x}_a) + \mathbf{r}^*_{\mathbf{x}} {\mathbf{J}^*}^{-1} \begin{bmatrix} \mathbf{x} - \mathbf{x}_a \\ -\tilde{\mathbf{D}}_a \end{bmatrix} \right\} \tag{7.6}$$

7.1 Monodimensional first derivatives of the shape functions

In this section we present the first derivatives for the same grid of LME program. In particular, in figure: 7.1, 7.2, 7.3, 7.4, we use the same grid with different values of α. In figure 7.5 we represent the derivatives of the shape functions of a random grid.

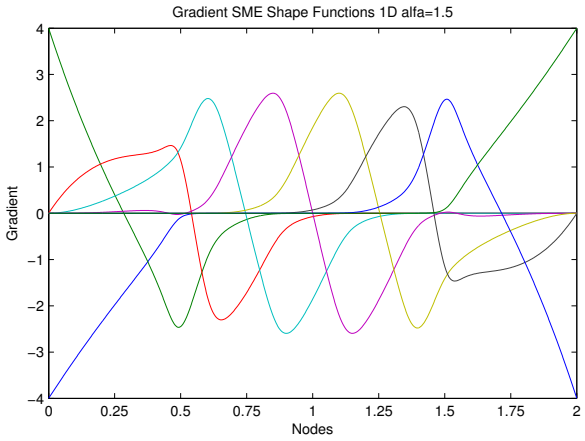


Figure 7.1: 1D first derivatives of the shape functions represented with 9 nodes and 900 definition points; α=1.5

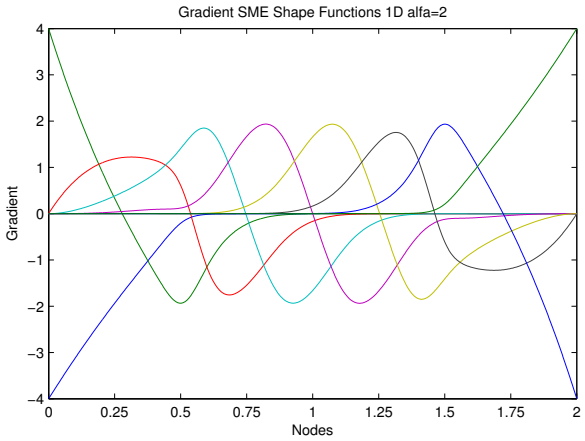


Figure 7.2: 1D first derivatives of the shape functions represented with 9 nodes and 900 definition points ; α=2.0

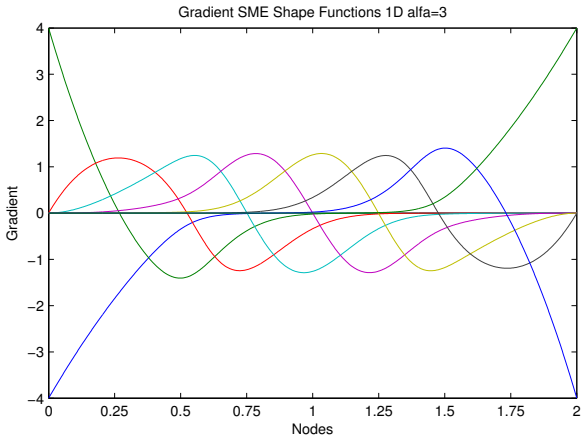


Figure 7.3: 1D first derivatives of the shape functions represented with 9 nodes and 900 definition points; α=3.0

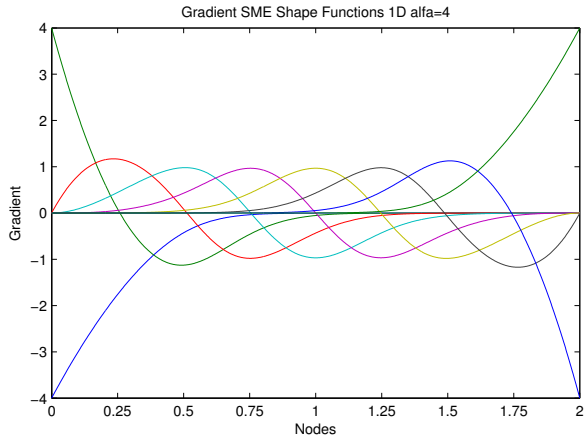


Figure 7.4: 1D first derivatives of the shape functions represented with 9 nodes and 900 definition points; α=4.0

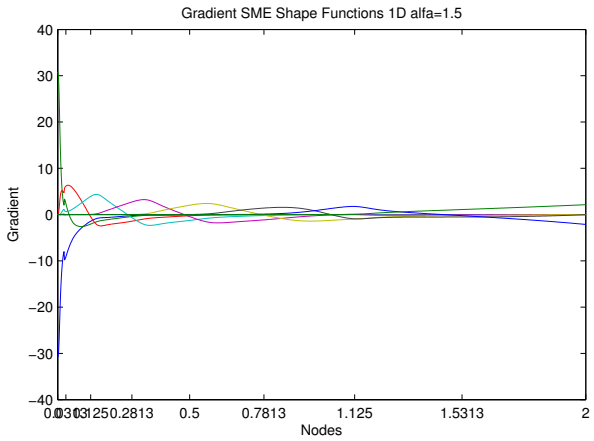


Figure 7.5: 1D first derivatives of the shape functions represented with 9 nodes and 900 definition points;random grid; α=1.5

It's possible to appreciate that for increasing values of α the derivatives spans about more nodes just like for the shape functions.

7.1.1 Program correction

To plot the gradient of the shape functions, we refer to 7.6, however, by applying the original formulation illustrated in previous chapter, the program creates a problem. If we consider the derivatives of the first and last shape functions, we can notice that in the first and the last point of the domain the first derivatives goes to 0; in fact in those particular positions we have:

$$(\mathbf{x} - \mathbf{x}_a) = 0;$$

and

$$\tilde{\mathbf{D}}_a = 0$$

so it's natural that in those points we have:

$$\nabla s_a^* = -s_a^* \left\{ 2\boldsymbol{\mu}^*(\mathbf{x} - \mathbf{x}_a) + \mathbf{r}_{\mathbf{x}}^* \mathbf{J}^{*^{-1}} \begin{bmatrix} \mathbf{x} - \mathbf{x}_a \\ -\tilde{\mathbf{D}}_a \end{bmatrix} \right\} = 0$$

This condition is right in a mathematical viewpoint but is not consistent in a physicist way if we look to the form of the shape functions. In figure 7.6 we re-plot the gradient illustrated in figure 7.1 without our correction to remark the difference with the original program: where we have

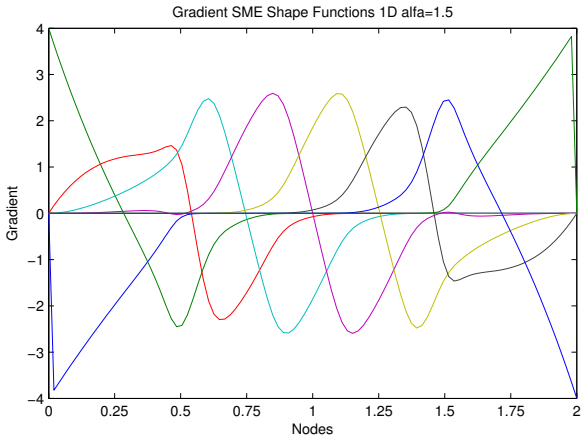


Figure 7.6: 1D first derivatives of the shape functions represented with 9 nodes and 100 definition points; original program; α=1.5

plotted the gradient for a specific value of α and we used 100 points to describe the functions, so we can better appreciate this bug of the program. This is clearly analytically correct, but it doesn't have a physicist consistency. In fact, since, the gradient (in 1D case) represent the angle ratio of the tangent line, in those specific points, it's wrong to have

$$\nabla s_a^* = 0$$

because it means the tangent line it's parallel to x-axis, which is impossible in relation with the form of the shape functions. To avoid this problem, we can reason in this way:

- fix the values of every $\mathbf{x_a}$;
- assume first **x point:**

  $$\mathbf{x}_1 = \mathbf{x}_1 + \epsilon$$

  where ϵ is a small value of your choosing, with only purpose to make:

  $$(\mathbf{x} - \mathbf{x}_a) \neq 0$$

  since we can't operate on $\tilde{\mathbf{D}}_a$ without a radical modify to program itself.
- assume last **x point:**

  $$\mathbf{x}_N = \mathbf{x}_N - \epsilon$$

  for the same reason over exposed.

In general it's reasonable to assume: $\epsilon = 10^{-8}$
In the case we have shown we put:

$$x_1 = 10^{-8} \qquad x_N = L - 10^{-8}$$

7.2 2D First Derivative shape functions representation

We present now the first derivatives of the bi-dimensional shape functions.

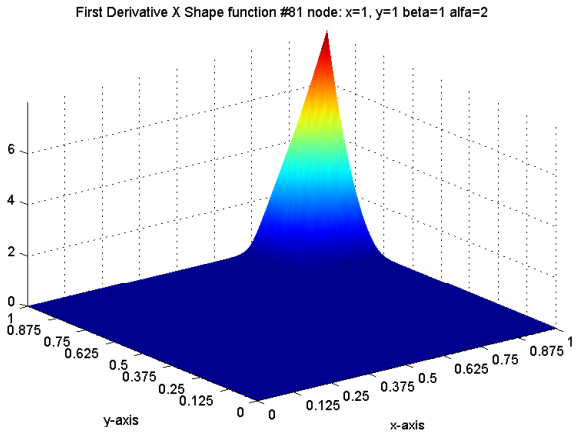


Figure 7.7: X derivative of the *angle* shape function represented with 81 nodes and 8100 definition points; α=2

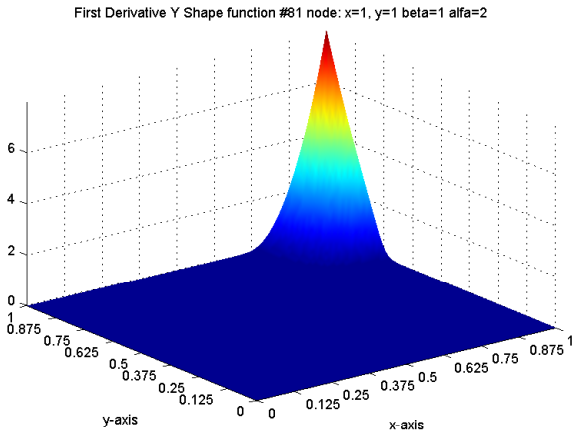


Figure 7.8: Y derivative of the *angle* shape function represented with 81 nodes and 8100 definition points; α=2

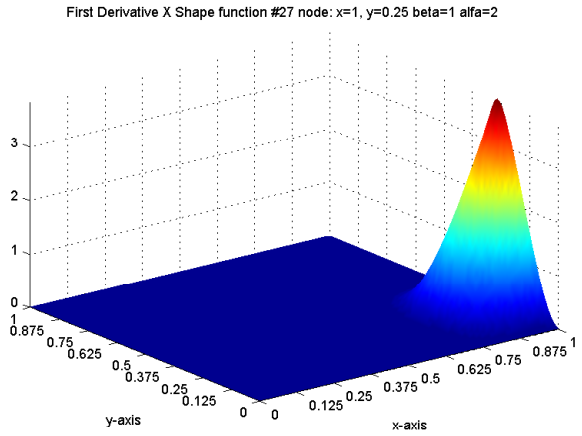


Figure 7.9: X derivative of the *border* shape function represented with 81 nodes and 8100 definition points; α=2

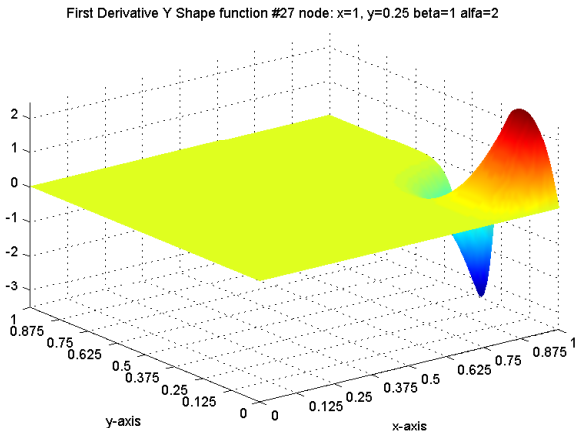


Figure 7.10: Y derivative of the *border* shape function represented with 81 nodes and 8100 definition points; α=2

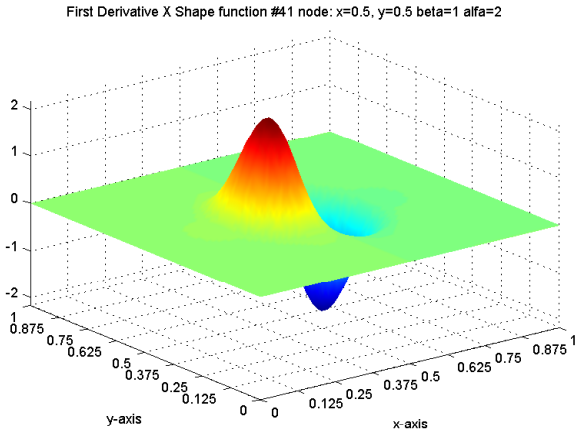


Figure 7.11: X derivative of the *center* shape function represented with 81 nodes and 8100 definition points; α=2

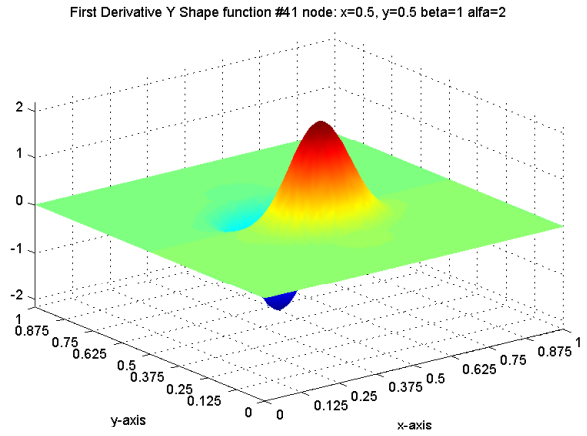


Figure 7.12: Y derivative of the *center* shape function represented with 81 nodes and 8100 definition points; α=2

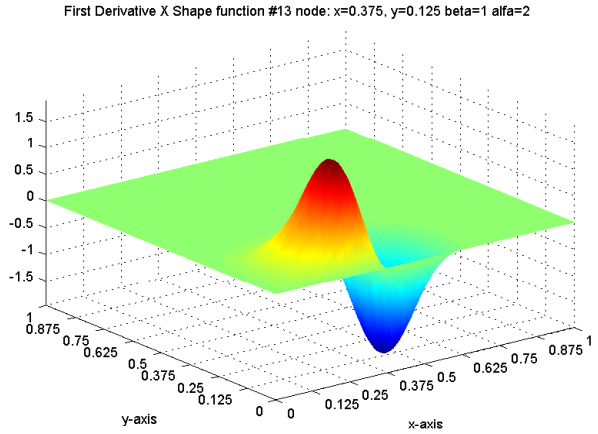


Figure 7.13: X derivative of the *random* shape function represented with 81 nodes and 8100 definition points; α=2

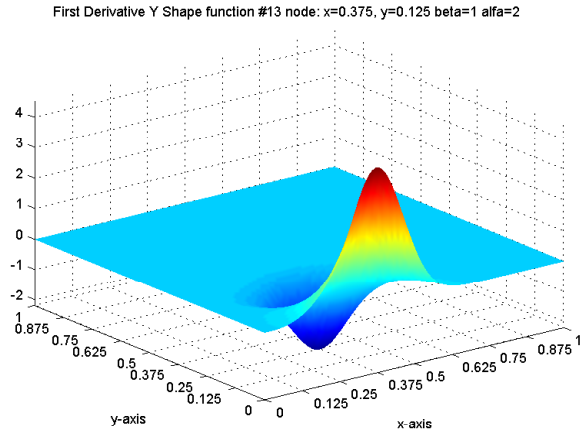


Figure 7.14: Y derivative of the *random* shape function represented with 81 nodes and 8100 definition points; α=2

7.2.1 Program correction

It is possible to recover also in 2D case the same problem illustrated for mono-dimensional domain. The greater difficulty we have to face it's the fact we find the bug in every border of the domain. In figure 7.15 we displace the derivatives that involve this problem:

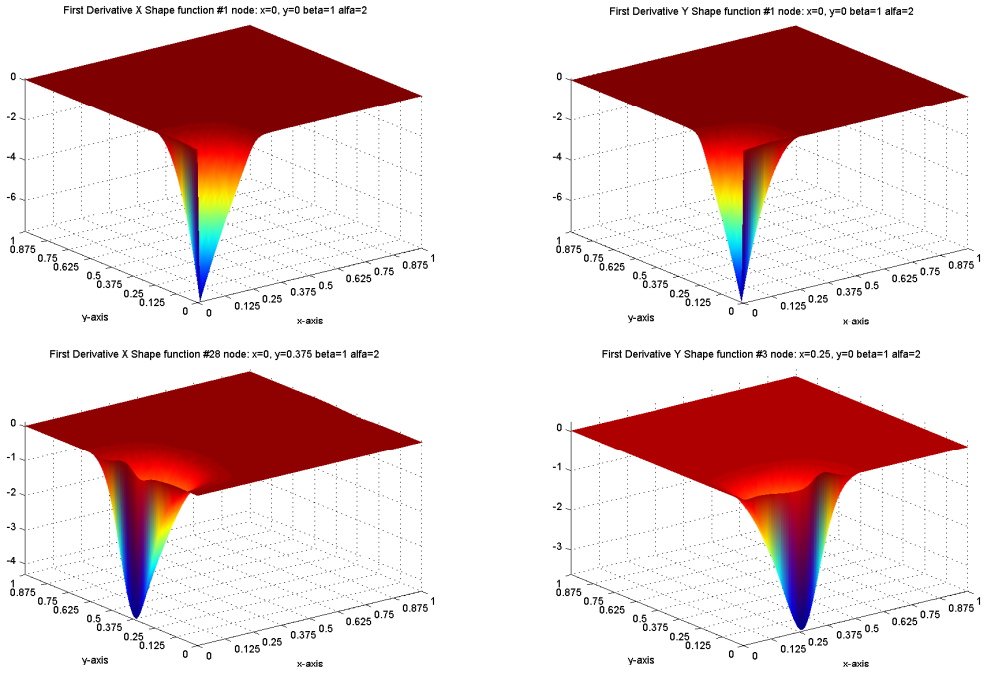


Figure 7.15: 2D shape functions X and Y gradient represented without any program correction

we can appreciate how each component of the gradient function near the border or the angle nodes present the extended version of the problem we had for 1D first derivative. The reason of this are analogous to ones exposed in chapter §7.1.1. In this case, clearly, the gradient represent the 2 unit vector normal to tangent plane.

