NTNU
Norwegian University of
Science and Technology

Isogeometric Analysis and Degenerated Mappings

Siv Bente Raknes

Master of Science in Physics and Mathematics
Submission date: February 2011
Supervisor: Trond Kvamsdal, MATH
Co-supervisor: Kjell Magne Mathisen, KT
Kjetil Johannessen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

In this thesis B-splines will be studied by programming an isogeometric finite element solver in MATLAB, designed to solve linear elasticity problems. We will look at some geometries having a degenerated mapping, and will further consider the effects different parameterizations of the spline basis have on the derivatives.

Assignment given: 06. September 2010
Supervisor: Trond Kvamsdal, MATH

Preface

This Master's thesis is my final work to obtain a Master of Science in Physics and Mathematics at the Norwegian University of Science and Technology. When educating for a Master of Science in Physics and Mathematics, one may from the third year choose between three main fields of study; technical physics, industrial mathematics and biophysics and medical technology. I have done my degree in the industrial mathematics program with a specialization within the field of applied mathematics. While working on this thesis I have been lucky to have three supervisors; Trond Kvamsdal and Kjetil Andre Johannessen at the Department of Mathematical Sciences, and Kjell Magne Mathisen at the Department of Structural Engineering.

I was first introduced to isogeometric analysis when I was working at SINTEF ICT this summer. My task during the summer internship was to convert the geometrical representation of a wind turbine blade from data points given in xy-coordinates to a full 3D model represented by NURBS. In my work I used a CAD program called Rhinoceros [31], a drawing program that uses NURBS as basis functions. I also used a C++ library called GoTools [12]. GoTools is developed by SINTEF and contains routines to handle spline objects. During the summer internship I was inspired to learn the theory behind splines and isogeometric analysis. As a consequence, I applied for a PhD fellowship in the field of nonlinear isogeometric analysis of thin-walled structures, and got accepted. The PhD will involve development, validation and implementation of prototype software for nonlinear isogeometric finite element analysis of thin-walled structures, starting right after I have turned in my Master's thesis. To get a flying start on the PhD the topic of my Master's thesis was thus chosen to involve the basic theory on isogeometric analysis. I am already familiar with the finite element method, and will in this thesis assume that the reader is familiar with it as well.

At the beginning I spent much time reading existing literature on the topic and learning the basic theory on isogeometric analysis, B-splines and NURBS. Thereafter, most of time was spent on implementing an isogeometric finite element solver for linear elasticity problems using NURBS. I also spent much time on studying degenerated mappings and implementing symbolic solvers in MATLAB. It was convenient for me to do all implementations in MATLAB, as it is the programming language I have the most experience with. However, when exploiting symbolical implementations, I experienced that MATLAB has its drawbacks.

Despite from late nights with writing and frustrating debugging, I have very much enjoyed working on this thesis. I would like to express my gratitude to my supervisors for inspiring me, introducing me to new challenges and for helping me carry out this work. I would like to thank them for showing much interest in my work, and for their help and support at the frequently advising sessions. I will also thank them for taking me to the first conference on isogeometric analysis, IGA 2011, at the University of Texas, Austin. At the conference I got up to date on ongoing research in this field, and got tons of inspiration for further studies. I enjoyed interesting conversations with PhD students and professors working within the same field, and got the opportunity to meet the persons that have written textbooks on splines and isogeometric analysis.

I hope you will find my thesis interesting!

Siv Bente Raknes Trondheim, 30^{th} of January 2011

Abstract

In this thesis we have given an introduction to isogeometric finite element analysis on linear elasticity problems in 2D using non uniform rational B-splines (NURBS) as basis functions. We have studied the theory of B-splines and have derived the equations needed to perform linear elasticity stress analysis. An isogeometric finite element solver has been programmed in MATLAB. We have also analyzed the effect degenerated mappings have on the derivatives of the basis functions. We started by looking at a quadrilateral collapsing to a triangle, considering different parameterizations and their impact on the derivatives. We found that the derivatives were no longer in H^1 and that our basis was not a proper basis for finite element analysis. Our solution to this problem is to form a new set of basis functions by summing the basis functions at the singular points. Further we have applied this approach on a circular surface and an infinite plate with a circular hole.

Contents

Chapter 1

Introduction

Isogeometric Analysis (IA) is a computational approach that integrates Finite Element Analysis (FEA) and Computer Aided Design (CAD). Isogeometric Analysis is developed in the purpose of utilizing the same data set in both design and analysis. In todays CAD and FEA packages one have to convert the data generated in design to a data set suitable for FEA. Converting the data is not trivial, as the computational geometric approach is different in CAD and FEA. IA makes it possible to utilize the NURBS geometry, which is the most used basis in CAD packages, in FEA directly. Isogeometric analysis is thus a great tool for optimizing models, as one easily can make refinements and perform testing and analysis during design and development.

Computer Aided Engineering

Computer Aided Engineering (CAE) involves using computer software for instance to create optimal designs or to perform analyses and simulations. In Computer Aided Engineering the finite element method is often applied as the analysis tool to solve partial differential equations by an piecewise polynomial approximation. Examples of fields within CAE is stress analysis [25], thermal and fluid flow analysis [4], fluid-structure interactions [34], computational fluid dynamics [30], kinematics [7], mechanical event simulation [9] and optimization of products or processes [10]. CAE often consists of a pre-processing phase, defining the model and environmental factors, an analysis solver and a post-processing part where results are visualized. The Finite Element Method (FEM) started developing in the 1950s [17], with all analyses made by hand. The method was therefore only applied on small and easy systems.

The next decades as analysts got experience with the method on different problems they started improving the algorithms and developing new basis function elements. Typically, Lagrange or Hermite polynomials are used as basis functions, and the problem domain is decomposed into non-overlapping elements of simple shapes like triangles, quadrilaterals etc. To solve partial differential equations by the FEM one first multiplies by a test function, also called a weighting function, before finding the variational formulation. The trial and weighting functions are most often represented by the same basis functions and used on the same elements. Next, the weak form is discretized in a finite dimensional space. To improve the FEM one can thus either improve the variational method, the spaces, the elements, or all those. For instance is the introduction of isoparametric elements an example of such an improvement. When computers were first used to perform

the analysis the computational efficiency was a very critical issue, making restriction on the choice of elements and degrees of freedom. Computational efficiency is still an issue. However, it turns out that higher order element uses more work per degrees of freedom but less degrees of freedom to converge.

Computer Aided Design

Earlier, before computers were easily accessible, designers worked at drawing boards using pencils. As from 1966 [17] Computer Aided Design made it possible to use computer technology in drawing and designing. Computers were now used to draw curves, surfaces and figures in 2D, or even solid 3D-objects. CAD is today very important in industrial design, and is extensively used in shipbuilding, automobile industries, aerospace industries, industrial and architectural design etc. [6]. CAD is also widely used in advertising and to produce computer animation for special effects in movies [18]. The CAD community evolved considerably in the 1970s, when it was possible to get computers with a graphical user interface.

As from 1972 [17] designers started using linear combinations of B-splines to represent curves. Non-uniform rational B-splines (NURBS) are a generalization of Bezier splines, and have been used in CAD programs since 1975 [17]. NURBS made it easier to generate and represent curves and surfaces with great flexibility and precision to handle both analytic and free form shapes [6]. By using NURBS we can represent conic section like circles, cylinders and spheres exactly. The development of NURBS began already in the 1950s by engineers that needed a mathematically precise representation of free form surfaces to model ship hulls, car bodies etc.. The precise representation was necessary to be able to do an exact reproduction of the models. The most used basis functions to represent geometries are NURBS. NURBS were in the beginning only used in proprietary CAD packages of car companies, but are today used in all standard CAD packages [6]. Today there exists many efficient numerical stable algorithms to generate NURBS objects, and refinement is easily done by knot insertion. Using NURBS lets us easily control the continuity, as C^{p-1}-continuity is obtained using p-th order NURBS.

Todays needs

Nowadays we want to make more and more complex constructions, resulting in the need of more efficient and accurate methods for design and analysis. It is a difficult task to improve the efficiency, as there exists a gap between FEA and CAD. The design and the analysis communities evolved independently of each other, as they had different goals and needs. Analysts have been looking at accurate systems which are easy to interpret, and at which is was possible to perform analysis computationally fast. Designers, on the other hand, made systems that were easy to manipulate, visualize and construct. As they looked at different geometric constructions, they developed to look at different systems and naturally concentrated on improving algorithms and such according to the respectively needs. With increasing computer power, a want to perform more complex FEA on CAD geometries occurred.

With increased computer power, the running time is no longer the biggest bottleneck for exploiting an efficient analysis. Instead, generating the geometrical mesh is. Before one can perform analysis on the CAD model one need to make the geometry suitable

for analysis in a way that preserves the geometry in sufficiently detail. Converting the geometric model is not trivial, and it takes a huge amount of man-time. In fact, for complex models, converting the CAD geometry to an analysis-suitable mesh might take over 80 percent of the total time spent on analysis [16]. To given an example on the complexity of typical constructions; an average automobile consists of more than 3000 parts and a nuclear submarine consists of more than 1,000,000 parts [17].

More and more complex constructions are being produced. With the industry living by the principle that time is money, there is also a growing need to decrease the time spent on making new grids for analysis. Consequently, there has been a lot of research on how to automate the process of converting from one system to the other. However, none of the algorithms and techniques developed turned out very successful in the industry. Instead of being an all automated process, the mesh generators are mostly used as an supplement. Analysts still both need and prefer to make and improve the grids by hand. Mesh generators only approximates the design, loosing the information on the exact geometry in the process. Making refinements are thus difficult. To perform mesh refinement one need to know the exact geometry, making the analysts dependent on a good communication with the designer. A seamless and automatic process was hence not obtained. Without an accurate geometry and mesh adaptivity, convergence and high-precision result are not possible to obtain. Shell buckling analysis is an good example on a very sensitive case where small errors in the geometric model may lead to huge errors in the analysis. Also, sliding contact between bodies will not be accurately modeled when the geometric description is not precise [16].

The development of isogeometric analysis

Researchers got a feeling that the problem needed to be solved differently. In 2003, Thomas Hughes at the University of Texas, Austin started working on a different approach aiming to fill the gap between FEA and CAD. Hughes believed that struggling with conversion, refinement and automatic mesh generators to connect the two systems was not the right way to approach the problem; rather he believed that the real issue lied in the fact that two different systems are being used. Hughes claims that using the same system in both design and analysis would be more convenient as well as giving more accurate results. Together with his PhD students, J. Cottrell and Y. Bazilevs, they continued the research on how to use the same basis functions in design and finite element analysis, an approach they called isogeometric analysis. As engineering design is dominated by NURBS, and NURBS are ofter more suited to represent geometries compared to Lagrange or Hermite polynomials, they found it better to change the bases used in analysis by the bases used in design. In 2005 they released the first paper on isogeometric analysis [16], presenting a seamless interaction between design and analysis.

CAD and FEA grew up and developed independently, but could with the concept of isogeometric analysis start developing towards the same goals. Hughes, Cottrell and Bazilevs have put a lot of work in initiating conversations between designers and analysts; to make them agree upon geometrical description and help them gaining knowledge of both sides. They encourage computational analysts to learn about isogeometric analysis, and to start exploiting this seamless interaction between design and analysis. To make the isogeometric approach successful and to break down the barriers between engineering design and analysis, reconstruction of the entire process is necessary, and changes must

be done by both sides. The analysts need to start working with different geometries and basis functions, and designers need to create better models that are suited for analysis.

By utilizing NURBS as basis functions one no longer have to make the finite element meshes explicitly, one can instead use the meshes made to represent the geometry in CAD programs. When the same geometric model is used in both design and analysis, analysis will be performed on a precise geometric model, resulting in faster and more accurate results. Hence, a much greater precision is obtained. Examples of fields where isogeometric analysis is suitable are [16] structural mechanics, fluid dynamics, acoustics, heat transfer and electromagnetics [1]. Optimization of CAD geometries has before the concept of isogeometric analysis been difficult and non existing in most industries, since the meshes haven't been equal and the mapping between CAD geometries and FE meshes has not been integrated in commercial FE solvers.

Through the introduction of isogeometric analysis, a great computational tools for optimizing designs will be available. Lets say we want to optimize the design of a ship. Building and testing the properties of the ship is extremely expensive and time consuming, we for sure want do much pre-testing on the computer. If we then need to make changes in the design and perform another FE analysis, a lot of time will be spent on making the new FE mesh. It is very time consuming to make a new FE meshes every time one want to check the effect of a small change in the design. Also, the FE analysis gives only an approximation of the design. With a precise isogeometric analysis tool in hand this process would be much more efficient. Another benefit of exploiting isogeometric analysis is that making refinements without changing the geometry is easily done by for instance inserting additional knots.

The concept of isogeometric analysis has proven to be successful in both the interest of designers as well as of analysts. Not needing to generate the geometrical mesh to perform analysis makes the process much more efficient, saving a lot of man-hours. NURBS-based isogeometric analysis has already been applied to for instance fluids [41], structures [22], fluid-structure interaction [42], phase-field modeling [13], electromagnetics, biomedical modeling [45], shape optimization [3, 38], modeling of structural vibrations [20] and structural dynamics and wave propagation [19].

This thesis

This thesis serves to be an introduction to isogeometric finite element analysis on linear elasticity problems. In addition to just learning the basics, we have attempted to solve problems caused by degenerated mappings. Example of a degenerated mapping is a quadrilateral where one line collapse to a point, forming a triangle. It could also be a hexahedron collapsing into a pyramid. If we have degenerated mappings, or singular parameterizations, it can happen that some of the resulting test functions are not well defined at the singular points and are not integrable. If so, the stiffness matrix integrals will not exist either. Singularities in the parameterization can be caused by intrinsic properties of the geometry of the object or by distortions of regular parameterizations, possibly due to mesh adaption in shape optimization [36]. If that is the case one can avoid the singularities via some constraints [36]. In many cases it is difficult or even impossible to avoid singularly parametrized objects, for instance when working with single patch circular domains [36]. E. Cohen et. al. emphasizes in the paper *Analysis-aware modeling: Understanding quality considerations in modeling for isogeometric analysis* [8] the impor-

tance of creating analysis suitable models. The main idea of analysis-aware modeling is that model properties and parameters must be selected to facilitate isogeometric analysis [8]. To obtain that, continual interaction and dialouge between designers and analysts is necessary. The quality of models are critical for the quality of the analysis. For instance, mesh distortion is a huge problem in finite element analysis, as it may lead to invalid computational analysis [33]. In this thesis we will look at degenerated mappings that cause the derivatives to be infinite, and for that reason creates a non-physical singularity at the degeneracy. Thomas Hughes stated this problem in his book on the finite element method from 1987 [15]. His approach for solving the problem was to create a new set of basis functions by adding the basis functions at the coalescing nodes. We want to bring this issue up in an isogeometric setting.

In Chapter 2 we start by studying B-splines and NURBS, before looking at isogeometric analysis in comparison to classical finite element analysis. Chapter 3 contains some theory of the finite element method that we need when studying degenerated mappings in Chapter 6. As we assume the reader to be familiar with classical finite element analysis, we have not given all details and proves. In chapter 4 we will give an introduction to 2D linear elasticity, defining terms like strain and stress, and deriving the equilibrium equation along with the strong and weak forms. In Chapter 5 we have considered the finite element discretization of the weak form, and have given details on how to solve the discrete problem from a computational point of view. As already mentioned, in Chapter 6 we have been looking at degenerated mappings, considering different parameterizations of a quadrilateral collapsing to a triangle. In Chapter 7 we have tested our theory on how to deal with degenerated mappings on actual problems. We have been looking at a circular surface and the well known problem; an infinite plate with a circular hole. We have done all programming in MATLAB. Implementations can be found in the appendix.

Chapter 2

Isogeometric analysis using NURBS

Isoparametric analysis involves using the same basis functions to represent design as well as to perform analysis, whereas isogeometric analysis also implies letting the geometry be the deciding factor on exactly what kind of basis functions to use. The isogeometric concept is that the basis functions that exactly represents the geometry is the same that spans the solution space in analysis. In CAD, Non Uniform Rational B-Splines (NURBS) are frequently used to represent geometries, and in CAE, the classical finite element method is commonly used as the analysis tool. By the isogeometric concept our goal is to fill the gap between design and analysis by introducing a seamless transfer of models. The main idea in isogeometric analysis is to use NURBS as basis functions in both design and in the finite element method.

NURBS are, as the name indicate, built from B-splines. To get a better understanding of NURBS we will therefore first take a look at B-splines. Toward the end of this chapter we will to some extent compare isogeometric analysis to classical finite element analysis.

Be aware that we in this thesis refer to the degree and the order of a polynomial as the same quantity, that is, a quadratic polynomial is of both order and degree two.

2.1 B-splines

B-splines are built from piecewise polynomial functions, i.e., from a set of polynomial functions that are defined on non-overlapping connected intervals. At the interval boundaries we require the polynomial functions to be continuous. B-splines are consequently smooth, differentiable and continuous functions within each subinterval. Across connected subintervals on the other hand, they are continuous but not necessarily differentiable. B-splines are used as basis functions in most CAD programs because they are very well suited for representing geometries consisting of curves or surfaces.

2.1.1 Knot vectors

A B-spline consists of n piecewise polynomial basis functions of degree p. To define the piecewise polynomial basis functions we make use of knot vectors. A knot vector, Ξ, is defined by a set of coordinates, or knots, that gives information on where the subintervals are connected. The knots are located at the interval boundaries. A knot vector is a one dimensional non-decreasing set of real valued coordinates $\Xi = \{\xi_1, \xi_2, ..., \xi_{n+p+1}\}$, where

$\xi_i \in \mathbf{R}$ is the i^{th} knot, i is the knot index, $i = 1, 2, ..., n + p + 1$, p is the polynomial order and n is the number of basis function [17, 27].

The knot-spans define element domains where the basis functions are smooth, that is, where they are C^∞. Across knots the basis functions are C^{p-m}, where p is the degree of the polynomial and m is the multiplicity of the knot. Knots may be located at the same coordinates; we then say that we have repeated knot values. Repeated knot values cause a reduction of the continuity of the basis functions across that knot, a property that is frequently exploited in CAD. If the first and last knot values are repeated $p + 1$ times, the knot vector is said to be *open*. An open knot vector forces the basis functions to interpolate the knots at the beginning and the end of the interval the knot vector represents. In general, the basis functions are not interpolating the interior knots.

2.1.2 Basis functions

The B-spline basis functions are defined recursively [16] by

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)$$

for $p = 1, 2, 3, \ldots$. For $p = 0$ we have that

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if} \quad \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

If the denominators in the factor we multiply the basis functions by are zero we define the factor to be zero. That is,

$$\frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} \equiv 0 \quad \text{if} \quad \xi_{i+p} - \xi_i = 0,$$
$$\frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} \equiv 0 \quad \text{if} \quad \xi_{i+p+1} - \xi_{i+1} = 0.$$

This formula is also known as the *Cox-de Boor recursion formula* [16]. Dynamic programming is recommended to improve the running time of this recursively formula. Else wise the same values will be calculated several times.

We have n basis functions, with $N_{i,p}$ being the i^{th} basis function of order p, $i \in [1, n]$. The number of basis functions is determined by the order and the number of knots; we have $n + p + 1$ knots resulting in n basis functions. Increasing the number of knots will consequently also increase the number of basis function. It is worth noticing that the basis functions are non-negative and that they form a partition of unity, i.e.

$$N_{i,p}(\xi) \geq 0 \quad \forall \xi,$$
$$\sum_{i=1}^{n} N_{i,p}(\xi) = 1.$$

Two other properties of the basis functions are that they have local support and local knots. The properties are stated in Lemma 2.6 in [27] and involve the following; Assume that we have the knot vector $\Xi = \{\xi_1, \xi_2, ..., \xi_{n+p+1}\}$. Then $N_i^p(\xi) = 0$ if ξ is outside the interval $[\xi_i, \xi_{i+p+1})$. Thus, the i^{th} B-spline $N_i^p(\xi)$ depends only on the knots $[\xi_i, \xi_{i+p+1})$.

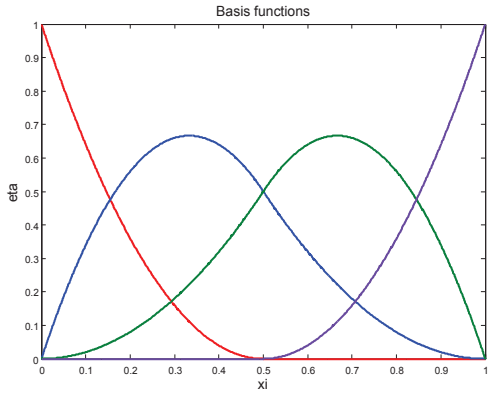


Figure 2.1: *Basis functions for knot vector* $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ *of degree* 2.

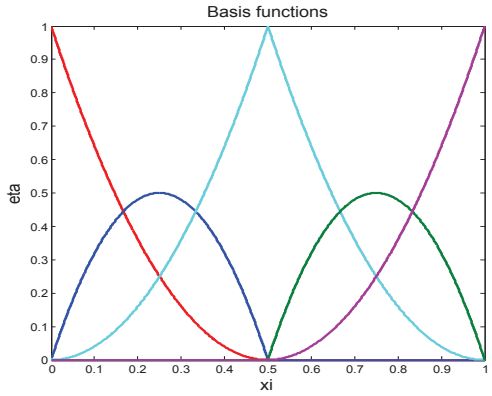


Figure 2.2: *Basis functions for knot vector* $\Xi = \{0, 0, 0, 0.5, 0.5, 1, 1, 1\}$ *of degree* 2.

An example

Consider the knot vector $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ of degree $p = 2$. This is an open knot vector, forcing the basis functions to interpolate the knots at the boundary of the domain. With seven knots and polynomials of degree two we will have $n = 7 - 2 - 1 = 4$ basis functions available to construct the B-spline curve. Since we have no repeated knots the basis functions will be $C^{p-m} = C^1$-continuous across the knot in the interior of the domain. The basis functions are shown in Figure 2.1.

We may increase the multiplicity of knots by inserting another knot at 0.5, resulting in the knot vector $\Xi = \{0, 0, 0, 0.5, 0.5, 1, 1, 1\}$, still of degree $p = 2$. The $n = 8-2-1 = 5$ basis functions will in this case be $C^{p-m} = C^0$-continuous across knots in the interior of the domain and will interpolate the knots at the beginning and end of the interval. The basis functions corresponding to this knot vector are shown in Figure 2.2.

2.1.3 B-spline curves

B-spline curves in $\mathbf{R}^d$ are made by a linear combination of B-spline basis function. The coefficients we multiply the basis functions by prior to adding them are called control points. The resulting B-spline curve does not need to interpolate the control points. However, we may force it to do so by utilizing a knot vector with sufficient multiplicity to insure that the basis functions, and hence the B-spline curve, will be $C^{p-m} = C^0$-continuous

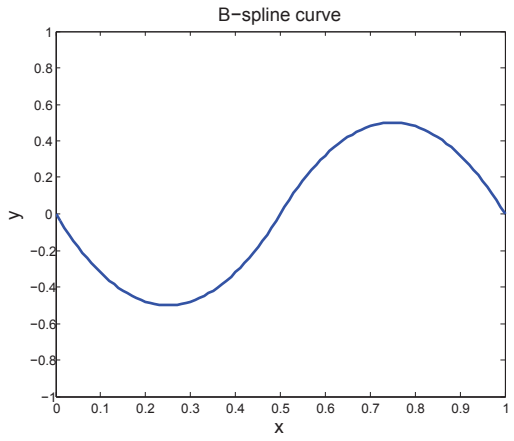


Figure 2.3: *B-spline curve in* $\mathbf{R}^1$ *for control points* $\mathbf{B} = [0, -1, 1, 0]$ *and knot vector* $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ *of degree* 2.

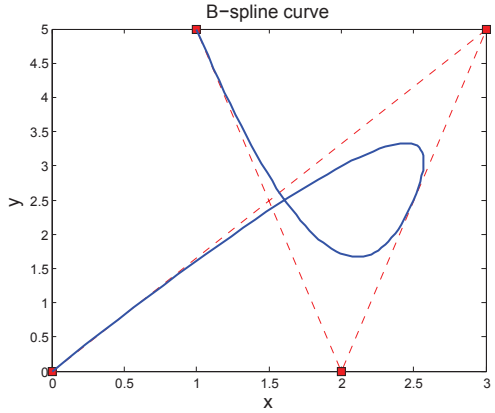


Figure 2.4: *B-spline curve for knot vector* $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ *of degree* 2 *and the control points* $\mathbf{B}_x = [0, 3, 2, 1]$ *and* $\mathbf{B}_y = [0, 5, 0, 5]$.

across that particular knot. For instance, the open knot vector $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ of degree $p = 2$ will force the B-spline curve to interpolate the first and the last control point. For the knot vector $\Xi = \{0, 0, 0, 0.5, 0.5, 1, 1, 1\}$ of degree $p = 2$ the B-spline curve will in addition interpolate the control point belonging to $\xi = 0.5$.

Say we have n basis functions $N_{i,p}(\xi)$, $i = 1, 2, \ldots, n$, with control points $\mathbf{B}_i \in \mathbf{R}^d$. The corresponding B-spline curve is then given by

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{B}_i.$$

This expression can be interpreted as a mapping taking us from a parameter space spanned by the knot vector to a physical space defined by the control points. More on this is to come in section 2.3.

For knot vector $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ of degree 2 and control points $\mathbf{B} = [0, -1, 1, 0]$ in $\mathbf{R}^1$ we get the curve shown in Figure 2.3. The same knot vector with control points $\mathbf{B}_x = [0, 3, 2, 1]$ and $\mathbf{B}_y = [0, 5, 0, 5]$ in $\mathbf{R}^2$ gives the curve shown in Figure 2.4. The control points are shown as red squares. We see that the curve only interpolates the first and last control points.

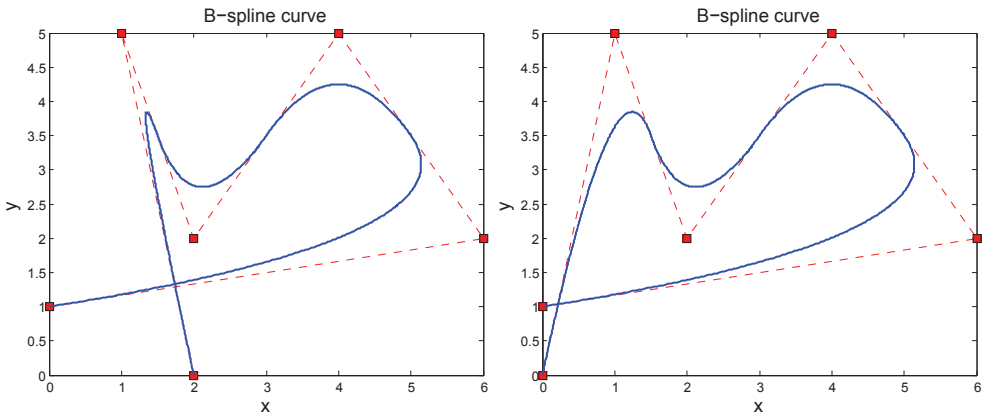


Figure 2.5: *B-spline curve for knot vector* $\Xi = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$ *of degree* 2 *and control points located at red squares.*

Because the basis functions have local support only a small part of the curve will be changed if we change a control point. Figure 2.5 illustrates this. Here we have changed the x-coordinate of first control point from 2 to 0. As you can see, only the part of the curve that is closest to the moved control point has changed. This is one of the reasons why B-splines are used in CAD programs; due to the local support of basis functions one can easily manipulate curves by dragging the control points.

2.1.4 B-spline surfaces

With the basic concept of B-spline curves fresh in mind we are now ready to take a look at B-spline surfaces. Assume as before that we have the knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, where $\xi_i \in \mathbf{R}$ is the i^{th} knot, i is the knot index, $i = 1, 2, \ldots, n+p+1$, p is the polynomial order and n is the number of basis function. In order to create a surface we need to introduce a second knot vector; $\mathcal{H} = \{\eta_1, \eta_2, \ldots, \eta_{m+q+1}\}$, where $\eta_j \in \mathbf{R}$ is the j^{th} knot, j is the knot index, $j = 1, 2, \ldots, m+q+1$, m is the polynomial order and q is the number of basis function. The basis functions for the surface is formed by a tensor product of the basis functions $N_{i,p}(\xi)$, $i = 1, 2, \ldots, n$, and $M_{j,q}(\eta)$, $j = 1, 2, \ldots, m$. Together with the control net $\mathbf{B}_{ij} \in \mathbf{R}^d$ the B-spline surface is given by

$$\mathbf{S}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \mathbf{B}_{ij}. \tag{2.1}$$

Notice that also the tensor product will form a partition of unity;
$\forall (\xi, \eta) \in [\xi_1, \xi_{n+p+1}] \times [\eta_1, \eta_{m+q+1}]$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) = \left(\sum_{i=1}^{n} N_{i,p}(\xi) \right) \left(\sum_{j=1}^{m} M_{j,q}(\eta) \right) = 1.$$

An example

Consider the knot vector $\mathcal{H} = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$ of degree $p = 2$ with corresponding basis functions as shown in Figure 2.6. Together with the basis functions made by the knot vector $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ of degree $q = 2$, shown in Figure 2.1 and the

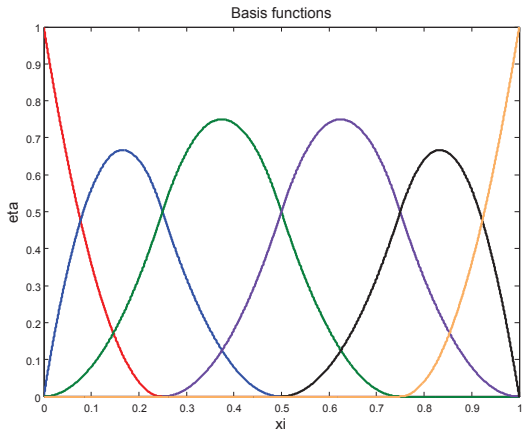


Figure 2.6: *Basis functions for knot vector* $\mathcal{H} = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$ *of degree* 2.

Table 2.1: Control net $\mathbf{B}_{i,j}$

(i,j)	1	2	3	4	5	6
1	(3,0)	(3.5,1)	(3,2)	(3,3)	(3.5,4)	(3,5)
2	(2,1)	(2.5,2)	(2,3)	(2,4)	(2.5,5)	(2,6)
3	(1,0)	(1.5,1)	(1,2)	(0.5,3)	(1,4)	(1,5)
4	(0,1)	(0.5,2)	(0,3)	(0.4,4)	(0,5)	(0.2,6)

control net given in Table 2.1 we can create a B-spline surface by the formula (2.1). The corresponding surface is shown in Figure 2.7.

2.1.5 Derivatives of B-spline basis functions

To do analysis on B-splines we often need to know the expression for the derivatives. For a polynomial of order p with a knot vector Ξ the derivative of the i^{th} basis function is given by [16]

$$\frac{d}{d\xi}N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}N_{i+1,p-1}(\xi). \tag{2.2}$$

By differentiating (2.2) k times we obtain

$$\frac{d^k}{d^k\xi}N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}\left(\frac{d^{k-1}}{d^{k-1}\xi}N_{i,p-1}(\xi)\right) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}\left(\frac{d^{k-1}}{d^{k-1}\xi}N_{i+1,p-1}(\xi)\right) \tag{2.3}$$

Using (2.2) we can express the right side of (2.3) by lower order basis functions. A generalized expression for higher derivatives of B-splines is thus given by [16]

$$\frac{d^k}{d^k\xi}N_{i,p}(\xi) = \frac{p!}{(p-k)!}\sum_{j=0}^{k}\alpha_{k,j}N_{i+j,p-k}(\xi),$$

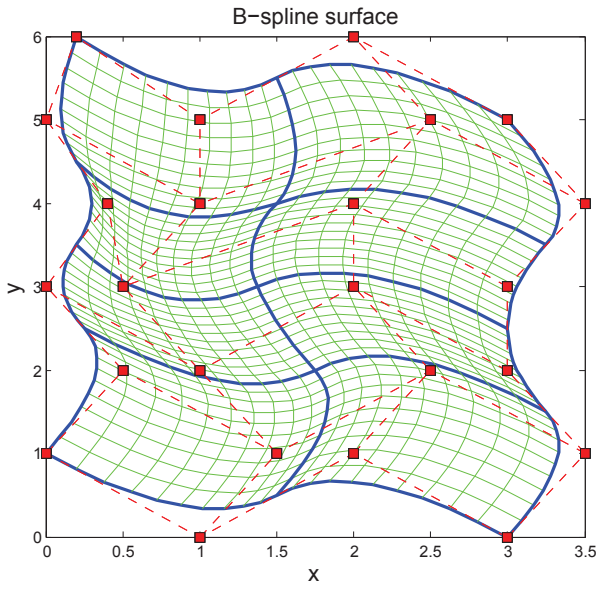


Figure 2.7: *B-spline surface for the control net* $\mathbf{B}_{i,j}$ *given in Table 2.1 and knot vectors* $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ *and* $\mathcal{H} = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$ *of degree* 2.

where

$$\begin{aligned}
\alpha_{0,0} &= 1, \\
\alpha_{k,0} &= \frac{\alpha_{k-1,0}}{\xi_{i+p-k+1} - \xi_i}, \\
\alpha_{k,j} &= \frac{\alpha_{k-1,j} - \alpha_{k-1,j-1}}{\xi_{i+p+j-k+1} - \xi_{i+j}} \quad j = 1, ..., k-1, \\
\alpha_{k,k} &= \frac{-\alpha_{k-1,k-1}}{\xi_{i+p+1} - \xi_{i+k}}.
\end{aligned}$$

2.1.6 Refinement; knot insertion

There are several ways to refine a B-spline. We could either insert additional knots, increase the order of the basis or do both. Inserting additional knots are called h-refinement and increasing the order is referred to as order elevation or p-refinement. Doing both is known as k-refinement. In this thesis we will only consider knot insertion. More about all three types of refinement can be found in [17].