To avoid this problem, we can follow this method:

- For angle nodes:

  we can impose those conditions:

  $$\mathbf{x}_{angle} = \mathbf{x}_{angle} \pm \epsilon \qquad \mathbf{y}_{new} = \mathbf{y}_{prevoius} \pm \epsilon$$

  where sign "+" or "-" is introduced with the purpose to move to interior nodes the previous value of $\vec{\mathbf{x}}$.

- For border nodes:

  in general we can adopt this way:

  $$\begin{bmatrix} \mathbf{x}_{border} \\ \mathbf{y}_{border} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{border} \\ \mathbf{y}_{border} \end{bmatrix} \pm \begin{bmatrix} \epsilon \\ 0 \end{bmatrix} or \pm \begin{bmatrix} 0 \\ \epsilon \end{bmatrix}$$

where we use the first or the second addend in order to move the border point to the line of the "next to boundary nodes", and with the same criteria we choose sign "+" or "-".

- For nodes belonging to not orthogonal lines:

  In this work we consider only bi-dimensional elements composed of orthogonal lines. If we want to extend the problem to elements with non orthogonal borderlines, we have to move every border node to interior ones on the direction described by the vector normal to the considered line. In general:

$$\begin{bmatrix} \mathbf{x}_{border} \\ \mathbf{y}_{border} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{border} \\ \mathbf{y}_{border} \end{bmatrix} \cdot \mathbf{n} \pm \begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix} \cdot \mathbf{n}$$

  where $\tilde{\mathbf{n}}$ is a unit vector normal to the considered line. The sign "+" or "-" is chosen in order to move the considered node to interior ones.

Like for the mono-dimensional case we can assume $\epsilon = 10^{-8}$

Chapter 8

Second derivatives of the shape functions

Thanks to the second order consistency condition, we can evaluate consistently the second derivatives of *Max Ent* approximants [8]. The second spatial derivatives of basis functions can be written in this way:

$$Hs_a^* = \frac{1}{s_a^*}\nabla s_a^* \otimes \nabla s_a^* + s_a^* H f_a^* - s_a^* \sum_{b=1}^{N} \frac{1}{s_b^*}\nabla_b^* \otimes \nabla s_b^* - s_a^* \sum_{b=1}^{N} s_b^* H f_b^* \tag{8.1}$$

where is simple to recognize the only unknown term is Hf_a^*. To compute it we must follow the mixed partial variable chain rule:

$$\begin{aligned}\frac{\partial^2 f_a}{\partial x^2} = Hf_a^* = \frac{\partial}{\partial x}\left[\frac{\partial f_a}{\partial \lambda}\frac{\partial \lambda}{\partial x}\right] + \frac{\partial}{\partial x}\left[\frac{\partial f_a}{\partial x}\frac{\partial x}{\partial x}\right] + \frac{\partial}{\partial x}\left[\frac{\partial f_a}{\partial \tilde{\mu}}\frac{\partial \tilde{\mu}}{\partial x}\right] = \\ \left(\frac{\partial^2 f_a}{\partial x^2}\frac{\partial x}{\partial x} + \frac{\partial^2 f_a}{\partial \lambda \partial x}\frac{\partial \lambda}{\partial x} + \frac{\partial^2 f_a}{\partial \mu \partial x}\frac{\partial \mu}{\partial x}\right)\frac{\partial x}{\partial x} + \frac{\partial f_a}{\partial x}\frac{\partial^2 x}{\partial x^2} \quad + \\ \left(\frac{\partial^2 f_a}{\partial \lambda^2}\frac{\partial \lambda}{\partial x} + \frac{\partial^2 f_a}{\partial x \partial \lambda}\frac{\partial x}{\partial x} + \frac{\partial^2 f_a}{\partial \mu \partial \lambda}\frac{\partial \mu}{\partial x}\right)\frac{\partial \lambda}{\partial x} + \frac{\partial f_a}{\partial \lambda}\frac{\partial^2 \lambda}{\partial x^2} \quad + \\ \left(\frac{\partial^2 f_a}{\partial \mu^2}\frac{\partial \mu}{\partial x} + \frac{\partial^2 f_a}{\partial x \partial \mu}\frac{\partial x}{\partial x} + \frac{\partial^2 f_a}{\partial \lambda \partial \mu}\frac{\partial \lambda}{\partial x}\right)\frac{\partial \mu}{\partial x} + \frac{\partial f_a}{\partial \mu}\frac{\partial^2 \mu}{\partial x^2} \quad .\end{aligned}$$

By applying the previous formulation to f_a^* we obtain:

$$\begin{aligned}Hf_a^* = -2\boldsymbol{\mu}^* + D\boldsymbol{\lambda}^* + (\mathbf{D}\boldsymbol{\lambda}^*)^{\mathbf{T}} + (\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}} f_a)(\mathbf{D}\boldsymbol{\mu}^*)^{\mathbf{T}} + (\mathbf{D}\boldsymbol{\mu}^*)(\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}} f_a)^{\mathbf{T}} + \\ \sum_{k=1}^{\frac{d}{2}(d+1)} (\mathbf{D}^{\mathbf{2}}_{\mathbf{x}}\tilde{\boldsymbol{\mu}}_k)^*(\partial_{\tilde{\boldsymbol{\mu}}_k} f_a)^* + \sum_{k=1}^{d} (\mathbf{D}^{\mathbf{2}}_{\mathbf{x}}\boldsymbol{\lambda}_k)^*(\partial_{\boldsymbol{\lambda}_k} f_a)^*\end{aligned} \tag{8.2}$$

where d is the dimension of our problem and k refers to the *k-th* component of the Lagrangian multiplier we have already calculated. Now by introducing 8.2 in 8.1 we have:

$$\begin{aligned} Hs_a^* = \frac{1}{s_a^*}\nabla s_a^* \otimes \nabla s_a^* - s_a^* \sum_{b=1}^{N} \frac{1}{s_b^*}\nabla_b^* \otimes \nabla s_b^* + s_a^*(\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}} f_a)(\mathbf{D}\boldsymbol{\mu}^*)^T + \\ s_a^*(\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}} f_a)^T(\mathbf{D}\boldsymbol{\mu}^*) - s_a^* \sum_{k=1}^{\frac{d}{2}(d+1)} \tilde{\mathbf{D}}_{ak}(\mathbf{D_x}^2\tilde{\mu}_k)^* + s_a^* \sum_{k=1}^{d}(\mathbf{x}_a - \mathbf{x}_{ak})(\mathbf{D_x^2}\boldsymbol{\lambda}_k)^* \end{aligned} \tag{8.3}$$

where now the unknowns are represented by $(\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}} f_a)$, $(\mathbf{D_x}^2\tilde{\boldsymbol{\mu}}_k)^*$, $(\mathbf{D_x}^2\boldsymbol{\lambda}_k)^*$. It's also interesting to notice how it's useful to evaluate $\frac{1}{s_a^*}\nabla_a^*$ in this way:

$$\frac{1}{s_a^*}\nabla_a^* = -\left\{ 2\boldsymbol{\mu}^*(\mathbf{x} - \mathbf{x}_a) + \mathbf{r}_{\mathbf{x}}^* \mathbf{J}^{*^{-1}} \begin{bmatrix} \mathbf{x} - \mathbf{x}_a \\ -\tilde{\mathbf{D}}_a \end{bmatrix} \right\}$$

to avoid error amplification in particular when we consider the shape functions far away from the investigated node. In those points, in fact, we have both the shape function both its gradient close to 0, so is very probably to have a consistent error amplification.

$(\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}} f_a)$ is the simplest unknown to find, in fact it's given by:

- mono-dimensional grid:
$$-2(x - x_a)$$
- bi-dimensional grid:
$$-2\begin{bmatrix} x_1 - x_{a1} & 0 & x_2 - x_{a2} \\ 0 & x_2 - x_{a2} & x_1 - x_{a1} \end{bmatrix}$$

The most complicated part of the program is the one which goes down to the evaluate the other two unknowns, in fact they are not explicitly available and they calculation includes a linear problem solving. We start from the analytic condition:

$$\mathbf{D_x^2 r}^* = \mathbf{0}$$

from it we can derive the following conditions:

$$\mathbf{D_x^2 g}\boldsymbol{\lambda}^* = \mathbf{0}$$

$$\mathbf{D_x^2 g}\tilde{\boldsymbol{\mu}}^* = \mathbf{0}$$

where we evaluate:

$$\begin{aligned} \mathbf{D_x^2 g}_{\boldsymbol{\lambda}_\mathbf{i}^*} = (\partial_{\mathbf{x}}\partial_{\mathbf{x}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^* + (\partial_{\mathbf{x}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^*(\mathbf{D}\boldsymbol{\lambda}^*)^T + (\mathbf{D}\boldsymbol{\lambda}^*)(\partial_{\mathbf{x}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^{*T} + (\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^*(\mathbf{D}\boldsymbol{\mu}^*)^T + \\ (\mathbf{D}\boldsymbol{\mu}^*)(\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^{*T} + (\mathbf{D}\boldsymbol{\lambda}^*)(\partial_{\boldsymbol{\lambda}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})(\mathbf{D}\boldsymbol{\mu}^*)^T + (\mathbf{D}\boldsymbol{\mu}^*)(\partial_{\boldsymbol{\lambda}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^T(\mathbf{D}\boldsymbol{\lambda}^*)^T + \\ (\mathbf{D}\boldsymbol{\lambda}^*)(\partial_{\boldsymbol{\lambda}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})(\mathbf{D}\boldsymbol{\lambda}^*)^T + (\mathbf{D}\boldsymbol{\mu}^*)(\partial_{\tilde{\boldsymbol{\mu}}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^*(\mathbf{D}\boldsymbol{\mu}^*)^T + \\ \sum_{k=1}^{d} \mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}\boldsymbol{\lambda}_\mathbf{k}}{}^*(\mathbf{D_x^2}\boldsymbol{\lambda}_k)^* + \sum_{k=1}^{\frac{d}{2}(d+1)} \mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}}\tilde{\boldsymbol{\mu}}_\mathbf{k}{}^*(\mathbf{D_x^2}\tilde{\boldsymbol{\mu}}_k)^* = \mathbf{0} \end{aligned} \tag{8.4}$$

and:

$$\begin{aligned}\mathbf{D^2_x g}_{\tilde{\boldsymbol{\mu}}^*_\mathbf{j}} = (\partial_\mathbf{x}\partial_\mathbf{x}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^* + (\partial_\mathbf{x}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^*(\mathbf{D}\boldsymbol{\lambda}^*)^T + (\mathbf{D}\boldsymbol{\lambda}^*)(\partial_\mathbf{x}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^{*T} + (\partial_\mathbf{x}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^*(\mathbf{D}\boldsymbol{\mu}^*)^T + \\ (\mathbf{D}\boldsymbol{\mu}^*)(\partial_\mathbf{x}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}\mathbf{i}})^{*T} + (\mathbf{D}\boldsymbol{\lambda}^*)(\partial_{\tilde{\boldsymbol{\mu}}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^T(\mathbf{D}\boldsymbol{\mu}^*)^T + (\mathbf{D}\boldsymbol{\mu}^*)(\partial_{\tilde{\boldsymbol{\mu}}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^T(\mathbf{D}\boldsymbol{\lambda}^*)^T + \\ (\mathbf{D}\boldsymbol{\lambda}^*)(\partial_{\boldsymbol{\lambda}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})(\mathbf{D}\boldsymbol{\lambda}^*)^T + (\mathbf{D}\boldsymbol{\mu}^*)(\partial_{\tilde{\boldsymbol{\mu}}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^*(\mathbf{D}\boldsymbol{\mu}^*)^T + \\ \sum_{k=1}^{d}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}}{\boldsymbol{\lambda}_\mathbf{k}}^*(\mathbf{D^2_x}\boldsymbol{\lambda}_k)^* + \sum_{k=1}^{\frac{d}{2}(d+1)}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}}{\tilde{\boldsymbol{\mu}}_\mathbf{k}}^*(\mathbf{D^2_x}\tilde{\boldsymbol{\mu}}_k)^*\end{aligned} \tag{8.5}$$

Given that $d = 2$ it's simple to find that i goes from 1 to 2 and j goes from 1 to 3. We notice also $\mathbf{D^2_x g}\boldsymbol{\lambda}$ and $\mathbf{D^2_x g}\tilde{\boldsymbol{\mu}}$ are third order tensors and so for every point of the domain we have to solve an algebraic system with 20 linear independent equations with 20 unknowns. This system can be divided into 4 linear independent systems with 5 unknowns for each one. This fact is very important in a implementation view, because, by using Gauss-Jordan method to invert the matrix of the coefficients of the linear system, the split of the system is suitable in order to reduce the machine time execution. We can easily solve the systems by knowing that:

$$\begin{bmatrix}\mathbf{D}\boldsymbol{\lambda} & \mathbf{D}\tilde{\boldsymbol{\mu}}\end{bmatrix} = -\mathbf{r}^*_\mathbf{x}\mathbf{J}^{-1}$$

where $\mathbf{r}^*_\mathbf{x}$ has been previously defined by in chapter §(8). Instead, we have already available all the terms $\mathbf{g}_{\lambda\lambda}, \mathbf{g}_{\lambda\tilde{\mu}}, \mathbf{g}_{\tilde{\mu}\lambda}, \mathbf{g}_{\tilde{\mu}\tilde{\mu}}$, in fact, since they are the Lagrange multipliers of Newton-Raphson solutions, they can be recovered in term $\mathbf{J}$. The remaining 12 derivatives presented in expressions 8.4 and 8.5 can be evaluated in this way:

$$(\partial_\mathbf{x}\partial_\mathbf{x}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^* = 4\sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})[\boldsymbol{\mu}^*(\mathbf{x} - \mathbf{x}_b)] \otimes [\boldsymbol{\mu}^*(\mathbf{x} - \mathbf{x}_b)]$$

$$(\partial_\mathbf{x}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}}) = -2\boldsymbol{\mu}^*\sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})(\mathbf{x} - \mathbf{x}_b) \otimes (\mathbf{x} - \mathbf{x}_b)$$

$$(\partial_\mathbf{x}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^* = 2\boldsymbol{\mu}^*\sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})(\mathbf{x} - \mathbf{x}_b) \otimes \tilde{\mathbf{D}}_b + \sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})(\partial_\mathbf{x}\partial_{\tilde{\boldsymbol{\mu}}}f_b)^*$$

$$(\partial_{\boldsymbol{\lambda}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^* = -\sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})(\mathbf{x} - \mathbf{x}_b) \otimes \tilde{\mathbf{D}}_b$$

$$(\partial_{\boldsymbol{\lambda}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^* = -\sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})(\mathbf{x} - \mathbf{x}_b) \otimes (\mathbf{x} - \mathbf{x}_b)$$

$$(\partial_{\tilde{\boldsymbol{\mu}}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\boldsymbol{\lambda}_\mathbf{i}})^* = \sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})\tilde{\mathbf{D}}_b \otimes \tilde{\mathbf{D}}_b$$

$$\begin{aligned}(\partial_\mathbf{x}\partial_\mathbf{x}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^* = -2\sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})\big[2\tilde{\mathbf{D}_{\mathbf{b_j}}}\boldsymbol{\mu}^*(\mathbf{x} - \mathbf{x}_b) + (\partial_\mathbf{x}\partial_{\tilde{\boldsymbol{\mu}}_j}f_b)^*\big] \otimes [\boldsymbol{\mu}^*(\mathbf{x} - \mathbf{x}_b)] \\ -2\boldsymbol{\mu}^*\sum_{b=1}^{N}s^*_b(\mathbf{x} - \mathbf{x}_b) \otimes (\partial_\mathbf{x}\partial_{\tilde{\boldsymbol{\mu}}}f_b)^* - 2\mathbf{N_j}\end{aligned}$$

$$(\partial_\mathbf{x}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^* = \sum_{b=1}^{N}s^*_b(\mathbf{x}_i - \mathbf{x}_{b_i})\big[2\tilde{\mathbf{D}_{b_j}}\boldsymbol{\mu}^*(\mathbf{x} - \mathbf{x}_b) + (\partial_\mathbf{x}\partial_{\tilde{\boldsymbol{\mu}}_j}f_b)^*\big] \otimes (\mathbf{x} - \mathbf{x}_b)$$

$$(\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}})^* = -\sum_{b=1}^{N} s_b^*(\mathbf{x}_i - \mathbf{x}_{b_i})\left[2\tilde{\mathbf{D}}_{b_j}\mu^*(\mathbf{x}-\mathbf{x}_b) + (\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}_j} f_b)^*\right]\otimes\tilde{\mathbf{D}}_\mathbf{b} - \sum_{b=1}^{N} s_b^*\tilde{\mathbf{D}}_{\mathbf{b_j}}(\partial_{\mathbf{x}}\partial_{\tilde{\boldsymbol{\mu}}} f_b)^*$$

$$(\partial_{\tilde{\boldsymbol{\mu}}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}}) = \sum_{b=1}^{N} s_b^*\tilde{\mathbf{D}}_{\mathbf{b_j}}\tilde{\mathbf{D}}_\mathbf{b}\otimes(\mathbf{x}-\mathbf{x}_b)$$

$$(\partial_{\boldsymbol{\lambda}}\partial_{\boldsymbol{\lambda}}\mathbf{g}_{\tilde{\mu}_\mathbf{j}}) = -\sum_{b=1}^{N} s_b^*\tilde{\mathbf{D}}_{\mathbf{b_j}}(\mathbf{x}-\mathbf{x}_b)\otimes(\mathbf{x}-\mathbf{x}_b)$$

$$(\partial_{\tilde{\boldsymbol{\mu}}}\partial_{\tilde{\boldsymbol{\mu}}}\mathbf{g}_{\tilde{\boldsymbol{\mu}}_\mathbf{j}}) = -\sum_{b=1}^{N} s_b^*\tilde{\mathbf{D}}_{\mathbf{b_j}}\tilde{\mathbf{D}}_\mathbf{b}\otimes\tilde{\mathbf{D}}_\mathbf{b}$$

where $\mathbf{N_j}$ in our instance of a 2D problem is represented by:

$$\left[\mathbf{N_j}\right]_{j=1,2,3} = \left[\mathbf{N_1 N_2 N_3}\right] = \left[\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}, \begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}, \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\right]$$

in a mono-dimensional problem $\mathbf{N_j}$ it's simply 1.

We can now evaluate the second spatial derivatives of the shape functions.

8.1 Mono-dimensional second derivatives of the shape functions

In the followings figures we displace the second derivatives of the mono-dimensional shape functions, both for the regular grid and for the random one.

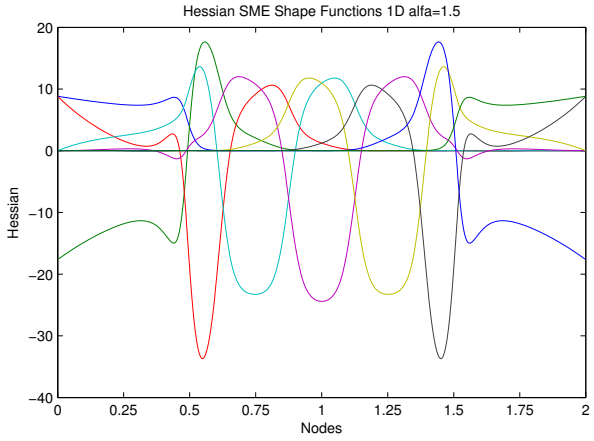


Figure 8.1: Second derivatives of the shape functions represented with 9 nodes and 900 definition points; α=1.5

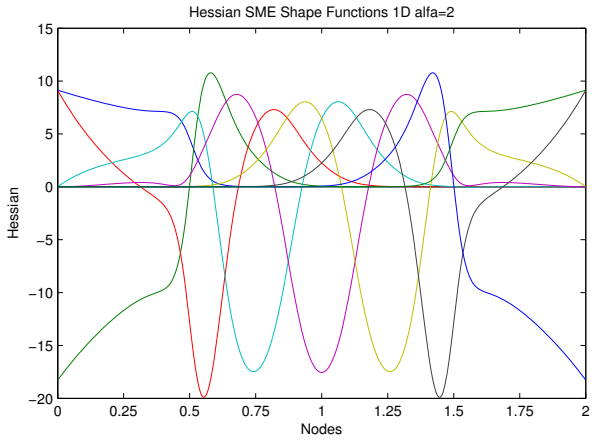


Figure 8.2: Second derivatives of the shape functions represented with 9 nodes and 900 definition points; α=2.0

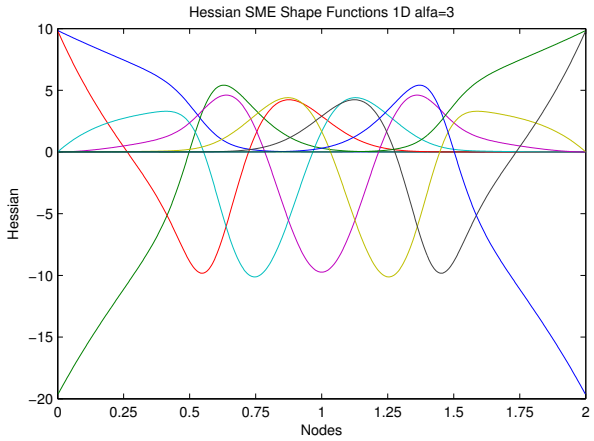


Figure 8.3: Second derivatives of the shape functions represented with 9 nodes and 900 definition points; α=3.0

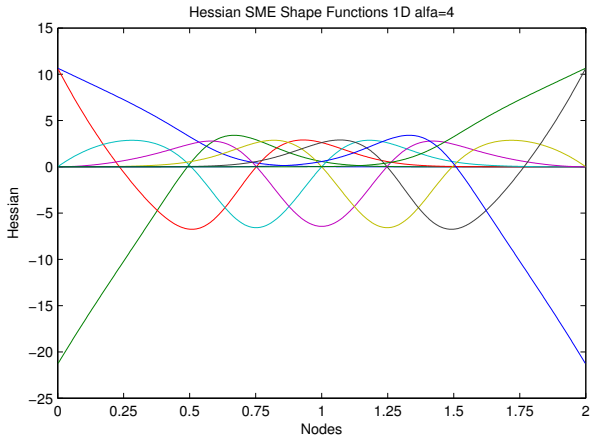


Figure 8.4: Second derivatives of the shape functions represented with 9 nodes and 900 definition points; α=4.0

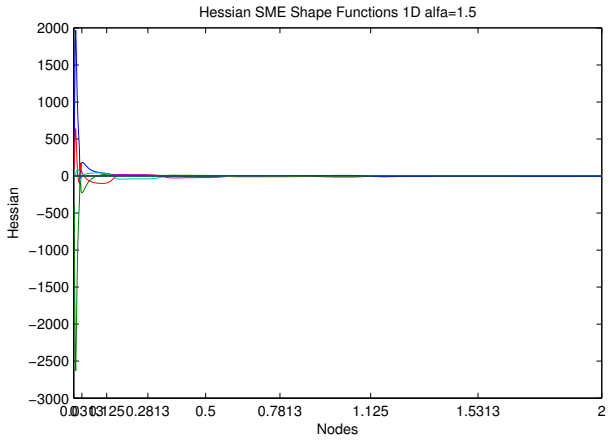


Figure 8.5: Second derivatives of the shape functions represented with 9 nodes and 900 definition points; random grid; α=1.5

8.1.1 Program correction

It's possible to find out an error very similar to that we found for the first derivatives, but in this case the values of the second derivatives near the borders goes to $+\infty$. To correct the bug, we can use an adapted Finite difference Method and work in this way:

$$f(x_1)'' = 2 \cdot f(x_2)'' - f(x_3)''$$

$$f(x_N)'' = 2 \cdot f(x_{N-2})'' - f(x_{N-3})''$$

given that we know that near the border the slope of the hessian function is quite constant. In general we'll have an error on the border which decrease with the increasing of the nodes.

8.1.2 Bi-dimensional second derivatives of the shape functions

By applying the program 8.1 in following figures we displace the second derivatives of the basis functions:

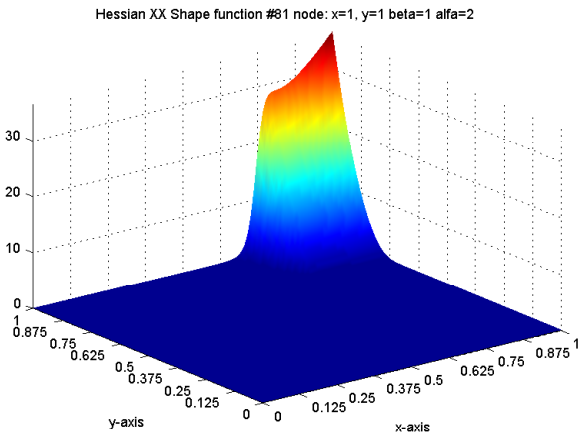


Figure 8.6: XX second derivative of the *angle* shape function represented with 81 nodes and 8100 definition points; α=2

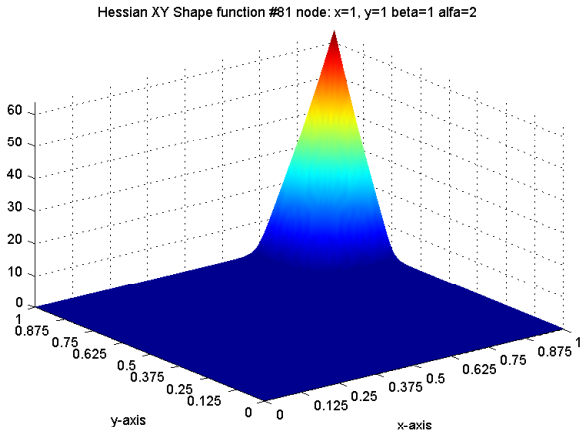


Figure 8.7: XY second derivative of the *angle* shape function represented with 81 nodes and 8100 definition points; α=2

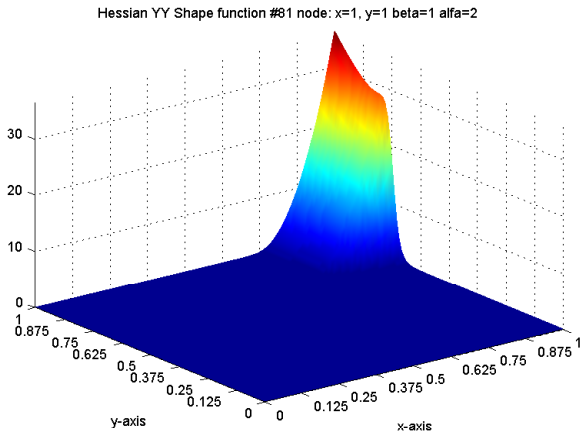


Figure 8.8: YY second derivative of the *angle* shape function represented with 81 nodes and 8100 definition points; α=2

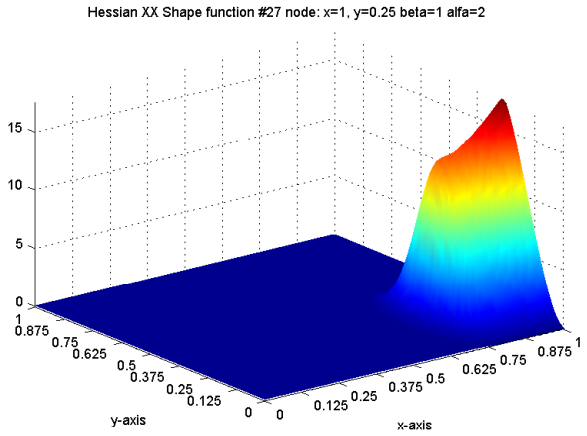


Figure 8.9: XX second derivative of the *border* shape function represented with 81 nodes and 8100 definition points; α=2

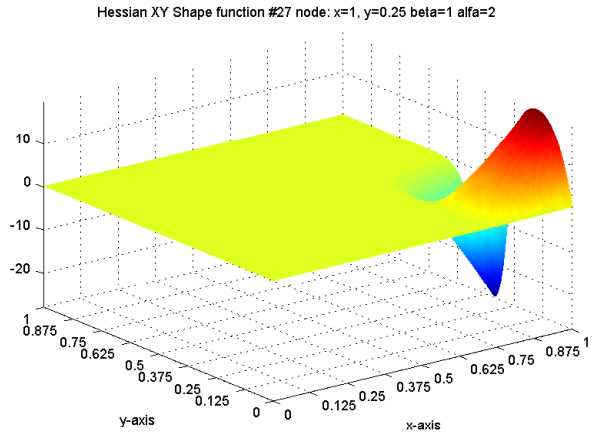


Figure 8.10: XY second derivative of the *border* shape function represented with 81 nodes and 8100 definition points; α=2

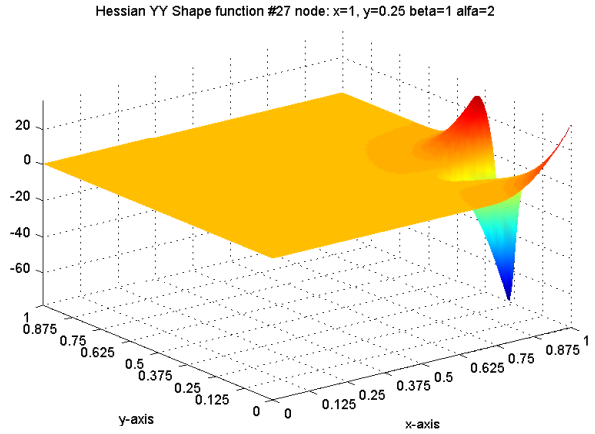


Figure 8.11: YY second derivative of the *border* shape function represented with 81 nodes and 8100 definition points; α=2

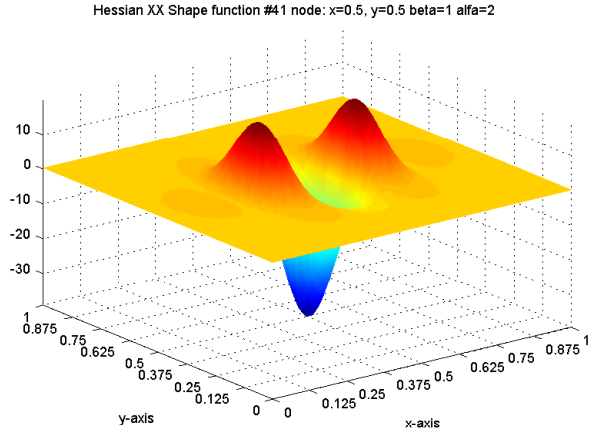


Figure 8.12: XX second derivative of the *center* shape function represented with 81 nodes and 8100 definition points; α=2