Inserting additional knots will enrich the basis without changing the curve geometrically or parametrically. Assume that we begin with the knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ with n corresponding basis function and the control points $\mathbf{B} = \{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n\}$. By inserting more knots we can extended Ξ to a knot vector $\tilde{\Xi} = \{\tilde{\xi}_1 = \xi_1, \tilde{\xi}_2, \ldots, \tilde{\xi}_{n+m+p+1} = \xi_{n+p+1}\}$, $\Xi \subset \tilde{\Xi}$. Our set of basis functions will now be extended to consist of $n+m$ basis functions. We also need to extend $\mathbf{B}$ to contain $n+m$ control points. We create the new control points $\tilde{\mathbf{B}} = \{\tilde{\mathbf{B}}_1, \tilde{\mathbf{B}}_2, \ldots, \tilde{\mathbf{B}}_{n+m}\}$ by a linear combination of the previous control points, $\mathbf{B}$. The new control points $\tilde{\mathbf{B}}$ are given by [17, 27]

$$\tilde{\mathbf{B}} = \mathbf{T}^p \mathbf{B}$$

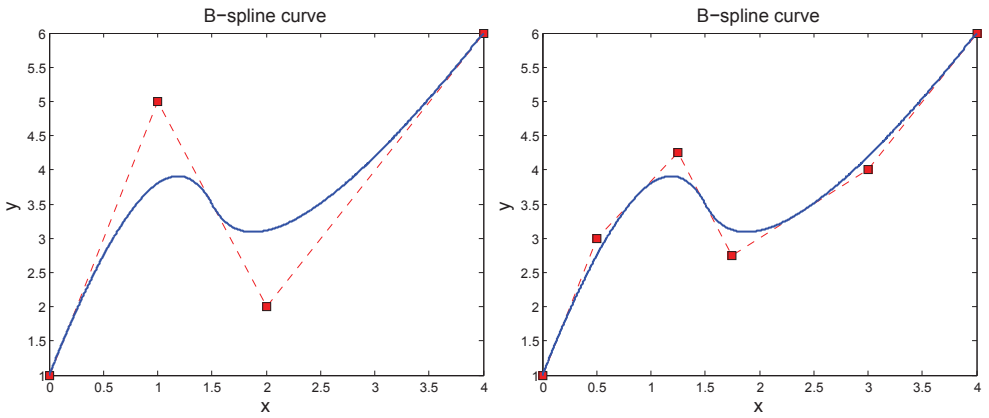


Figure 2.8: *Knot refinement from* $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ *to* $\tilde{\Xi} = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$. *Notice that the curve remains unchanged.*

where

$$T_{i,j}^{q+1} = \frac{\tilde{\xi}_{i+q} - \xi_j}{\xi_{j+q} - \xi_j} T_{i,j}^{q} + \frac{\xi_{j+q+1} - \tilde{\xi}_{i+q}}{\xi_{j+q+1} - \xi_{j+1}} T_{i,j+1}^{q}$$

for $q = 0, 1, 2, \ldots, p-1$ and

$$T_{i,j}^{0} = \begin{cases} 1 & \tilde{\xi}_i \in [\xi_j, \xi_{j+1}) \\ 0 & \text{otherwise} \end{cases} .$$

Knot insertion can also be used to create repeated knots. The continuity of the basis will then be reduced while the curve will be the same. In Figure 2.8 we have exploited a knot insertion, increasing the knot vector $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ of order 2 to an extended knot vector $\tilde{\Xi} = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$. Observe that the B-spline curve is similar before and after additional knots were inserted.

2.2 NURBS

NURBS, non uniform rational B-splines, are piecewise rational polynomials built from B-splines. The term *non uniform* refers to using non uniform knot vectors. The term *rational* refers to the fact that NURBS are a combination of B-splines basis functions multiplied by a weighting factor, divided by the sum of the B-spline basis functions multiplied by the same weights. If all the weights are equal to one, NURBS will be equal to B-splines.

NURBS are very adequate to represent geometrical entities, especially conic sections like circles and ellipses. Using second order Lagrange polynomials on quadrilateral elements in standard FEA we would need quite many nodes to give a good approximation of a circle. When using NURBS on the other hand, we are able to represent a circle exactly by only nine points. NURBS are because of that ability, among others, frequently used in CAD to represent geometry. Using NURBS to represent both the physical domain used in analysis and the solution will give more accurate results than first reconstructing and simplifying the geometry to create an FE mesh and then approximating the solution using Lagrange polynomials. NURBS provides better accuracy and robustness, generalizing and improving traditional piecewise polynomial basis functions [16]. Using NURBS also

provides great flexibility. For instance, by changing the order, the location or the multiplicity of the knot vectors, we are able to change the basis functions to possess desired properties. Making changes in the control net will also contribute to change the mapping and consequently the resulting curve or surface.

2.2.1 Geometric perspective

NURBS are regular B-spline curves that have been projected to a space of one dimension smaller. That is, NURBS in $\mathbf{R}^d$ are made from a projecting transformation of B-spline curves in $\mathbf{R}^{d+1}$ [17, 24, 26]. Conic sections like circles or ellipses are thus made by projecting piecewise quadratic B-spline curves from the (x, y, z)-plane to the $(x, y, z = 1)$-plane.

Assume that we have the B-spline curve $\mathbf{C}^w(\xi)$ with control points $\mathbf{B}_i^w \in \mathbf{R}^{d+1}$. The projected NURBS control points $\mathbf{B}_i$ are then given from the B-spline control points by

$$(\mathbf{B}_i)_j = \frac{(\mathbf{B}_i^w)_j}{w_i} \quad j = 1, \ldots, d$$
$$w_i = (\mathbf{B}_i)_{d+1}$$

Here $(\mathbf{B}_i)_j$ is the j^{th} component of the vector $\mathbf{B}_i$ and w_i is the i^{th} weight. In $\mathbf{R}^3$, the weights corresponds to the z-coordinates of the B-spline curve. Dividing the NURBS control points by the weight are thus the same as applying a projective transformation on them. The same transformations need to be exploited on every point on the curve. With B-spline curves in $\mathbf{R}^3$ that implies dividing all points by its height, see Figure 2.9. To exploit the same transformation on all points we divide all points by the weighting function

$$W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) w_i,$$

where $N_{i,p}(\xi)$ are the B-spline basis functions. The NURBS curve, $\mathbf{C}(\xi)$, can now be defined as

$$(\mathbf{C}(\xi))_j = \frac{(\mathbf{C}^w(\xi))_j}{W(\xi)} \quad j = 1, \ldots, d$$

where $\mathbf{C}(\xi)^w = \sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{B}_i^w = \sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{B}_i w_i$.

2.2.2 Basis functions

With the geometric point of view established we can define the NURBS basis functions as

$$R_i^p(\xi) = \frac{N_{i,p}(\xi) w_i}{W(\xi)} \tag{2.4}$$

where

$$W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) w_i$$

as before. NURBS curves are then given by

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} R_i^p(\xi)\mathbf{B}_i.$$

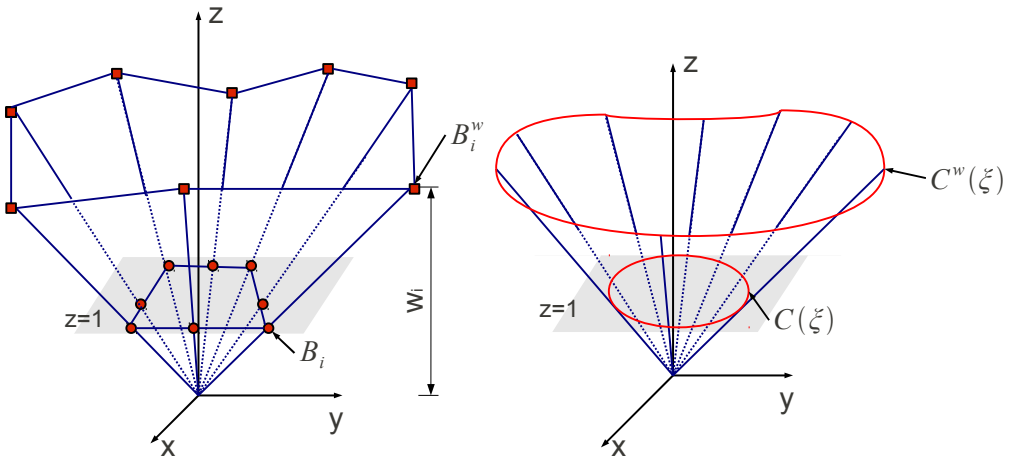


Figure 2.9: *Projective transformation from a quadratic B-spline in* $\mathbf{R}^3$ *to a circle in* $\mathbf{R}^2$*. The figure is reconstructed from Figure 2.28 in [17].*

This is the mapping from the parameter space to the physical space, as we will discuss in Chapter 2.3.

In 2D the basis function are given as a tensor product by

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_{i,p}(\xi)M_{j,q}(\eta)w_{i,j}}{W(\xi,\eta)}, \tag{2.5}$$

with

$$W(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\xi)M_{j,q}(\xi)w_{i,j}.$$

Surfaces are in the same way given by

$$\mathbf{S}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q}(\xi,\eta)\mathbf{B}_{i,j}.$$

Extension to three dimensions are done analogously. Notice that if all weights are equal to one, NURBS and B-spline are the same.

2.2.3 Derivatives of basis functions

Using NURBS in analysis we often need to know the derivatives of the basis functions. We get the first derivative by differentiating (2.4) with respect to ξ using the quotient rule. We thus obtain

$$\frac{d}{d\xi}R_i^p(\xi) = w_i\frac{N_{i,p}(\xi)'W(\xi) - N_{i,p}(\xi)W(\xi)'}{(W(\xi))^2},$$

where

$$W(\xi)' = \sum_{i=1}^{n} N_{i,p}(\xi)'w_i.$$

Generalizing to higher order terms we get that [17]

$$\frac{d^k}{d^k\xi}R_i^p(\xi) = \frac{w_i\frac{d^k}{d^k\xi}N_{i,p}(\xi) - \sum_{j=1}^{k}\frac{k!}{j!(k-j)!}W^{(j)}(\xi)\frac{d^{k-1}}{d^{k-1}\xi}R_i^p(\xi)}{W(\xi)}.$$

In 2D, differentiating (2.5) with respect to ξ yields

$$\frac{d}{d\xi}R_{i,j}^{p,q}(\xi,\eta) = w_{i,j}\frac{\left(\frac{d}{d\xi}N_{i,p}(\xi)\right)M_{j,q}(\eta)W(\xi,\eta) - N_{i,p}(\xi)M_{j,q}(\eta)\left(\frac{d}{d\xi}W(\xi,\eta)\right)}{(W(\xi,\eta))^2}.$$

With respect to η we get

$$\frac{d}{d\eta}R_{i,j}^{p,q}(\xi,\eta) = w_{i,j}\frac{N_{i,p}(\xi)\left(\frac{d}{d\eta}M_{j,q}(\eta)\right)W(\xi,\eta) - N_{i,p}(\xi)M_{j,q}(\eta)\left(\frac{d}{d\eta}W(\xi,\eta)\right)}{(W(\xi,\eta))^2}.$$

Here

$$\frac{d}{d\xi}W(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m}\left(\frac{d}{d\xi}N_{i,p}(\xi)\right)M_{j,q}(\eta)w_{i,j}$$

and

$$\frac{d}{d\eta}W(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m}N_{i,p}(\xi)\left(\frac{d}{d\eta}M_{j,q}(\eta)\right)w_{i,j}.$$

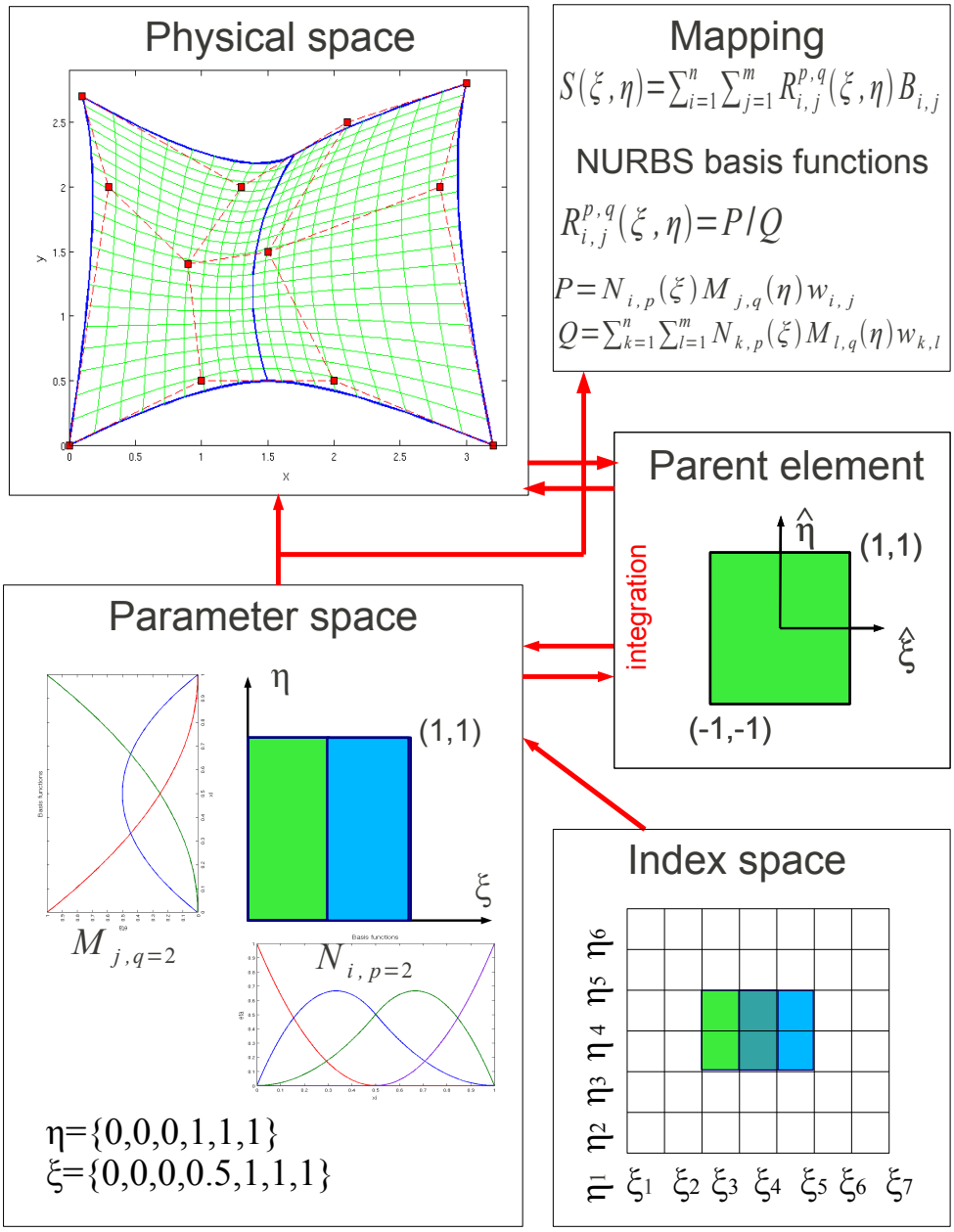


Figure 2.10: *The different spaces. We here see a physical mesh consisting of one patch. In the physical space the control mesh is shown as red dotted lines and the control points are shown as red squares. Notice that the control elements are bilinear quadrilaterals. The blue lines in the physical space shows how it is divided into knot spans, corresponding to how the parameter space is divided into two elements by the knots. The support of each element is shown in colors in the index space. The green element in the parameter space has support on the green and the turquoise area in the index space. The blue area in the parameter space has support on the turquoise and the blue area in the index space. Notice that the support is overlapping for knots of degree greater than one. Knot vectors of degree two, as we are dealing with here, have support in the area that is spanned by two knots in each direction.*

2.3 Spaces and mappings

In classical finite element analysis we are working on different domains; we have the physical mesh, the physical elements and the parent domain. The physical mesh is where the geometry is represented with help from nodes. The physical mesh is divided into non overlapping physical elements. The parent element is where we perform integration by utilizing Gaussian quadrature. All physical elements are mapped to the same parent element, and we can apply the inverse mapping to return to the physical element after the integration is exploited. The physical elements are defined by the nodal coordinates, and the degrees of freedom are the values of the basis function at the nodes. Due to compact support, the local basis functions only have support on neighboring elements. The basis functions are interpolating the nodes and are often called shape functions.

In isogeometric analysis we are also working on different domains. We have the physical mesh, the control mesh, the parameter space, the index space and the parent element, all shown in Figure 2.10.

2.3.1 The physical space

The physical space is where the actual geometry is represented by a linear combination of the basis functions and the control points. The basis functions are usually not interpolating the control points. The physical mesh is a decomposition of the geometry and can be divided into elements in two different ways; we can either divide it by patches or by knot spans.

A **patch** can be thought of as a subdomain. Patches are curves in 1D, surfaces in 2D and volumes in 3D. Many geometries can be modeled by a single patch, as is the case for all geometries we will consider in this thesis. Each patch can again be divided into knot spans. **Knot spans** are bounded by the knots and define element domains where the basis functions are smooth. The basis functions are smooth in the interior of the elements, and are at the boundary, that is, across knots, C^{p-m}-continuous. p is the polynomial degree and m is the multiplicity of the particular knot. Knot spans are the smallest elements. In the parent domain knots are points in 1D, lines in 2D and planes in 3D. In the physical space they are points in 1D, curves in 2D and surfaces in 3D.

2.3.2 The control mesh

The control mesh is defined by the control points. The control mesh interpolates all control points. It controls the geometry, but does generally not coincide with the physical mesh. In 1D the control elements are straight lines between two control points. In 2D the mesh consists of bilinear quadrilaterals defined by four control points. In 3D the control elements are trilinear hexahedras defined by eight control points [17]. The control variables are located at the control points and are the degrees of freedom [17]. The control mesh may be degenerated, for instance from a quadrilateral to a triangle. Control meshes may be severely distorted while the physical geometry still remains well defined.

2.3.3 The parameter space

The parameter space is where the NURBS basis functions $R_{i,j}^{p,q}$ are defined and where the local elements are given by the knots. The knot vector is uniform if all the knots are equally distributed in the parameter space. The parameter space is local to patches. In our finite element code we hence have to loop over all patches in addition to looping over all elements on the current patch. We map a patch of multiple elements in the parameter space into the physical space. Each element in the physical space is thus an image of the corresponding element in the parameter space [17]. The mapping from the parameter space to the physical space is for surfaces given by

$$\mathbf{S}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q}(\xi,\eta)\mathbf{B}_{i,j},$$

a mapping that is global to the whole patch.

2.3.4 The index space

In the index space of a patch each knot can be uniquely identified, that included knots having multiplicity greater than one. The index space is spanned by the area $[1, n+p+1]$ in the i-direction and $[1, m+q+1]$ in the j-direction, and is thus given by $[1, n+p+1] \times [1, m+q+1]$.

2.3.5 The parent element

The parent element is the constant area $[-1,1] \times [-1,1]$ and is where we perform the integration. We map ξ and η in the parameter space to $\hat{\xi}$ and $\hat{\eta}$ in the parent element to make it easier to exploit Gaussian quadrature. The mapping from the parent element to the parameter space is given by

$$\xi(\hat{\xi}) = \frac{(\xi_{i+1}-\xi_i)\hat{\xi} + (\xi_{i+1}+\xi_i)}{2},$$
$$\eta(\hat{\eta}) = \frac{(\eta_{i+1}-\eta_i)\hat{\eta} + (\eta_{i+1}+\eta_i)}{2}.$$

In Chapter 5 we will take a closer look on the different mappings from a computational point of view.

2.4 Isogeometric analysis vs finite element analysis

In this section we will mention some of the differences and similarities between isogeometric analysis and standard finite element analysis. The concept of isogeometric analysis is that the basis functions that are used to model the exact geometry are also used as a basis for the solution field. In classical FEA it is the other way around; the basis functions we choose to approximate the unknown solution field is used to approximate the already known geometry. NURBS based Galerkin finite element method is somewhat similar to classical FEA, only now, different basis functions are being used.

Table 2.2: Comparing isogeometric analysis and finite element analysis

Isogeometric analysis	Finite element analysis
Control points	Nodal points
Control variables	Nodal variables
Knots	Mesh
Exact geometry	Approximated geometry
NURBS basis functions	Lagrange basis functions
Basis not interpolating control points	Basis interpolating nodes
Patches	Subdomains
Compact support	
Partition of unity	

The code architecture of isogeometric analysis is shown in Figure 2.11. The routines shown in blue differ from those used in classical FEA. To convert a finite element code into a single patch isogeometric code we first of all need a different input. We do no longer take an FE mesh and nodal points as input to describe the geometry, but rather knot vectors and control points. The connectivity array that links the local shape function numbering to a global shape function numbering is also different. In isogeometric analysis, the connectivity array is calculated automatically from knot vectors and their polynomial orders, see Appendix A.2.2. Both the connectivity array and the global stiffness matrix are dependent of the basis that is being used. Thus, they will also differ from connectivity arrays and global matrices in classical FEA. Also, the routines that evaluate the basis functions and their derivatives are necessarily different when using NURBS. In addition, the output data needs to be represented differently.

In isogeometric analysis exact geometry is employed at all levels of discretization. In classical FEA on the other hand, we apply a piecewise polynomial approximation. Rather applying isogeometric analysis we obtain not only a greater accuracy of the solution, but also, we need no external descriptions of the geometry to do refinements. In both IA and FEA, the solution of the weak form is a linear combination of the basis functions. In IA, the coefficients are the control variables, while in FEA they are the nodal variables. In IA, control points and control variables are generally not interpolated, unlike the nodal points and variables in classical FEA. Both methods are isoparametric implementations of Galerkin's method [16] utilizing an element approach with basis functions having compact support. In both approaches, the bases being used forms a partition of unity, and the bandwidth of matrices corresponds to the given polynomial order and are equal. In classical FEA, nodal basis function can be positive or negative, while NURBS basis functions are only positive. In FEA, the degrees of freedom are located at the nodes, while in IA they are located at the control points. In FEA the continuity of the basis functions are fixed, while in IA we can easily control the continuity to be as desired. Table 2.2 sums up some of the differences and similarities.

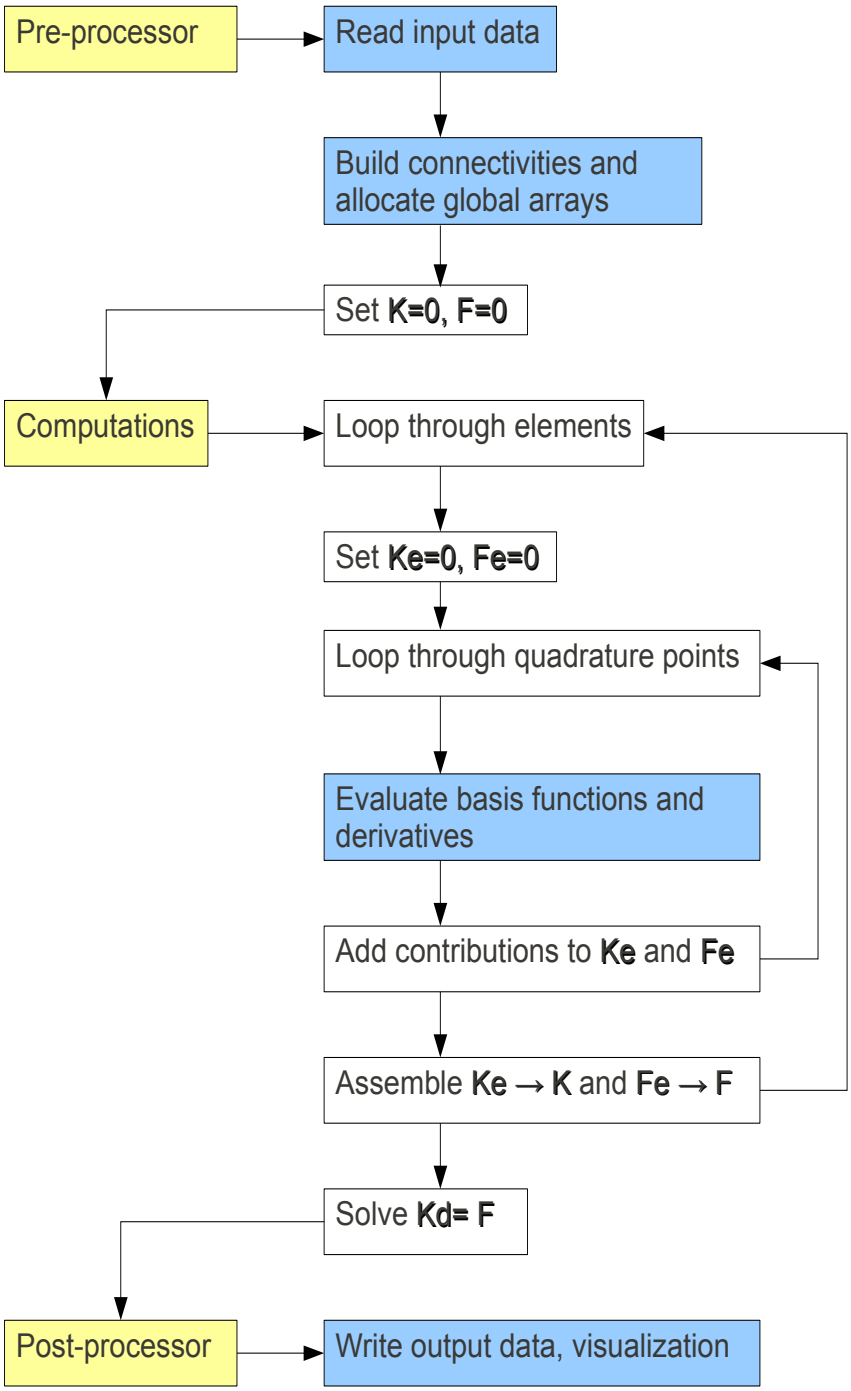


Figure 2.11: *Flow chart for one patch isogeometric analysis. The routines in blue differ from those in classical finite element analysis. All routines have been programmed in MATLAB for a 2D linear elasticity problem, and are given in detail in the appendix.*

Chapter 3

Theory of the finite element method

In this chapter we are going to look at some of the theory behind the finite element method in order to understand what a finite element is. In the first section we will refresh our knowledge by reviewing some definitions we need to be familiar with in order to establish and study the space of basis functions. Next we will consider the variational formulation, and at last we will look at what determines a finite element. As we assume the reader to be familiar with the finite element method we will not prove the applied theorems. The definitions and theorems with proves are thus to be found in [5, 32, 44].

3.1 Some definitions

Linear space

A set X is a linear space if

(i) $v_1 + v_2 \in X \quad \forall v_1, v_2 \in X$,

(ii) $\alpha v \in X \quad \forall v \in X$,$\forall \alpha \in \mathbf{R}$.

Linear functionals

L is a linear functional if $L : X \mapsto \mathbf{R}$ such that $L(\alpha v_1 + v_2) = \alpha L(v_1) + L(v_2) \quad \forall \alpha \in \mathbf{R}, \forall v_1, v_2 \in X$.

Bilinear functionals

A is a bilinear functional if $A : X \times Y \mapsto \mathbf{R}$ such that

(i) $A(u, \bar{v})$ is a linear functional in u for fixed $\bar{v}$,

(ii) $A(\bar{u}, v)$ is a linear functional in v for fixed $\bar{u}$.

Inner product space

An inner product space is a vector space V over the field $\mathbb{F}$ together with the inner product $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{F}$ that satisfies the following axioms for all vectors $x, y, z \in V$ and all scalars $a \in \mathbb{F}$:

(i) $\langle x, y\rangle = \overline{\langle y, x\rangle}$,

(ii) $\langle ax, y\rangle = a\langle x, y\rangle$,

(iii) $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$,

(iv) $\langle x, x\rangle \geq 0$ with equality only for $x = 0$.

Norm

Given a linear vector space V, a norm, $\|\cdot\|$, is a function on V with values in $\mathbf{R}^+$ with properties

(i) $\|v\| \geq 0 \quad \forall v \in V$,

(ii) $\|v\| = 0$ if and only if $v = 0$,

(iii) $\|c \cdot v\| = |c| \, \|v\| \quad \forall c \in \mathbf{R}, v \in V$,

(iv) $\|v + u\| \leq \|v\| + \|u\| \quad \forall v, u \in V$.

Metric space

A metric space is a pair (X, d), where X is a set and d is a distance function on X, i.e. $d : X \times X \to \mathbf{R}$. The distance function d must for all element $x, y, z \in X$ satisfy

(i) d is real valued, finite and non-negative,

(ii) $d(x, y) = 0$ if and only if $x = y$,

(iii) $d(x, y) = d(y, x)$,

(iv) $d(x, y) \leq d(x, z) + d(z, y)$.

Completeness

A metric space X is complete if any Cauchy sequence ($x_n \in X$ such that $\|x_n - x_m\|_X \to 0$ as $n, m \to \infty$) converges to a member of X.

Banach space

A normed linear space $(V, \|\cdot\|)$ is called a Banach space if it is complete with respect to the metric induced by the norm $\|\cdot\|$. A Banach space is a complete metric space, where the metric is induced by the norm

$$\|\cdot\|_V ; d(x, y) = \|x - y\|_V .$$

Hilbert space

A Hilbert space is a Banach space with a norm induced by the inner product of the Hilbert space, $\|x\|_H = \sqrt{(x, x)_H}$.

Bounded coercive form

A bilinear form $a(\cdot,\cdot)$ on a normed linear space H is *bounded* or *continuous*, if there exists a $C < \infty$ such that

$$|a(u,v)| \leq C \|u\| \|v\| \quad \forall u, v \in H.$$

$a(\cdot,\cdot)$ is *coercive* if $\exists\ \alpha > 0$ such that

$$a(u,u) \geq \alpha \|v\|^2 \quad \forall v \in H.$$

3.2 The Hilbert space

The Hilbert space H is a linear space with the inner product $(\cdot,\cdot)_H$ and the norm $\|u\|_H \equiv \sqrt{(u,u)}$. The Hilbert space is a complete inner product space. For $m > 0$ the Hilbert space is given by

$$H^m(\Omega) \equiv \left\{ v | \int_\Omega v^2 dA < \infty, \int_\Omega \left(\frac{dv}{dx}\right)^2 dA < \infty, \ldots, \int_\Omega \left(\frac{d^m v}{dx^m}\right)^2 dA < \infty \right\},$$

with the inner product

$$(u,v)_{H^m(\Omega)} = \sum_{j=0}^{m} \int_0^1 \frac{d^j u}{dx^j} \frac{d^j v}{dx^j} dx$$

and the norm

$$\|u\|_{H^m(\Omega)} = \left(\sum_{j=0}^{m} \int_0^1 \left(\frac{d^j u}{dx^j}\right)^2 dx \right)^{\frac{1}{2}}.$$

Given the Hilbert space H we can define a dual space H' as the space of all bounded linear functionals $L(v)$, where $L(v)$ is bounded if $L(v) \leq C \|v\|_H\ \forall v \in H$. The norm of $L(v)$ is given by

$$\|L\|_{H'} = \sup_{v \in H, v \neq 0} \frac{L(v)}{\|v\|_H}.$$

By Riesz representation theorem [5] any continuous linear functional L on a Hilbert space can be represented uniquely as

$$L(v) = (u,v)$$

for some $u \in H$. Further we have that

$$\|L\|_{H'} = \|u\|_H .$$

3.3 Formulation of the variational problem

Suppose that

(1) $(H, (\cdot,\cdot))$ is a Hilbert space.

(2) V is a closed subspace of H.

(3) $a(\cdot,\cdot)$ is a bilinear form on V, not necessarily symmetric.