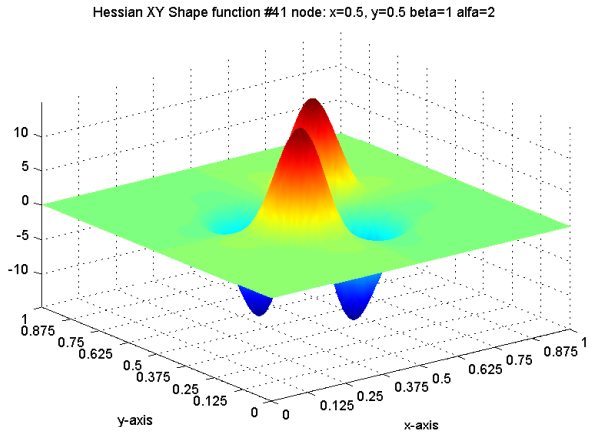


Figure 8.13: XY second derivative of the *center* shape function represented with 81 nodes and 8100 definition points; α=2

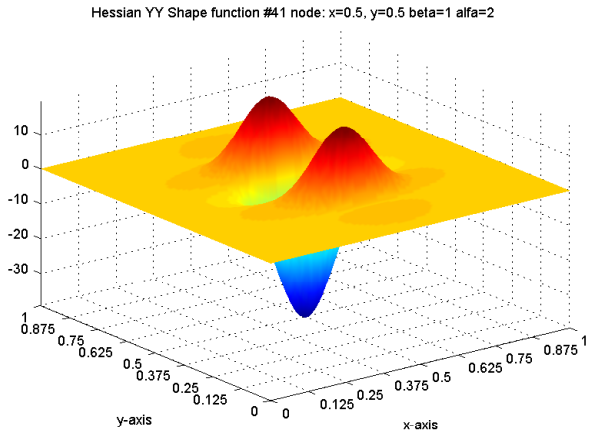


Figure 8.14: YY second derivative of the *center* shape function represented with 81 nodes and 8100 definition points; α=2

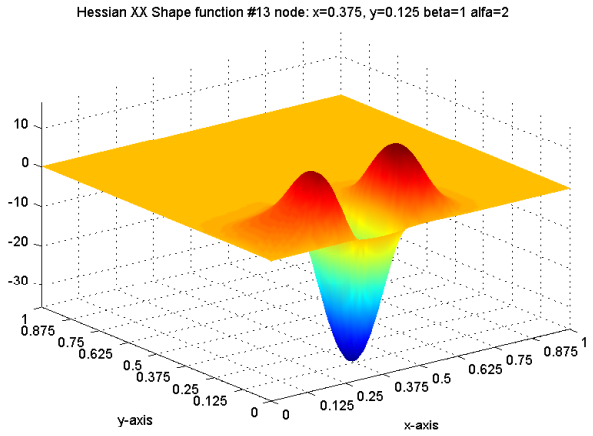


Figure 8.15: XX second derivative of the *random* shape function represented with 81 nodes and 8100 definition points; α=2

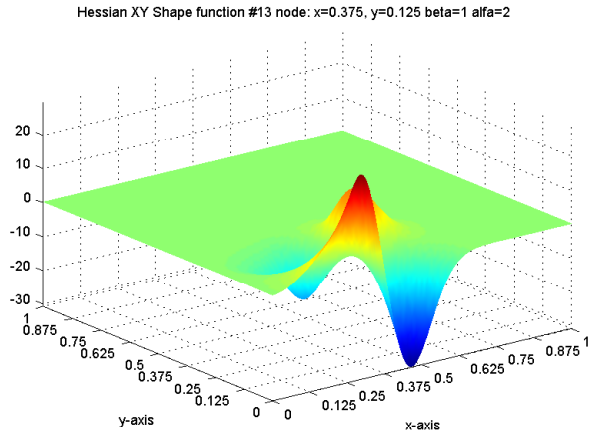


Figure 8.16: XY second derivative of the *random* shape function represented with 81 nodes and 8100 definition points; α=2

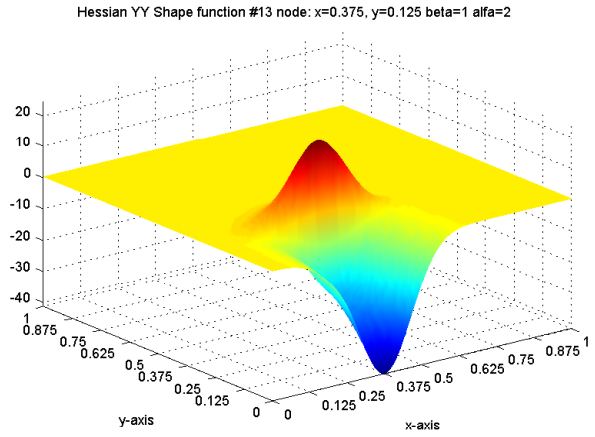


Figure 8.17: YY second derivative of the *random* shape function represented with 81 nodes and 8100 definition points; α=2

8.1.3 Program correction

The original program present the same problem we had in the mono-dimensional case. To avoid this bug it's possible to operate in this way:

- Border nodes
  for this kind of nodes, it's possible to apply the following scheme:
  - (0;0)-(L_x;0) border

    $$f''_{xx,xy,yx,yy}(num,:) = f''_{xx,xy,yx,yy}(num+1,:) * 2 - f''_{xx,xy,yx,yy}(num-2,:)$$

    with: num=$N_y \cdot$ i+1 for $i = 1 \rightarrow N_x - 2$
  - (0;L_y)-(L_x;L_y) border

    $$f''_{xx,xy,yx,yy}(num,:) = f''_{xx,xy,yx,yy}(num-1,:) * 2 - f''_{xx,xy,yx,yy}(num-2,:)$$

    with: num=$N_y \cdot$ i+N_y for $i = 1 \rightarrow N_x - 2$

– (0;0)-(0;L_y) border

$$f''_{xx,xy,yx,yy}(i,:) = f''_{xx,xy,yx,yy}(num,:) * 2 - f''_{xx,xy,yx,yy}(num + N_y,:)$$

with: num=i+N_y for $i = 2 \rightarrow N_y - 1$

– (0;L_x)-(L_x;L_y) border

$$f''_{xx,xy,yx,yy}(i,:) = f''_{xx,xy,yx,yy}(num,:) * 2 - f''_{xx,xy,yx,yy}(num - N_y,:)$$

with: num=i-N_y for $i = N_x N_y - N_y + 2 \rightarrow N_x N_y$

- Angle nodes

$$f''_{xx,xy,yx,yy}(1,:) = f''_{xx,xy,yx,yy}(N_y + 2,:) * 2 - f''_{xx,xy,yx,yy}(N_y + 3,:)$$

$$f''_{xx,xy,yx,yy}(N_y,:) = f''_{xx,xy,yx,yy}(2N_y - 1,:) * 2 - f''_{xx,xy,yx,yy}(3N_y - 2,:)$$

$$f''_{xx,xy,yx,yy}(num,:) = f''_{xx,xy,yx,yy}(num - N_y + 1,:) * 2 - f''_{xx,xy,yx,yy}(num - 2N_y + 2,:)$$

with num=$N_y(N_x - 1) + 1$

$$f''_{xx,xy,yx,yy}(tot,:) = f''_{xx,xy,yx,yy}(tot - N_y - 1,:) * 2 - f''_{xx,xy,yx,yy}(tot - 2N_y - 2,:)$$

with tot=$N_y N_x$

where, we call "N_x" and "N_y" the number of grid points in each directions.

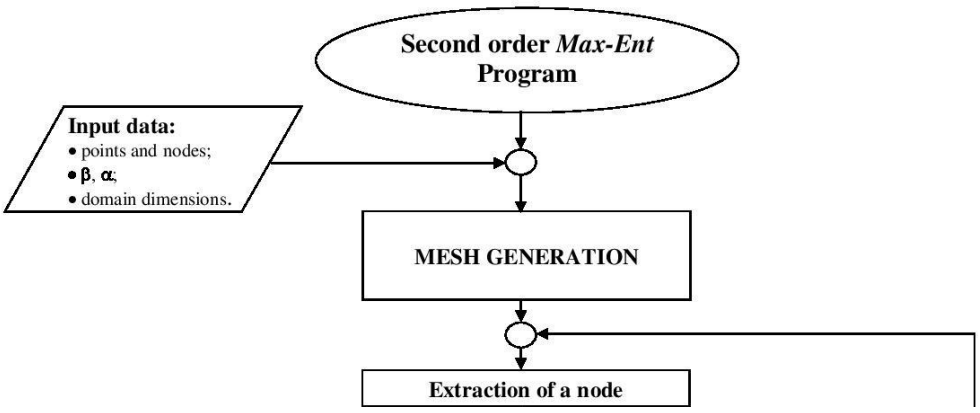
Chapter 9

Computer implementation of Second order *max-ent* program

In this chapter we want to outline the principal features of the minimizing program. Since the program is very similar to LME we have quite the same problems, in particular we want to reduce the computational time and the memory space used to save temporary and final results. The program has been written in MATLAB®for the part of mesh generation and results; FORTRAN 77®, instead has been used for the core of the program which involves the Newton-Raphson minimizing and the derivatives calculation.

9.1 General features of SME program

To create a code which is able to reproduce the SME program, we must consider that the most important and onerous part of the problem is the optimization one, made with the Newton-Raphson iterative method. Our goal is to create a program which is faster and less expensive (in memory terms) as possible, in particular since in this case the Newton-Raphson method involves 5 unknowns and not 2 as in LME.
We present now a simple flow chart to delineate the principal steps of the program.



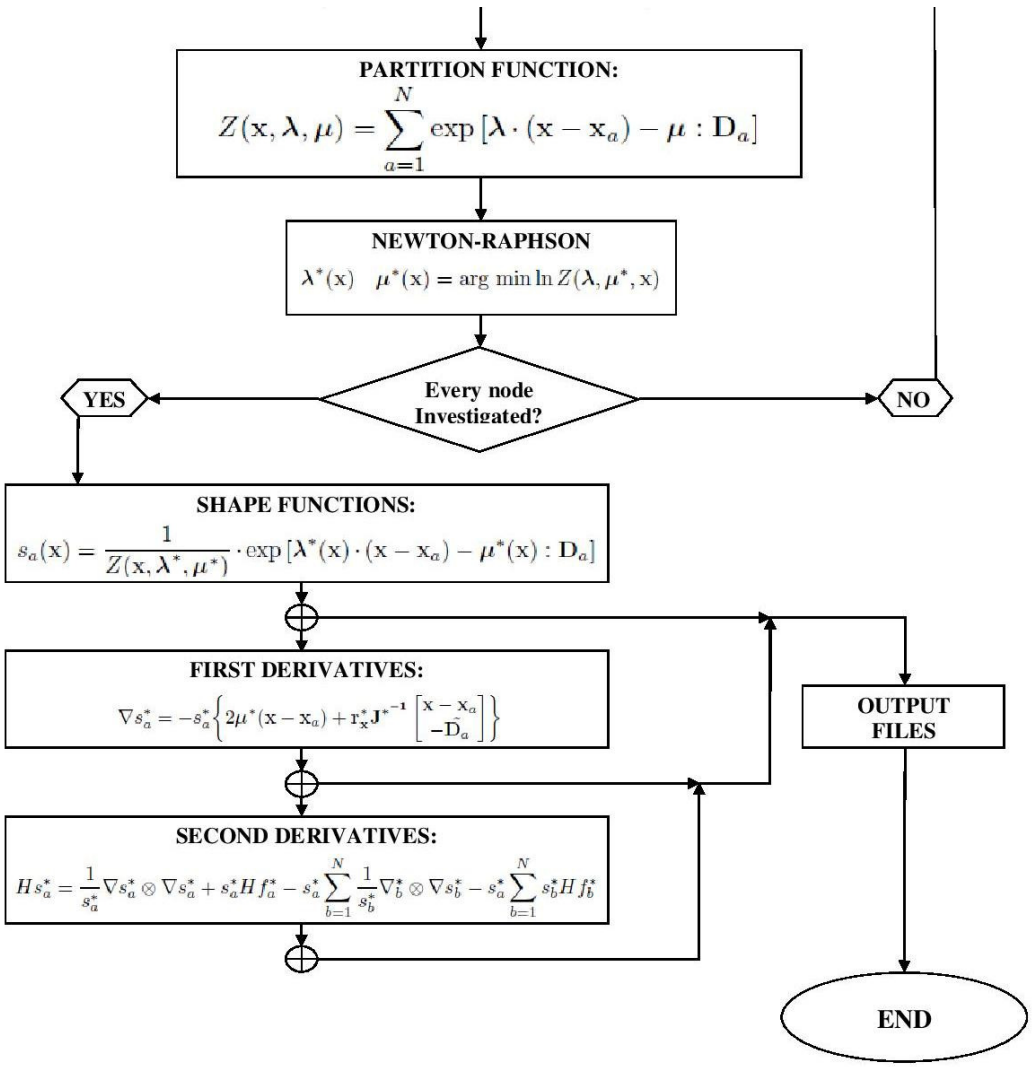


Figure 9.1: Second order *Max-Ent* program flowchart

where we have used MATLAB®for the mesh generation and FORTRAN 77®for the rest of the program. In flow-chart 9.1 we have omitted two possibly part of the program that are not essential:

- Program controls: we can control if the minimizing program has worked right by checking proprieties:
  - positivity propriety: 6.1;
  - unity propriety: 6.2;
  - linear propriety: 6.3;
  - quadratic propriety: 6.4;
  - equality of the mixed second partial derivatives.
- Results displacements: we can realize some plots in which we displace the shape functions with associated derivatives and the program controls; in this way we have a better vision of the final results.

it's possible to insert this part of the program after the output files. In this case is simplest to work with MATLAB®.

9.2 Program optimization

The computational problems involved by SME program are quite the same then LME. In particular our goal is to be able to save memory and to reduce the machine time. The strategy we can adopt is similar to which we have used for LMEs: first of all we introduce sparse matrices in order to save RAM memory. The approach we use aim at evaluate the value of any shape functions only in the points next to the investigated node, where we know the shape function has significant ordinates different to 0. It's possible to evaluate in this way the radius from the node were we want to compute the shape functions:

$$R_a = \max\left(\sqrt{-\frac{\alpha}{2}\ln\left(Tol_0\right)}\cdot h_a; 4h_a\right)$$

where: Tol_0 is the fixed tolerance below that we consider 0 the value of the shape function, h_a is the value of the maximum nodal spacing around the investigated node. In general it's reasonable to assume $Tol_0 = 10^{-9}$

Such we have illustrated, is very important in order to reduce the memory usage and, considerably also the computational time, but if we want to make more performing the program, we have to consider the most expensive part, in terms of time, is the Newton-Raphson optimization method, that in this case involves the inversion of a [5x5] matrix. To reduce the number of the iterations, the easier way is to choose the right point $\boldsymbol{\lambda}, \boldsymbol{\mu}$ form where we start the iteration. In our program we have selected as first attempt solution, the optimal solution $\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*$ of the closest node. In this way we are able to have very good results in term of number of iterations saved. According to [9], instead, it is possible to choose as initial guesses:

$$\boldsymbol{\lambda} = \mathbf{0} \qquad \text{and} \qquad \boldsymbol{\mu} = \frac{1}{2}\cdot\frac{1}{\mathbf{d}_{a,CN}}$$

where CN, means the closest node. At the end we can conclude to use for $\boldsymbol{\lambda}$ our solution and for $\boldsymbol{\mu}$ the suggested strategy, in this way the program is optimized the best.
In table 9.1 and 9.2 we report some interesting results about the regular and optimized program:

Nodes for side	Memory employed	Machine time	Iterations
	[Mb]	[s]	
20	28.8	2.93	7770
30	147	10.12	18678
40	465	33.56	34922
50	1134	89.71	56818
60	2235	178.4	84970
70	4153	315.2	119221
80	6987	496.5	161352
90	10562	716.35	207562
100	15236	1012.78	262264
120	28756	1865.62	388652
140	45620	3245.65	536251
160	68354	4865.43	698621
180	98625	7256.51	876235
200	132564	10235.69	1068352

Table 9.1: Computational values for the ORIGINAL PROGRAM

Nodes for side	Memory employed	Machine time	Iterations
	[Mb]	[s]	
20	4.92	0.66	4430
30	14.88	1.98	8089
40	25.09	5.93	12629
50	42.31	14.17	17945
60	61.50	29.80	24401
70	84.87	56.11	31582
80	118.08	96.14	40215
90	141.45	155.20	49352
100	193.11	221.30	60070
120	287.82	456.40	81256
140	488.95	786.56	102532
160	640.83	1345.20	129653
180	931.89	2156.80	163254
200	1225.83	3256.80	198625

Table 9.2: Computational values for the OPTIMIZED PROGRAM

To make easier the comprehension of the importance of the optimization, we plot the results in figure 9.2. We can note the decreasing of the time and the number of iterations, and like for LME we note the time has a lower decreasing then iterations. This condition is due to by the fact the most of saved time belongs to the part of shape functions computation, while the computation time of the derivatives remains quite the same.