(4) $a(\cdot,\cdot)$ is bounded or continuous on V.

(5) $a(\cdot,\cdot)$ is coercive on V.

The variational problem is then given as [5];
Given $l \in V'$, find $u \in V$ such that

$$a(u,v) = l(v) \quad \forall v \in V. \tag{3.1}$$

The existence and uniqueness of the solution is guaranteed by the Lax-Milgram theorem [5]; Given a Hilbert space $(V,(\cdot,\cdot))$, a continuous, coercive bilinear form $a(\cdot,\cdot)$ and a continuous linear functional $l \in V'$, there exists a unique $u \in V$ such that

$$a(u,v) = l(v) \quad \forall v \in V.$$

The Galerkin approximation can now be given as; Given finite-dimensional subspaces $V_h, V_h^D \subset V$ and $l \in V'$, find $u_h \subset V_h^D$ such that

$$a(u_h,v) = l(v) \quad \forall v \in V_h.$$

If $a(u,v)$ involves only first derivatives and we have Dirichlet boundary conditions,

$$V = \left\{ v \in H^1(\Omega) | v|_\Gamma = 0 \right\},$$
$$V^D = \left\{ u \in H^1(\Omega) | u|_\Gamma = \bar{u} \right\},$$

where $H^1(\Omega) \equiv \left\{ v | \int_\Omega v^2 dA < \infty, \int_\Omega \left(\frac{dv}{dx}\right)^2 dA < \infty, \int_\Omega \left(\frac{dv}{dy}\right)^2 dA < \infty \right\}$.

3.4 The finite element

The finite element is defined by [5]:

Let

(i) $\mathcal{K} \subseteq \mathbf{R}^n$ be a bounded closed set with nonempty interior and piecewise smooth boundary (the element domain),

(ii) $\mathcal{P}$ be a finite-dimensional space of functions $\mathcal{K}$ (the space of shape functions),

(iii) $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ be a basis for $\mathcal{P}'$ (the set of nodal variables).

Then $(\mathcal{K}, \mathcal{P}, \mathcal{N})$ is called a finite element.

When $(\mathcal{K}, \mathcal{P}, \mathcal{N})$ is a finite element then the basis $\{\phi_1, \phi_2, \ldots, \phi_k\}$ of $\mathcal{P}$ dual to $\mathcal{N}$, that is, the basis $N_i(\phi_j) = \delta_{ij}$, is the nodal basis of $\mathcal{P}$.

Lets look at the following example to better understand the definition of a finite element; Let $\mathcal{K} = [-1,1]$ and $\mathcal{P}$ be the set of all linear polynomials of degree less than or

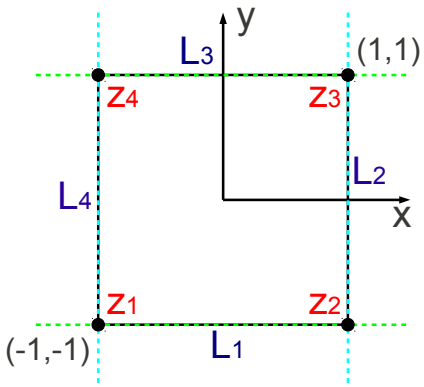


Figure 3.1: *The lines* L_1, L_2, L_3, L_4 *for the square* $[-1,1] \times [-1,1]$ *with the nodes* z_1, z_2, z_3, z_4.

equal to 1. Let $\mathcal{N} = \{N_1, N_2\}$ and $N_1(v) = v(-1)$, $N_2(v) = v(1)$ $\forall v \in \mathcal{P}$. Then $(\mathcal{K}, \mathcal{P}, \mathcal{N})$ is a finite element with the nodal basis $\phi_1(x) = \frac{1}{2}(1-x)$, $\phi_2(x) = \frac{1}{2}(1+x)$. In general in 1D; Let $\mathcal{K} = [a,b]$ and $\mathcal{P}_k$ be the set of all polynomials of degree less than or equal to k. Let $\mathcal{N} = \{N_0, N_1, \ldots, N_k\}$ and $N_i(v) = v(a + \frac{(b-a)i}{k})$ $\forall v \in \mathcal{P}_k$, $i = 0, 1, \ldots, k$. Then $(\mathcal{K}, \mathcal{P}_k, \mathcal{N}_k)$ is a finite element.

Lemma (3.1.4) in [5] says that if $\mathcal{P}$ is a k-dimensional vector space and $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ is a subset of the dual space $\mathcal{P}'$, then the two following statements are equivalent;

(i) $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ is a basis for $\mathcal{P}'$.

(ii) Given $v \in \mathcal{P}$ with $N_i v = 0$ for $i = 1, 2, \ldots, k$, then $v \equiv 0$.

Notice that $k = \dim \mathcal{P} = \dim \mathcal{P}'$ and that $\mathcal{N}$ determines $\mathcal{P}$ if $\phi \in \mathcal{P}$ with $N(\phi) = 0$ $\forall N \in \mathcal{N}$ implies that $\phi = 0$.

Assume we want to construct a finite element on the square $[-1,1] \times [-1,1]$ using linear basis functions. In the following we will refer to the hyperplane $\{x : L(x) = 0\}$, where L is a non-degenerated linear function, as L. More precisely, L is a line if we are in R^2, a plane in R^3 and a hyperplane in R^n for $n > 3$. Lemma (3.1.10) in [5] then says that if we let P be a polynomial of degree $k \geq 1$ that vanishes on a hyperplane L and Q be a polynomial of degree $(k-1)$, then $P = LQ$. We will use this to construct the finite element shown in Figure 3.1. First we let $\mathcal{K} = [-1,1] \times [-1,1]$, $\mathcal{P}$ be the set of all bilinear polynomials $\{c_0 + c_1 x + c_2 y + c_3 xy\}$ and $\mathcal{N} = \{N_1, \ldots N_4\}$ where $N_i(v) = v(z_i)$. To show that $\mathcal{N}$ determines $\mathcal{P}$ we suppose that a polynomial $P \in \mathcal{P}$ vanishes at all the nodes z_1, z_2, z_3 and z_4. We let L_1, L_2, L_3 and L_4 be non-trivial linear functions that define the edges of the square. The restriction to P to any side of the square is then a first order polynomial of one variable, and we can write $P = cL_1L_2$ for some constant c. We then have that $0 = P(z_4) = cL_1(z_4)L_2(z_4) \Rightarrow c = 0$ since $L_1(z_3) \neq 0$ and $L_2(z_3) \neq 0$. Hence, $P(x,y) \equiv 0$ on $\mathcal{K}$ and $\mathcal{N}$ determines $\mathcal{P}$. $(\mathcal{K}, \mathcal{P}, \mathcal{N})$ is thus a finite element. We can now

construct the basis functions, getting that

$$\phi_1(x,y) = cL_3(x)L_2(y) = c(1-x)(1-y) = \frac{1}{4}(1-x)(1-y),$$
$$\phi_2(x,y) = cL_3(x)L_4(y) = c(1+x)(1-y) = \frac{1}{4}(1+x)(1-y),$$
$$\phi_3(x,y) = cL_1(x)L_4(y) = c(1+x)(1+y) = \frac{1}{4}(1+x)(1+y),$$
$$\phi_4(x,y) = cL_1(x)L_2(y) = c(1-x)(1+y) = \frac{1}{4}(1-x)(1+y),$$

since we require that $\phi_1(-1,-1) = 1$, $\phi_2(1,-1) = 1$, $\phi_3(1,1) = 1$ and $\phi_4(-1,1) = 1$.

In this thesis we need to consider the definition of a finite element in an isogeometric setting, thus, we have to translate the meaning of the different terms. In isogeometric analysis the term *element* in the definition corresponds to a *patch*. Be aware that in calculations we often talk about elements referring to the knot spans, more precisely, the images of knot spans in the physical space. If we consider the definition of a finite element in an isogeometric setting, an element domain is thus referring to a patch and the nodal variables corresponds to the control variables.

We will also only consider subspaces of the Hilbert space H^1 as the space of basis functions. The control variables will thus lie in the dual space of H^1. The dimension of the basis function space must be equal to the dimension of the dual space, which again is equal to the dimension of the degrees of freedom. Hence, the number of basis functions and the number of degrees of freedom must coincide.

Chapter 4

2D Linear elasticity

In this thesis we will consider solid materials that undergoes small deformations when they are subjected to stress, and that, if the stress forces are not too large, will return to their natural shape. We will assume that we have a linear relationship between stress and strain components, and that the stress states do not produce yielding. Hence, we can apply linear elasticity theory, a simplification of the nonlinear theory of elasticity that is a branch of continuum mechanics [2]. The assumptions mentioned above are reasonable for many engineering design scenarios, which is why linear elasticity is much used in structural analysis together with the finite element method. In this chapter we will give a short introduction on principals, terms and quantities in structural mechanics related to linear elasticity problems. We will start by defining quantities like strain, stress and traction, before deriving the equilibrium equations on strong and weak form.

4.1 Strain

Strain is a dimensionless quantity describing the relative amount of deformation of a body. It is a measure of the relative displacement between particles in the material and measures how much a given displacement differs locally from a rigid-body displacement [23]. To obtain an expression for strain we start by writing the displacement vector as

$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix}.$$

Strain is the same as elongation divided by original length. Extensional strains are thus given by [37]

$$\epsilon_{xx} = \lim_{\Delta x \to 0} \frac{u_x(x + \Delta x, y) - u_x(x, y)}{\Delta x} = \frac{\partial u_x}{\partial x}$$

$$\epsilon_{yy} = \lim_{\Delta y \to 0} \frac{u_y(x, y + \Delta y) - u_y(x, y)}{\Delta y} = \frac{\partial u_y}{\partial y},$$

whereas shear strain,γ_{xy}, is given by

$$\gamma_{xy} = \frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} = \alpha_1 + \alpha_2.$$

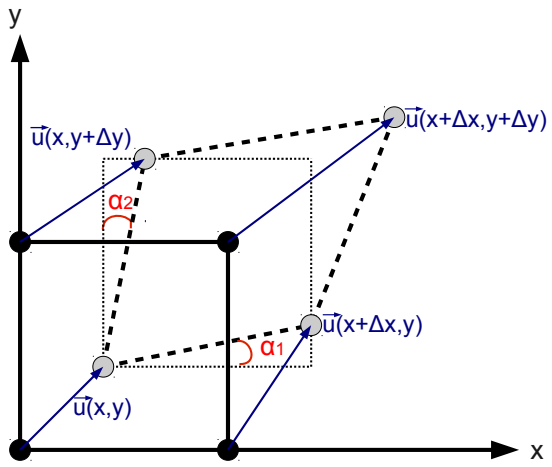


Figure 4.1: *Deformation of a control volume. The figure is reconstructed from Figure 9.1 in [11].*

Shear strain measures the change in angle between unit vectors in x- and y-directions, see Figure 4.1. The total strain can hence be expressed as

$$\epsilon = \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \nabla_S \mathbf{u}.$$

Recall that these are the linearized equations that are valid only for small deformations where higher order terms can be ignored.

4.2 Stress

Stress is a quantity measuring the strength of forces causing deformation of a body, and is defined to be force per unit area. In two dimensions stresses are forces per unit area acting on the planes normal to the x- or y-axis [11]. Stress can in vector form be expressed as

$$\sigma = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix},$$

where the first subscript denotes the direction of the normal to the plane at where the stress is acting and the second subscript denotes the direction of the force, see Figure 4.2. Notice that $\sigma_{xy} = \sigma_{yx}$.

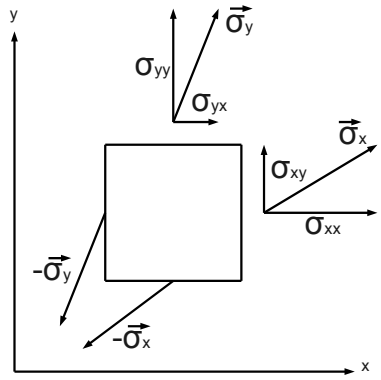


Figure 4.2: *Stress components. The figure is reconstructed from Figure 9.3 in [11].*

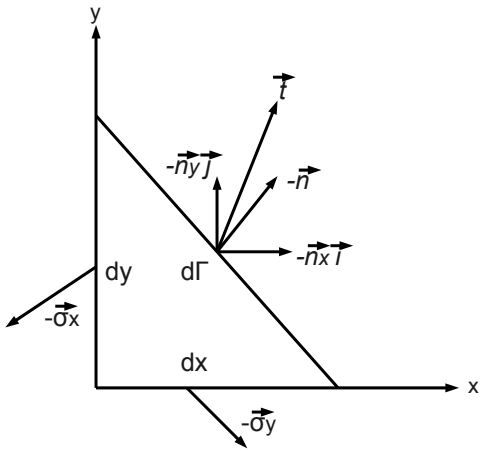


Figure 4.3: *Relationship between stress and traction. The figure is reconstructed from Figure 9.4 in [11].*

4.3 Traction

Traction is like stress also a quantity measured in force per unit area. Traction is the same as stress, only associated with a specific surface, such as the outer boundary of a domain. Stress vectors could give information about traction at a point on any surface.

Figure 4.3 shows the relationship between stress and traction. Making the forces in Figure 4.3 to be in equilibrium we must require that

$$\mathbf{t}d\Gamma - \sigma_x dy - \sigma_y dx = \mathbf{0}.$$

Using that $dy = n_x d\Gamma$, $dx = n_y d\Gamma$, dividing by $d\Gamma$ and multiplying by unit vectors gives the expressions

$$t_x = \sigma_{xx} n_x + \sigma_{xy} n_y = \sigma_x \mathbf{n},$$
$$t_y = \sigma_{xy} n_x + \sigma_{yy} n_y = \sigma_y \mathbf{n}.$$

4.4 Hooke's law for plane stress

When stress and strain are small enough they are proportional quantities [43]. The linear relationship between them is called Hooke's law and is expressed as

$$\frac{\text{Stress}}{\text{Strain}} = \text{Elastic modulus.}$$

In our case, studying linear elasticity problems, Hooke's law is given as

$$\sigma = \mathbf{D}\epsilon.$$

In two dimensions $\mathbf{D}$ will be a 3×3 matrix, looking differently depending on whether we assume plane strain or plane stress. Plane stress implies that we assume the body to be thin compared to the dimension in the xy-plane. In other words, σ_{zz} could be neglected. We will assume plane stress and isotropic material using the following expression for $\mathbf{D}$:

$$\mathbf{D} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-2\nu)/2 \end{bmatrix},$$

where E is Young's modulus and ν is Poisson's ratio. Young's modulus and Poisson's ratio are constants depending on the material properties. Notice that D is symmetric positive definite. When $\nu \neq 0$ D will couple the different directions.

4.5 Assumptions

The theory of linear elasticity is often applied in stress analysis for problems in solid mechanics. However, for the theory to be applicable some assumptions needs to be fulfilled. First of all we need the behavior of the material we are analyzing to be linear. Also, when external forces are acting on the structure, deformations must be small and no gaps or overlaps could occur. In addition, dynamic effects have to be negligible. Dynamic effects will be small if the d'Alembert forces, which are equal to the mass of the body times the acceleration, are small compared to the the loads acting on the body [11]. In order to fulfill these assumptions the following requirements must be satisfied:

(i) Deformations must be smooth.

(ii) Hooke's law must be satisfied.

(iii) The body must be in equilibrium.

Assumption (i) will be satisfied if $\epsilon = \nabla_{\mathbf{S}}\mathbf{u}$, which is referred to as the kinematic equation. Assumption (ii), Hooke's law, is given as $\sigma = \mathbf{D}\epsilon$ and is called the constitutive equation. Assumption (iii) is called the equilibrium equation and will be derived in the next section.

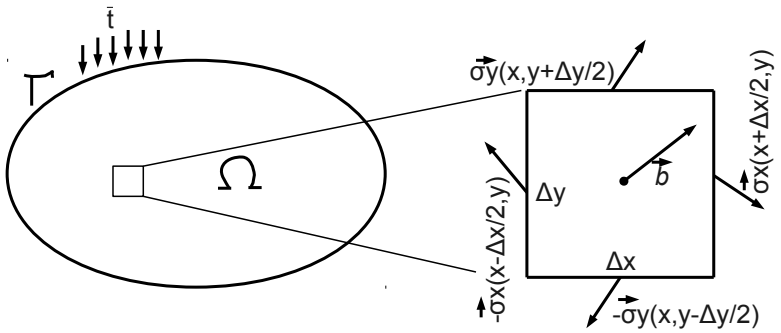


Figure 4.4: *Traction and body forces acting on a infinitesimal element. The figure is reconstructed from Figure 9.5 in [11].*

4.6 The equilibrium equation

We are considering a body where tractions, $\mathbf{t}$, are acting along the boundary Γ and body forces per unit volume, $\mathbf{b}$, are acting on the whole body. Body forces are for instance gravity forces, magnetic forces or thermal stresses. Figure 4.4 shows how stresses and body forces are acting on an inner infinitesimal element of the domain Ω. For the infinitesimal element to be in equilibrium we must require that

$$\begin{aligned} &-\sigma_x(x-\frac{\Delta x}{2},y)\Delta y+\sigma_x(x+\frac{\Delta x}{2},y)\Delta y \\ &-\sigma_y(x,y-\frac{\Delta y}{2})\Delta x+\sigma_y(x,y+\frac{\Delta y}{2})\Delta x+\mathbf{b}(x,y)\Delta x\Delta y=\mathbf{0}. \end{aligned}$$

If we divide by $\Delta x\Delta y$ and take the limit as $\Delta x \to 0$, $\Delta y \to 0$ we obtain the equation

$$\frac{\partial \sigma_x}{\partial x}+\frac{\partial \sigma_y}{\partial y}+\mathbf{b}=\mathbf{0}.$$

Inserting for σ_x and σ_y gives us in vector form

$$\nabla\cdot\sigma_x+b_x=0, \qquad \nabla\cdot\sigma_y+b_y=0.$$

The body will hence be in equilibrium if the following equation is satisfied:

$$\nabla_S^T\sigma+\mathbf{b}=\mathbf{0}.$$

4.7 Strong form

The relations for 2D linear elasticity is given by the equilibrium equation, the kinematics equation and the constitutive equation. Two types of boundary conditions will be considered, prescribed traction and prescribed displacement. Prescribed traction is written as

$$\sigma_x\cdot\mathbf{n}=\bar{t}_x \quad \text{and} \quad \sigma_y\cdot\mathbf{n}=\bar{t}_y \quad \text{on} \quad \Gamma_t, \tag{4.1}$$

whereas prescribed displacement is written as

$$\mathbf{u}=\bar{\mathbf{u}} \quad \text{on} \quad \Gamma_u.$$

We have that $\Gamma_u \cup \Gamma_t = \Gamma$ and that $\Gamma_u \cap \Gamma_t = 0$. The strong form for linear elasticity is hence formed by the three equations for the boundary conditions together with the three following equations

$$\nabla \cdot \sigma_x + b_x = 0, \quad \nabla \cdot \sigma_y + b_y = 0 \tag{4.2}$$

and

$$\sigma = \mathbf{D}\nabla_{\mathbf{S}}\mathbf{u}.$$

4.8 Weak form

Let $\mathbf{w} = \begin{bmatrix} w_x & w_y \end{bmatrix}^T$ be weight functions such that $\mathbf{w} = \mathbf{0}$ on Γ_u and let the trial solutions be such that $\mathbf{u} = \bar{\mathbf{u}}$ on Γ_u. To obtain the weak form we first multiplying equation (4.1) and equation (4.2) by admissible weight functions and integrate over the domain Ω. We then get

$$\int_\Omega w_x \nabla \cdot \sigma_x d\Omega + \int_\Omega w_x b_x d\Omega = 0 \quad \forall w_x \in U_0, \tag{4.3}$$

$$\int_\Omega w_y \nabla \cdot \sigma_y d\Omega + \int_\Omega w_y b_y d\Omega = 0 \quad \forall w_y \in U_0, \tag{4.4}$$

$$\int_{\Gamma_t} w_x (\bar{t_x} - \sigma_x \cdot \mathbf{n}) d\Gamma = 0 \quad \forall w_x \in U_0 \quad \text{and} \tag{4.5}$$

$$\int_{\Gamma_t} w_y (\bar{t_y} - \sigma_y \cdot \mathbf{n}) d\Gamma = 0 \quad \forall w_y \in U_0, \tag{4.6}$$

where $U_0 = \{\mathbf{w} | \mathbf{w} \in H^1, \mathbf{w} = \mathbf{0} \quad \text{on} \quad \Gamma_u\}$. Applying Green's theorem to the first terms in equation (4.3) and equation (4.4) before adding the two equations yields

$$\int_\Omega (\nabla w_x \cdot \sigma_x + \nabla w_y \cdot \sigma_y) d\Omega = \oint_{\Gamma_t} (w_x \sigma_x \cdot \mathbf{n} + w_y \sigma_y \cdot \mathbf{n}) d\Gamma + \int_\Omega (w_x b_x + w_y b_y) d\Omega, \tag{4.7}$$

since w_x and w_y vanish on Γ_u. Substituting equation (4.5) and equation (4.6) into equation (4.7), writing in vector form and using that $\nabla w_x \cdot \sigma_x + \nabla w_y \cdot \sigma_y = (\nabla_S \mathbf{w})^T \sigma$ yields

$$\int_\Omega (\nabla_S \mathbf{w})^T \sigma d\Omega = \int_{\Gamma_t} \mathbf{w}^T \bar{\mathbf{t}} d\Gamma + \int_\Omega \mathbf{w}^T \mathbf{b} d\Omega \qquad \forall \mathbf{w} \in U_0.$$

Using that $\sigma = \mathbf{D}\nabla_{\mathbf{S}}\mathbf{u}$ we can write the weak form as:

Weak form:
Find $\mathbf{u} \in U$ such that

$$a(u, w) = l(w) \qquad \forall \mathbf{w} \in U_0$$

where

$$\begin{aligned} a(u, w) &= \int_\Omega (\nabla_{\mathbf{S}} \mathbf{w})^T \mathbf{D} \nabla_{\mathbf{S}} \mathbf{u} d\Omega, \\ l(w) &= \int_{\Gamma_t} \mathbf{w}^T \bar{\mathbf{t}} d\Gamma + \int_\Omega \mathbf{w}^T \mathbf{b} d\Omega, \\ U &= \{\mathbf{u} | \mathbf{u} \in H^1, \mathbf{u} = \bar{\mathbf{u}} \quad \text{on} \quad \Gamma_u\}, \\ U^0 &= \{\mathbf{w} | \mathbf{w} \in H^1, \mathbf{w} = \mathbf{0} \quad \text{on} \quad \Gamma_u\}. \end{aligned} \tag{4.8}$$

Chapter 5

Isogeometric linear elasticity problems

5.1 The finite element discretization

We consider the domain Ω with boundary Γ in the physical space, the domain $\tilde{\Omega}$ in the parameter space and the domain $\hat{\Omega}$ in the parent element, see Figure 5.1. The mapping $\mathbf{x} : \tilde{\Omega} \mapsto \Omega$ is the geometrical mapping that maps values in the parameter domain to values in the physical domain. When we are talking about elements in this setting we are referring to the images of knot spans under that mapping. We denote the knot spans by $\tilde{\Omega}^e$ in the parameter space and Ω^e in the physical space. e runs from $1, \ldots, n_{el}$, where n_{el} denotes the total number of elements. We are counting all knot spans independently of its measure. That is, we also count knot spans that have zero measure because of knots with multiplicity greater than one. We can thus say that we are considering the elements in the index space.

We let n_{np} be the total number of basis functions, also corresponding to the total number of control points. $\mathbf{R}$ are the shape functions and $\mathbf{B}^{cp} = \begin{bmatrix} \mathbf{B}_x^{cp} & \mathbf{B}_y^{cp} \end{bmatrix}$ the control points. n_{en} is the number of basis functions that have support on the current element.

The mapping $\mathbf{x}$ is given by

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n_{np}} R_i(\xi,\eta) B_{x_i}^{cp} \\ \sum_{i=1}^{n_{np}} R_i(\xi,\eta) B_{y_i}^{cp} \end{bmatrix}$$

We express the trial solution, $\mathbf{u}(x,y)$, the weight functions, $\mathbf{w}(x,y)$, and the body force, $\mathbf{b}(x,y)$, as

$$\begin{aligned} \mathbf{u}(x,y) &= \mathbf{R}(\xi,\eta)\mathbf{d} \quad (x,y) \in \Omega, \\ \mathbf{w}(x,y) &= \mathbf{R}(\xi,\eta)\mathbf{w} \quad (x,y) \in \Omega, \\ \mathbf{b}(x,y) &= \mathbf{R}(\xi,\eta)\mathbf{b} \quad (x,y) \in \Omega, \end{aligned}$$

where

$$\mathbf{d} = \begin{bmatrix} u_{x1} & u_{y1} & u_{x2} & u_{y2} & \ldots & u_{xn_{np}} & u_{yn_{np}} \end{bmatrix}^T$$

are the displacements at the control points and

$$\mathbf{w} = \begin{bmatrix} w_{x1} & w_{y1} & w_{x2} & w_{y2} & \ldots & w_{xn_{np}} & w_{yn_{np}} \end{bmatrix}^T$$

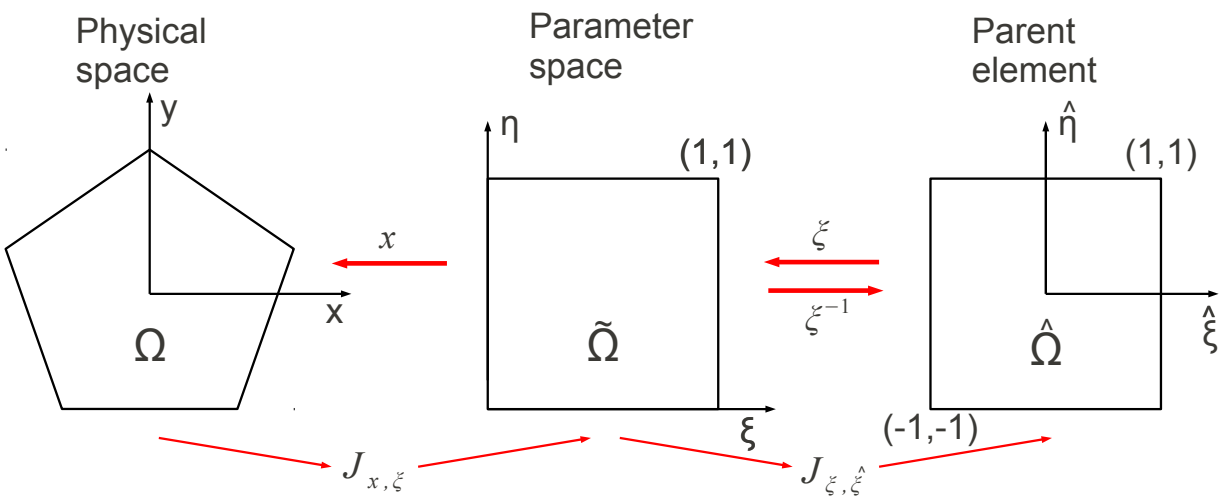


Figure 5.1: *Mapping between the physical space, the parameter space and the parent element.*

are the values of the weight functions at the control points. Notice that we have two degrees of freedom at each control point, one for the x-direction and one for the y-direction.

In the weak form we compute the integral over the domain, Ω, as a sum of integrals computed over each element domain, Ω^e;

$$\sum_{e=1}^{nel} \left\{ \int_{\Omega^e} (\nabla_{\mathbf{S}} \mathbf{w}^e)^T \mathbf{D}^e \nabla_{\mathbf{S}} \mathbf{u}^e d\Omega - \int_{\Gamma_{t^e}} \mathbf{w}^{eT} \bar{\mathbf{t}} d\Gamma - \int_{\Omega^e} \mathbf{w}^{eT} \mathbf{b} d\Omega \right\} = 0. \tag{5.1}$$

Now let

$$\begin{aligned} \mathbf{u}^e(x,y) &= \mathbf{R}^e(\xi,\eta)\mathbf{d}^e \quad (x,y) \in \Omega^e, \\ \mathbf{w}^e(x,y) &= \mathbf{R}^e(\xi,\eta)\mathbf{w}^e \quad (x,y) \in \Omega^e, \\ \mathbf{b}^e(x,y) &= \mathbf{R}^e(\xi,\eta)\mathbf{b}^e \quad (x,y) \in \Omega^e, \end{aligned}$$

where

$$\begin{aligned} \mathbf{d}^e &= \left[\begin{array}{cccccc} u^e_{x1} & u^e_{y1} & u^e_{x2} & u^e_{y2} & \dots & u^e_{x_{nen}} & u_{y^e_{nen}} \end{array} \right]^T, \\ \mathbf{w}^e &= \left[\begin{array}{cccccc} w^e_{x1} & w^e_{y1} & w^e_{x2} & w^e_{y2} & \dots & w^e_{x_{nen}} & w_{y^e_{nen}} \end{array} \right]^T, \\ \mathbf{b}^e &= \left[\begin{array}{cccccc} b^e_{x1} & b^e_{y1} & b^e_{x2} & b^e_{y2} & \dots & b^e_{x_{nen}} & b_{y^e_{nen}} \end{array} \right]^T. \end{aligned}$$

The notation $(*)^e_{x1}$ refers to local control point number 1 in element e in the x-direction. The element shape function matrix, $\mathbf{R}^e$, is given by

$$\mathbf{R}^e = \left[\begin{array}{cccccc} R^e_1 & 0 & R^e_2 & 0 & \dots & R^e_{n_{en}} & 0 \\ 0 & R^e_1 & 0 & R^e_2 & \dots & 0 & R^e_{n_{en}} \end{array} \right].$$

We then express the strain as

$$\epsilon^e = \left[\begin{array}{c} \epsilon^e_{xx} \\ \epsilon^e_{yy} \\ \gamma^e_{xy} \end{array} \right] = \nabla_{\mathbf{S}} \mathbf{u}^e = \nabla_{\mathbf{S}} \mathbf{R}^e \mathbf{d}^e = \mathbf{B}^e \mathbf{d}^e,$$

where

$$\mathbf{B}^e = \nabla_{\mathbf{S}}\mathbf{R}^e = \begin{bmatrix} \frac{\partial R_1^e}{\partial x} & 0 & \frac{\partial R_2^e}{\partial x} & 0 & \cdots & \frac{\partial R_{n_{en}}^e}{\partial x} & 0 \\ 0 & \frac{\partial R_1^e}{\partial y} & 0 & \frac{\partial R_2^e}{\partial y} & \cdots & 0 & \frac{\partial R_{n_{en}}^e}{\partial y} \\ \frac{\partial R_1^e}{\partial y} & \frac{\partial R_1^e}{\partial x} & \frac{\partial R_2^e}{\partial y} & \frac{\partial R_2^e}{\partial x} & \cdots & \frac{\partial R_{n_{en}}^e}{\partial y} & \frac{\partial R_{n_{en}}^e}{\partial x} \end{bmatrix}.$$

Inserting the above into equation (5.1) we obtain

$$\sum_{e=1}^{nel} \left\{ \int_{\Omega^e} \mathbf{w}^{eT}\mathbf{B}^{eT}\mathbf{D}^e\mathbf{B}^e\mathbf{d}^e d\Omega - \int_{\Gamma_{t^e}} \mathbf{w}^{eT}\mathbf{R}^{eT}\bar{\mathbf{t}} d\Gamma - \int_{\Omega^e} \mathbf{w}^{eT}\mathbf{R}^{eT}\mathbf{R}^e\mathbf{b}^e d\Omega \right\} = 0$$

If we let

$$\mathbf{K}^e = \int_{\Omega^e} \mathbf{B}^{eT}\mathbf{D}^e\mathbf{B}^e d\Omega$$

be the element stiffness matrix and

$$\mathbf{f}^e = \int_{\Gamma_{t^e}} \mathbf{R}^{eT}\bar{\mathbf{t}} d\Gamma + \int_{\Omega^e} \mathbf{R}^{eT}\mathbf{R}^e\mathbf{b}^e d\Omega$$

be the external force matrix we get that

$$\sum_{e=1}^{nel} \mathbf{w}^{eT} \left\{ \mathbf{K}^e\mathbf{d}^e - \mathbf{f}^e \right\} = 0.$$

Let $\mathbf{K}$ be the global stiffness matrix and $\mathbf{f}$ be the global force matrix. Assembling the system and using that $\mathbf{w}$ is arbitrary make our discrete problem look like

$$\mathbf{Kd} = \mathbf{f}.$$

5.2 Solving the discrete problem

In order to calculate the stiffness matrix and force matrix we will approximate the integrals utilizing Gaussian quadrature. To do that we map the parameter space and the physical space to the parent element, cf. Figure 2.10 and Figure 5.1, where we perform the integration. Assume we have the knot vectors $\Xi = \{\xi_1, \xi_2, ..., \xi_{n+p+1}\}$ and $\mathcal{H} = \{\eta_1, \eta_2, ..., \eta_{m+q+1}\}$. The mapping $\xi : \hat{\Omega} \mapsto \tilde{\Omega}$ from the parent element to the parameter space is thus given by

$$\xi = \begin{bmatrix} \xi(\hat{\xi}) \\ \eta(\hat{\eta}) \end{bmatrix} = \begin{bmatrix} \frac{(\xi_{i+1}-\xi_i)\hat{\xi}+(\xi_{i+1}+\xi_i)}{2} \\ \frac{(\eta_{i+1}-\eta_i)\hat{\eta}+(\eta_{i+1}+\eta_i)}{2} \end{bmatrix}. \tag{5.2}$$

The Jacobian matrix taking us from the parameter space to the parent element is given by

$$\mathbf{J}_{\xi,\hat{\xi}}(\hat{\xi},\hat{\eta}) = \begin{bmatrix} \frac{\partial\xi}{\partial\hat{\xi}} & \frac{\partial\xi}{\partial\hat{\eta}} \\ \frac{\partial\eta}{\partial\hat{\xi}} & \frac{\partial\eta}{\partial\hat{\eta}} \end{bmatrix} = \begin{bmatrix} \frac{(\xi_{i+1}-\xi_i)\hat{\xi}}{2} & 0 \\ 0 & \frac{(\eta_{i+1}-\eta_i)\hat{\eta}}{2} \end{bmatrix}.$$

We only have an explicit expression for the basis functions $R_i(\xi,\eta)$ in terms of ξ and η. However, to be able to calculate $\mathbf{B}$ we need to know ∇R_i, the derivatives with respect to

x and y. To find the expression for ∇R_i we will first look at the mapping $\mathbf{x} : \tilde{\Omega} \mapsto \Omega$ from the parameter space to the physical space. The mapping $\mathbf{x}$ have the Jacobian matrix

$$\mathbf{J}_{\mathbf{x},\xi} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix},$$

with the inverse

$$(\mathbf{J}_{\mathbf{x},\xi})^{-1} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix}.$$

We now let

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix}, \quad \hat{\nabla} = \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix}.$$

Hence we have that

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = (\mathbf{J}_{\mathbf{x},\xi})^{-T} \hat{\nabla}.$$

Defining $\mathbf{G} = (\mathbf{J}_{\mathbf{x},\xi})^{-T}$ we get that

$$\nabla R_i(x,y) = \begin{bmatrix} \frac{\partial R_i}{\partial x} \\ \frac{\partial R_i}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial R_i(\xi,\eta)}{\partial \xi} \\ \frac{\partial R_i(\xi,\eta)}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} R_i(\xi,\eta) = \mathbf{G}\hat{\nabla} R_i(\xi,\eta).$$

We are thus able to calculate $\mathbf{B}$, as we already know the expression for $\hat{\nabla} R_i(\xi,\eta)$, cf. Chapter 2.2.3.

To map Ω to $\hat{\Omega}$ to perform the integration we need to know the Jacobian $\mathbf{J}_{\mathbf{x},\hat{\xi}}$. We observe that

$$\mathbf{J}_{\mathbf{x},\hat{\xi}} = \begin{bmatrix} \frac{\partial x}{\partial \hat{\xi}} & \frac{\partial x}{\partial \hat{\eta}} \\ \frac{\partial y}{\partial \hat{\xi}} & \frac{\partial y}{\partial \hat{\eta}} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial \xi}{\partial \hat{\xi}} & \frac{\partial \xi}{\partial \hat{\eta}} \\ \frac{\partial \eta}{\partial \hat{\xi}} & \frac{\partial \eta}{\partial \hat{\eta}} \end{bmatrix} = \mathbf{J}_{\mathbf{x},\xi}\mathbf{J}_{\xi,\hat{\xi}}.$$

We can thus express the stiffness matrix as

$$\begin{aligned} \mathbf{K}^e &= \int_{\Omega^e} \mathbf{B}^{eT}\mathbf{D}^e\mathbf{B}^e d\Omega = \int_{-1}^{1}\int_{-1}^{1} \mathbf{B}^{eT}\mathbf{D}^e\mathbf{B}^e |\mathbf{J}_{\mathbf{x},\hat{\xi}}{}^e| d\hat{\xi} d\hat{\eta} \\ &= \sum_{i=1}^{n_{gp}}\sum_{j=1}^{n_{gp}} \mathbf{B}^{eT}(\xi(\hat{\xi}_i),\eta(\hat{\eta}_j))\mathbf{D}^e\mathbf{B}^e(\xi(\hat{\xi}_i),\eta(\hat{\eta}_j))|\mathbf{J}_{\mathbf{x},\hat{\xi}}{}^e(\hat{\xi}_i,\hat{\eta}_j)|\rho_i\rho_j, \end{aligned}$$

where n_{gp} is the number of gauss points and ρ_i are the corresponding weights. Recall the force matrix

$$\mathbf{f}^e = \int_{\Gamma_{t^e}} \mathbf{R}^{eT}\overline{\mathbf{t}} d\Gamma + \int_{\Omega^e} \mathbf{R}^{eT}\mathbf{R}^e\mathbf{b}^e d\Omega = \mathbf{f}^e_\Gamma + \mathbf{f}^e_\Omega.$$

We calculate the element body force matrix in the following way

$$\begin{aligned} \mathbf{f}^e_\Omega &= \int_{\Omega^e} \mathbf{R}^{eT}\mathbf{R}^e\mathbf{b}^e d\Omega = \int_{-1}^{1}\int_{-1}^{1} \mathbf{R}^{eT}\mathbf{R}^e\mathbf{b}^e |\mathbf{J}_{\mathbf{x},\hat{\xi}}{}^e| d\hat{\xi} d\hat{\eta} \\ &= \sum_{i=1}^{n_{gp}}\sum_{j=1}^{n_{gp}} \mathbf{R}^{eT}(\xi(\hat{\xi}_i),\eta(\hat{\eta}_j))\mathbf{R}^e(\xi(\hat{\xi}_i),\eta(\hat{\eta}_j))\mathbf{b}^e|\mathbf{J}_{\mathbf{x},\hat{\xi}}{}^e(\hat{\xi}_i,\hat{\eta}_j)|\rho_i\rho_j. \end{aligned}$$

Assume prescribed traction at the physical domain at the boundary Γ. The boundary force vector is then calculated as either

$$\begin{aligned}\mathbf{f}_\Gamma^e = \int_{\Gamma_{t^e}} \mathbf{R}^{eT}\bar{\mathbf{t}}d\Gamma &= \int_{-1}^{1} \mathbf{R}^{eT}(\xi(\hat{\xi}=a),\eta(\hat{\eta}))\bar{\mathbf{t}}|\mathbf{J}_{\mathbf{x},\hat{\xi}}{}^e|d\hat{\eta} \\ &= \sum_{j=1}^{n_{gp}} \mathbf{R}^{eT}(\xi(\hat{\xi}=a),\eta(\hat{\eta}_j))\bar{\mathbf{t}}|\mathbf{J}_{\mathbf{x},\hat{\xi}}{}^e(\hat{\xi}=a,\hat{\eta}_j)|\rho_j,\end{aligned}$$

for the part of the boundary we ξ is constant, or

$$\begin{aligned}\mathbf{f}_\Gamma^e = \int_{\Gamma_{t^e}} \mathbf{R}^{eT}\bar{\mathbf{t}}d\Gamma &= \int_{-1}^{1} \mathbf{R}^{eT}(\xi(\hat{\xi}),\eta(\hat{\eta}=b))\bar{\mathbf{t}}|\mathbf{J}_{\mathbf{x},\hat{\xi}}{}^e|d\hat{\xi} \\ &= \sum_{i=1}^{n_{gp}} \mathbf{R}^{eT}(\xi(\hat{\xi}_i),\eta(\hat{\eta}=b))\bar{\mathbf{t}}|\mathbf{J}_{\mathbf{x},\hat{\xi}}{}^e(\hat{\xi}_i,\hat{\eta}=b)|\rho_i\end{aligned}$$

for the part of the boundary we η is constant.

After assembling all contributions to $\mathbf{K}$ and $\mathbf{f}$ we can solve with respect to $\mathbf{d}$ and further calculate strain and stresses. Recall that $\epsilon^e = \mathbf{B}^e\mathbf{d}^e$ and $\sigma^e = \mathbf{D}^e\epsilon^e = \mathbf{D}^e\mathbf{B}^e\mathbf{d}^e$. Details on how this is programmed in MATLAB can be found in the appendix.

Chapter 6

Degenerated mappings

During analysis we have experienced some difficulties with non-physical singularities, that is, singular points existing in the mapping but not in the actual physical problem. Because of this we want to study degenerated mappings in hope of finding a way to deal with this issue. We will look at some different degenerated mappings, starting by looking at a triangle.

6.1 Triangle, degenerated quadrilateral

In this chapter we are going to study what happens when a quadrilateral is degenerated to a triangle. Consider the quadrilateral shown in Figure 6.1. Let the length of one of the sides decrease to a length ϵ. We are going to look at mappings and basis functions when $\epsilon \to 0$. We will also consider what happens when the side is completely degenerated to a point, that is, when $\epsilon = 0$.

6.1.1 Bilinear Lagrange basis functions

We will first take a look at what is happening with Lagrange basis functions during such a degeneration in order to gain some experience on the issue and to establish a hypothesis on how B-spline basis function will behave. Thereafter, with the behavior of Lagrange basis functions established, we will consider the same case exploiting B-spline basis function in isogeometric analysis.

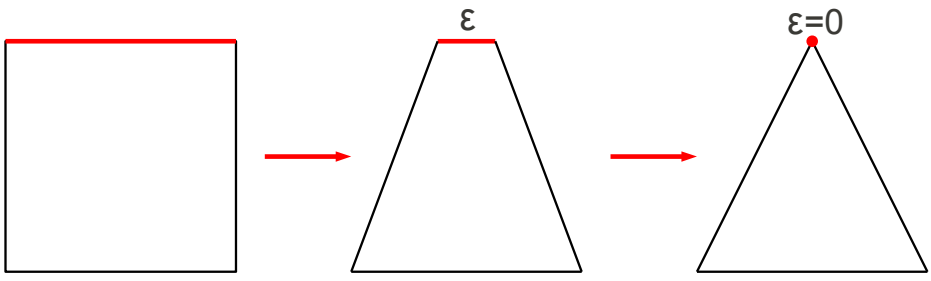


Figure 6.1: *Degenerating a quadrilateral to a triangle.*

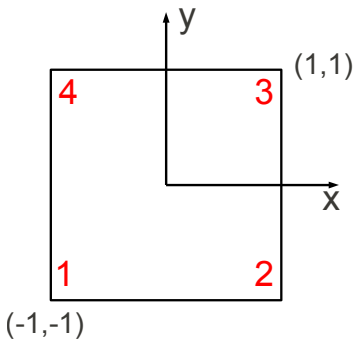


Figure 6.2: *Illustration of the quadrilateral showing the numbering of the basis functions.*

Table 6.1: Derivatives of ϕ with respect to x and y.

$\frac{d\phi_1}{dx} = -\frac{1}{4}(1-y)$	$\frac{d\phi_1}{dy} = -\frac{1}{4}(1-x)$
$\frac{d\phi_2}{dx} = \frac{1}{4}(1-y)$	$\frac{d\phi_2}{dy} = -\frac{1}{4}(1+x)$
$\frac{d\phi_3}{dx} = \frac{1}{4}(1+y)$	$\frac{d\phi_3}{dy} = \frac{1}{4}(1+x)$
$\frac{d\phi_4}{dx} = -\frac{1}{4}(1+y)$	$\frac{d\phi_4}{dy} = \frac{1}{4}(1-x)$

Consider the quadrilateral shown in Figure 6.2 with the basis functions

$$\phi_1(x,y) = \frac{1}{4}(1-x)(1-y),$$
$$\phi_2(x,y) = \frac{1}{4}(1+x)(1-y),$$
$$\phi_3(x,y) = \frac{1}{4}(1+x)(1+y),$$
$$\phi_4(x,y) = \frac{1}{4}(1-x)(1+y),$$

and the corresponding derivatives that are given in Table 6.1. This is the same quadrilateral as we looked at in chapter 3.4, when we constructed a finite element. We now let the top side of the quadrilateral collapse to a point, forming the triangle shown in Figure 6.3. The basis functions of the triangle are given by

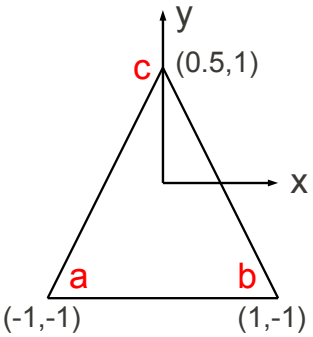


Figure 6.3: *Triangle, degenerated quadrilateral. The numbering of the basis functions is shown by red letters.*

Table 6.2: Derivatives of ψ with respect to x and y.

$\frac{d\psi_a}{dx} = -\frac{1}{4}(1-y)$	$\frac{d\psi_a}{dy} = -\frac{1}{4}(1-x)$
$\frac{d\psi_b}{dx} = \frac{1}{4}(1-y)$	$\frac{d\psi_b}{dy} = -\frac{1}{4}(1+x)$
$\frac{d\psi_c}{dx} = 0$	$\frac{d\psi_c}{dy} = \frac{1}{2}$

$$\psi_a(x,y) = \frac{1}{4}(1-x)(1-y),$$
$$\psi_b(x,y) = \frac{1}{4}(1+x)(1-y),$$
$$\psi_c(x,y) = \frac{1}{2}(1+y),$$

and the derivatives are given in Table 6.2. We observe that if we add the two basis functions of the quadrilateral at the degenerated side, ϕ_3 and ϕ_4, we obtain the basis function of the triangle at the degenerated point;

$$\psi_c(x,y) = \phi_3(x,y) + \phi_4(x,y) = \frac{1}{4}(1+x)(1+y) + \frac{1}{4}(1-x)(1+y) = \frac{1}{2}(1+y)$$

Also adding the derivatives result in the same derivatives as those of the triangle;

$$\frac{d\psi_c(x,y)}{dx} = \frac{d\phi_3}{dx} + \frac{d\phi_4}{dx} = \frac{1}{4}(1+y) - \frac{1}{4}(1+y) = 0$$
$$\frac{d\psi_c(x,y)}{dy} = \frac{d\phi_3}{dy} + \frac{d\phi_4}{dy} = \frac{1}{4}(1+x) + \frac{1}{4}(1-x) = \frac{1}{2}$$

We thus see that the set of basis functions should be reduced when we have a degeneracy, and that the new set of basis functions is formed by adding the basis functions spanning the line that has been degenerated to a point. Recall the definition of a finite element given in Chapter 3.4 and that the dimension of the space of shape functions must coincide with the number of degrees of freedom. The original quadrilateral has four basis functions and four degrees of freedom, while the degenerated quadrilateral have four basis functions and only three degrees of freedom. The degenerated quadrilateral is thus contradicting the definition of a finite element, and can consequently not be defined as a proper finite element. If we consider the basis functions of the triangle on the other hand, ψ_a, ψ_b and ψ_c, we will have three basis functions and three degrees of freedom. The triangle could hence be considered a proper finite element. Our hypothesis is thus that we need to redefine the set of basis functions when the mapping is degenerated, and that the new set of basis functions is formed by adding the basis functions corresponding to the degeneracy.

6.1.2 Bilinear B-spline basis functions

With the Lagrange basis function fresh in mind, we will now study the same case with B-spline basis function. Our hypothesis is that a similar behavior is expected from B-spline basis functions. The triangle in Figure 6.4 with $\epsilon \to 0$ is modeled by the knot vectors

$$\Xi = \{0,0,1,1\},$$
$$\mathcal{H} = \{0,0,1,1\},$$

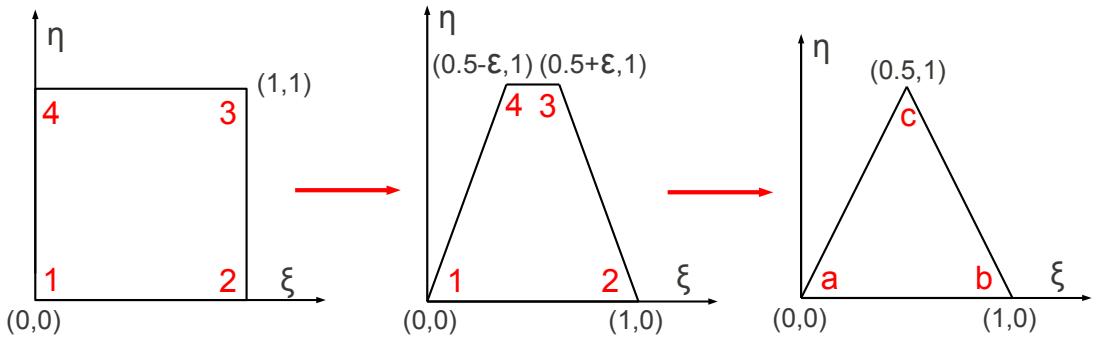


Figure 6.4: *Degenerating a quadrilateral; geometry and numbering of basis functions.*

Table 6.3: Control net $\mathbf{B}_{i,j}$

(i,j)	Control point $\mathbf{B}_{i,j}$	Weight w_{ij}	(ξ, η)
(1,1)	(0,0)	1	(0,0)
(2,1)	(1,0)	1	(1,0)
(1,2)	$(\frac{1}{2} - \epsilon, 1)$	1	(0,1)
(2,2)	$(\frac{1}{2} + \epsilon, 1)$	1	(1,1)

and the control net and weights given in Table 6.3. The B-spline basis functions are for this geometry given by

$$\begin{aligned} R_1 &= (\eta - 1)(\xi - 1), \\ R_2 &= -\xi(\eta - 1), \\ R_3 &= -\eta(\xi - 1), \\ R_4 &= \eta\xi, \end{aligned}$$

in the parameter space. The derivatives with respect to ξ and η are given in Table 6.4.

We have that

$$\frac{\mathbf{dx}}{\mathbf{d}\xi} = \begin{bmatrix} \frac{dx}{d\xi} & \frac{dx}{d\eta} \\ \frac{dy}{d\xi} & \frac{dy}{d\eta} \end{bmatrix} = \begin{bmatrix} 2\epsilon\eta - \eta + 1 & \frac{(2\epsilon-1)(2\xi-1)}{2} \\ 0 & 1 \end{bmatrix},$$

resulting in the Jacobian

$$J_{\mathbf{x},\xi} = \left| \frac{\mathbf{dx}}{\mathbf{d}\xi} \right| = 2\epsilon\eta - \eta + 1.$$

When $\epsilon = 0$ and $\eta = 1$, $J_{\mathbf{x},\xi}$ is zero. Consequently, the mapping will be singular at the point c corresponding to the degenerated line.

Recall that

$$\nabla R_i(x, y) = G\hat{\nabla} R_i(\xi, \eta) = (\mathbf{J}_{\mathbf{x},\xi})^{-T} \hat{\nabla} R_i(\xi, \eta).$$

Table 6.4: Derivatives with respect to ξ and η.

$\frac{dR_1}{d\xi} = \eta - 1$	$\frac{dR_1}{d\eta} = \xi - 1$
$\frac{dR_2}{d\xi} = 1 - \eta$	$\frac{dR_2}{d\eta} = -\xi$
$\frac{dR_3}{d\xi} = -\eta$	$\frac{dR_3}{d\eta} = 1 - \xi$
$\frac{dR_4}{d\xi} = \eta$	$\frac{dR_4}{d\eta} = \xi$

Table 6.5: Derivatives with respect to x and y

$\frac{dR_1}{dx} = \frac{\eta-1}{2\epsilon\eta-\eta+1}$	$\frac{dR_1}{dy} = \frac{\epsilon(2\xi-1)}{\eta(2\epsilon-1)+1} - \frac{2\epsilon-1}{4\epsilon-2}$
$\frac{dR_2}{dx} = -\frac{\eta-1}{2\epsilon\eta-\eta+1}$	$\frac{dR_2}{dy} = -\frac{2\epsilon-1}{4\epsilon-2} - \frac{\epsilon(2\xi-1)}{\eta(2\epsilon-1)+1}$
$\frac{dR_3}{dx} = -\frac{\eta}{2\epsilon\eta-\eta+1}$	$\frac{dR_3}{dy} = \frac{1}{2} - \frac{\xi-\frac{1}{2}}{2\epsilon\eta-\eta+1}$
$\frac{dR_4}{dx} = \frac{\eta}{2\epsilon\eta-\eta+1}$	$\frac{dR_4}{dy} = \frac{\xi-\frac{1}{2}}{2\epsilon\eta-\eta+1} + \frac{1}{2}$

To find the Jacobian matrix $\left(\mathbf{J}_{\mathbf{x},\xi}\right)^{-T}$ we first use that

$$\frac{\mathbf{d}\xi}{\mathbf{dx}} = \left(\frac{\mathbf{dx}}{\mathbf{d}\xi}\right)^{-1} = \begin{bmatrix} \frac{d\xi}{dx} & \frac{d\xi}{dy} \\ \frac{d\eta}{dx} & \frac{d\eta}{dy} \end{bmatrix} = \begin{bmatrix} \frac{1}{2\epsilon\eta-\eta+1} & -\frac{(2\epsilon-1)(2\xi-1)}{2(2\epsilon\eta-\eta+1)} \\ 0 & 1 \end{bmatrix}.$$

$\left(\mathbf{J}_{\mathbf{x},\xi}\right)^{-T}$ is thus given by

$$\left(\mathbf{J}_{\mathbf{x},\xi}\right)^{-T} = \begin{bmatrix} \frac{1}{2\epsilon\eta-\eta+1} & 0 \\ -\frac{(2\epsilon-1)(2\xi-1)}{2(2\epsilon\eta-\eta+1)} & 1 \end{bmatrix},$$

yielding the derivatives with respect to x and y as shown in Table 6.5. We see that when $\eta = 1$, $G \to \pm\infty$ as $\epsilon \to 0$. When $\epsilon = 0$ G is not defined. The derivatives with respect to x and y will in consequence either not be defined.

We will do further analysis of the derivatives. Consider the derivatives with respect to x and y when $\epsilon = 0$;

$$\begin{aligned} \frac{dR_1}{dx} &= -1, & \frac{dR_1}{dy} &= -\frac{1}{2}, \\ \frac{dR_2}{dx} &= 1, & \frac{dR_2}{dy} &= -\frac{1}{2}, \\ \frac{dR_3}{dx} &= -\frac{\eta}{-\eta+1}, & \frac{dR_3}{dy} &= \frac{1}{2} - \frac{\xi-\frac{1}{2}}{-\eta+1}, \\ \frac{dR_4}{dx} &= \frac{\eta}{-\eta+1}, & \frac{dR_4}{dy} &= \frac{\xi-\frac{1}{2}}{-\eta+1} + \frac{1}{2}. \end{aligned}$$

We observe that $\frac{dR_1}{dx}$, $\frac{dR_1}{dy}$, $\frac{dR_2}{dx}$ and $\frac{dR_2}{dy}$ are in H^1. Now we want to find out whether or not $\frac{dR_3}{dx}$, $\frac{dR_3}{dy}$, $\frac{dR_4}{dx}$, $\frac{dR_4}{dy}$ are in H^1. Integrating the square of the derivatives over the

computational domain gives

$$\begin{aligned}
\int_0^1\int_0^1\left(\frac{dR_3}{dx}\right)^2 d\eta d\xi &= \int_0^1\int_0^1 \frac{\eta^2}{(-\eta+1)^2} d\eta d\xi = \int_0^1\int_0^1 1+\frac{2(\eta-1)+1}{(-\eta+1)^2} d\eta d\xi \\
&= \left[\eta - 2\ln(1-\eta)+\frac{1}{1-\eta}\right]_0^1 = \lim_{a\to 0}\left(-2\ln(a)+\frac{1}{a}\right)\to\infty \\
\int_0^1\int_0^1\left(\frac{dR_3}{dy}\right)^2 d\eta d\xi &= \int_0^1\int_0^1\left(\frac{1}{2}-\frac{\xi-\frac{1}{2}}{-\eta+1}\right)^2 d\eta d\xi = \int_0^1 \frac{1}{4}+\frac{1}{12(1-\eta)^2} d\eta \\
&= \frac{1}{4}+\left[\frac{1}{12(1-\eta)}\right]_0^1 = \frac{1}{4}+\frac{1}{12}\cdot\lim_{a\to 0}(\frac{1}{a})-\frac{1}{12}\to\infty \\
\int_0^1\int_0^1\left(\frac{dR_4}{dx}\right)^2 d\eta d\xi &= \int_0^1\int_0^1 \frac{\eta^2}{(-\eta+1)^2} d\eta d\xi = \left[\eta-2\ln(1-\eta)+\frac{1}{1-\eta}\right]_0^1\to\infty \\
\int_0^1\int_0^1\left(\frac{dR_4}{dy}\right)^2 d\eta d\xi &= \int_0^1\int_0^1\left(\frac{1}{2}+\frac{\xi-\frac{1}{2}}{-\eta+1}\right)^2 d\eta d\xi = \frac{1}{4}+\frac{1}{12}\cdot\lim_{a\to 0}(\frac{1}{a})-\frac{1}{12}\to\infty
\end{aligned}$$

Hence, $\frac{dR_3}{dx}, \frac{dR_3}{dy}, \frac{dR_4}{dx}, \frac{dR_4}{dy} \notin H^1$. Our basis functions can as a consequence not be used as a basis in finite element analysis.

As with bilinear Lagrange basis functions we try to form a new set of basis functions by summing the basis functions and the derivatives at the degenerated line, yielding

$$\begin{aligned}
R_c &= R_3 + R_4 = -\eta(\xi-1)+\eta\xi = \eta, \\
\frac{dR_c}{dx} &= \frac{dR_3}{dx}+\frac{dR_4}{dx} = -\frac{\eta}{2\epsilon\eta-\eta+1}+\frac{\eta}{2\epsilon\eta-\eta+1} = 0, \\
\frac{dR_c}{dy} &= \frac{dR_3}{dy}+\frac{dR_4}{dy} = \frac{1}{2}-\frac{\xi-\frac{1}{2}}{2\epsilon\eta-\eta+1}+\frac{\xi-\frac{1}{2}}{2\epsilon\eta-\eta+1}+\frac{1}{2} = 1.
\end{aligned}$$

We see that R_c is a valid basis function and that R_c, $\frac{dR_c}{dx}$ and $\frac{dR_c}{dy}$ are in H^1. If we rather use the basis functions $R_a = R_1$, $R_b = R_2$ and R_c to represent the triangle, all basis function will be in H^1.

Numerical results

We have solved this case with $\epsilon = 0$ symbolically in MATLAB. Figure 6.5 shows the control points and the points that are used in visualizations.

Figure 6.6 shows the Jacobian $J_{\mathbf{x},\xi}$ plotted both in the physical space and the parameter space. We observe that the mapping is singular at the point c of the triangle and for $\eta = 1$.

Figure 6.7 shows the basis functions $R_1, \ldots, R_4$ in the parameter space. The basis functions are equal to one at "its corner" and equal to zero at all other corners. The basis functions are apparently representing the parameter space correctly.

Figure 6.8 shows the basis functions $R_1, \ldots, R_4$ in the physical space. We observe that R_3 and R_4, because of the degeneracy, are not representing the physical space very well. In accordance with our hypothesis, we have summed the basis functions R_3 and R_4. The sum, R_c, is shown in Figure 6.9. We see that R_c is a valid basis function.

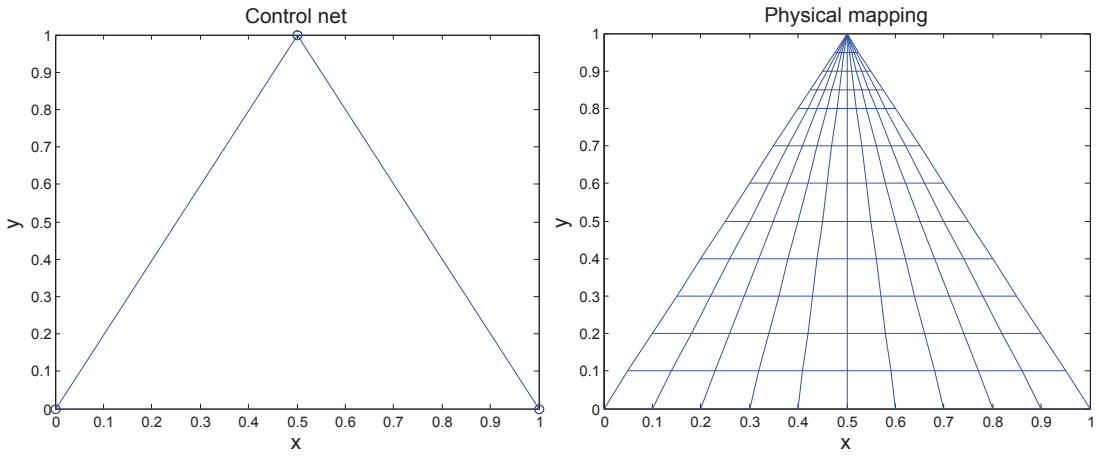


Figure 6.5: *Control net and visualization points.*

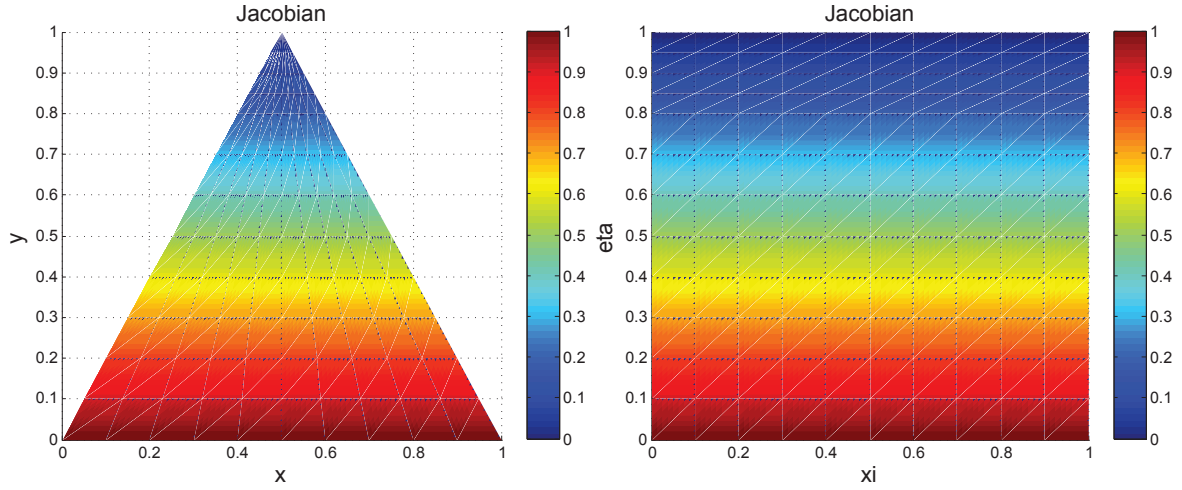


Figure 6.6: *Jacobian $J_{\mathbf{x},\xi}$ in physical space and parameter space.*

Figure 6.10 shows the derivatives of the basis functions $R_1, \ldots, R_4$ with respect to x. We see that, as expected, $\frac{dR_3}{dx}$ and $\frac{dR_4}{dx}$ are tending to $\pm\infty$ and are not defined as $\eta \to 1$. As they are not defined at $\eta = 1$ and $\eta = 0.95$ is our last visualization point, MATLAB are not able to plot the basis functions in the area $[0.95 < \eta \leq 1]$. If our visualization point had been closer to $\eta = 1$ we would have see that the derivatives would grow very big. They are tending to infinity, but MATLAB have difficulties plotting that. Figure 6.11 shows the derivatives of the basis functions $R_1, \ldots, R_4$ with respect to y. Similarly as for the derivatives with respect to x, we see that $\frac{dR_3}{dy}$ and $\frac{dR_4}{dy}$ are tending to $\pm\infty$ and are not defined as $\eta \to 1$.