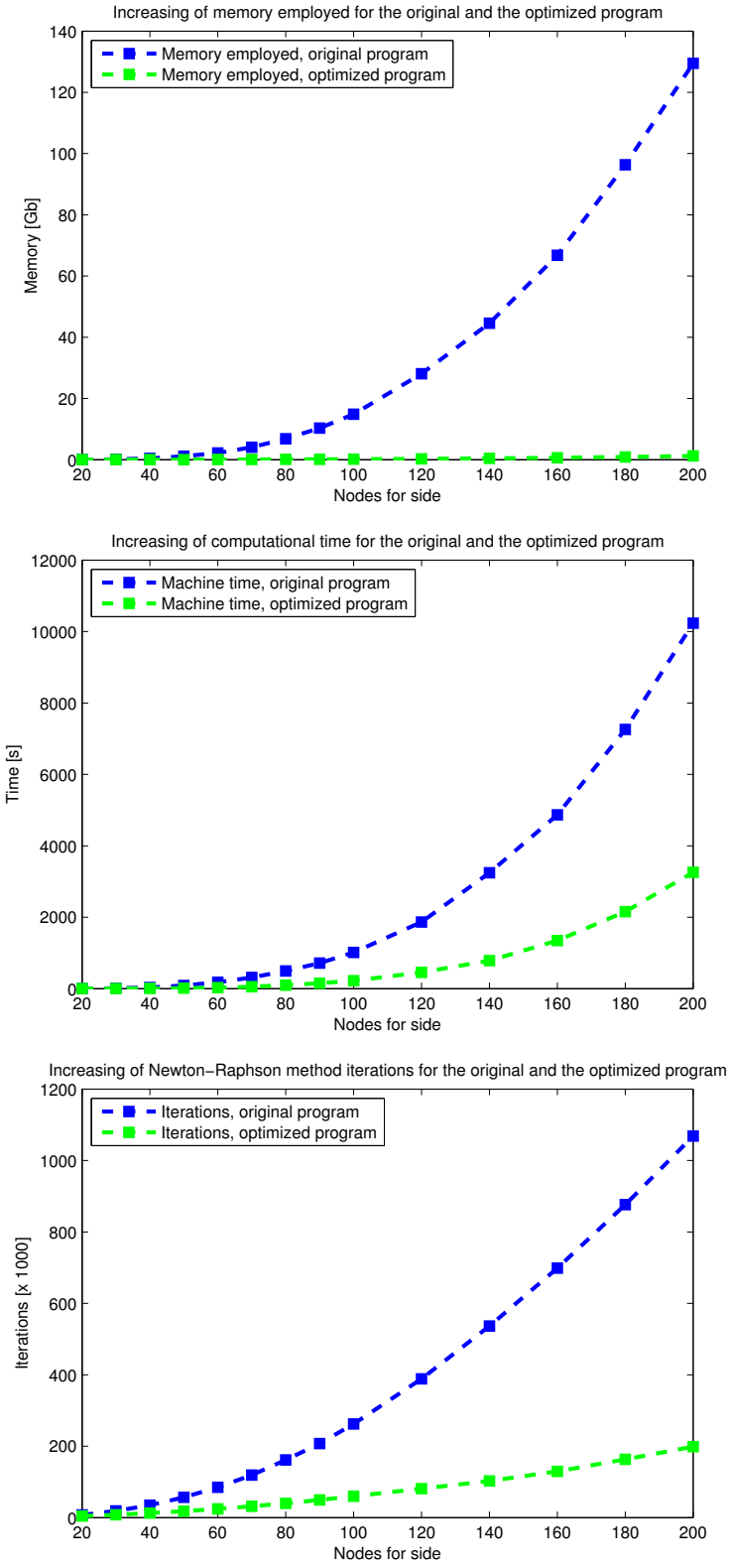


Figure 9.2: Comparison between the straight and the optimized program, in terms of memory, time and iterations

9.3 Matrix inversion

Our optimized program produce sparse matrices which are not symmetric. This involves a problem when we have to invert it in order to solve any application problem, in fact it's very hard to build up a subroutine for FORTRAN 77®. To avoid the problem it's possible to use Intel MKL PARDISO®or in alternative PARDISO 5.0.0 (open source). Those software are able to solve an unsymmetrical linear system also with sparse matrices. In figure 9.3 we present Intel MKL PARDISO®working scheme.

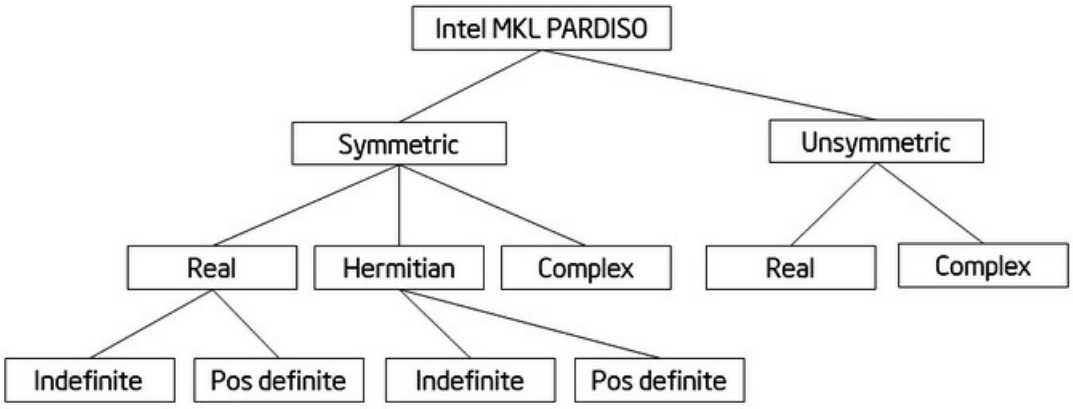


Figure 9.3: Intel MKL PARDISO®software

Chapter 10

Max-Ent approximation schemes applications

Our purpose is now to text the efficiency of those approximation schemes. In particular we refers to elastic problems in convex domain and not convex domain. In this work our goal is to investigate the effective consistency of the second derivatives of the approximants so we use the *Collocation method* in problem solving. First we test the approximants with the very simple Poisson's equation, then we move to more practical test.

10.1 Poisson's equation

By working with bi-dimensional domain, we introduce the 2D membrane problem:

$$\nabla^2 u(x,y) = f(x,y)$$

whose solution is well known:

$$u(x,y) = \sin(\pi x) \cdot \sin(\pi y) \qquad f(x,y) = -2\pi^2 \sin(\pi x) \cdot \sin(\pi y)$$

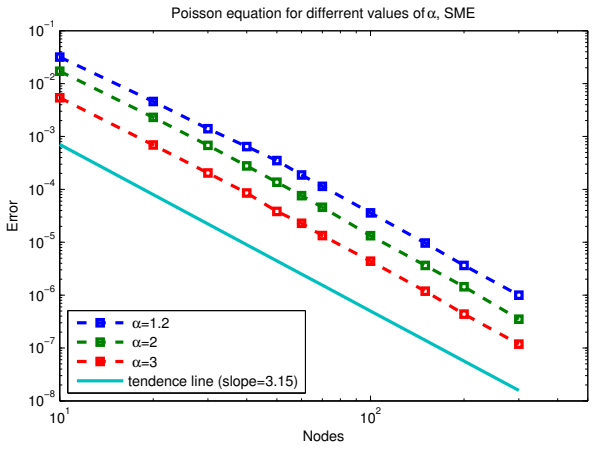


Figure 10.1: Error progression of Poisson problem for different values of β; regular grid

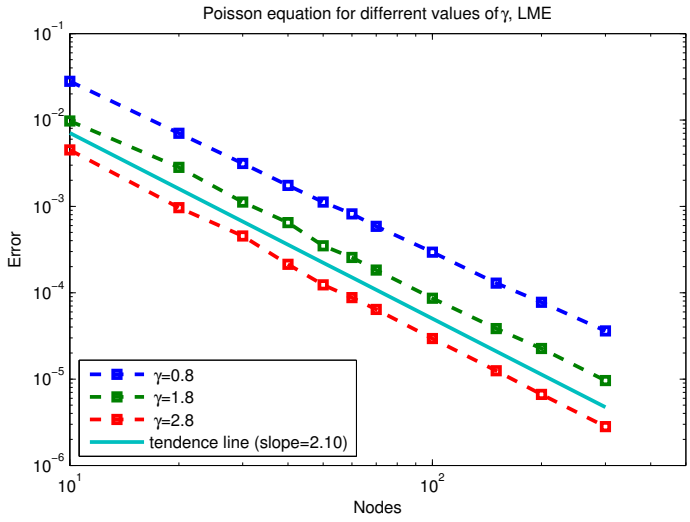


Figure 10.2: Error progression of Poisson problem for different values of α, regular grid

In figure 10.1 and 10.2 we have displaced the approximation error for both LME and SME programs by changing the control parameters. We can note we SME program has a superior convergence order then LME. This is completely justified by the fact SMEs have a second order consistency condition.
In figure 10.3 we displace the errors obtained with LME program with random grid:

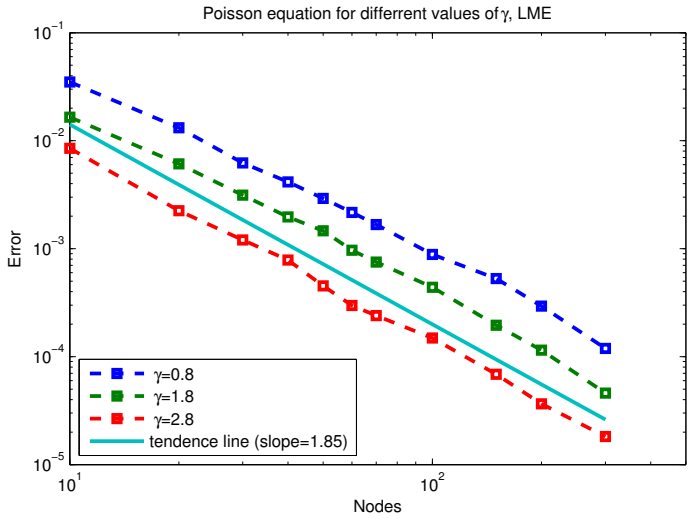


Figure 10.3: Error progression of Poisson problem for different values of β; random grid

In this case the slope of convergence line is lower then for the random grid, but is quite similar to regular case.

10.2 Elastic problem

Our goal is to prove the efficiency of the approximants for some simple elastic problems. We'll consider different problems with fixed a random distribution of nodes. We know that the elastic one is the simplest way to prove its efficiency, but we must know how the approximants works, to evaluate if it's usefull to proceed with more difficult tests.

10.2.1 General framework of the problem

First of all we have to choose a way to solve the problem. We have two different chooses to consider:

- WEAK FORM:

  this type of procedure is certain the most used in mechanics problems, but requires the calculation of integers upon the functions derivatives domain;

- STRONG FORM:

  in this case we need to compute the second derivatives of the shape functions, which, generally, requires more processing time and memory package, but allow us to avoid the calculation of integers, which could be more onerous for high numbers of nodes;

In this work we choose to test the second way, first of all, cause we have an analytic way to calculate them, second to test the method with the most uncertain pick.

10.2.2 Quick review of elasticity

In this section we give a brief review of general concepts of elasticity. First we introduce the 3 fields equation of elasticity problem:

- equilibrium equation:
$$\sigma_{ij,j} + b_i = 0$$
- constitutive equation:
$$\sigma_{ij} = D_{ijhk}\epsilon_{hk}$$
- compatibility equation:
$$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$$

associated with the commons boundary conditions:

- imposed displacement:
$$u_i = \bar{u_i}$$
- imposed stress :
$$\sigma_{ij}n_j = \bar{t_i}$$

Our goal is now to rewrite the 3 equations to obtain equations which have for unknowns only the displacements ones [30]. First we write the 2^{nd} equation by introducing the constitutive tensor:

$$D_{ijhk} = \begin{bmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & 2\mu \end{bmatrix}$$

so we obtain this 3 equations:

$$\sigma_{xx} = (\lambda + 2\mu)\epsilon_{xx} + \lambda\epsilon_{yy}$$
$$\sigma_{yy} = \lambda\epsilon_{xx} + (\lambda + 2\mu)\epsilon_{yy}$$
$$\sigma_{xy} = 2\mu\epsilon_{xy}$$

then we introduce in those equations the compatibility ones to obtain the 2 equations that dominates our problem:

$$(\lambda + 2\mu)u_{,xx} + \lambda v_{,yx} + \mu(u_{,yy} + v_{,xy}) + b_x = 0$$
$$\lambda u_{,xy} + (\lambda + 2\mu)v_{,yy} + \mu(u_{,yx} + v_{,yy}) + b_y = 0$$

now we re-adapt them to have a more useful formulation:

$$u_{,xx}(\lambda + 2\mu) + v_{,xy}(\lambda + \mu) + u_{,yy}(\mu) + b_x = 0$$
$$v_{,xx}\mu + u_{,xy}(\lambda + \mu) + v_{,yy}(\lambda + 2\mu) + b_y = 0$$

as inspected, we have now equations with only displacements unknowns that, of course, involve the second derivatives of the field. Since we are going to face a finite element problem we can introduce the classical approximation by using the shape functions and their derivatives we have already at our available.

$$(\lambda + 2\mu)\left(\sum_{j=1}^{N} \phi^j_{,xx} u^j\right) + \mu\left(\sum_{j=1}^{N} \phi^j_{,yy} u^j\right) + (\lambda + \mu)\left(\sum_{j=1}^{N} \phi^j_{,xy} v^j\right) + b_x = 0$$

$$\mu\left(\sum_{j=1}^{N} \phi^j_{,xx} v^j\right) + (\lambda + 2\mu)\left(\sum_{j=1}^{N} \phi^j_{,yy} v^j\right) + (\lambda + \mu)\left(\sum_{j=1}^{N} \phi^j_{,xy} u^j\right) + b_x = 0$$

The same procedure is needed to compute the boundary conditions, so, for the imposed displacements we have:

$$\left(\sum_{j=1}^{N} \phi^j_{,xx} u^j\right) = \bar{u_i}$$

$$\left(\sum_{j=1}^{N} \phi^j_{,xx} v^j\right) = \bar{v_i}$$

and for the stress border conditions:

$$\sigma_{xx} = (\lambda + 2\mu)\frac{\partial u}{\partial x} + \lambda\frac{\partial v}{\partial y}$$

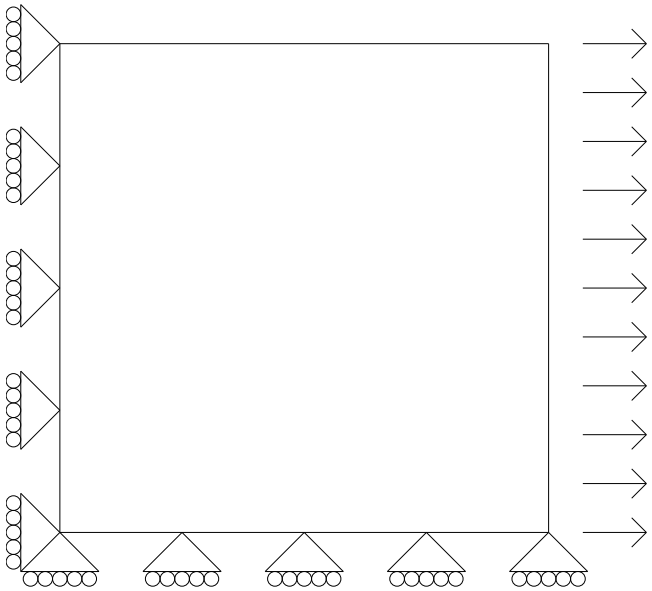
$$\sigma_{yy} = \lambda\frac{\partial u}{\partial x} + (\lambda + 2\mu)\frac{\partial v}{\partial y}$$

$$\sigma_{xy} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)$$

By introducing the expression of our first and second derivatives approximants, we can solve easly the elastic problem by using a strong formulation.