Figure 6.12 shows $\frac{dR_c}{dx} = \frac{dR_3}{dx} + \frac{dR_4}{dx}$ and $\frac{dR_c}{dy} = \frac{dR_3}{dy} + \frac{dR_4}{dy}$. We see that $\frac{dR_c}{dx}$ and $\frac{dR_c}{dy}$ are defined everywhere and are, as expected, equal to zero and one. To calculate $\frac{dR_c}{dx}$ and $\frac{dR_c}{dy}$ we have summed the terms symbolically in MATLAB before evaluating them. If we had evaluated the derivatives before adding them, $\frac{dR_c}{dx}$ and $\frac{dR_c}{dy}$ would not be defined. To obtain the correct expressions for $\frac{dR_c}{dx}$ and $\frac{dR_c}{dy}$ we need to exploit the fact that the terms in $\frac{dR_3}{dx}$, $\frac{dR_4}{dx}$, $\frac{dR_3}{dy}$ and $\frac{dR_4}{dy}$ that are not defined as $\eta = 1$ will cancel when we add them. To illustrate this we will consider a simpler example; Let $F_1 = \frac{1}{x}$, $F_2 = -\frac{1}{x}$ and $F_3 = x$. The

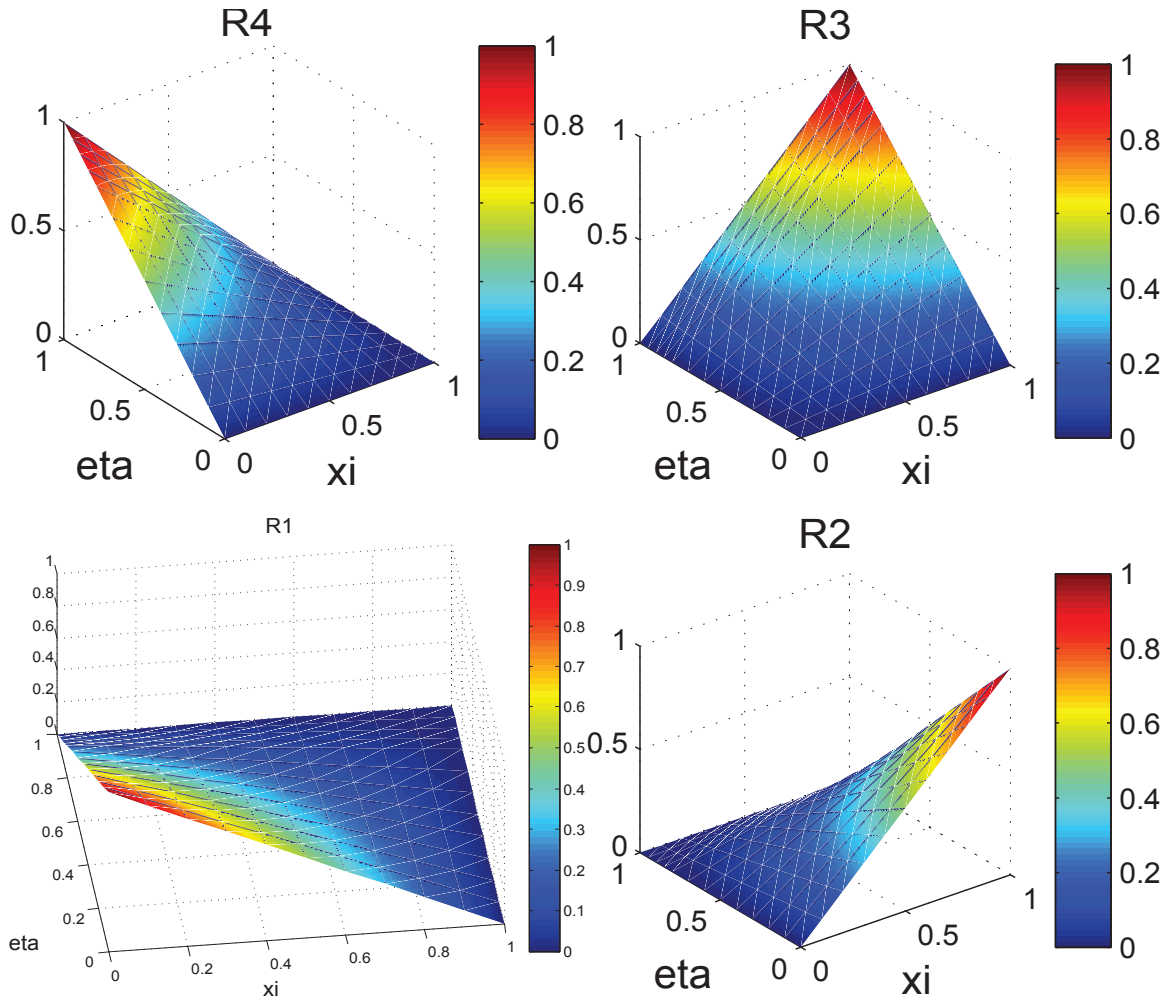


Figure 6.7: *The basis functions R_1, R_2 R_3 and R_4 in the parameter space.*

sum $F = F_1 + F_2 + F_3$ evaluated in $x = 0$ is thus given by

$$F = F_1 + F_2 + F_3 = \frac{1}{x} - \frac{1}{x} + x = x = 0.$$

If we had evaluated the individual terms in $x = 0$ before adding them we would obtain

$$F = F_1 + F_2 + F_3 = NAN - NAN + x = NAN.$$

NAN is an abbreviation for *not a number*. Hence, from a computational point of view, for instance when programming in MATLAB, F would not be defined, even though we know perfectly well that F is continuous and defined for all x. We thus have to be aware of *how* we performing the analysis, such that we add the basis functions *before* evaluating them. In MATLAB we have solved this issue by utilizing the symbolic toolbox, more precisely *syms*, *vpa*, *eval* and *simple* [28]. The downsides of symbolic calculations are that the running time is increasing tremendously and that we consequently won't be able to perform complex or huge analysis.

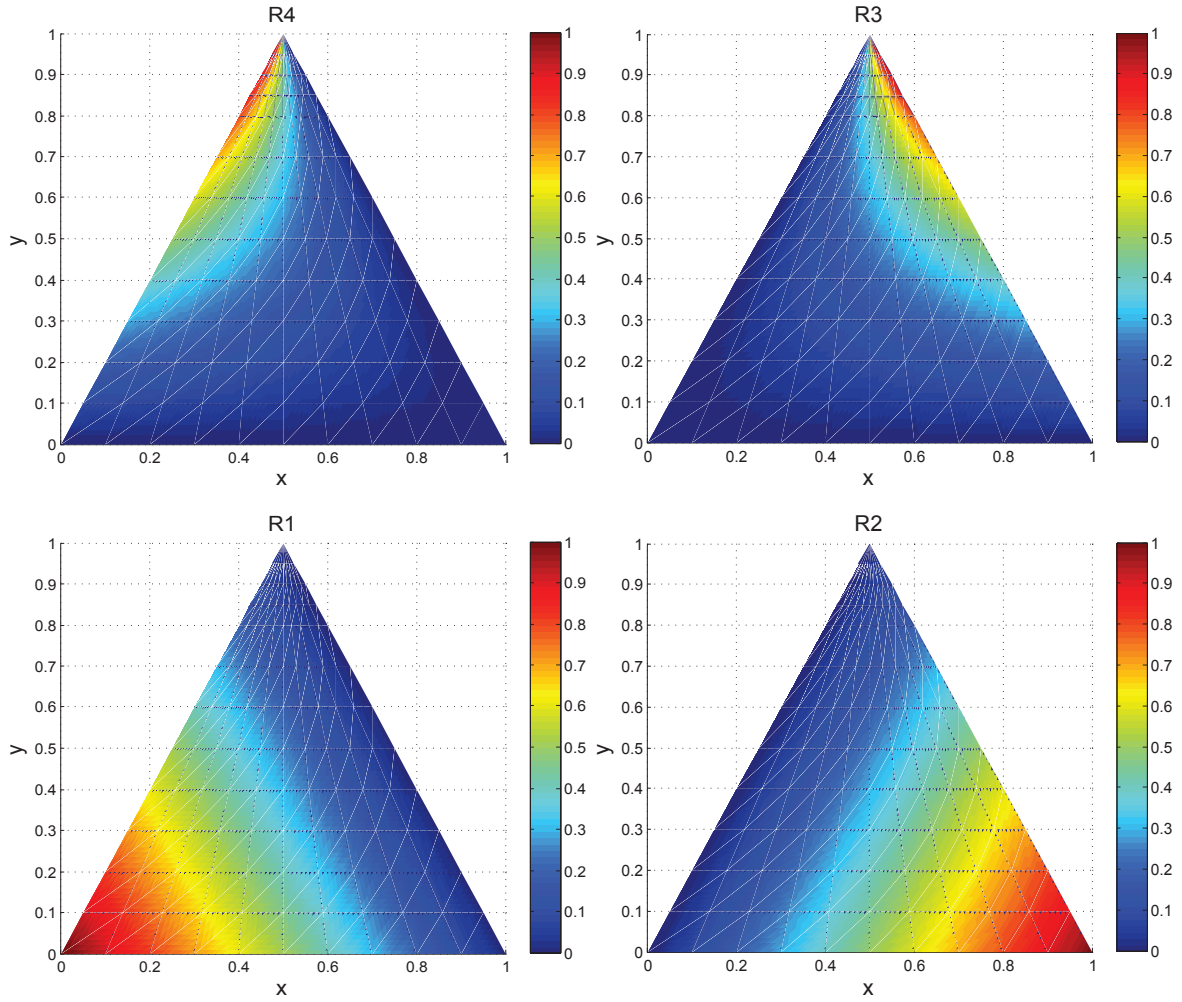


Figure 6.8: *The basis functions R_1, R_2 R_3 and R_4 in the physical space.*

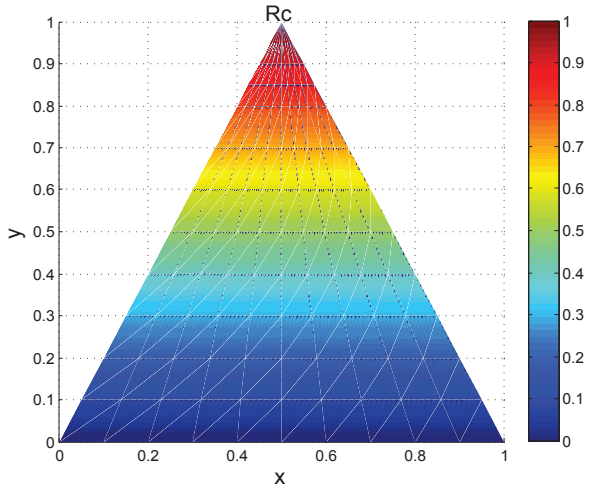


Figure 6.9: *R_c, the sum of basis functions at the degenerated side.*

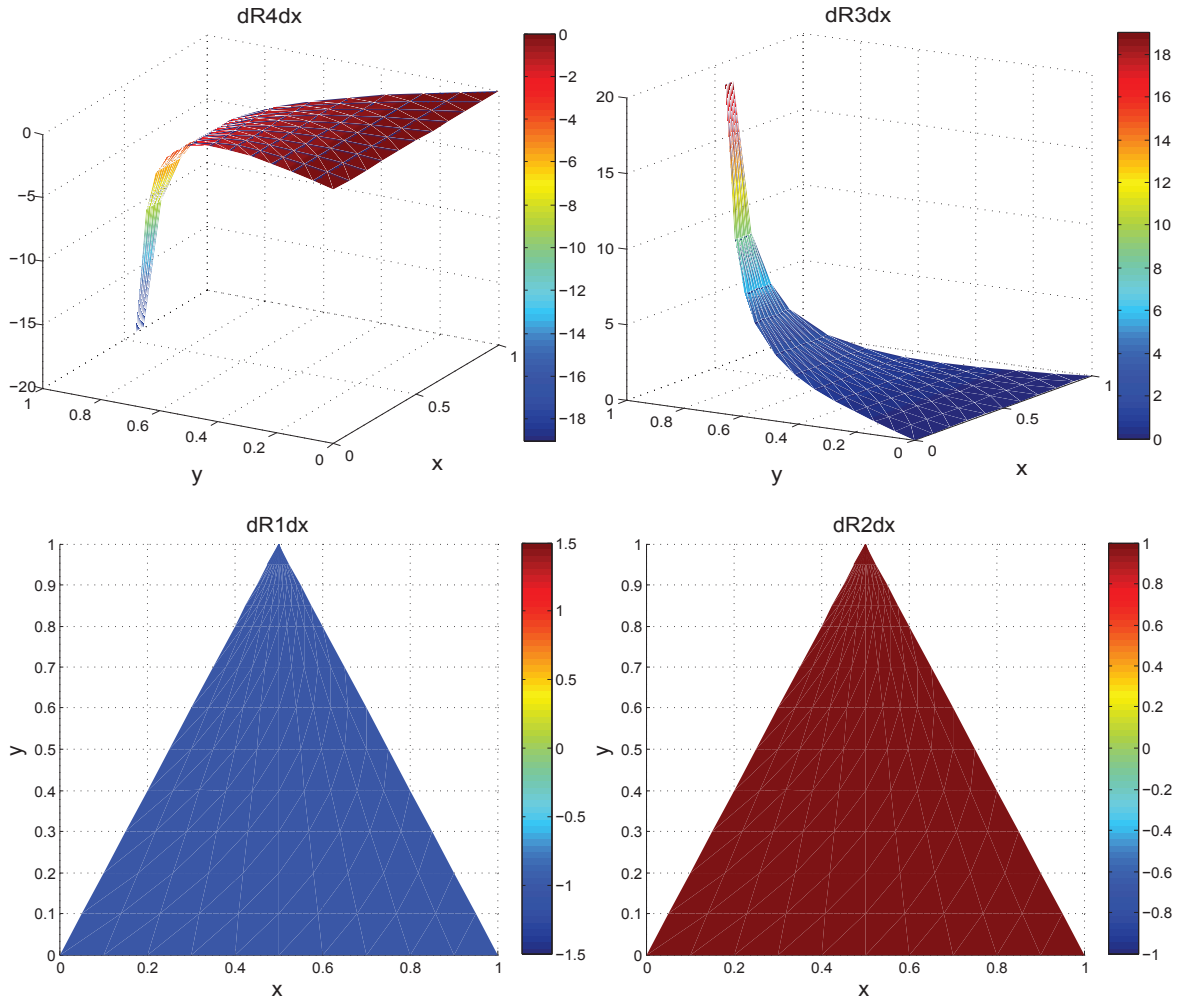


Figure 6.10: *Derivatives of basis functions with respect to x.*

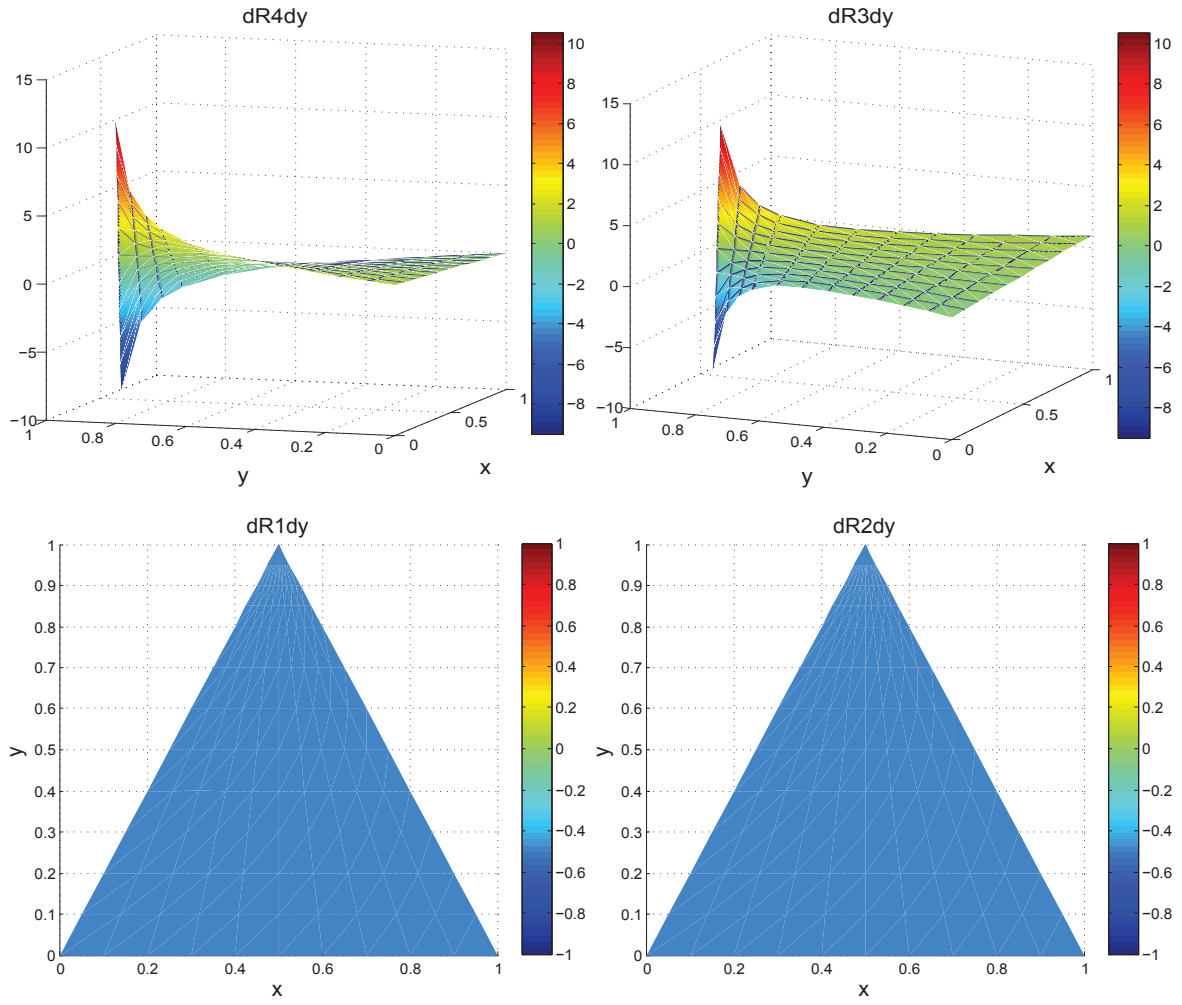


Figure 6.11: *Derivatives of basis functions with respect to y.*

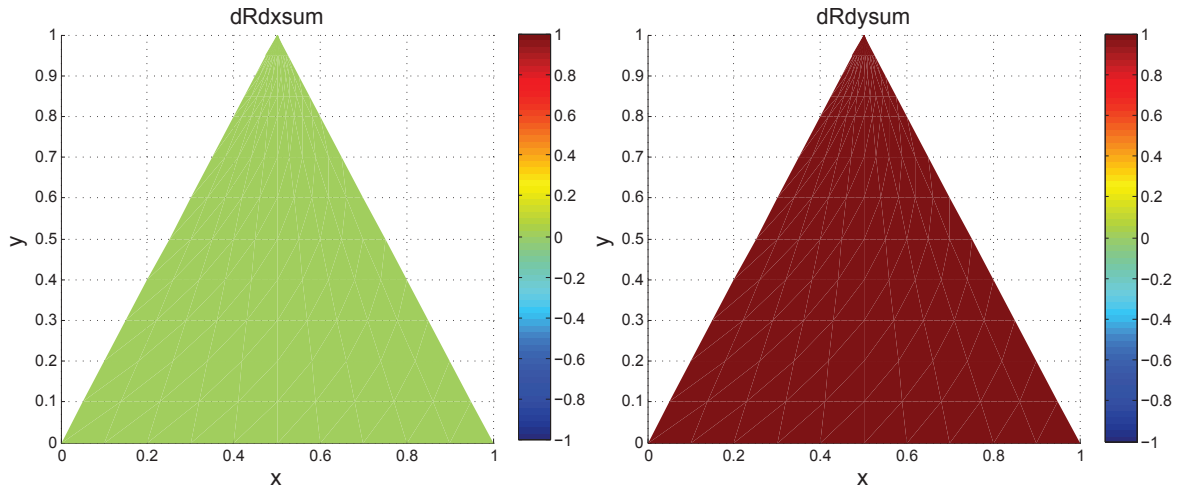


Figure 6.12: $\frac{dR_c}{dx}$ *and* $\frac{dR_c}{dy}$*, the sum of the derivatives at the degenerated side.*

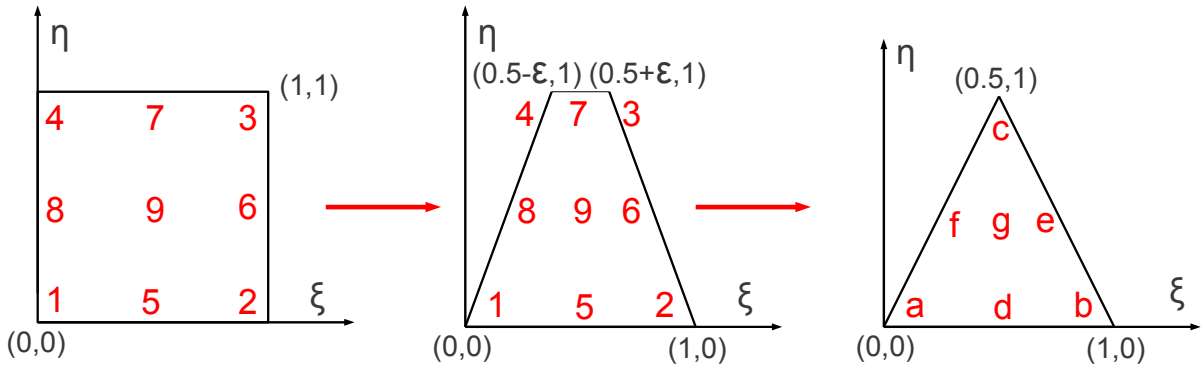


Figure 6.13: *Degenerating a quadrilateral; geometry and numbering for quadratic basis functions.*

Table 6.6: Control net $\mathbf{B}_{i,j}$ for modeling the degenerated quadrilateral with quadratic basis functions.

(i,j)	Control point $\mathbf{B}_{i,j}$	Weight w_{ij}	(ξ, η)
(1,1)	(0,0)	1	(0,0)
(2,1)	(0.5,0)	1	(0.5,0)
(3,1)	(1,0)	1	(1,0)
(1,2)	(0.25,0.5)	1	(0,0.5)
(2,2)	(0.5,0.5)	1	(0.5,0.5)
(3,2)	(0.75,0.5)	1	(1,0.5)
(1,3)	(0.5-ϵ,1)	1	(0,1)
(2,3)	(0.5,1)	1	(0.5,1)
(3,3)	(0.5+ϵ,1)	1	(1,1)

6.1.3 Quadratic B-spline basis functions

We have also looked at the same case utilizing quadratic basis functions to study if the behaviour is similar. Consider the degenerated quadrilateral given in Figure 6.13. The triangle in Figure 6.13 with $\epsilon \to 0$ is modeled by the knot vectors

$$\Xi = \{0, 0, 0, 1, 1, 1\},$$
$$\mathcal{H} = \{0, 0, 0, 1, 1, 1\},$$

and the control net and weights given in Table 6.6.

Table 6.7: Derivatives with respect to ξ and η.

$\frac{dR_1}{d\xi} = 2(\eta-1)^2(\xi-1)$	$\frac{dR_1}{d\eta} = 2(\eta-1)(\xi-1)^2$
$\frac{dR_2}{d\xi} = 2\xi(\eta-1)^2$	$\frac{dR_2}{d\eta} = 2\xi^2(\eta-1)$
$\frac{dR_3}{d\xi} = 2\eta^2\xi$	$\frac{dR_3}{d\eta} = 2\eta\xi^2$
$\frac{dR_4}{d\xi} = 2\eta^2(\xi-1)$	$\frac{dR_4}{d\eta} = 2\eta(\xi-1)^2$
$\frac{dR_5}{d\xi} = -2(2\xi-1)(\eta-1)^2$	$\frac{dR_5}{d\eta} = -4\xi(\eta-1)(\xi-1)$
$\frac{dR_6}{d\xi} = -4\eta\xi(\eta-1)$	$\frac{dR_6}{d\eta} = 2\xi^2(2\eta-1)$
$\frac{dR_7}{d\xi} = -2\eta^2(2\xi-1)$	$\frac{dR_7}{d\eta} = -4\eta\xi(\xi-1)$
$\frac{dR_8}{d\xi} = -4\eta(\eta-1)(\xi-1)$	$\frac{dR_8}{d\eta} = -2(2\eta-1)(\xi-1)^2$
$\frac{dR_9}{d\xi} = 4\eta(2\xi-1)(\eta-1)$	$\frac{dR_9}{d\eta} = 4\xi(2\eta-1)(\xi-1)$

The quadratic B-spline basis functions are in the parameter space given by

$$
\begin{aligned}
R_1 &= (\eta-1)^2(\xi-1)^2,\\
R_2 &= \xi^2(\eta-1)^2,\\
R_3 &= \eta^2\xi^2,\\
R_4 &= \eta^2(\xi-1)^2,\\
R_5 &= -2\xi(\eta-1)^2(\xi-1),\\
R_6 &= -2\eta\xi^2(\eta-1),\\
R_7 &= -2\eta^2\xi(\xi-1),\\
R_8 &= -2\eta(\eta-1)(\xi-1)^2,\\
R_9 &= 4\eta\xi(\eta-1)(\xi-1).
\end{aligned}
$$

The derivatives with respect to ξ and η are given in Table 6.7.

We have that

$$
\frac{\mathbf{dx}}{\mathbf{d}\xi} = \begin{bmatrix} \frac{dx}{d\xi} & \frac{dx}{d\eta} \\ \frac{dy}{d\xi} & \frac{dy}{d\eta} \end{bmatrix} = \begin{bmatrix} 2\epsilon\eta^2 - \eta + 1 & \frac{(4\epsilon\eta-1)(2\xi-1)}{2} \\ 0 & 1 \end{bmatrix},
$$

resulting in the Jacobian

$$
J_{\mathbf{x},\xi} = \left|\frac{\mathbf{dx}}{\mathbf{d}\xi}\right| = 2\epsilon\eta^2 - \eta + 1.
$$

When $\epsilon = 0$ and $\eta = 1$, $J_{\mathbf{x},\xi}$ is zero. Consequently, the mapping will be singular at the point c corresponding to the degenerated line, similarly as for the bilinear basis functions.

To find the Jacobian matrix $(\mathbf{J}_{\mathbf{x},\xi})^{-T}$ we use that

$$
\frac{\mathbf{d}\xi}{\mathbf{dx}} = \left(\frac{\mathbf{dx}}{\mathbf{d}\xi}\right)^{-1} = \begin{bmatrix} \frac{d\xi}{dx} & \frac{d\xi}{dy} \\ \frac{d\eta}{dx} & \frac{d\eta}{dy} \end{bmatrix} = \begin{bmatrix} \frac{1}{2\epsilon\eta^2-\eta+1} & -\frac{(4\epsilon\eta-1)(2\xi-1)}{2(2\epsilon\eta^2-\eta+1)} \\ 0 & 1 \end{bmatrix}.
$$

$(\mathbf{J}_{\mathbf{x},\xi})^{-T}$ is thus given by

$$
(\mathbf{J}_{\mathbf{x},\xi})^{-T} = \begin{bmatrix} \frac{1}{2\epsilon\eta^2-\eta+1} & 0 \\ -\frac{(4\epsilon\eta-1)(2\xi-1)}{2(2\epsilon\eta^2-\eta+1)} & 1 \end{bmatrix},
$$

yielding the derivatives with respect to x and y as shown in Table 6.8. We see that when

Table 6.8: Derivatives with respect to x and y.

$\frac{dR_1}{dx} = \frac{2(\eta-1)^2(\xi-1)}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_1}{dy} = -\frac{(\eta-1)(\xi-1)(4\epsilon\eta-\eta+4\epsilon\eta^2\xi-8\epsilon\eta\xi+1)}{2\epsilon\eta^2-\eta+1}$
$\frac{dR_2}{dx} = \frac{2\xi(\eta-1)^2}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_2}{dy} = -\frac{\xi(\eta-1)(\eta+4\epsilon\eta-4\epsilon\eta^2+4\epsilon\eta^2\xi-8\epsilon\eta\xi-1)}{2\epsilon\eta^2-\eta+1}$
$\frac{dR_3}{dx} = \frac{2\eta^2\xi}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_3}{dy} = -\frac{\eta\xi(\eta-2\xi-4\epsilon\eta^2+4\epsilon\eta^2\xi)}{2\epsilon\eta^2-\eta+1}$
$\frac{dR_4}{dx} = \frac{2\eta^2(\xi-1)}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_4}{dy} = \frac{\eta(\xi-1)(-4\epsilon\xi\eta^2+\eta+2\xi-2)}{2\epsilon\eta^2-\eta+1}$
$\frac{dR_5}{dx} = -\frac{2(2\xi-1)(\eta-1)^2}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_5}{dy} = -\frac{(\eta-1)(\eta+4\epsilon\eta-4\epsilon\eta^2+16\epsilon\eta\xi^2+8\epsilon\eta^2\xi-8\epsilon\eta^2\xi^2-16\epsilon\eta\xi-1)}{2\epsilon\eta^2-\eta+1}$
$\frac{dR_6}{dx} = -\frac{4\eta\xi(\eta-1)}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_6}{dy} = \frac{2\xi(2\xi-1)(\eta-1)(\eta-2)}{2\epsilon\eta^2-\eta+1} - 2\xi(2\eta+3\xi-2\eta\xi-2)$
$\frac{dR_7}{dx} = -\frac{2\eta^2(2\xi-1)}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_7}{dy} = \frac{8\eta^3\xi^2-8\eta^3\xi+4\eta^3}{2\eta^2} + \frac{\eta(2\xi-1)^2(\eta-2)}{2\epsilon\eta^2-\eta+1}$
$\frac{dR_8}{dx} = -\frac{4\eta(\eta-1)(\xi-1)}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_8}{dy} = 2(\xi-1)(2\eta\xi-3\xi+1) + \frac{2(2\xi-1)(\eta-1)(\eta-2)(\xi-1)}{2\epsilon\eta^2-\eta+1}$
$\frac{dR_9}{dx} = \frac{4\eta(2\xi-1)(\eta-1)}{2\epsilon\eta^2-\eta+1}$	$\frac{dR_9}{dy} = 8\eta\xi-12\xi-4\eta-8\eta\xi^2+12\xi^2-\frac{2(2\xi-1)^2(\eta-1)(\eta-2)}{2\epsilon\eta^2-\eta+1}+4$

Table 6.9: Analysis of the derivatives.

$\int_0^1\int_0^1\left(\frac{dR_1}{dx}\right)^2 d\eta d\xi = \frac{4}{9}$	$\int_0^1\int_0^1\left(\frac{dR_1}{dy}\right)^2 d\eta d\xi = \frac{1}{9}$
$\int_0^1\int_0^1\left(\frac{dR_2}{dx}\right)^2 d\eta d\xi = \frac{4}{9}$	$\int_0^1\int_0^1\left(\frac{dR_2}{dy}\right)^2 d\eta d\xi = \frac{1}{9}$
$\int_0^1\int_0^1\left(\frac{dR_3}{dx}\right)^2 d\eta d\xi = \infty$	$\int_0^1\int_0^1\left(\frac{dR_3}{dy}\right)^2 d\eta d\xi = \infty$
$\int_0^1\int_0^1\left(\frac{dR_4}{dx}\right)^2 d\eta d\xi = \infty$	$\int_0^1\int_0^1\left(\frac{dR_4}{dy}\right)^2 d\eta d\xi = \infty$
$\int_0^1\int_0^1\left(\frac{dR_5}{dx}\right)^2 d\eta d\xi = \frac{4}{9}$	$\int_0^1\int_0^1\left(\frac{dR_5}{dy}\right)^2 d\eta d\xi = \frac{67}{3} - 32\ln(2)$
$\int_0^1\int_0^1\left(\frac{dR_6}{dx}\right)^2 d\eta d\xi = \frac{16}{9}$	$\int_0^1\int_0^1\left(\frac{dR_6}{dy}\right)^2 d\eta d\xi = \frac{11}{45}$
$\int_0^1\int_0^1\left(\frac{dR_7}{dx}\right)^2 d\eta d\xi = \infty$	$\int_0^1\int_0^1\left(\frac{dR_7}{dy}\right)^2 d\eta d\xi = \infty$
$\int_0^1\int_0^1\left(\frac{dR_8}{dx}\right)^2 d\eta d\xi = \frac{16}{9}$	$\int_0^1\int_0^1\left(\frac{dR_8}{dy}\right)^2 d\eta d\xi = \frac{11}{45}$
$\int_0^1\int_0^1\left(\frac{dR_9}{dx}\right)^2 d\eta d\xi = \frac{16}{9}$	$\int_0^1\int_0^1\left(\frac{dR_9}{dy}\right)^2 d\eta d\xi = \frac{8}{15}$

$\eta = 1$, $G \to \pm\infty$ as $\epsilon \to 0$. When $\epsilon = 0$, G is not defined. The derivatives with respect to x and y will in consequence not be defined either.

We will do further analysis of the derivatives in the same way as we did with the derivatives of the bilinear basis functions. We have considered the derivatives with respect to x and y when $\epsilon = 0$ and have integrated the square of the derivatives over the computational domain. The result is given in Table 6.9.

We observe that $\frac{dR_4}{dx}, \frac{dR_7}{dx}, \frac{dR_3}{dx}, \frac{dR_4}{dy}, \frac{dR_7}{dy}, \frac{dR_3}{dy} \notin H^1$. The quadratic basis functions can as a consequence not be used as a basis in finite element analysis.

We try to form a new set of basis functions by summing the basis functions and the derivatives at the degenerated line, yielding

$$
\begin{aligned}
R_c &= R_4 + R_7 + R_3 = \eta^2(\xi-1)^2 - 2\eta^2\xi(\xi-1) + \eta^2\xi^2 = \eta^2, \\
\frac{dR_c}{dx} &= \frac{dR_4}{dx} + \frac{dR_7}{dx} + \frac{dR_3}{dx} = \frac{2\eta^2(\xi-1)}{2\epsilon\eta^2-\eta+1} - \frac{2\eta^2(2\xi-1)}{2\epsilon\eta^2-\eta+1} + \frac{2\eta^2\xi}{2\epsilon\eta^2-\eta+1} = 0, \\
\frac{dR_c}{dy} &= \frac{dR_4}{dy} + \frac{dR_7}{dy} + \frac{dR_3}{dy} = 2\eta.
\end{aligned}
$$

We see that R_c is a valid basis function and that R_c, $\frac{dR_c}{dx}$ and $\frac{dR_c}{dy}$ are in H^1. If we rather

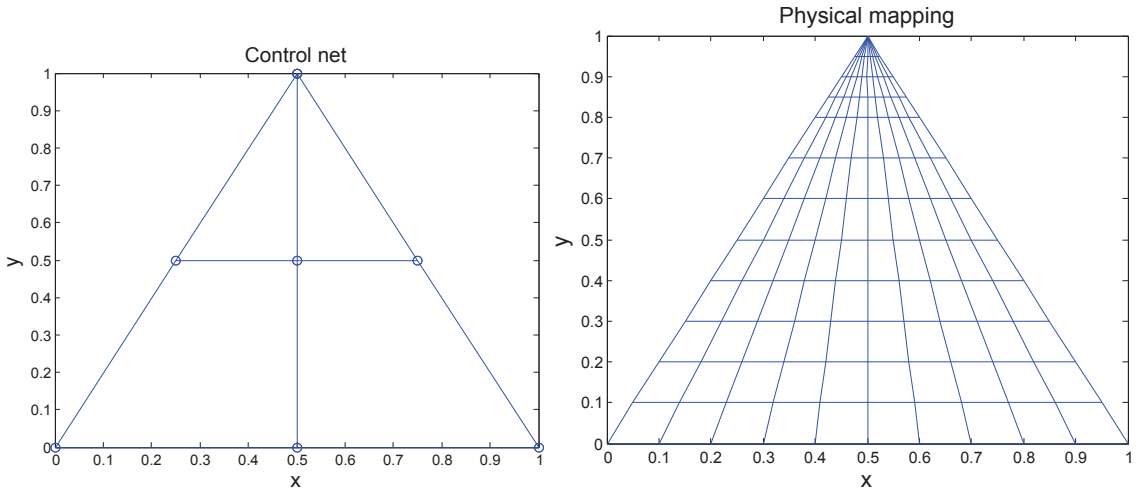


Figure 6.14: *Control net and visualization points.*

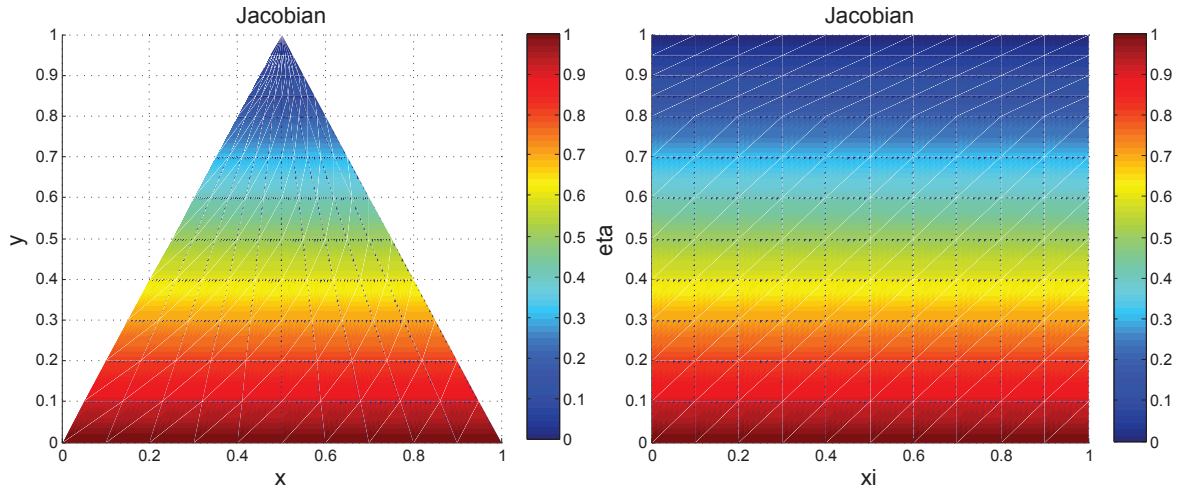


Figure 6.15: *Jacobian $J_{\mathbf{x},\xi}$ in physical space and parameter space.*

use the basis functions $R_a = R_1$, $R_b = R_2$, R_c, $R_d = R_5$, $R_e = R_6$, $R_f = R_8$ and $R_g = R_9$ to represent the triangle, all basis function will be in H^1.

Numerical results

We have solved this case with $\epsilon = 0$ symbolically in MATLAB. Figure 6.14 shows the control points and the points that are used in visualizations.

Figure 6.15 shows the Jacobian $J_{\mathbf{x},\xi}$ plotted both in the physical space and the parameter space. We observe that the mapping is singular at the point c of the triangle and for $\eta = 1$.

Figure 6.16 shows the basis functions $R_1, \ldots, R_9$ in the parameter space. The basis functions are apparently representing the space correctly.

Figure 6.17 shows the basis functions $R_1, \ldots, R_9$ in the physical space. We observe that R_3, R_4 and R_7 not are representing the physical space very well. We have summed the basis functions R_3, R_4 and R_7. The sum, R_c, is shown in Figure 6.18. We see that R_c is a valid basis function.

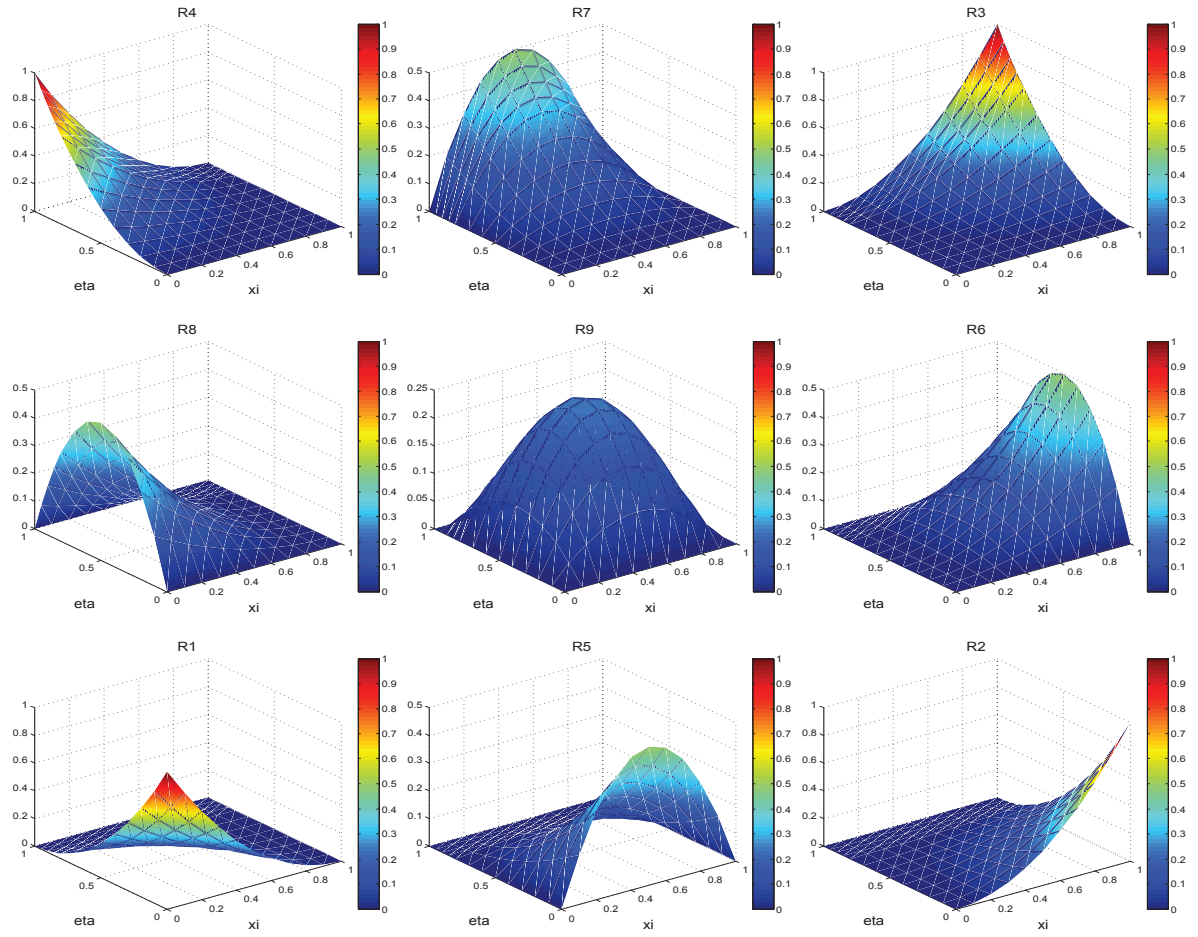


Figure 6.16: *The quadratic basis functions $R_1, \ldots, R_9$ in the parameter space.*

Figure 6.19 shows the derivatives of the basis functions $R_1, \ldots, R_9$ with respect to x. We see that, as expected, $\frac{dR_4}{dx}$, $\frac{dR_4}{dx}$ and $\frac{dR_7}{dx}$ are tending to $\pm\infty$ and are not defined as $\eta \to 1$. Figure 6.20 shows the derivatives of the basis functions $R_1, \ldots, R_9$ with respect to y. Similarly as for the derivatives with respect to x, we see that $\frac{dR_3}{dy}$, $\frac{dR_4}{dy}$ and $\frac{dR_7}{dy}$ are tending to $\pm\infty$ and are not defined as $\eta \to 1$. Figure 6.21 shows $\frac{dR_c}{dx} = \frac{dR_3}{dx} + \frac{dR_4}{dx} + \frac{dR_7}{dx}$ and $\frac{dR_c}{dy} = \frac{dR_3}{dy} + \frac{dR_4}{dy} + \frac{dR_7}{dy}$. We see that $\frac{dR_c}{dx}$ and $\frac{dR_c}{dy}$ are defined everywhere and are, as expected, equal to zero and 2η.

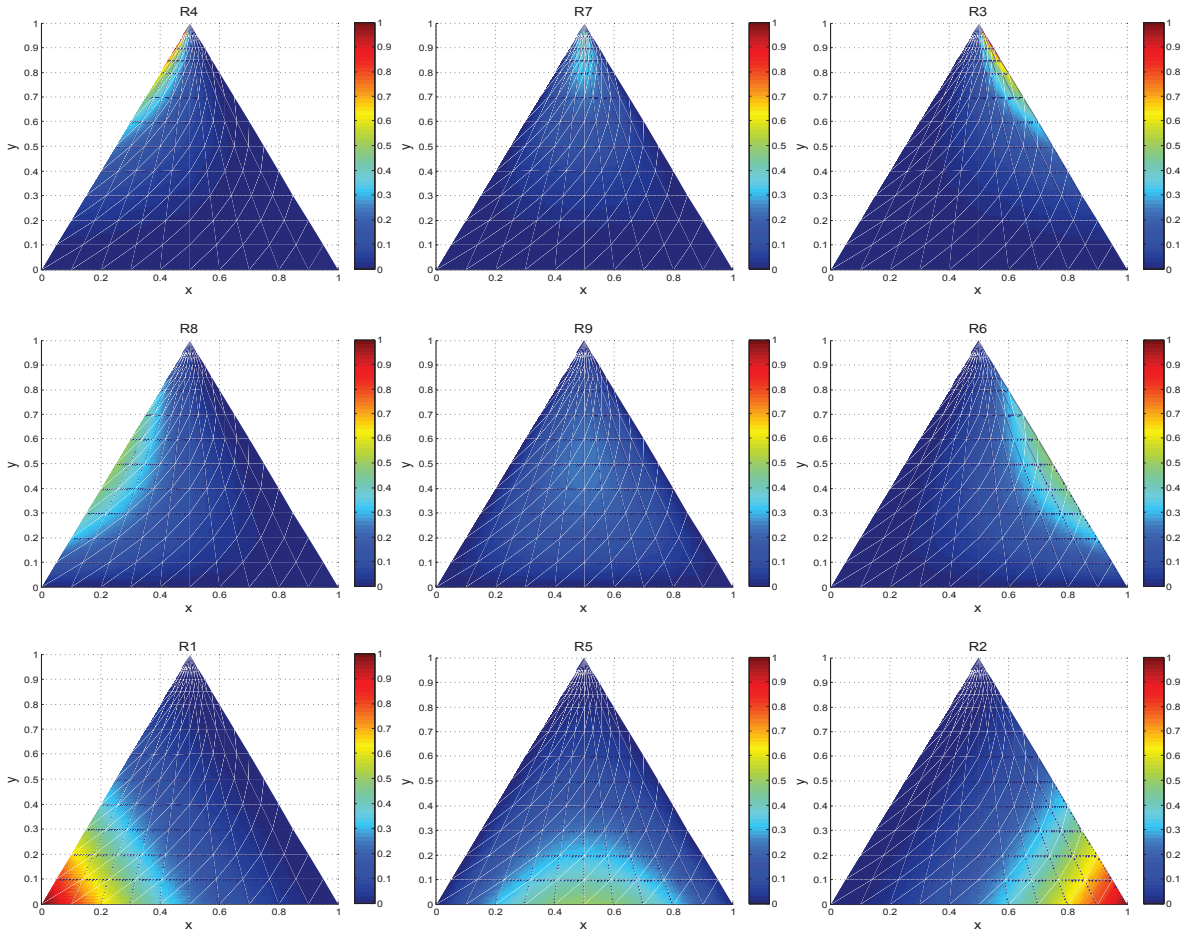


Figure 6.17: *The quadratic basis functions $R_1, \ldots, R_9$ in the physical space.*

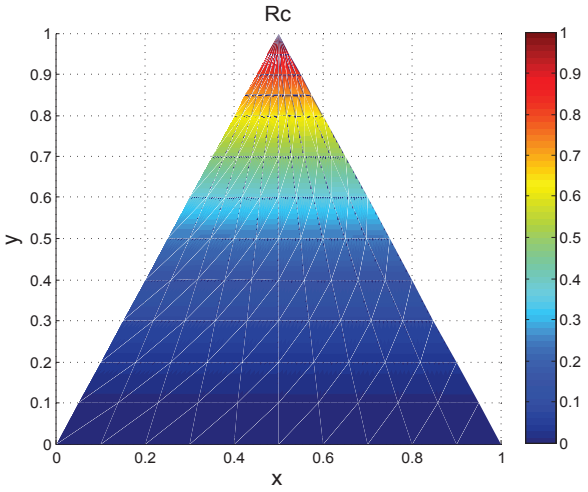


Figure 6.18: *R_c, the sum of basis functions at the degenerated side.*

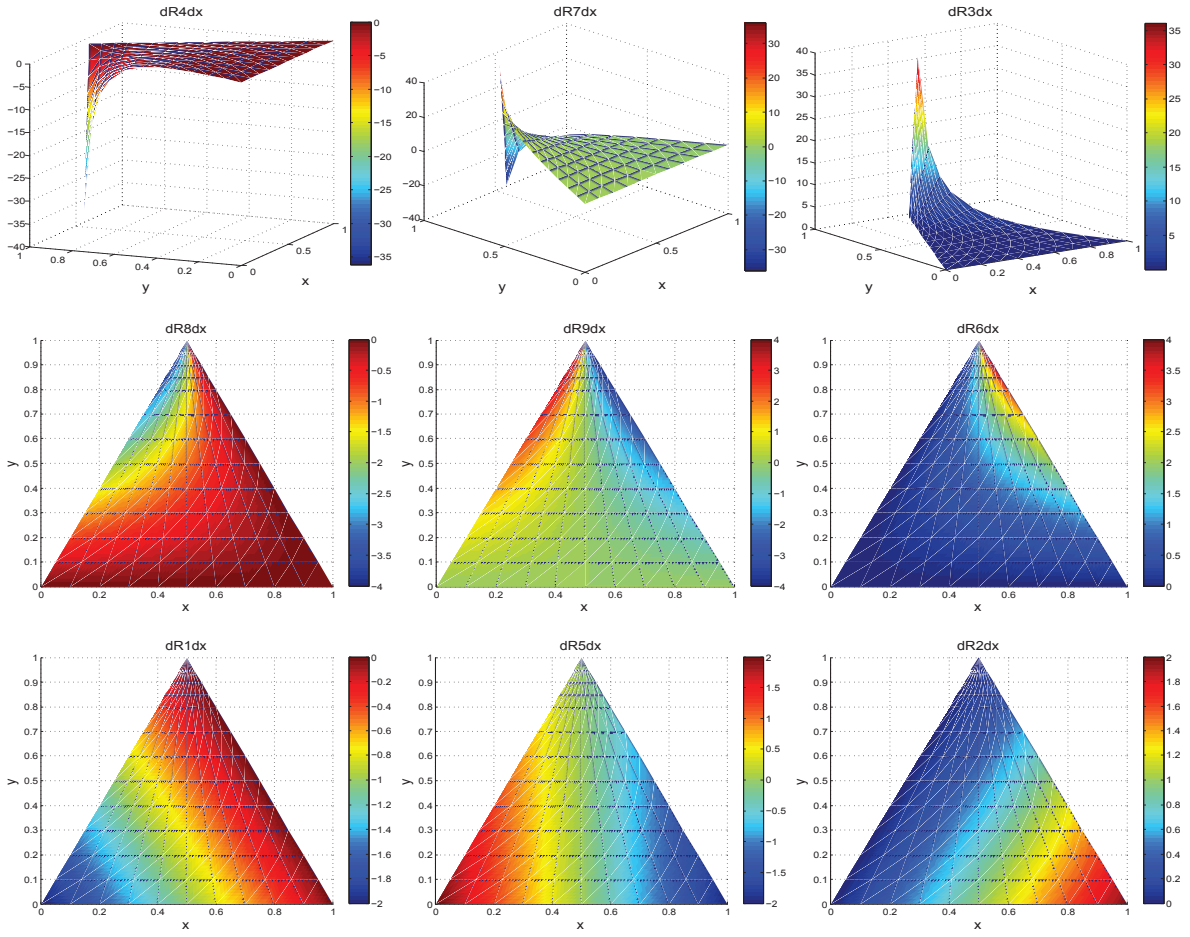


Figure 6.19: *Derivatives of the basis functions with respect to x.*

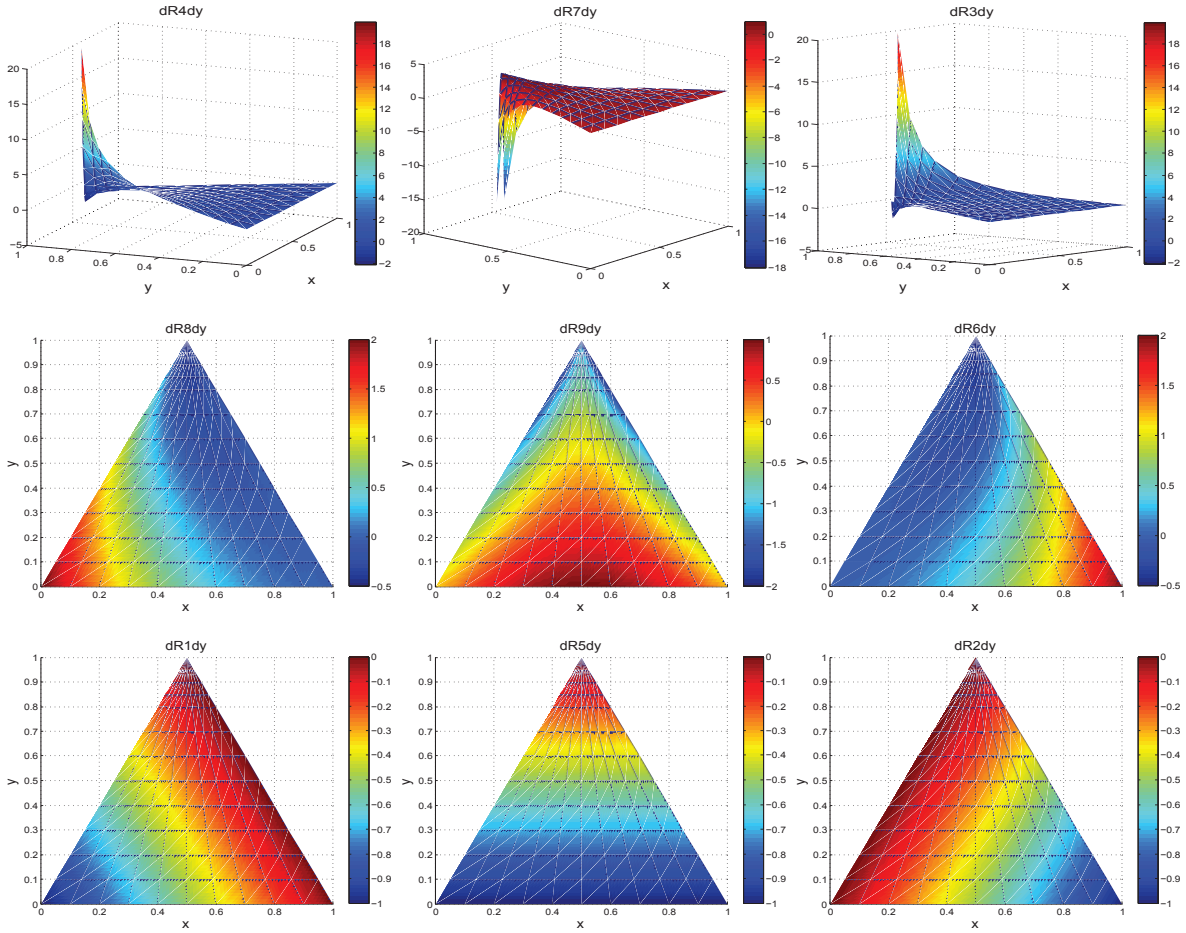


Figure 6.20: *Derivatives of the basis functions with respect to y.*

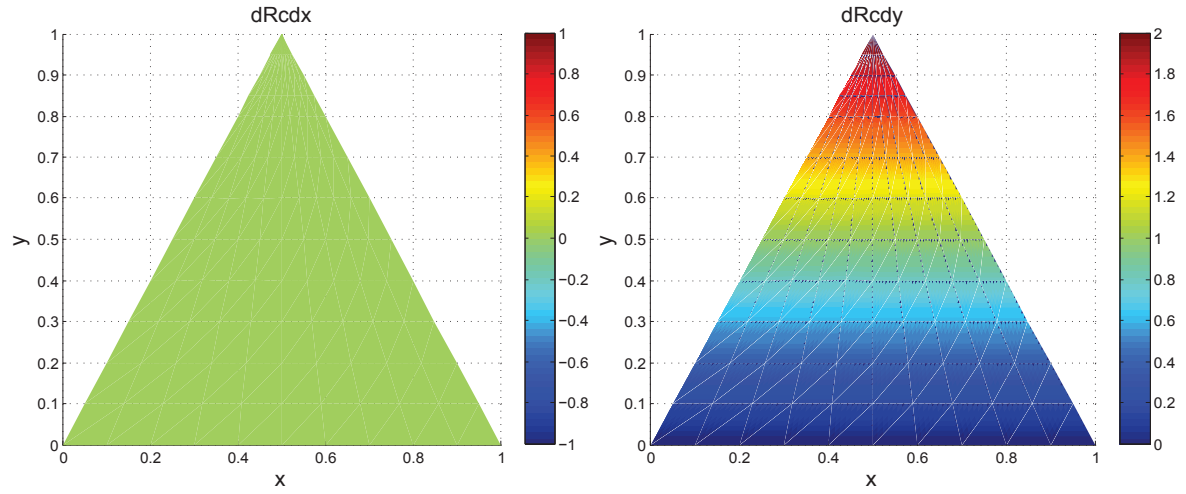


Figure 6.21: $\frac{dR_c}{dx}$ *and* $\frac{dR_c}{dy}$*, the sum of the derivatives at the degenerated side.*

6.2 Discussion

In this chapter we have considered a quadrilateral that was degenerated to a triangle. We observed that the original quadrilateral had the same number of basis functions as the number of degrees of freedom. When two or more control points coincided our mapping got degenerated and the number of degrees of freedom was reduced. When the quadrilateral was degenerated, but was spanned by the same set of basis functions, the number of basis functions were greater than the number of degrees of freedom. We found that the first derivatives of the basis functions at the degeneracy were infinite. Consequently, the basis functions were no longer in H^1. Because of that our degenerated quadrilateral could not be considered as a proper finite element, and the finite element method could not be applied.

We observed that if we added the basis functions spanning the line that was degenerated to a point, we could form a new set of basis functions. The new set of basis functions were then all in H^1 and the number of basis functions were the same as the number of degrees of freedom. The new set of basis functions was also equal to the basis functions of a triangle. The degenerated quadrilateral was thus a proper finite element. Based on these observations we came up with a hypothesis saying that when a mapping is degenerated we need to redefine the set of basis functions, with the new set of basis functions being formed by adding the basis functions corresponding to the degeneracy. We now want to discuss this hypothesis further.

We start by looking at the properties of the new set of basis functions. Recall that our original set of basis functions $\mathcal{R} = \{R_1, \dots, R_n\}$ are non-negative and that they form a partition of unity, i.e.

$$\begin{aligned} R_i(\xi) &\geq 0 \quad \forall \xi, \\ \sum_{i=1}^{n} R_i(\xi) &= 1. \end{aligned}$$

Our new set of basis function, $\tilde{\mathcal{R}} = \{\tilde{R}_1 = R_1, \tilde{R}_k = (R_i + R_{j\neq i}), \dots, \tilde{R}_m = R_n\}$ will possess the same properties since

$$\begin{aligned} \tilde{R}_k &= \underbrace{R_i}_{\geq 0} + \underbrace{R_j}_{\geq 0} \geq 0, \\ \sum_{i=1}^{m} \tilde{R}_i &= \sum_{i=1}^{n} R_i = 1. \end{aligned}$$

The new set of basis functions are still non-negative and form a partition of unity, indicating that our new basis is a valid basis.

We are now going to study if our element domain still is a proper finite element after it has been degenerated. Confirm the definition of a finite element in Chapter 3.4. Consider the finite element $(\mathcal{K}, \mathcal{P}, \mathcal{N})$ where $\mathcal{P}$ is the set of all bilinear polynomials of degree less than or equal to k. Assume that the element domain, $\mathcal{K}$, is degenerated due to coinciding control variables, $N_i = N_j \in \mathcal{N}$. The dimension of $\mathcal{P}'$ is then reduced as the degrees of freedom is reduced. The dimension of $\mathcal{P}$ is then no longer equal to the dimension of $\mathcal{P}'$, and our basis is no longer linear independent as $\dim \mathcal{N} \neq \dim \mathcal{P}$.

In this particular case our basis functions were no longer in H^1 after degenerating from a quadrilateral to a triangle. In consequence, we are no longer considering the variational

problem (3.1) as the assumption now are violated. The bilinear form $a(\cdot, \cdot)$ is not bounded on V and is not in V'. Due to the fact that the basis functions are not in H^1, the finite element method could not be applied. Forming a new set of basis functions as explained earlier would in this case solve the problems caused by the degenerated mapping. It still remains to prove if this approach is general and will generate valid basis functions for all types of degenerated mappings.

Chapter 7

Numerical results

7.1 Circular surface

We want to model a circular surface by the knot vectors

$$\Xi = \{0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1\},$$
$$\mathcal{H} = \{0, 0, 1, 1\},$$

of order $p = 2$ and $q = 1$ respectively. The control net and weights are given in Table 7.1. The control net and the physical mapping are shown in Figure 7.1. The circular surface consists of four knot spans, as shown in the figure of the physical mapping. The basis functions have a different expression in each knot span. As we have four knot spans, each basis function will be given by a unique polynomial in each of the four non-overlapping intervals corresponding to the knot spans. When we are calculating the basis functions symbolically, we therefore only need to find the expression for the basis functions in each knot span, and not in all the visualization points.

Figure 7.2 shows the Jacobian $J_{\mathbf{x},\xi}$ plotted in the physical space. We observe that the mapping is singular where we have coinciding control points. Some of the derivatives with respect to the physical coordinates x and y are shown in Figure 7.3. We see that the derivatives are tending to $\pm\infty$ at the center of the circle, due to the inverse of the Jacobian tending to infinity. As we are not able to plot values that huge in MATLAB,

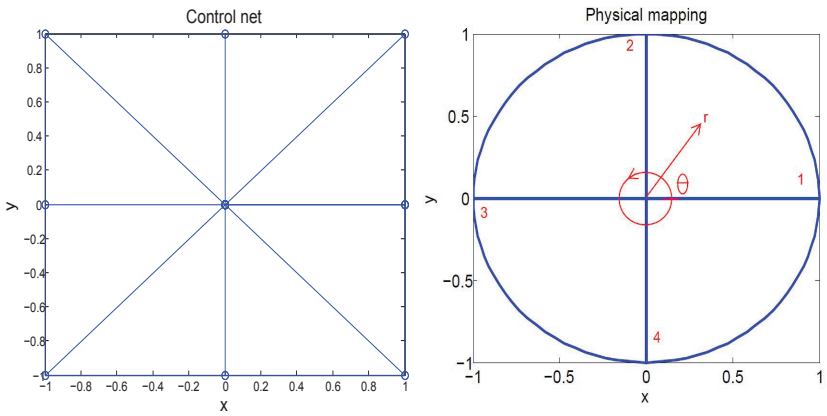


Figure 7.1: *Control net and physical mapping.*

Table 7.1: Control net $\mathbf{B}_{i,j}$

(i,j)	Control point $\mathbf{B}_{i,j}$	Weight w_{ij}
(1,1)	(0,0)	1
(2,1)	(0,0)	1
(3,1)	(0,0)	1
(4,1)	(0,0)	1
(5,1)	(0,0)	1
(6,1)	(0,0)	1
(7,1)	(0,0)	1
(8,1)	(0,0)	1
(9,1)	(0,0)	1
(1,2)	(1,0)	1
(2,2)	(1,1)	$\frac{1}{\sqrt{2}}$
(3,2)	(0,1)	1
(4,2)	(-1,1)	$\frac{1}{\sqrt{2}}$
(5,2)	(-1,0)	1
(6,2)	(-1,-1)	$\frac{1}{\sqrt{2}}$
(7,2)	(0,-1)	1
(8,2)	(1,-1)	$\frac{1}{\sqrt{2}}$
(9,2)	(1,0)	1

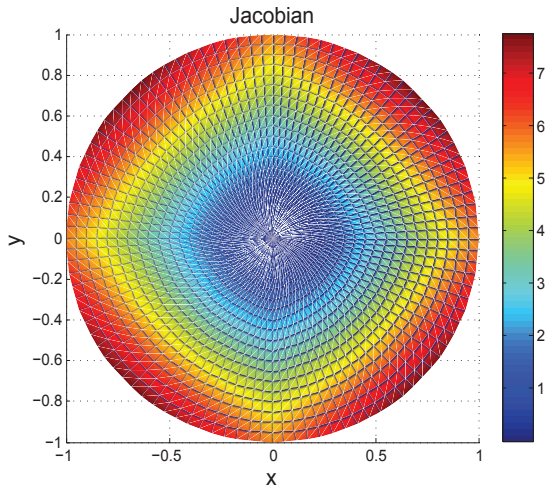


Figure 7.2: *Jacobian $J_{\mathbf{x},\xi}$ in the physical space.*

we see a hole in the circular surface at the origin. Notice that the Jacobian is dependent on the angle θ. The knot spans are symmetric, but within each knot span, the mapping is dependent of both r and θ. Even though the knot vector is uniform, the quadratic parametrization is not uniform in the physical space.