10.2.3 Plate in traction with LME

We consider a simple traction problem of plate in traction on a side:



We use the following data to solve the problem:

$$E = 2.11 \cdot 10^{11} MPa \quad \text{Young's modulus}$$

$$\nu = 0.3 \quad \text{Poissont's Ratio}$$

$$q = 1N \quad \text{Distributed force}$$

$$L = 1m \quad \text{Square dimension}$$

For this problem we know the exact solution; by considering a strain plain problem we have:

$$\epsilon_{zz} = \epsilon_{zx} = \epsilon_{zy} = 0$$

so from the mono-axial stress condition $\sigma_{yy} = \sigma_{xy} = 0$ and from the strain condition $\epsilon_{zz} = 0$ we can derive:

$$\sigma_{zz} = \nu \sigma_{xx}$$

and so:

$$\epsilon_{xx} = \frac{\sigma_{xx}}{E} - \frac{\nu}{E}\sigma_{zz} = \frac{\nu^2}{E}\sigma_{xx}$$

$$\epsilon_{yy} = -\frac{\nu}{E}(\sigma_{xx} + \sigma_{zz}) = -\frac{\nu}{E}(1+\nu)\sigma_{xx}$$

now by imposing the uniform stress $\sigma_{xx} = 1$ we obtain:

$$\epsilon_{xx} = 0.4313 \cdot 10^{-11}$$

$$\epsilon_{yy} = -0.1848 \cdot 10^{-11}$$

that is also the displacement of the north-est node cause L=1 m. Now we apply the LME approximants and we hope that we can caught the right solution, since the approximants have a 1^{st} order consistency. We test the program in for a regular and for an irregular grid of nodes. In both cases we use a simple grid with 81 nodes. In the tables 10.1 we presents the computational results in terms of nodal displacements for the uniform grid and we displace them in figure 10.8 by applying an amplification factor.

Nodal displacements X - direction								
$\cdot 10^{-11}$ m								
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313

Nodal displacements Y - direction								
$\cdot 10^{-11}$ m								
-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848
-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617
-0.1386	-0.1386	-0.1386	-0.1386	-0.1386	-0.1386	-0.1386	-0.1386	-0.1386
-0.1155	-0.1155	-0.1155	-0.1155	-0.1155	-0.1155	-0.1155	-0.1155	-0.1155
-0.0924	-0.0924	-0.0924	-0.0924	-0.0924	-0.0924	-0.0924	-0.0924	-0.0924
-0.0693	-0.0693	-0.0693	-0.0693	-0.0693	-0.0693	-0.0693	-0.0693	-0.0693
-0.0462	-0.0462	-0.0462	-0.0462	-0.0462	-0.0462	-0.0462	-0.0462	-0.0462
-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231
0	0	0	0	0	0	0	0	0

Table 10.1: Nodal displacements of the plate in traction (LME) with uniform grid

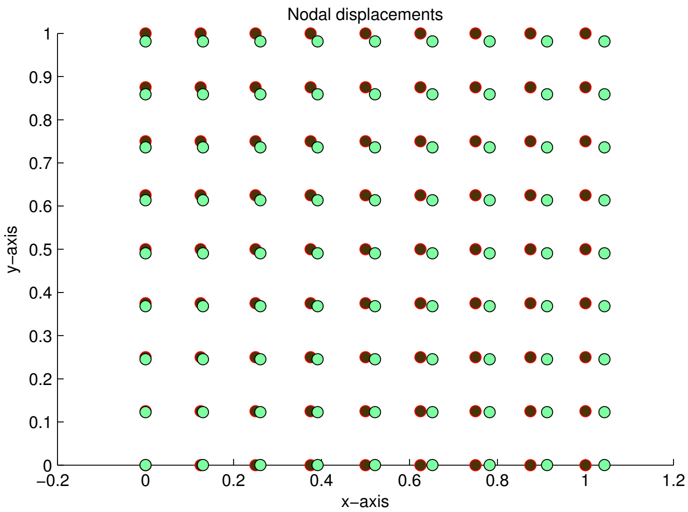


Figure 10.4: Nodal displacements of the plate in traction (LME) with uniform grid

From the nodal displacements we can easily recover the stresses we report in figure 10.5.

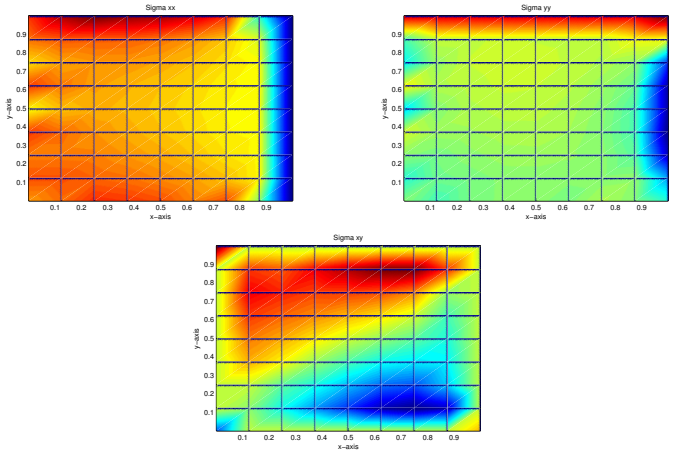


Figure 10.5: Stresses of the plate in traction (LME) with uniform grid

In the tables 10.3 we presents the computational results in terms of nodal displacements for the random grid and we displace them in figure 10.8 by applying an amplification factor.

Nodal displacements X - direction								
$\cdot 10^{-11}$ m								
0	0.0539	0.1078	0.1617	0.2156	0.2695	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.2690	0.1270	0.1322	0.3115	0.2108	0.3774	0.4313
0	0.0539	0.2262	0.3502	0.2612	0.1230	0.1963	0.3774	0.4313
0	0.0539	0.2605	0.1154	0.3186	0.3455	0.1323	0.3774	0.4313
0	0.0539	0.2698	0.1289	0.2321	0.2051	0.1823	0.3774	0.4313
0	0.0539	0.1621	0.2605	0.0897	0.1803	0.2547	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313

Nodal displacements Y - direction								
$\cdot 10^{-11}$ m								
-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848
-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617
-0.1386	-0.1386	-0.1020	-0.1299	-0.0471	-0.1033	-0.0553	-0.1386	-0.1386
-0.1155	-0.1155	-0.0791	-0.0653	-0.1021	-0.0648	-0.0533	-0.1155	-0.1155
-0.0924	-0.0924	-0.0608	-0.1059	-0.0744	-0.1291	-0.0708	-0.0924	-0.0924
-0.0693	-0.0693	-0.1028	-0.0682	-0.1190	-0.1363	-0.0838	-0.0693	-0.0693
-0.0462	-0.0462	-0.0486	-0.0637	-0.1482	-0.1394	-0.0373	-0.0462	-0.0462
-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231
0	0	0	0	0	0	0	0	0

Table 10.2: Nodal displacements of the plate in traction (LME) with random grid

In figure 10.6 we give a representation of the nodal displacements, then in figure 10.7, we displace the stresses upon the plate in traction:

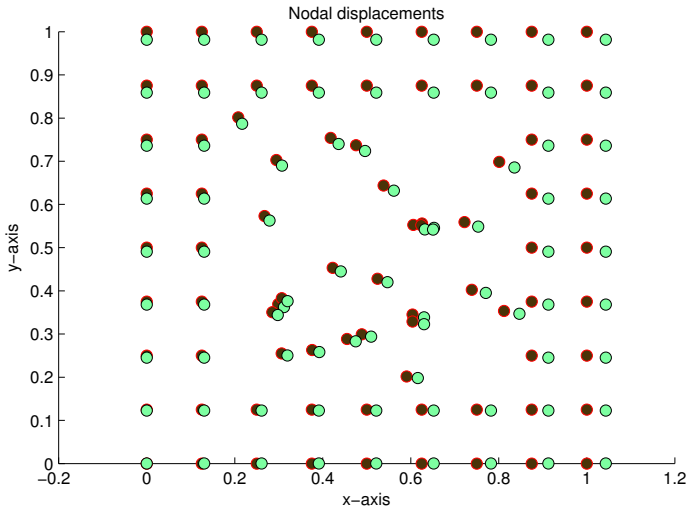


Figure 10.6: Nodal displacements of the plate in traction (LME) with random grid

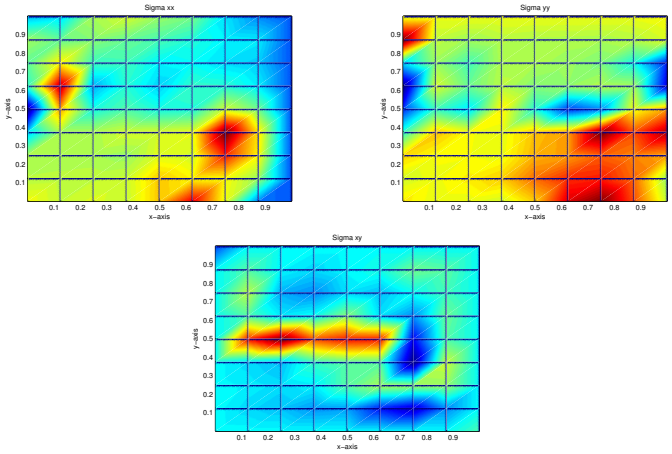


Figure 10.7: Stresses of the plate in traction (LME) with random grid

In both situations we have assumed $\gamma = 0.6$ and so $\beta = 38$ (classical values) and we can notice that the results we have obtained are exactly the analytic solutions of the problem. Note that we have maintained uniformly distributed the boundary and next to boundary nodes. This is a common choice for this kind of problems according to [8].

10.2.4 Plate in traction with SME

We approach, now to the same problem by using the SME approximants, which have a second order consistency condition, so we expect to reach the same exact results we had with LME approximation schemes. In this case, we take the parameter $\alpha = 3$ (classical value).
First we presents the computational results for a random grid of nodes:

Nodal displacements X - direction $\cdot 10^{-11}$ m								
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2696	0.3235	0.3774	0.4313

Nodal displacements Y - direction $\cdot 10^{-11}$ m								
-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848
-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617
-0.1386	-0.1386	-0.1386	-0.1386	-0.1386	-0.1386	-0.1386	-0.1386	-0.1386
-0.1155	-0.1155	-0.1155	-0.1155	-0.1155	-0.1155	-0.1155	-0.1155	-0.1155
-0.0924	-0.0924	-0.0924	-0.0924	-0.0924	-0.0924	-0.0924	-0.0924	-0.0924
-0.0693	-0.0693	-0.0693	-0.0693	-0.0693	-0.0693	-0.0693	-0.0693	-0.0693
-0.0462	-0.0462	-0.0462	-0.0462	-0.0462	-0.0462	-0.0462	-0.0462	-0.0462
-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231
0	0	0	0	0	0	0	0	0

Table 10.3: Nodal displacements of the plate in traction (SME) with uniform grid

In figure 10.8 we give a representation of the nodal displacements, then in figure 10.9, we displace the stresses upon the plate in traction:

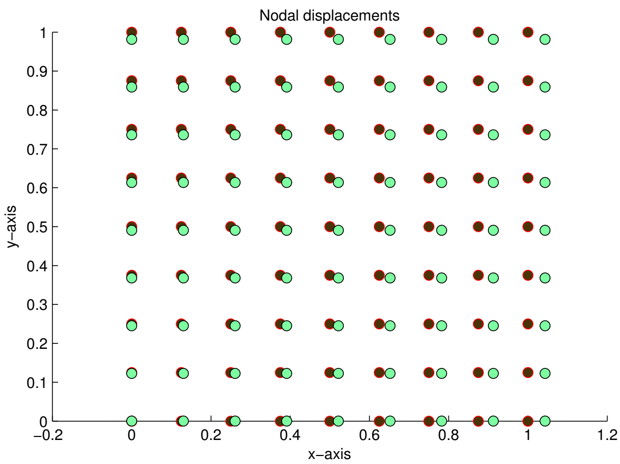


Figure 10.8: Nodal displacements of the palte in traction (SME) with uniform grid

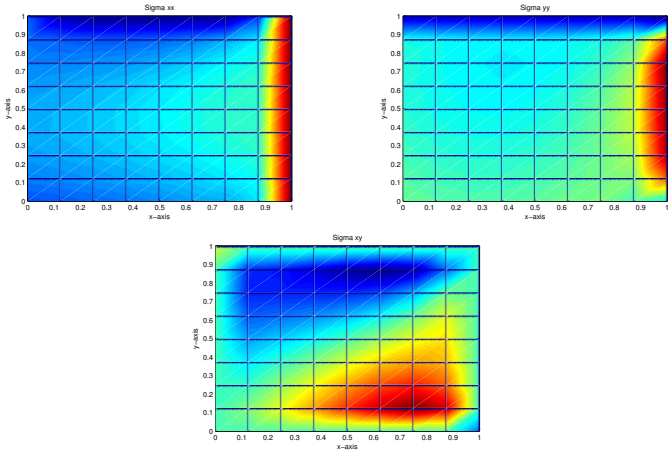


Figure 10.9: Stresses of the plate in traction (SME) with uniform grid

We analyze now the case of a random grid of nodes: as we have already explained, in this case the Newton-Raphson iterative method does not converge anyhow. To show this situation, we have created a program depending form a parameter : ϵ which is able to control the density of interior nodes: when $\epsilon = 0$ we have a regular node set, and when $\epsilon = 1$ we have a completely random node set.

- ϵ=0.3

Nodal displacements X - direction								
$\cdot 10^{-11}$ m								
0	0.0539	0.1078	0.1617	0.2156	0.2695	0.3234	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2695	0.3234	0.3774	0.4313
0	0.0539	0.1094	0.1743	0.2232	0.2557	0.3331	0.3774	0.4313
0	0.0539	0.1140	0.1584	0.2247	0.2730	0.3106	0.3774	0.4313
0	0.0539	0.1158	0.1471	0.2190	0.2769	0.3325	0.3774	0.4313
0	0.0539	0.1121	0.1482	0.2287	0.2569	0.3182	0.3774	0.4313
0	0.0539	0.1064	0.1777	0.2091	0.2630	0.3236	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2695	0.3234	0.3774	0.4313
0	0.0539	0.1078	0.1617	0.2156	0.2695	0.3234	0.3774	0.4313

Nodal displacements Y - direction								
$\cdot 10^{-11}$ m								
-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848
-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617	-0.1617
-0.1386	-0.1386	-0.1384	-0.1427	-0.1324	-0.1329	-0.1447	-0.1386	-0.1386
-0.1155	-0.1155	-0.1163	-0.1094	-0.1132	-0.1188	-0.1103	-0.1155	-0.1155
-0.0924	-0.0924	-0.0856	-0.0947	-0.0928	-0.0953	-0.0895	-0.0924	-0.0924
-0.0693	-0.0693	-0.0636	-0.0731	-0.0698	-0.0738	-0.0664	-0.0693	-0.0693
-0.0462	-0.0462	-0.0407	-0.0439	-0.0401	-0.0399	-0.0498	-0.0462	-0.0462
-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231	-0.0231
0	0	0	0	0	0	0	0	0

Table 10.4: Nodal displacements of the plate in traction (SME) with random grid ϵ=0.3

In figure 10.10 we give a representation of the nodal displacements, then in figure 10.11, we

displace the stresses upon the plate in traction:

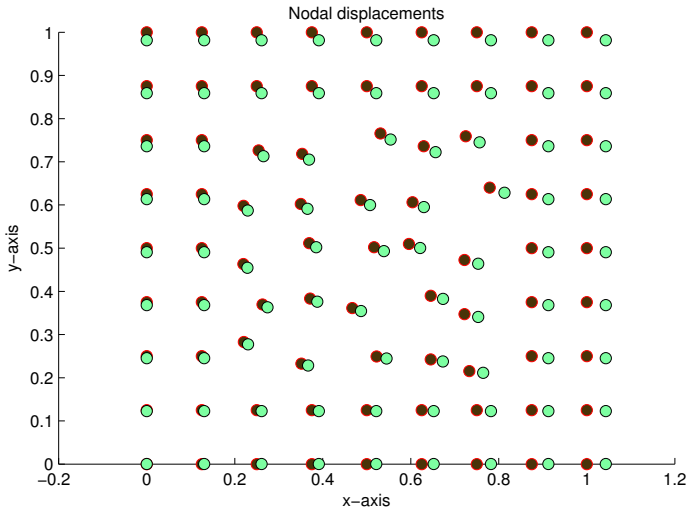


Figure 10.10: Nodal displacements of the plate in traction (SME) with random grid ϵ=0.3

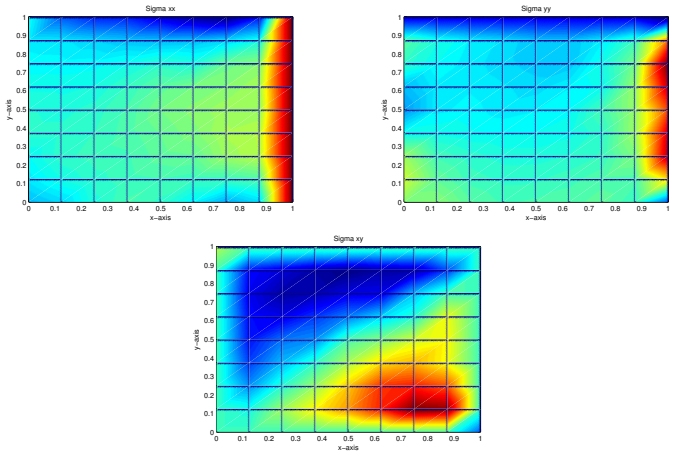


Figure 10.11: Stresses of the plate in traction (SME) with random grid, ϵ=0.3

As we can note, in this case the nodes are quite regular, and we are able to find the right solution.