If we as explained in Chapter 6 sum the nine basis functions at the middle of the circle, we get the new basis function R_{sum}, as shown in Figure 7.4. We see that R_{sum} is a valid basis function. As we have four knot spans we need to calculate four expressions

for R_{sum}. Doing that we find that R_{sum} is given by

$$R_{sum} = \begin{cases} -\frac{\mathrm{r}-1}{-4\,\mathrm{r}\left(4\sqrt{2}-8\right)\left(\frac{\theta}{2\pi}\right)^2+4\,\mathrm{r}\left(\sqrt{2}-2\right)\left(\frac{\theta}{2\pi}\right)+1} & \text{if} \quad 0 \leq \theta < \frac{\pi}{2} \\ \frac{\mathrm{r}-1}{4\,\mathrm{r}\left(4\sqrt{2}-8\right)\left(\frac{\theta}{2\pi}\right)^2-4\,\mathrm{r}\left(3\sqrt{2}-6\right)\left(\frac{\theta}{2\pi}\right)+4\,\mathrm{r}\left(\frac{\sqrt{2}}{2}-1\right)-1} & \text{if} \quad \frac{\pi}{2} \leq \theta < \pi \\ \frac{\mathrm{r}-1}{4\,\mathrm{r}\left(4\sqrt{2}-8\right)\left(\frac{\theta}{2\pi}\right)^2-4\,\mathrm{r}\left(5\sqrt{2}-10\right)\left(\frac{\theta}{2\pi}\right)+4\,\mathrm{r}\left(\frac{3\sqrt{2}}{2}-3\right)-1} & \text{if} \quad \pi \leq \theta < \frac{3\pi}{2} \\ \frac{\mathrm{r}-1}{4\,\mathrm{r}\left(4\sqrt{2}-8\right)\left(\frac{\theta}{2\pi}\right)^2-4\,\mathrm{r}\left(7\sqrt{2}-14\right)\left(\frac{\theta}{2\pi}\right)+4\,\mathrm{r}\left(3\sqrt{2}-6\right)-1} & \text{if} \quad \frac{3\pi}{2} \leq \theta < 2\pi \end{cases}$$

for $0 \leq r \leq 1$. We see that the basis function is dependent of the angle. A rational B-spline representation of a circle is geometrically exact, but the parametrization is not uniform. We have parametrized the circle using uniform knot vectors. However, the quadratic parametrization will not be uniform in the physical space. Therefore, our basis functions will be dependent on the angle.

We have also summed the derivatives of the basis function at the degenerated mapping in the middle of the circle symbolically. The summed derivatives are shown from different angles in Figure 7.5 and 7.6. We see that the derivatives of R_{sum} with respect to x and y are finite and are C^0-continuous at the origin. Using this approach and forming a new set of basis functions will hence make us able to calculate the correct derivatives in the middle of the circle. We are thus able to calculate stresses correctly, and are not obtaining non-physical singularities due to a degenerated mapping.

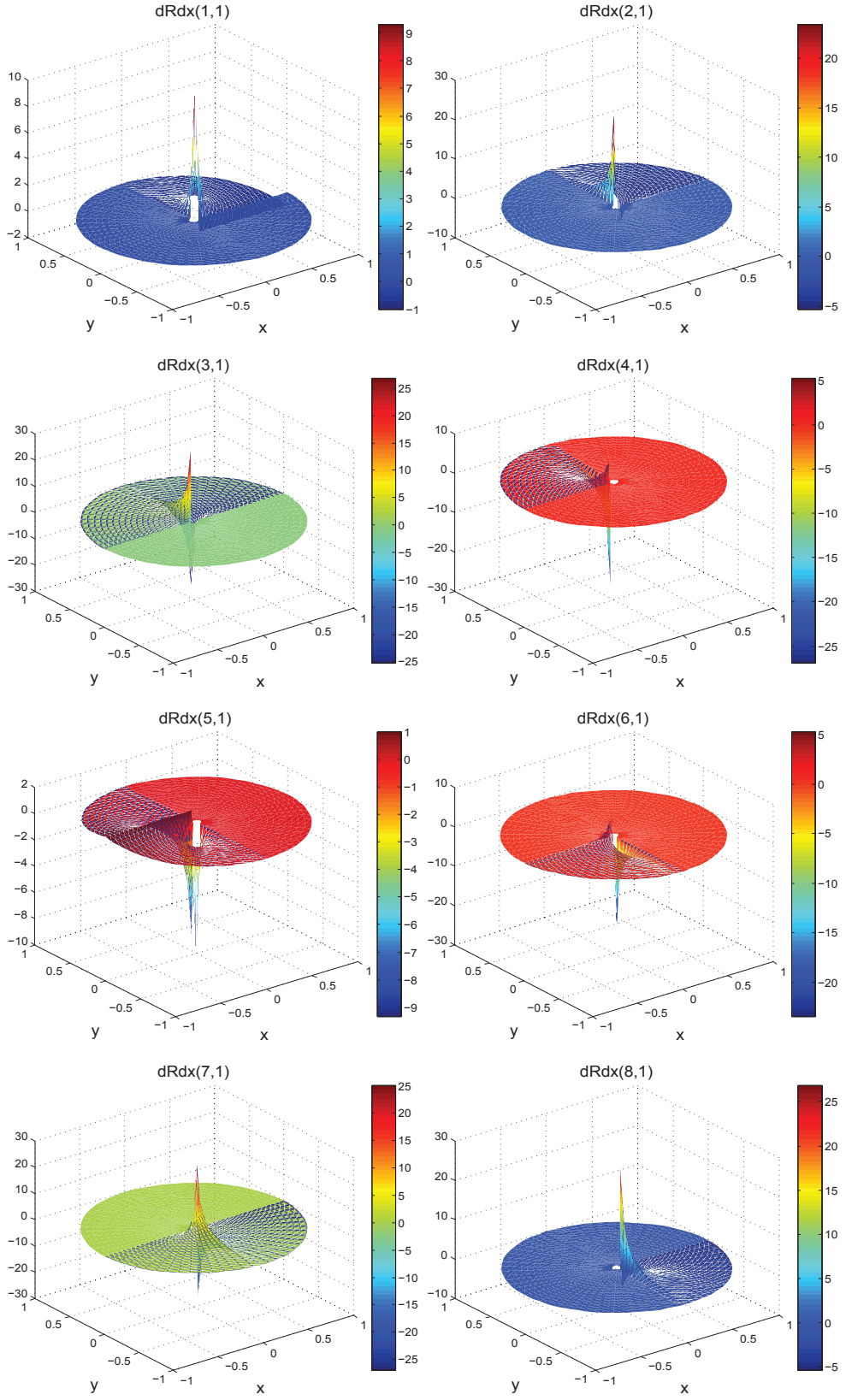


Figure 7.3: *Some of the derivatives with respect to x. We see that the derivatives tend to $\pm\infty$.*

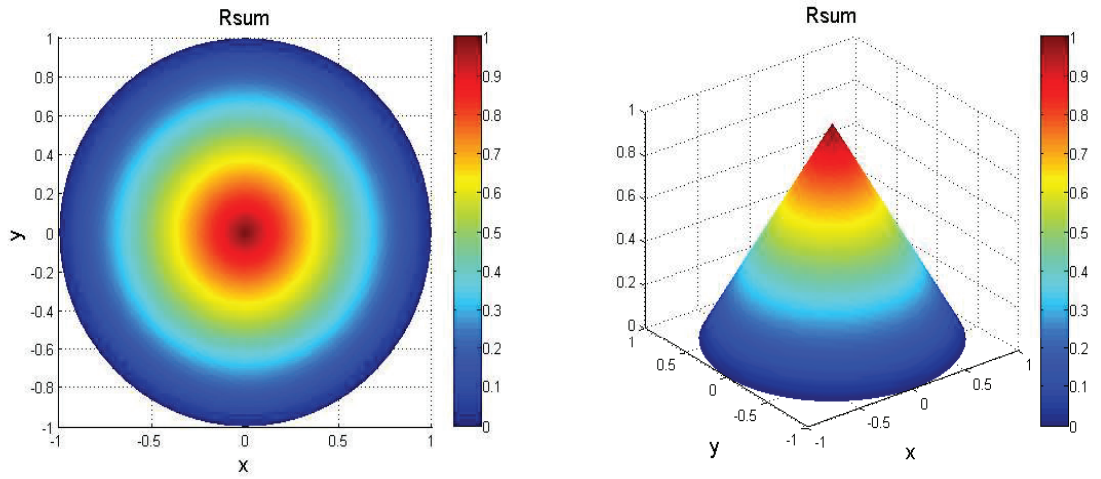


Figure 7.4: R_{sum}, *the sum of basis functions at the degenerated point in the middle of the circle.*

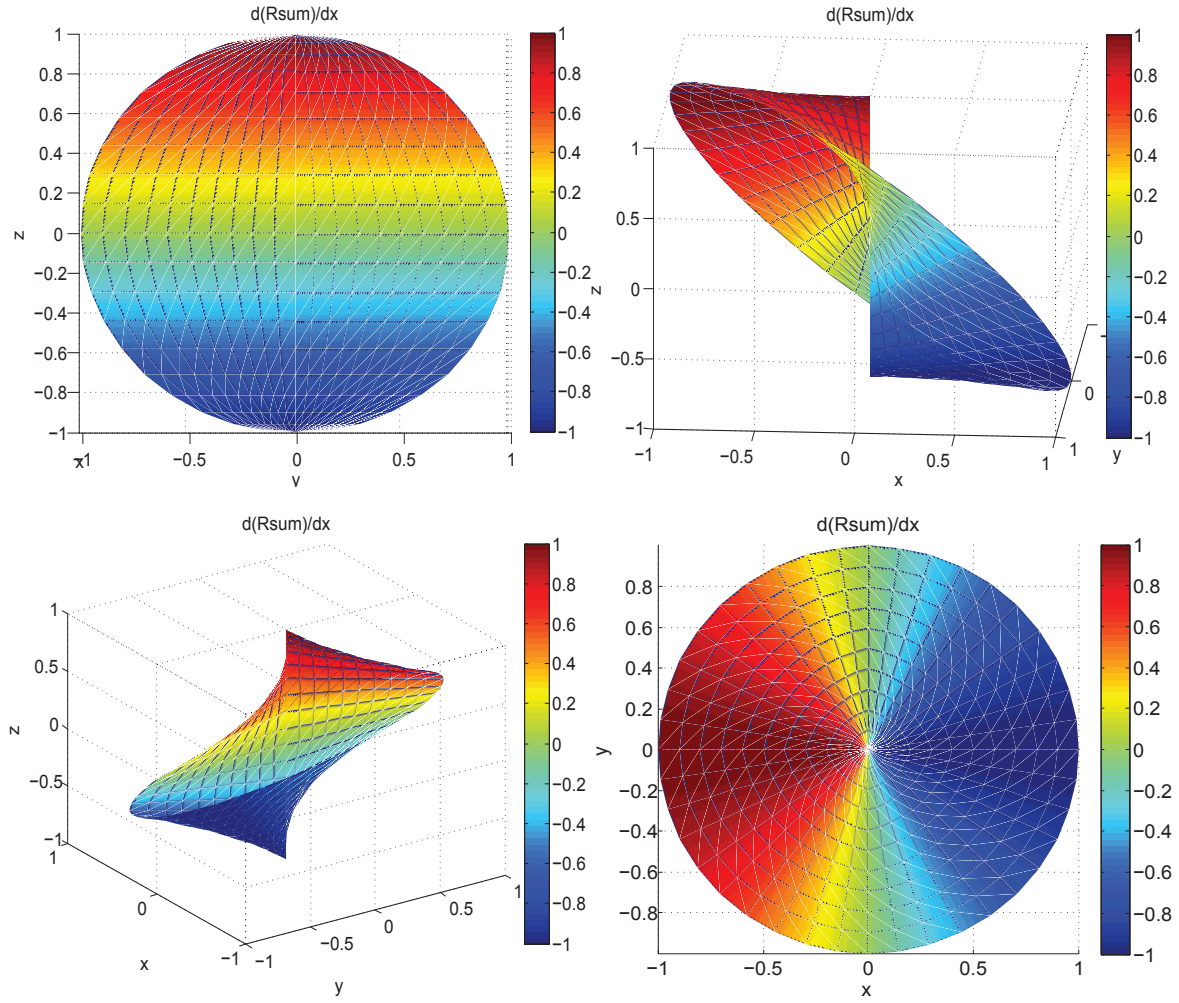


Figure 7.5: $\frac{dR_{sum}}{dx}$, *the sum of the derivatives with respect to* x *at the degenerated point in the middle of the circle.*

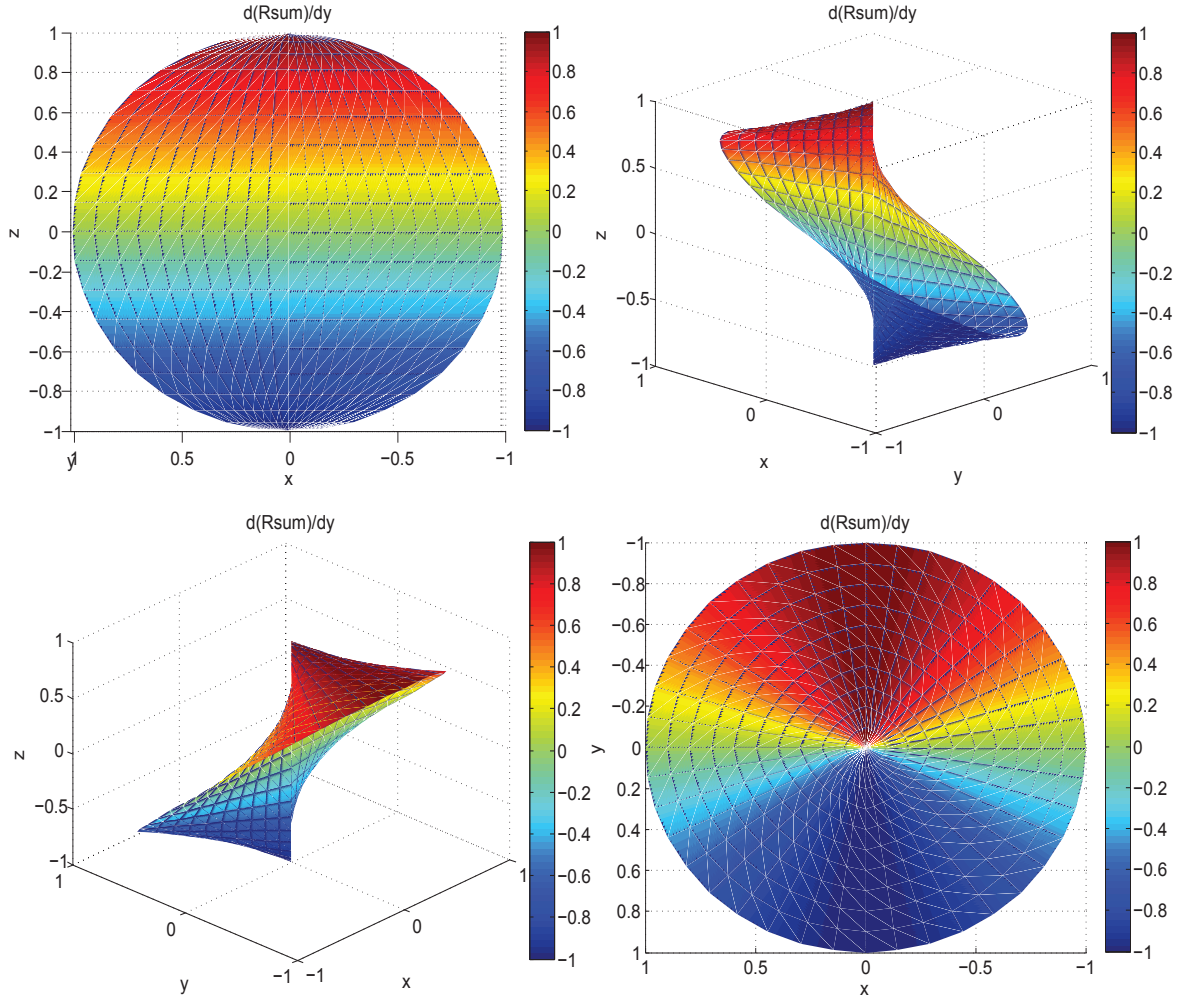


Figure 7.6: $\frac{dR_{sum}}{dy}$, *the sum of the derivatives with respect to* y *at the degenerated point in the middle of the circle.*

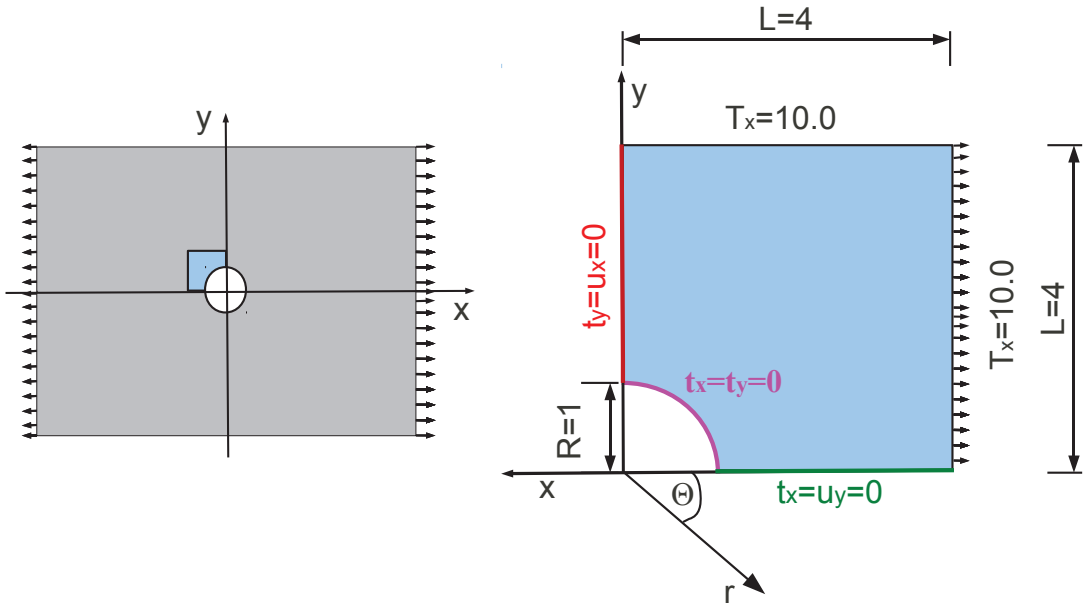


Figure 7.7: *Infinite plate with circular hole.*

Table 7.2: Problem description.

T_x	=	10.0
R	=	1
L	=	4
E	=	10^5
ν	=	0.3
r^2	=	x^2+y^2
θ	=	$\arctan(y,x)$

7.2 Infinite plate with circular hole

To verify that our MATLAB-code is correct we have for one thing looked at a already solved linear elasticity problem, the infinite plate with a circular hole. We will solve the equations

$$\nabla \cdot \sigma(\mathbf{u}) = 0 \quad \text{in} \quad \Omega, \tag{7.1}$$

$$\mathbf{u} = g_D \quad \text{on} \quad \Gamma_D, \tag{7.2}$$

$$\sigma \cdot \mathbf{n} = g_N \quad \text{on} \quad \partial\Gamma_N. \tag{7.3}$$

We have considered the same control net, weights and boundary conditions as employed in [17] and [16]. The problem is shown in Figure 7.7. Due to symmetry we need only to consider the area shown in blue. The part of the plate we are looking at is a square of length $L = 4$ with one fourth of the circular hole with radius $R = 1$. We have set Young's modulus to $E = 10^5$ and Poisson's ration to $\nu = 0.3$. We also use the relations $r^2 = x^2+y^2$ and $\theta = \arctan(y, x)$. Table 7.2 summarizes the quantities used in our problem.

Table 7.3: Control net $\mathbf{B}_{i,j}$

i/j	1	2	3
1	(-1,0)	(-2.5,0)	(-4,0)
2	(-1,$\sqrt{2}$-1)	(-2.5,0.75)	(-4,4)
3	(1-$\sqrt{2}$,1)	(-0.75,2.5)	(-4,4)
4	(0,1)	(0,2.5)	(0,4)

Table 7.4: Weights

(i,j)	1	2	3
1	1	1	1
2	s	1	1
3	s	1	1
4	1	1	1

At the arc of the circle we have Neumann boundary conditions $\sigma \cdot \mathbf{n} = 0$. Recall that

$$\sigma = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix}.$$

At the connected boundary in y-direction we have boundary conditions $u_x = 0$ and $t_y = 0$. At the connected boundary in x-direction we have $u_y = 0$ and $t_x = 0$, cf. Figure 7.7. At the two remaining sides we have prescribed traction with $T_x = 10.0$. As our plate is infinite we can use the exact solution to calculate the prescribed traction [21]. The exact solution is given by [35]

$$\begin{aligned}
\sigma_{xx} &= T_x \cdot \left\{ 1 - \frac{R^2}{r^2} \left(\frac{3}{2}\cos(2\theta) + \cos(4\theta) \right) + \frac{3R^4}{2r^4}\cos(4\theta) \right\}, \\
\sigma_{yy} &= T_x \cdot \left\{ -\frac{R^2}{r^2} \left(\frac{1}{2}\cos(2\theta) - \cos(4\theta) \right) - \frac{3R^4}{2r^4}\cos(4\theta) \right\}, \\
\sigma_{xy} &= T_x \cdot \left\{ -\frac{R^2}{r^2} \left(\frac{1}{2}\sin(2\theta) + \sin(4\theta) \right) + \frac{3R^4}{2r^4}\sin(4\theta) \right\},
\end{aligned}$$

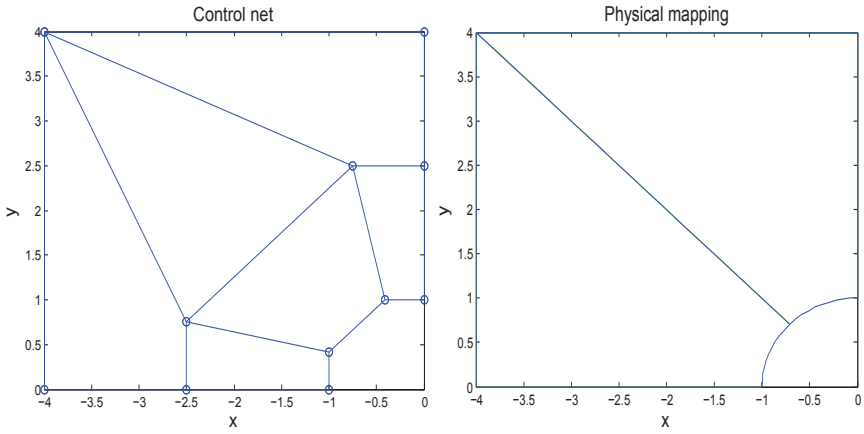
with r, R and θ as in Figure 7.7 and Table 7.2.

To solve our problem we employ the knot vectors

$$\begin{aligned}
\Xi &= \{0, 0, 0, 0.5, 1, 1, 1\}, \\
\mathcal{H} &= \{0, 0, 0, 1, 1, 1\},
\end{aligned}$$

of order $p = q = 2$ and the control net given in Table 7.3. The corresponding weights are given in Table 7.4, where $s = \frac{1}{2}(1 + \frac{1}{\sqrt{2}})$. The control net and physical mapping is shown in Figure 7.8. Notice that we will have a singularity at $(-4, 4)$ resulting in the Jacobian $J_{\mathbf{x},\xi}$ being equal to zero at that point, see Figure 7.9.

Figure 7.10 shows the displacements. Notice that the displacement in x-direction is zero at $x = 0$ and that the displacement in y-direction is zero at $y = 0$, as given in the boundary conditions. From our boundary conditions it is also expected that we have a greater displacement in the x-direction than in the y-direction.



(a) *Control net.* (b) *Physical mapping.*

Figure 7.8: *Control net and physical mapping for* $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$, $\mathcal{H} = \{0, 0, 0, 1, 1, 1\}$.

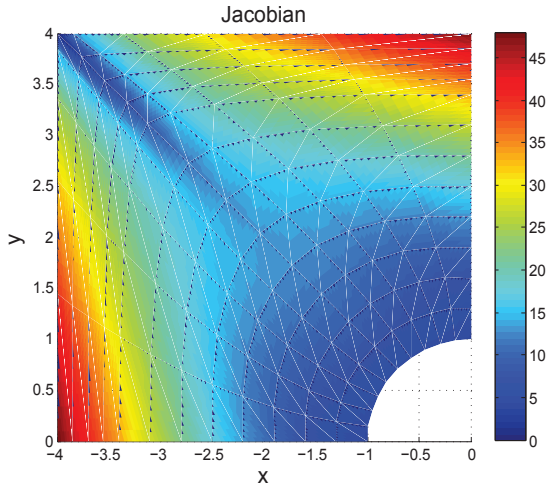


Figure 7.9: *Jacobian* $J_{\mathbf{x},\xi}$ *in the physical space.*

We want to solve this issue by summing the two basis functions belonging to the degenerated point. Figure 7.12 shows the two basis functions before and after we have added them in R_{sum}.

Figure 7.13 shows the derivatives of R_{sum} with respect to x and y from different angles. We see that the derivatives are smooth and finite.

We have done refinements by inserting new knots and have calculated the stresses. Figures 7.15, 7.16 and 7.17 shows the stresses σ_{xx}, σ_{yy} and σ_{xy} before and after refinements. The new control net is shown in Figure 7.14. We can compared our solution against the exact solution which is also shown in the figures 7.15, 7.16 and 7.17.

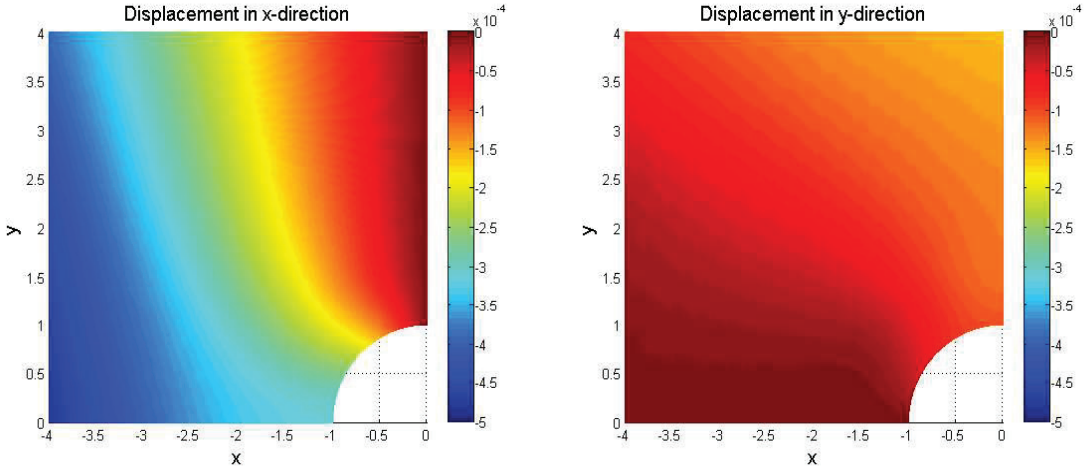


Figure 7.10: *Displacement in x- and y-direction when* $T_x = 10.0$.

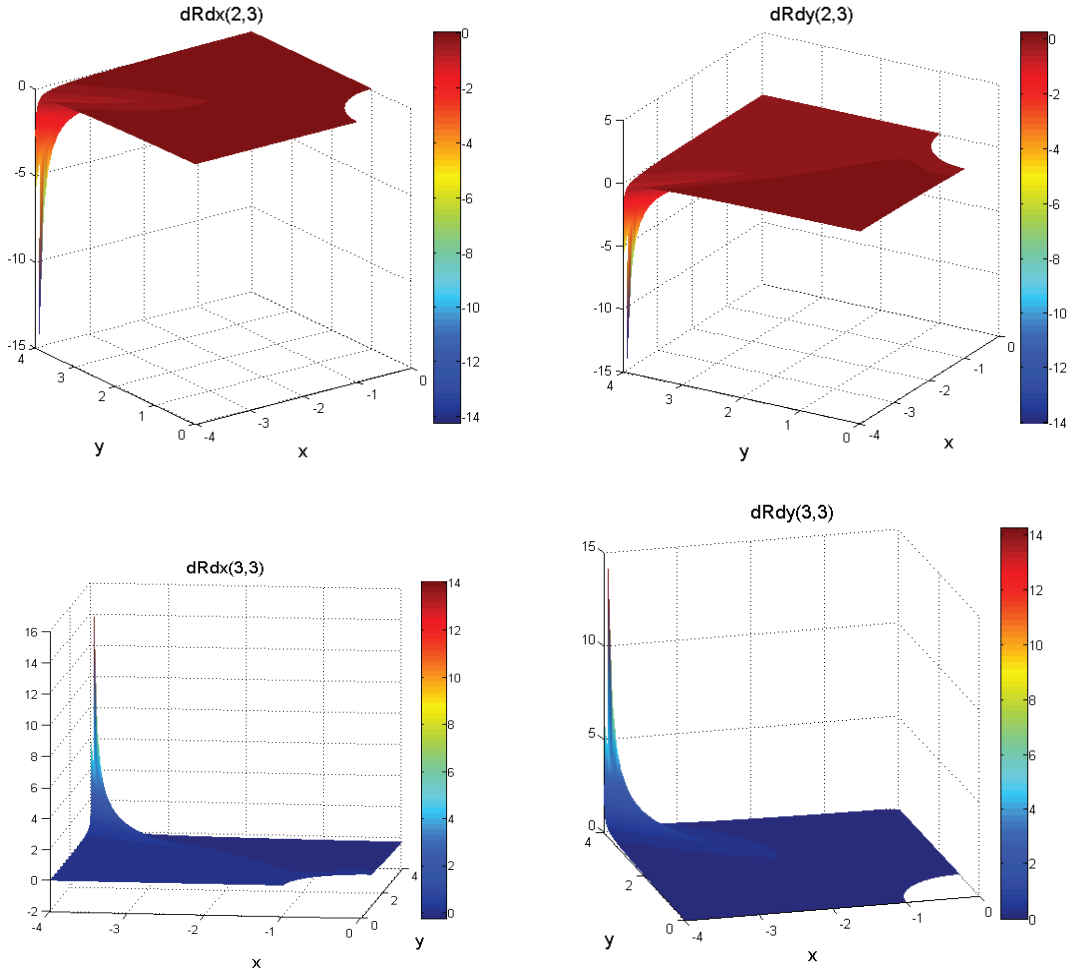


Figure 7.11: *Derivatives of the basis functions at the degenerated point.*

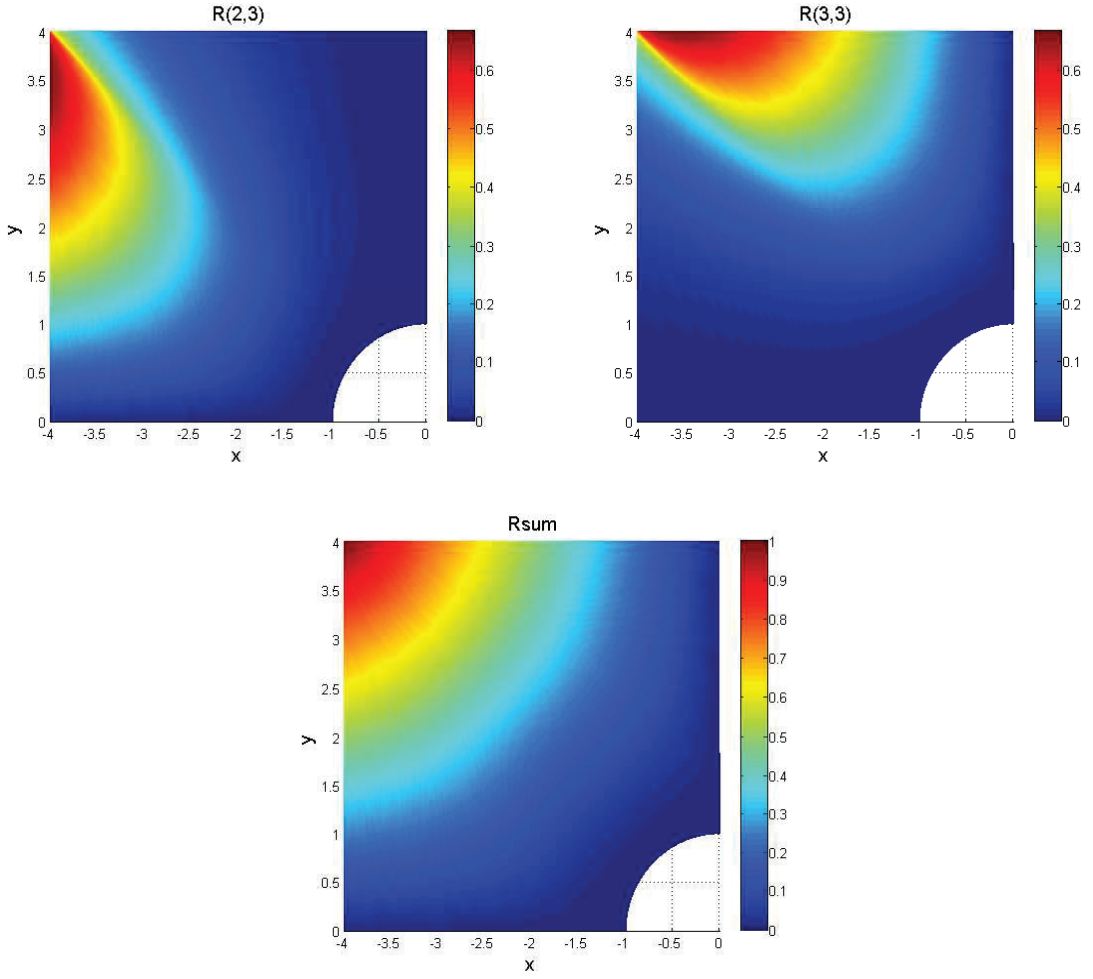


Figure 7.12: R_{sum} *is shown at the bottom figure and is the sum of the two basis functions shown above.*