- ϵ=0.5

Nodal displacements X - direction								
$\cdot 10^{-11}$ m								
0	0.0539	0.1078	0.1619	0.2156	0.2695	0.3233	0.3773	0.4312
0	0.0540	0.1079	0.1617	0.2156	0.2695	0.3234	0.3773	0.4312
0	0.0539	0.1000	0.1533	0.2133	0.2543	0.3134	0.3774	0.4313
0	0.0539	0.1030	0.1756	0.2182	0.2703	0.3351	0.3774	0.4313
0	0.0539	0.1320	0.1874	0.2076	0.2556	0.3335	0.3774	0.4313
0	0.0539	0.0883	0.1447	0.1945	0.2939	0.3184	0.3774	0.4314
0	0.0539	0.1341	0.1851	0.2258	0.2713	0.3451	0.3774	0.4314
0	0.0540	0.1079	0.1621	0.2158	0.2697	0.3236	0.3774	0.4314
0	0.0539	0.1080	0.1619	0.2159	0.2696	0.3236	0.3774	0.4314

Nodal displacements X - direction								
$\cdot 10^{-11}$ m								
-0.1848	-0.1848	-0.1846	-0.1848	-0.1848	-0.1848	-0.1848	-0.1848	-0.1847
-0.1617	-0.1617	-0.1614	-0.1617	-0.1618	-0.1618	-0.1617	-0.1617	-0.1616
-0.1385	-0.1385	-0.1495	-0.1475	-0.1365	-0.1300	-0.1438	-0.1386	-0.1385
-0.1155	-0.1155	-0.1178	-0.1174	-0.1175	-0.1058	-0.1270	-0.1154	-0.1154
-0.0924	-0.0924	-0.0965	-0.0985	-0.0963	-0.0878	-0.0931	-0.0924	-0.0923
-0.0693	-0.0693	-0.0627	-0.0587	-0.0721	-0.0661	-0.0803	-0.0693	-0.0692
-0.0462	-0.0462	-0.0362	-0.0352	-0.0527	-0.0552	-0.0491	-0.0462	-0.0462
-0.0231	-0.0231	-0.0232	-0.0231	-0.0230	-0.0231	-0.0231	-0.0231	-0.0231
0	0	0	0	0	0	0	0	0

Table 10.5: Nodal displacements of the plate in traction (SME) with random grid ϵ=0.5

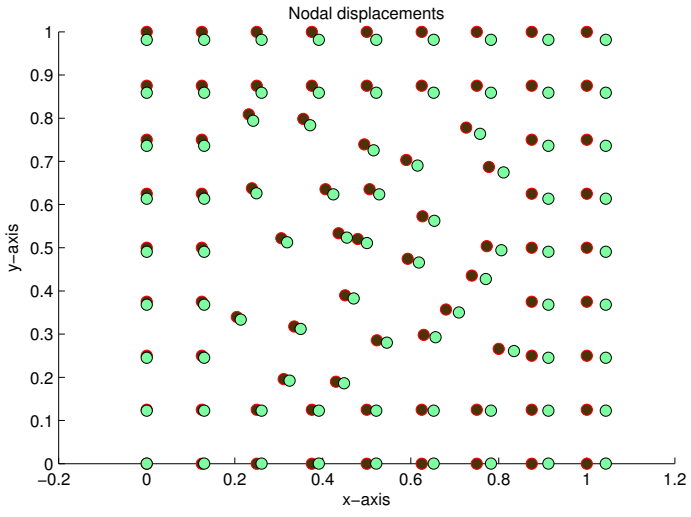


Figure 10.12: Nodal displacements of the plate in traction (SME) with random grid ϵ=0.5

In this case we can notice how the solution is not identical to the analytic one. In fact also the stresses, displaced in 10.13, we can appreciate how the follow of every component is not correct.

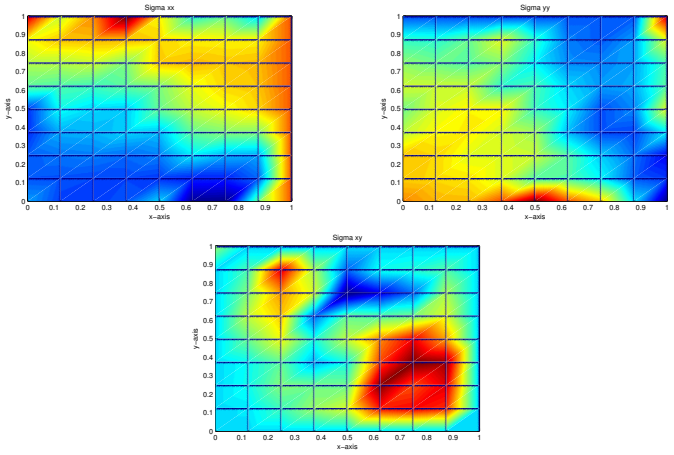


Figure 10.13: Stresses of the plate in traction (LME) with uniform grid

- ϵ=0.8

Nodal displacements X - direction								
$\cdot 10^{-11}$ m								
0	0.0525	0.1055	0.1603	0.2152	0.2684	0.3231	0.3771	0.4309
0	0.0543	0.1098	0.1579	0.2127	0.2677	0.3217	0.3770	0.4308
0	0.0539	0.1236	0.1440	0.1867	0.2397	0.3120	0.3762	0.4301
0	0.0536	0.0902	0.1793	0.2194	0.2319	0.3167	0.3757	0.4296
0	0.0535	0.1059	0.2011	0.2424	0.3060	0.3106	0.3754	0.4291
0	0.0537	0.1363	0.1655	0.2184	0.2786	0.3433	0.3750	0.4288
0	0.0535	0.1253	0.1468	0.1820	0.2795	0.3128	0.3748	0.4286
0	0.0535	0.1069	0.1604	0.2139	0.2675	0.3210	0.3748	0.4286
0	0.0535	0.1069	0.1604	0.2139	0.2675	0.3210	0.3748	0.4286

Nodal displacements Y - direction								
$\cdot 10^{-11}$ m								
-0.1802	-0.1805	-0.1759	-0.1771	-0.1802	-0.1809	-0.1828	-0.1831	-0.1832
-0.1577	-0.1575	-0.1537	-0.1551	-0.1560	-0.1585	-0.1592	-0.1599	-0.1604
-0.1341	-0.1343	-0.1515	-0.1367	-0.1462	-0.1388	-0.1458	-0.1368	-0.1375
-0.1120	-0.1119	-0.1012	-0.1159	-0.1181	-0.1052	-0.1041	-0.1139	-0.1143
-0.0891	-0.0892	-0.0966	-0.0836	-0.0984	-0.0740	-0.0959	-0.0910	-0.0915
-0.0668	-0.0668	-0.0610	-0.0835	-0.0610	-0.0614	-0.0646	-0.0683	-0.0686
-0.0445	-0.0445	-0.0487	-0.0596	-0.0528	-0.0579	-0.0542	-0.0455	-0.0456
-0.0223	-0.0222	-0.0223	-0.0223	-0.0224	-0.0225	-0.0226	-0.0228	-0.0228
0	0	0	0	0	0	0	0	0

Table 10.6: Nodal displacements of the plate in traction (SME) with random grid ϵ=0.8

In figure 10.14 we give a representation of the nodal displacements by applying an amplification factor, then in figure 10.15, we displace the stresses upon the plate in traction:

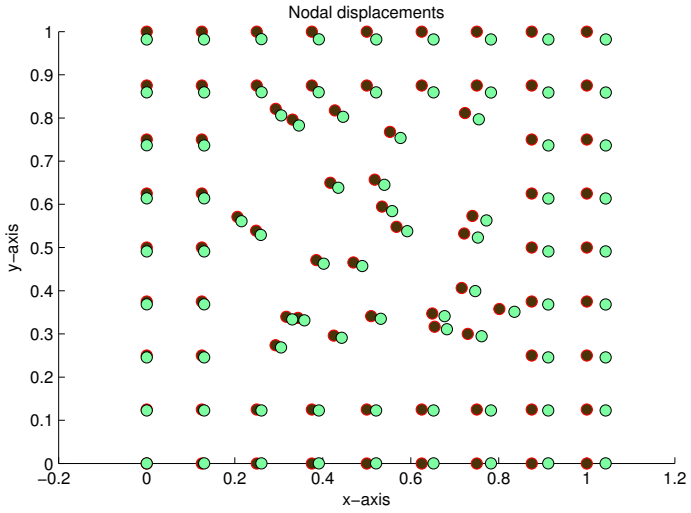


Figure 10.14: Nodal displacements of the plate in traction (SME) with random grid ϵ=0.8

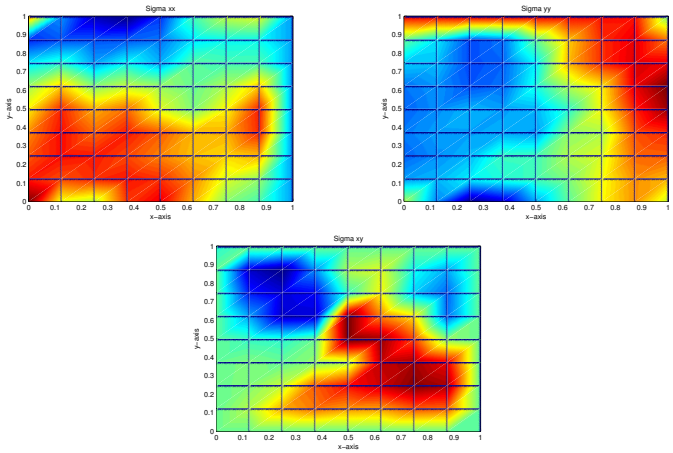


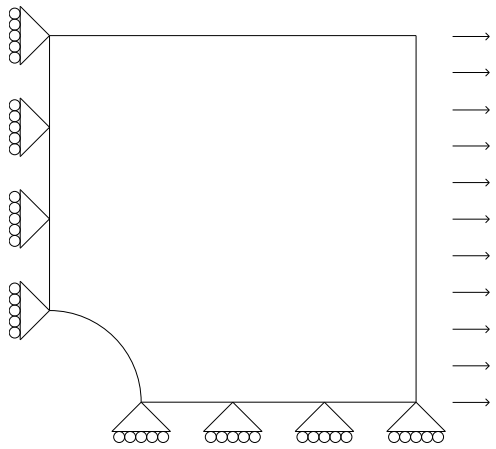
Figure 10.15: Stresses of the plate in traction (LME) with uniform grid

It's clear that in this case, since we are approaching to a more randomized solution, we have the problems over described, in fact the results are more sparse from the right solution.

Those tests confirm the SME program is less efficient then the LME one when we have a random set of nodes.

10.2.5 Quarter of plate with hole (LME)

In this section we want to test the program with this traction problem:



First of all we have to remark this type of domain is not a convex one. This is very important, cause from the start of our study we have adopted only convex domain in order to have consistent shape functions; although we want to investigate the response of our approximants in this situations. What we wish is to have a good solution in all of the plate and possibly not a good one on the border of the hole.
We use analogous data used for the plate in simple traction:

$$E = 2.11 \cdot 10^{11} MPa \quad \text{Young's modulus}$$

$$\nu = 0.3 \quad \text{Poissont's Ratio}$$

$$\sigma_0 = 1N \quad \text{Distributed force}$$

$$L = 1m \quad \text{Square dimension}$$

$$a = 0.2m \quad \text{Radious of the hole}$$

For this specific problem has been studied an analytic solution: if we consider the load σ_0 we can derive the solution in terms of stresses from it in this way [3]:

$$\sigma_{xx} = \sigma_0 \left[1 - \frac{a^2}{r^2} \left(\frac{3}{2} \cos(2\theta) + \cos(4\theta) \right) + \frac{3a^4}{2r^4} \cos(4\theta) \right]$$

$$\sigma_{xy} = \sigma_0 \left[-\frac{a^2}{r^2} \left(\frac{1}{2} \sin(2\theta) + \sin(4\theta) \right) + \frac{3a^4}{2r^4} \sin(4\theta) \right]$$

$$\sigma_{yy} = \sigma_0 \left[-\frac{a^2}{r^2} \left(\frac{1}{2} \cos(2\theta) - \cos(4\theta) \right) - \frac{3a^4}{2r^4} \cos(4\theta) \right]$$

and the solution in displacement terms can be so written:

$$u = \frac{1+\nu}{E} \sigma_0 \left(\frac{1}{1+\nu} r \cos\theta + \frac{2}{1+\nu} \frac{a^2}{r} \cos\theta + \frac{1}{2} \frac{a^2}{r} \cos 3\theta - \frac{1}{2} \frac{a^4}{r^3} \cos 3\theta \right)$$

where r is the distance of the considered point from x-axis origin and θ is the angle measured from the positive x-axis counterclockwise.

We have to impose also the following boundary conditions:

$$\boldsymbol{\sigma}\mathbf{n}\cdot\mathbf{n}=\mathbf{0} \quad \text{and} \quad \boldsymbol{\sigma}\mathbf{n}\cdot\mathbf{t}=\mathbf{0} \quad \text{on the quarter hole and north side}$$

$$\mathbf{s}\cdot\mathbf{n}=\mathbf{0} \quad \text{and} \quad \boldsymbol{\sigma}\mathbf{n}\cdot\mathbf{t}=\mathbf{0} \quad \text{on the west and south side}$$

$$\boldsymbol{\sigma}\mathbf{n}\cdot\mathbf{n}=\boldsymbol{\sigma}_0 \quad \text{and} \quad \boldsymbol{\sigma}\mathbf{n}\cdot\mathbf{t}=\mathbf{0} \quad \text{on the east side}$$

Since we have operated with a displacement approach, we follow this work line also in this contest. In figure 10.16 we plot the node grid:

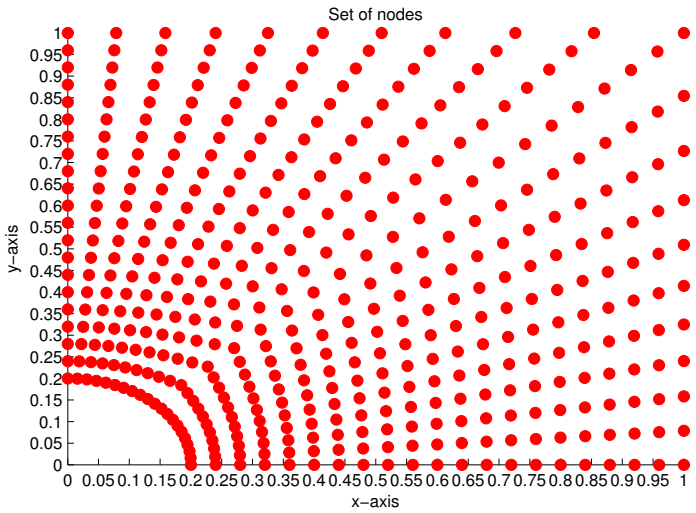


Figure 10.16: Set of nodes for quarter plate with hole problem

by applying the previuos border conditions, we obtain the following nodal displacements:

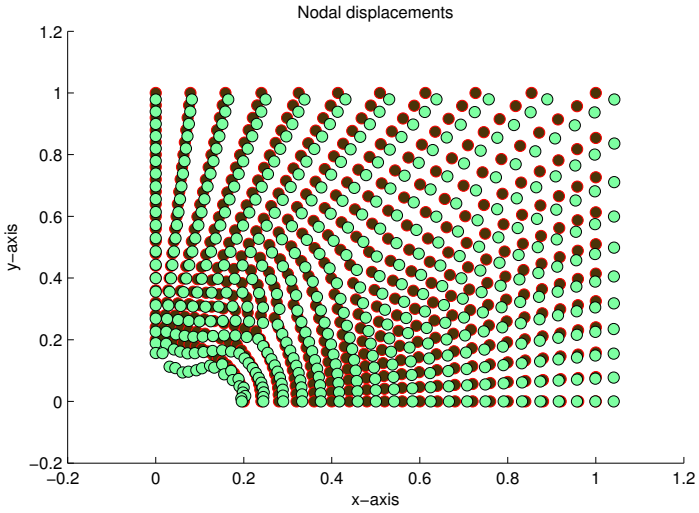


Figure 10.17: Nodal displacements for the plate with quarter hole problem

In table 10.7 we present the x displacements of the side in traction of the plate:

It's possible to note the results are very poor in particular near the concave border. This was

Nodal displacements	**Nodal displacements**
Exact	**Approximate**
$0.5237 \cdot 10^{-11}$	$0.4335 \cdot 10^{-11}$
$0.5231 \cdot 10^{-11}$	$0.4207 \cdot 10^{-11}$
$0.5213 \cdot 10^{-11}$	$0.4284 \cdot 10^{-11}$
$0.5185 \cdot 10^{-11}$	$0.4264 \cdot 10^{-11}$
$0.5149 \cdot 10^{-11}$	$0.4237 \cdot 10^{-11}$
$0.5105 \cdot 10^{-11}$	$0.4230 \cdot 10^{-11}$
$0.5057 \cdot 10^{-11}$	$0.4222 \cdot 10^{-11}$
$0.5007 \cdot 10^{-11}$	$0.4213 \cdot 10^{-11}$
$0.4957 \cdot 10^{-11}$	$0.4204 \cdot 10^{-11}$
$0.4911 \cdot 10^{-11}$	$0.4196 \cdot 10^{-11}$
$0.4869 \cdot 10^{-11}$	$0.4192 \cdot 10^{-11}$

Table 10.7: Comparison between analytic and approximate results on north-est nodal displacement

presumable since to have a correct solution of the minimizing problem the entropy must be defined in a convex domain. We try to avoid this bug by introducing a *false* node used to make convex the domain has we can see in figure 10.18. The domain it's now convex, but we have a node

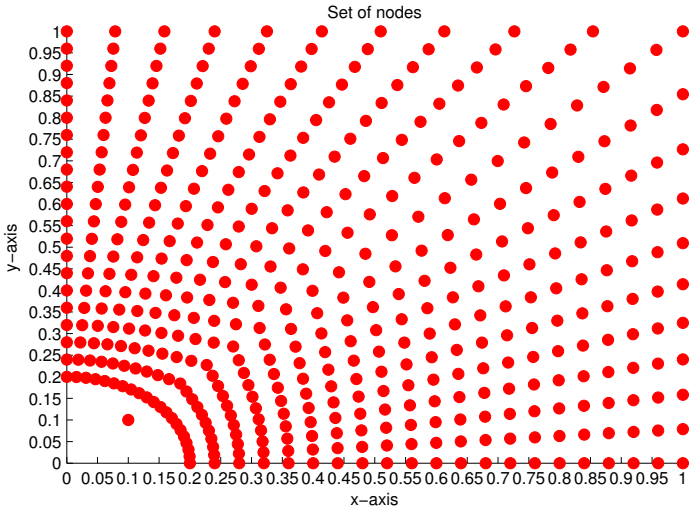


Figure 10.18: Set of nodes for quarter plate with hole problem; correction

that is not part of the domain. During the computational process the additional node generates a shape function with connected derivatives. We have, so, to remove the additional row and column from every matrix. In this case we can appreciate how both the proprieties (2.4), (2.3), (2.5), are respected.

We expect, with this attempt to have better results near the concave border and in all the domain. Another good attempt to solve the problem is to increase the parameter β as it's possible in order to have more local shape functions, so to reduce the error given to the concave border. In figure 10.19 we plot the displacement obtained:

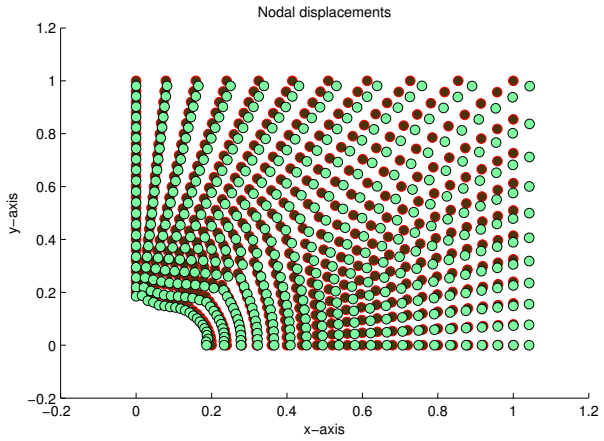


Figure 10.19: Nodal displacements for the corrected plate with quarter hole problem

Indeed, we note an improvement in method's performances, even if we don't have a correct solution on the concave domain. To test effectively the goodness of the solution it's possible to evaluate the error between the correct solution and the approximated one in this way:

$$Err = \frac{\sqrt{\sum_{i=1}^{N} ||u_{i_{an}} - u_{i_{lme}}||^2}}{\sqrt{\sum_{i=1}^{N} ||u_{i_{an}}||^2}}$$

which can be normalized on the mean value of the correct solution. In figure 10.20 we displace the decreasing error for the increasing of the number of nodes and in figure 10.21 we displace the σ_{xx} component of stress:

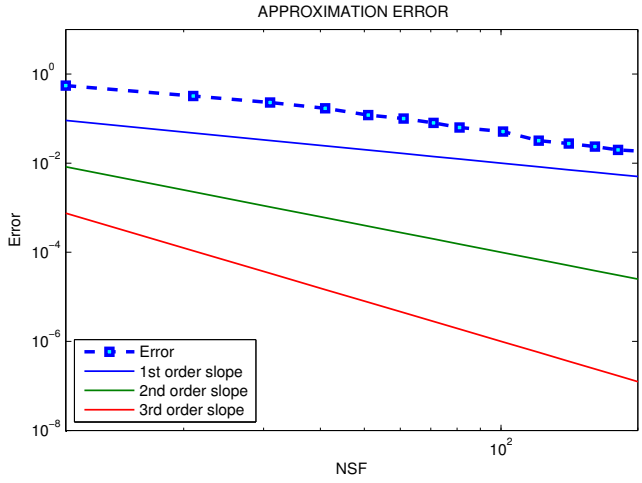


Figure 10.20: Approximation error with trend lines, for the quarter hole problem

As we note, the decreasing line as a slope next to 2. Surely the concavity of a little part of the domain has influenced the results of the approximation. By this example we can deduce if we want to face problems with high irregular grids, we will not be able to obtain a good approximations.

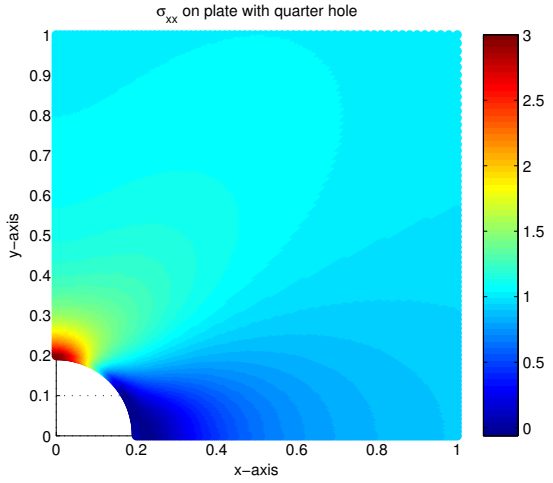


Figure 10.21: σ_{xx} component of stress, for the quarter hole problem

We can appreciate we are able to obtain good results in terms of stresses in this case, by using an high number of nodes.

Chapter 11

Conclusions and future perspectives

In this work we have outlined the principal characteristics of the *meshfree* maximum entropy approximants. First we have delineated the principal mathematical aspects of the approximants: from the probabilistic to the statistic background, then we have moved to a more physicist way to look the approximants, observing the similarity between the partition function we have used here and the one used in statistic mechanical. In particular we have put our attention to the analytic construction of the shape functions and their first and second derivatives. During the progression of the work we have appreciated how the construction of every single shape function is expensive in terms of computational time, in fact it involves the loop over all the nodes of the domain, and we have checked the high dependence from the control parameters α and β. We have, also, introduced 2 different consistency condition on the approximation schemes. In particular we have noticed the increasing of consistency condition involves an high computational burden, in fact both in the evaluation of the shape functions and the associated derivatives, we have a consistent increasing of the machine time due to the presence of 5 unknowns in front of 2 we had in LME program. In our approximation tests we have proved the effective more efficiency of the second order approximants then the first order one's, where we have a regular grid of nodes, but we found some problems where we approach to random grid domains with SMEs. In fact the second order approximation schemes, even if show an higher slope in approximation tests, present itself like a very sensible method, in specific we have shown how a simple parameter (d_a) which relaxes the second order consistency condition appears to be essential in order to have the correct evaluation of the basis functions. With the LMEs approximation schemes, that seam to work better, we have approached to elasticity problems in a convex domain. In this case we have found good results by reaching the exact solution. As last point of our work we have faced the quarter plate with hole problem, with which we wanted to test the efficiency of the approximants in a concave domain. With this type of grid the approximants are not able to give a correct solution so we have introduced a *false node* in order to make convex the domain in which we construct the shape functions so to avoid the problem of the minimization program.
In our work we have dedicated, also, two chapters to the problem of the computer implementation, cause to build up the shape functions and relative derivatives involves an high computational burden. We have outlined, in particular, the high machine time, which is, just in part, shot down by the usage of sparse matrices.

As final review of our work we can conclude those family of approximants is well working everywhere for regular grid of nodes. The LME program shows a good behavior also for random grid of nodes, even if it's very heavy in a computational way and gives best performances in convex domain.

As a further development of this work we purpose to confront this method with other classical method used for mesh-less problem like SPLINES, NURBS or a MODIFIED PARTICLE METHOD [3], in order to test if it's convenient to go ahead with this approximation schemes.

As future perspective it's also possible to find a way to construct the shape functions only with a determinate number of nodes and not by using all the nodes of the domain so to make the methods more flexible in a common use.

Bibliography

[1] Marino Arroyo and Michael Ortiz. Local maximum-entropy approximation schemes. In *Meshfree Methods for Partial Differential Equations III*, pages 1–16. Springer, 2007.

[2] Marino Arroyo and Michael Ortiz. Local maximum-entropy approximation schemes: a seamless bridge between finite elements and meshfree methods. *International journal for numerical methods in engineering*, 65(13):2167–2202, 2006.

[3] D Asprone, F Auricchio, A Montanino, and A Reali. A modified finite particle method: Multi-dimensional elasto-statics and dynamics. *International Journal for Numerical Methods in Engineering*, 99(1):1–25, 2014.

[4] SN Atluri, HT Liu, and ZD Han. Meshless local petrov-galerkin (mlpg) mixed collocation method for elasticity problems. *CMC-TECH SCIENCE PRESS-*, 4(3):141, 2006.

[5] A Bompadre, LE Perotti, CJ Cyron, and M Ortiz. Convergent meshfree approximation schemes of arbitrary order and smoothness. *Computer Methods in Applied Mechanics and Engineering*, 221:83–103, 2012.

[6] Agustin Bompadre, Bernd Schmidt, and Michael Ortiz. Convergence analysis of meshfree approximation schemes. *SIAM Journal on Numerical Analysis*, 50(3):1344–1366, 2012.

[7] Thierry Coupez. Metric construction by length distribution tensor and edge based error for anisotropic adaptive meshing. *Journal of Computational Physics*, 230(7):2391–2405, 2011.

[8] Christian J Cyron, M Arroyo, and Michael Ortiz. Smooth, second order, non-negative meshfree approximants selected by maximum entropy. *International Journal for Numerical Methods in Engineering*, 79(13):1605–1632, 2009.

[9] Christian J Cyron, Marino Arroyo, Michael Ortiz, and WA Wall. Second order maximum-entropy approximation schemes. 2008.

[10] Geoff Gordon and Ryan Tibshirani. Karush-kuhn-tucker conditions. *Optimization*, 10(725/36):725.

[11] F Greco and N Sukumar. Derivatives of maximum-entropy basis functions on the boundary: Theory and computations. *International Journal for Numerical Methods in Engineering*, 94(12):1123–1149, 2013.

[12] Antonio Huerta, Ted Belytschko, Sonia Fernández-Méndez, and Timon Rabczuk. Meshfree methods. 2004.

[13] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[14] Jagat Narain Kapur. *Maximum-entropy models in science and engineering*. John Wiley & Sons, 1989.

[15] Aleksandr Iakovlevich Khinchin. *Mathematical foundations of information theory*, volume 434. Courier Corporation, 1957.

[16] Daniel Millán, Adrian Rosolen, and Marino Arroyo. Finite deformation thin shell analysis from scattered points with maximum-entropy approximants. 2010.

[17] Daniel Millán, N Sukumar, and Marino Arroyo. Cell-based maximum-entropy approximants. *Computer Methods in Applied Mechanics and Engineering*, 284:712–731, 2015.

[18] Alejandro A Ortiz. *Maximum-Entropy Meshfree Method for Linear and Nonlinear Elasticity*. PhD thesis, UNIVERSITY OF CALIFORNIA DAVIS, 2011.

[19] A Ortiz, MA Puso, and N Sukumar. Maximum-entropy meshfree method for compressible and near-incompressible elasticity. *Computer Methods in Applied Mechanics and Engineering*, 199(25):1859–1871, 2010.

[20] Christian Peco, Daniel Millán, Adrian Rosolen, and Marino Arroyo. Efficient implementation of galerkin meshfree methods for large-scale problems with an emphasis on maximum entropy approximants. *Computers & Structures*, 150:52–62, 2015.

[21] VT Rajan. Optimality of the delaunay triangulation in r. *Discrete & Computational Geometry*, 12(1):189–202, 1994.

[22] A Rosolen, D Millán, and M Arroyo. Second-order convex maximum entropy approximants with applications to high-order pde. *International Journal for Numerical Methods in Engineering*, 94(2):150–182, 2013.

[23] A Rosolen, C Peco, and M Arroyo. An adaptive meshfree method for phase-field models of biomembranes. part i: Approximation with maximum-entropy basis functions. *Journal of Computational Physics*, 249:303–319, 2013.

[24] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[25] N Sukumar and RJ-B Wets. Deriving the continuity of maximum-entropy basis functions via variational analysis. *SIAM Journal on Optimization*, 18(3):914–925, 2007.

[26] N Sukumar and RW Wright. Overview and construction of meshfree basis functions: from moving least squares to entropy approximants. *International Journal for Numerical Methods in Engineering*, 70(2):181–205, 2007.

[27] Aldo Tagliani. Existence and stability of a discrete probability distribution by maximum entropy approach. *Applied Mathematics and Computation*, 110(2):105–114, 2000.

[28] Z Ullah, C Augarde, and W Coombs. Local maximum entropy shape functions based fe–efgm coupling. *Int. J. Numer. Methods Eng., submitted for publication*.

[29] Z Ullah, WM Coombs, and CE Augarde. An adaptive finite element/meshless coupled method based on local maximum entropy shape functions for linear and nonlinear problems. *Computer Methods in Applied Mechanics and Engineering*, 267:111–132, 2013.

[30] Young-Cheol Yoon, Sang-Ho Lee, and Ted Belytschko. Enriched meshfree collocation method with diffuse derivatives for elastic fracture. *Computers & Mathematics with Applications*, 51(8):1349–1366, 2006.

[31] T Zhu and SN Atluri. A modified collocation method and a penalty formulation for enforcing the essential boundary conditions in the element free galerkin method. *Computational Mechanics*, 21(3):211–222, 1998.