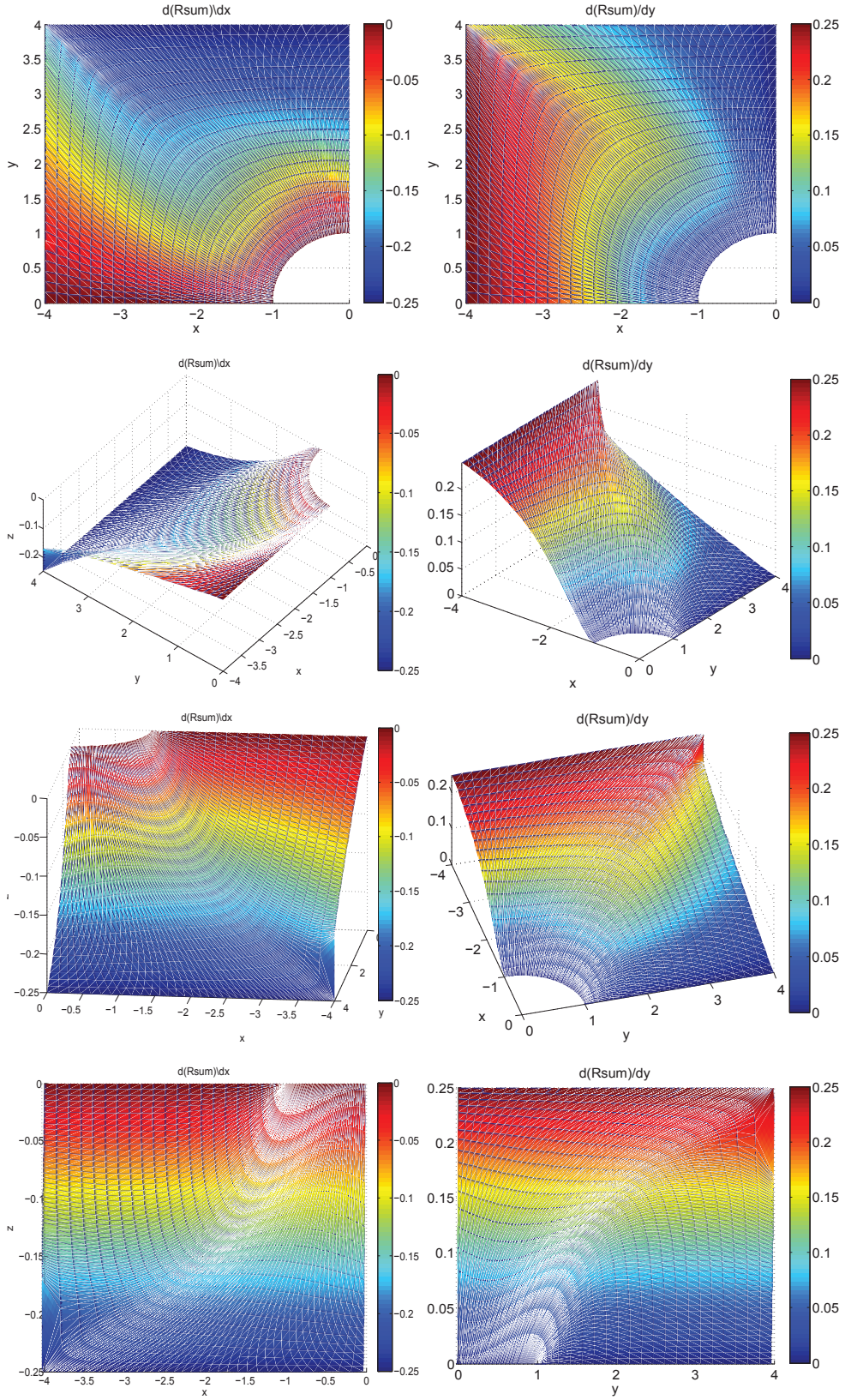


Figure 7.13: $\frac{dR_{sum}}{dx}$ *and* $\frac{dR_{sum}}{dy}$*, the summed derivatives of the basis functions at the degenerated point.*

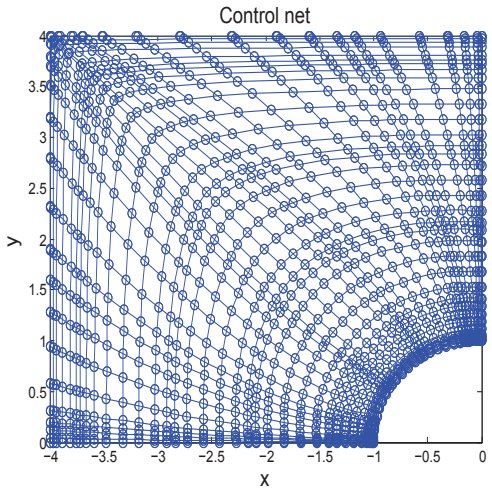
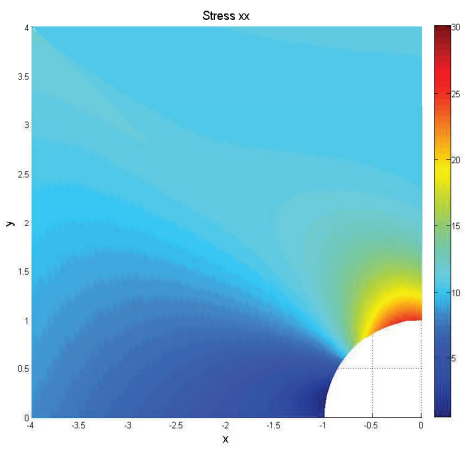
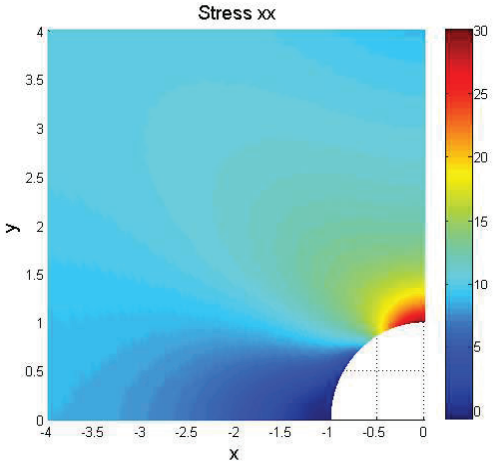


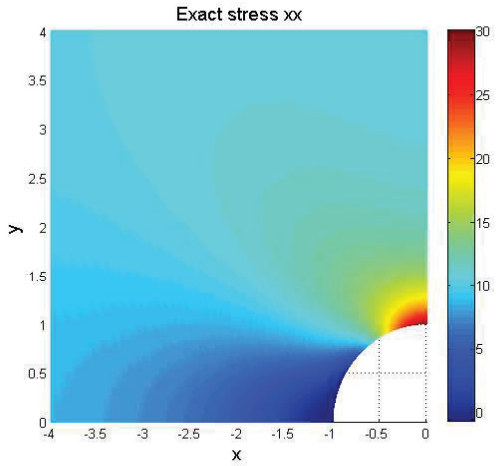
Figure 7.14: *Control net after refinement by knot insertion.*



(a) Numerical stress σ_{xx} when $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ and $\mathcal{H} = \{0, 0, 0, 1, 1, 1\}$

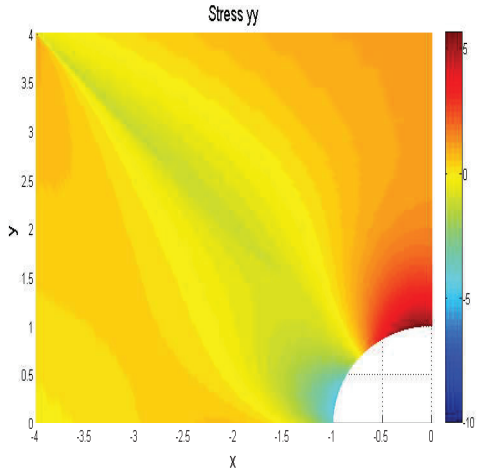


(b) Numerical stress σ_{xx} after refinements

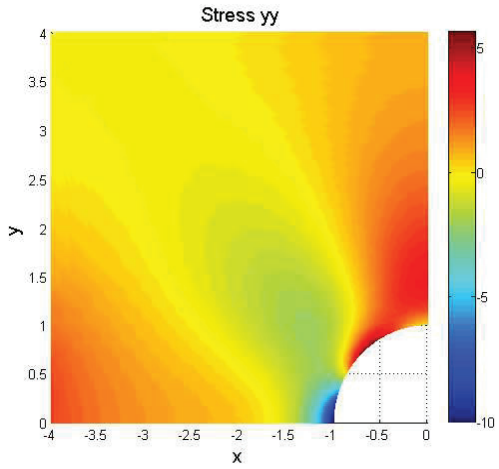


(c) Exact stress σ_{xx}

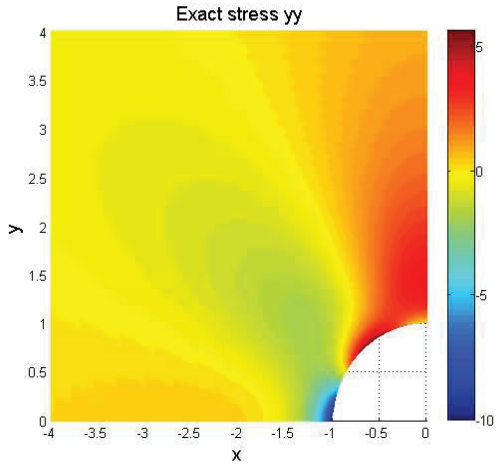
Figure 7.15: *Numerical and exact stress σ_{xx}.*



(a) Numerical stress σ_{yy} when $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ and $\mathcal{H} = \{0, 0, 0, 1, 1, 1\}$

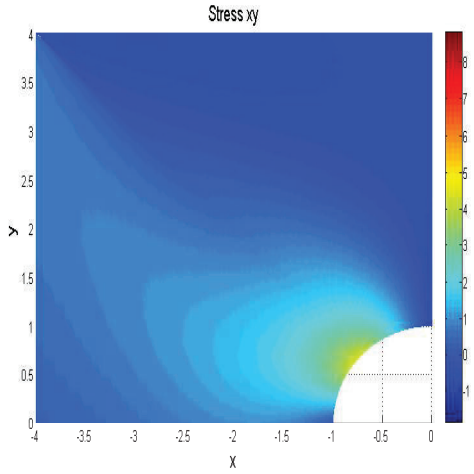


(b) Numerical stress σ_{yy} after refinements

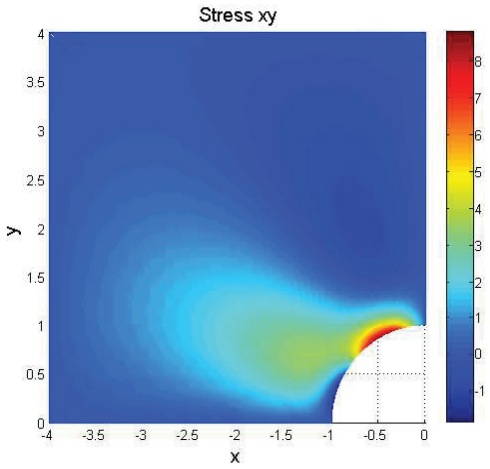


(c) Exact stress σ_{yy}

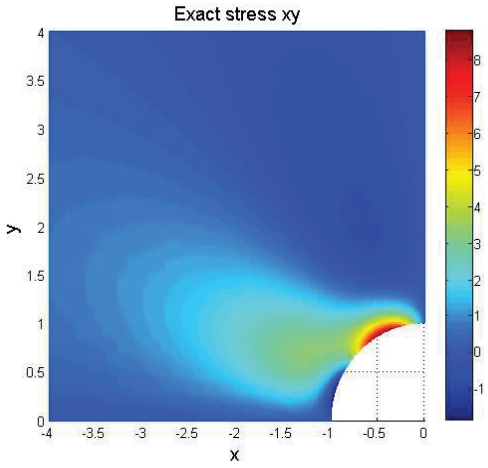
Figure 7.16: *Numerical and exact stress* σ_{yy}.



(a) Numerical stress σ_{xy} when $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ and $\mathcal{H} = \{0, 0, 0, 1, 1, 1\}$



(b) Numerical stress σ_{xy} after refinements



(c) Exact stress σ_{xy}

Figure 7.17: *Numerical and exact stress* σ_{xy} *when* $T_x = 10.0$.

Chapter 8

Concluding remarks

In this thesis we have programmed a NURBS isogeometric finite element solver and studied mappings where quadrilaterals have been degenerated to triangles. We observed that the original quadrilateral had the same number of basis functions as the number of degrees of freedom. When two or more control points coincided our mapping got degenerated and the number of degrees of freedom was reduced. When the quadrilateral was degenerated, but was spanned by the same set of basis functions, the number of basis functions was greater than the number of degrees of freedom. Because of that our degenerated quadrilateral could not be considered as a proper finite element and the finite element method could not be applied. For the finite element $(\mathcal{K}, \mathcal{P}, \mathcal{N})$ we found that the dimension of $\mathcal{P}'$ was reduced as the degrees of freedom was reduced. The dimension of $\mathcal{P}$ was then no longer equal to the dimension of $\mathcal{P}'$, and the basis functions were consequently not linear independent. We also found that the first derivatives of the basis functions at the degeneracy were infinite. As a result, the basis functions were no longer in H^1, meaning that they were not integrable and the stiffness matrix integral would not exist. Also, we would then not be considering the variational problem (3.1), as the assumption would be violated. The bilinear form would no longer be not bounded on V and would not be in V'. Hence, the finite element method could not be applied.

To solve the problems caused by the degenerated mapping we formed a new set of basis functions by adding the basis functions corresponding to the degeneracy. We successfully applied that approach for a degenerated parameterization of a circular surface and of an infinite plate with a circular hole. The new set of basis functions were then all in H^1 and the number of basis functions were the same as the number of degrees of freedom. We found that the new set of basis functions were non-negative, that they formed a partition of unity and that the degenerated element was still a proper finite element. Thus, it still remains to prove if this approach is general and will generate valid basis functions for all types of degenerated mappings.

To be able to evaluate the derivatives correctly we had to add them symbolically and simplify before evaluating. We did this by applying the symbolic toolbox in MATLAB. We have only done calculations on small problems, still we soon discovered that the symbolic calculations are very slow. Especially considering the algorithms to make simplifications and to perform the evaluations. Another downside is that we use a lot of memory when storing symbolic matrices. When doing symbolic calculations one must also be aware of how the evaluations are performed. That we discovered when looking at the quadrilateral in Figure 6.4 with the symbolic value ϵ in addition to ξ and η. When we were evaluating

the derivatives for the case of $\epsilon = 0$, we could not just simplify the derivatives and then evaluate with respect to ϵ, ξ and η. Rather we had to first evaluate the symbolic expression with $\epsilon = 0$, then simplify, and at last evaluate with respect to ξ and η. Else we would only obtain values being infinity or *not a number*. Also, one needs to be attentive when using built in functions to find inverses etc.. At first we used the built in function to find the inverse of the Jacobian matrix. After evaluating the derivatives at the degenerated point we had troubles with the value having the wrong sign. At this point the determinant of the Jacobian is zero. After debugging we discovered that the evaluation algorithm would return a values for the determinant being $-1e-10$ instead of 0.0. When evaluating $\frac{1}{det\mathbf{J}}$ this gave a huge error. We could hence not apply the built in function and had to calculate the inverse of the Jacobian by multiplying $\frac{1}{|\det \mathbf{J}|}$ to the matrix. In our case we knew that the determinant of the Jacobian was positive everywhere, so this method was applicable. We are not sure if there are many other similar cases one can meet when performing symbolic calculations, and can only conclude that one need to be aware of such problems. The most crucial downside of symbolic calculations are however the running time and the use of memory that increases tremendously. Because of that we will not be able to perform complex or huge analysis. We do not recommend to do such calculations in MATLAB, but it could however be interesting to try other programming languages. In future work it also remains to prove if this approach of adding basis functions is general and will generate valid basis functions for all types of degenerated mappings.

There are many degenerations that leads to a triangle. The approach we used to solve this issue seems from our calculations to give good results. However, the execution of it has to be done differently. Based on our experiences, symbolic calculations should be avoided. In future work it might be more interesting to consider parameterizations leading to the mappings not being degenerated. In particular, it would be interesting to look at triangular B-splines [29] and triangular NURBS [14]. There are ongoing research on using triangular B-splines in isogeometric analysis. As the work is in progress we were not able to find any papers on it. However, Gang XU, a Postdoctoral Fellow at the GALAAD Team, INRIA Sophia Antipolis, are currently working on the issue. When the research papers are available they will probably be posted at the urls [39] and [40].

Bibliography

[1] A. Buffa, G. Sangalli, and R. Vázquez. Isogeometric analysis in electromagnetics: B-splines approximation. *Computer Methods in Applied Mechanics and Engineering*, 199:1143–1152, 2010.

[2] A. J. M. Spenser. *Continuum mechanics*. Dover, 2004.

[3] A.P. Nagy, M.M. Abdalla, and Z. Gurda. Isogeometric sizing and shape optimisation of beam structures. *Computer Methods in Applied Mechanics and Engineering*, 199:1216–1230, 2010.

[4] B. Xiao and P. Weng. Integrated analysis of the electromagnetical, thermal, fluid flow fields in a Tokamak. *Fusion Engineering and Design*, 81(8-14):1549–1554, 2006. Proceedings of the Seventh International Symposium on Fusion Nuclear Technology.

[5] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 3. edition, 2008.

[6] D. F. Rogers. *An introduction to NURBS with historical perspective*. Academic Press, 2001.

[7] E. C. Teo, Q. H. Zhang, and R. C. Huang. Finite element analysis of head-neck kinematics during motor vehicle accidents: Analysis in multiple planes. *Medical Engineering and Physics*, 29(1):54–60, 2007.

[8] E. Cohen, T. Martin, R.M. Kirby, T. Lyche, and R.F. Riesenfeld. Analysis-aware modeling: Understanding quality considerations in modeling for isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199:334–356, 2010.

[9] E. Laniado Jácome, J. Meneses Alonso, and V. Diaz López. A study of sliding between rollers and races in a roller bearing with a numerical model for mechanical event simulations. *Tribology International*, 43(11):2175–2182, 2010.

[10] E. Nakamachi, H. Kuramae, H. Sakamoto, and H. Morimoto. Process metallurgy design of aluminum alloy sheet rolling by using two-scale finite element analysis and optimization algorithm. *International Journal of Mechanical Sciences*, 52(2):146–157, 2010.

[11] J. Fish and T. Belytschko. *A First Course in Finite Elements*. Wiley, England, 2008.

[12] Gotools. *http://www.sintef.no/Projectweb/Geometry-Toolkits/GoTools/*.

[13] H. Gomez, V. Calo, and T. J. R. Hughes. Isogeometric Analysis of Phase-Field Models: Application to the Cahn-Hilliard Equation. In *ECCOMAS Multidisciplinary Jubilee Symposium*, volume 14, pages 1–16. Springer Netherlands, 2009.

[14] H. Qin and D. Terzopoulos. Triangular NURBS and their dynamic generalizations. *Computer Aided Geometric Design*, 14(4):325 – 347, 1997.

[15] T. J. R. Hughes. *The Finite Element Method, linear static and dynamic finite element analysis*. Prentice-Hall, 1987.

[16] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric Analysis: CAD, Finite Elements, NURBS, Exact Geometry and Mesh Refinement. *Computer Methods in Applied Mechanics and Engineering*, 194:4135–4195, 2005.

[17] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. *Isogeometric Analysis: Toward Unification of CAD and FEA*. Wiley, 2009.

[18] I. V. Kerlow. *The art of 3D computer animation and effects*. Wiley, 2004.

[19] J. A. Cottrell, T. J. R. Hughes, and A. Realiand G. Sangalli. *Isogeometric discretizations in structural dynamics and wave propagation*, pages 13–16. Number June. 2007.

[20] J. A. Cottrell, Y. Bazilevs, and T. J. R. Hughes. Isogeometric analysis of structural vibrations. *Computer Methods in Applied Mechanics and Engineering*, 195(41-43):5257–5296, 2006.

[21] J. E. Akin. *Finite Element Analysis with error estimators, an introduction to the FEM and adaptive error analysis for engineering students*. Elsevier, 2005.

[22] J. Lu and X. Zhou. Cylindrical element: Isogeometric model of continuum rod. *Computer Methods in Applied Mechanics and Engineering*, 200(1-4):233–241, 2011.

[23] J. Lubliner. *Plasticity theory*. Pearson Education Inc, 1990.

[24] K. A. Johannessen. An adaptive isogeometric finite element analysis. Master's thesis, Norwegian University of Science and Technology, June 2009.

[25] L.A.M. Mendes and L.M.S.S. Castro. Hybrid-mixed stress finite element models in elastoplastic analysis. *Finite Elements in Analysis and Design*, 45(12):863 – 875, 2009.

[26] P. S. Larsen. A comparison of accuracy and computational efficiency between the finite element method and the isogeometric analysis for two dimensional Poisson problems. Master's thesis, Norwegian University of Science and Technology, June 2009.

[27] T. Lyche and K. Mørken. Spline Methods. Lecture notes in INF-MAT5340011 Spline Methods at the University of Oslo, May 2008.

[28] MATLAB symbolic toolbox. *http://www.mathworks.com/help/toolbox/symbolic/*.

[29] P. Fong and H.-P. Seidel. An implementation of triangular B-spline surfaces over arbitrary triangulations. *Computer Aided Geometric Design*, 10(3-4):267 – 275, 1993.

[30] R. C. Almeida, R. A. Feijóo, A. C. Galeão, C. Padra, and R S. Silva. Adaptive finite element computational fluid dynamics using an anisotropic error estimator. *Computer Methods in Applied Mechanics and Engineering*, 182(3-4):379–400, 2000.

[31] Homepage of Rhinoceros. *http://www.rhino3d.com/*.

[32] E. Rønquist, A T. Patera, and T. Kvamsdal. Lecture notes in TMA4220 *Numerical solution of partial differential equations by finite element methods, Norwegian University of Science and Technology* .

[33] S. Lipton, J. A. Evans, Y. Bazilevs, T. Elguedj, and T. J. R. Hughes. Robustness of isogeoemtric structural discretizations under severe mesh distortion. *Computer Methods in Applied Mechanics and Engineering*, 199:357–373, 2010.

[34] S.R. Idelsohn, E. Oñate, F. Del Pin, and N. Calvo. Fluid-structure interaction using the particle finite element method. *Computer Methods in Applied Mechanics and Engineering*, 195(17-18):2100–2123, 2006. Fluid-Structure Interaction.

[35] Exact stress solution for infinite plate with circular hole. *http://en.wikiversity.org/wiki/Introduction_to_Elasticity/Plate_with_hole_in_tension*.

[36] T. Takacs. Existens of stiffness matrix integrals for singularly parametrized domains in isogeometric analysis. November 2010.

[37] W. Slaughter. *The linearized theory of elastscity*. Birkhäuser, 2002.

[38] X. Qian. Full analytical sensitivities in NURBS based isogeometric shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 199:2059–2071, 2010.

[39] G. XU. GALAAD Team, INRIA Sophia Antipolis, Isogeometric analysis using triangular B-splines. *http://www-sop.inria.fr/members/Gang.Xu/English/isogeometric%20analysis.html*.

[40] G. XU. GALAAD Team, INRIA Sophia Antipolis, Isogeometric analysis using triangular B-splines. *http://www-sop.inria.fr/galaad/wiki/index.php/Isogeometric_analysis_and_shape_optimization*.

[41] Y. Bazilevs and T .Hughes. NURBS-based isogeometric analysis for the computation of flows about rotating components. *Computational Mechanics*, 43:143–150, 2008.

[42] Y. Bazilevs, V. Calo, Y. Zhang, and T. Hughes. Isogeometric Fluid-structure Interaction Analysis with Applications to Arterial Blood Flow. *Computational Mechanics*, 38:310–322, 2006.

[43] Young and Freedman. *University Physics*. PEARSON Addison Weasley, USA, 2004.

[44] N. Young. *An introduction to Hilbert space*. Cambridge, 1988.

[45] Y. Zhang, Y. Bazilevs, S. Goswami, C. Bajaj, and T. Hughes. Patient-Specific Vascular NURBS Modeling for Isogeometric Analysis of Blood Flow. In *Proceedings of the 15th International Meshing Roundtable*, pages 73–92. Springer Berlin Heidelberg, 2006.

Appendix A

Implementations in MATLAB; Isogeometric finite element solver for infinite plate with circular hole

A.1 Main method

```
function FEM()

%nel     number of elements
%nnp     number of global basis functions
%nen     number of local basis functions
%INC     NURBS coordinates array
%IEN     Connenctivty array
% ContrPts Control points

[knot_xi, knot_eta, p_xi, p_eta, ContrPts,n,m]=geometry();

%build connenctivities and allocate global arrays
[INC,IEN,nel,nnp,nen]=buildArrays(n,m,p_xi,p_eta);

%Prescribe body forces
Fbody_y=0*ones(nnp,1);
Fbody_x=0*ones(nnp,1);

Fbody=zeros(2*nnp,1);
Fbody(1:2:2*nnp-1)=Fbody_x;
Fbody(2:2:2*nnp)=Fbody_y;


[dof,ud,ndof,D,tractionboundary,tpre,boundary_west,
    boundary_north]=preprocessor(INC,IEN,nel,nnp,nen,ContrPts);
```

```
Support_element_boundary_west=6;
Support_element_boundary_north=6;

[A,F,DBGausspt,ngp,Mass]=computation(INC,IEN,nel,nnp,nen,dof,ud,
    ndof,knot_xi,knot_eta, p_xi,p_eta,ContrPts,D,tractionboundary
    ,tpre,boundary_west,boundary_north,Fbody,n,m,
    Support_element_boundary_west,Support_element_boundary_north)
    ;

%Solve Au=F by the conjugate gradient method
tol=1e-6;
maxit=50;
U=pcg(A,F,tol,maxit);
solution=ud;

%Expanding U to include the nodes at the Dirichlet boundary
for i=1:2*nnp
        if dof(i)>0
        solution(i)=U(dof(i));
        end
end

postprocessor(D,solution,INC,IEN,nen,nel,nnp,DBGausspt,ngp,
    ContrPts,n,m, p_xi,p_eta,knot_xi,knot_eta,U);
end
```

A.2 Pre-prosessor

A.2.1 Geometry - Infinite plate with circular hole

```
function [knot_xi, knot_eta, p_xi, p_eta, B,n,m]=geometry()

knot_xi=[0,0,0,0.5,1,1,1];
knot_eta=[0,0,0,1,1,1];
p_xi=2;
p_eta=2
n=length(knot_xi)-p_xi-1;
m=length(knot_eta)-p_eta-1;
s=(1+1/sqrt(2))/2;
B = zeros(n,m,3);

B(:,1,1) = [-1 -1 1-sqrt(2) 0];
B(:,1,2) = [0 sqrt(2)-1 1 1];
B(:,1,3) = [1 s s 1];

B(:,2,1) = [-2.5 -2.5 -0.75 0];
```

```
B(:,2,2) = [0  0.75   2.5  2.5];
B(:,2,3) = [1  1  1    1];

B(:,3,1) = [-4  -4   -4  0];
B(:,3,2) = [0  4  4    4];
B(:,3,3) = [1  1  1  1];

end
```

A.2.2 Building the connectivity arrays

```
function [INC,IEN,nel,nnp,nen]=buildArrays(n,m,p,q)
%number of elements
nel=(n-p)*(m-q);
%number of global basis functions
nnp=n*m;
%number of local basis functions
nen=(p+1)*(q+1);
%NURBS coordinates array
INC=zeros(nnp,2);
%Connenctivty array
IEN=zeros(nen,nel);
%increment global function number
A=0;
%increment element number
e=0;
%global function number
B=0;
%local function number
b=0;
for j=1:m
    for i=1:n
        A=A+1;
        %assign NURBS coordinates
        INC(A,1)=i;
        INC(A,2)=j;
        if (i-(p+1)>-1e-10 && j-(q+1)>-1e-10)
            e=e+1;
            for jloc=0:q
                for iloc=0:p
                    B=A-jloc*n-iloc;
                    b=jloc*(p+1)+iloc+1;
                    IEN(b,e)=B;
                end
            end
        end
    end
```

```
end
end
```

A.2.3 Pre-prosessor

```
function [dof,ud,ndof,D,tractionboundary,tpre,boundary_west,
    boundary_north]=preprocessor(INC,IEN,nel,nnp,nen,ContrPts)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Description of variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %nel     number of elements
    %nnp     number of global basis functions
    %nen     number of local basis functions
    %INC     NURBS coordinates array   size=(nnp,2)
    %IEN     Connenctivty array        size=(nen,nel)

    %D - constitutive matrix. Hooke's law: Stress=D*Strain
    %E - Young's modulus [Pa]
    %ny - Poisson's ration
    %tractionboundary - boundary with prescribed traction
    %strain - [strain_xx;strain_yy;strain_xy]
    %tpre - prescribed traction [stress_xx+stress_xy;stress_yy+
        stress_xy]
    %dof - vector that tags the nodes at the dirichlet boundary
    %ud - prescribed displacement
    %ndof - number of degrees of freedom
    %nodesboundary_west - stores the nodes on the part of the
        boundary called boundary_west
    %nodesboundary_north - stores the nodes on the part of the
        boundary called boundary_north

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%INPUT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%remember to make 'tractionboundary', 'tpre' and prescribed
    displacement fit the computational domain

%Young's modulus [Pa]
E=10^5;

%Poisson's ration
ny=0.3;

%Constitutive matrix; plane stress
D=(E/(1-ny^2))*[1 ny 0;ny 1 0;0 0 0.5*(1-ny)];
```

```
%Constitutive matrix; plane strain
%D=(E/(1+ny)*(1-2*ny))*[1-ny ny 0;ny 1-ny 0;0 0 0.5*(1-2*ny)];

%Tractionboudary
tractionboundary=[1 2]; %[boundary_west boundary_north]

% %Prescribed strain
% strain=[-1;-1;-1];
% %Calculating the prescribed traction knowing the prescribed
    strain
% stress=D*strain;
% tx=stress(1)+stress(3);
% ty=stress(2)+stress(3);
% tpre=[tx tx;0 0]; %ex: tpre=[tx_tractionboundary(1) tx_trb(2);
    ty_trb(2) ty_trb(2)];

tpre=[0 0;0 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nnpxy=nnp*2;

%Number of degrees of freedom
ndof=0;

%dof tags the nodes at the dirichlet boundary
dof=-1*ones(nnpxy,1);

%ud stores the already known values of u
ud=zeros(nnpxy,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Finding the boundary nodes, the nodes at the interface frame,
    the degrees of freedom and the values in ud
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
boundary_west=[];
boundary_north=[];

degdof=-1;
k=1;
for i=1:nnp
    inc=INC(i,:);
    Cx=ContrPts(inc(1),inc(2),1);
    Cy=ContrPts(inc(1),inc(2),2);
    if Cx>-1e-10
        ud(k)=0;
    else
        ndof=ndof+1;
```

```
        dof(k)=ndof;
    end

    if Cy<1e-10
        ud(k+1)=0;
    else
        ndof=ndof+1;
        dof(k+1)=ndof;
    end

     if Cx+4<1e-10
        boundary_west=[boundary_west k k+1];
     end

     if Cy-4>-1e-10
        boundary_north=[boundary_north k k+1];
     end

     if(Cx+4<1e-10 && Cy-4>-1e-10)
                          if degdof<0
                 degdof=ndof;
      else
           dof(k)=degdof-1;
           dof(k+1)=degdof;
           ndof=ndof-2;
      end
     end

    k=k+2;
end

end
```

A.3 Computations

A.3.1 Calculating the basis functions and their derivatives

```
function [N N_diff]=getBasisandDerivatives(p_xi,xi,knot_xi,d)
%d=1 returs the first derivative, d=2 returns the second
   derivative etc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Function to calculate basis functions recursively
%-------------------------------------------------------------------------------
function N=calcBspline(knot,xi,i,p,pstatic)
if(i-(length(knot)-pstatic)<-1e-10)
if (M(i,p+1)<inf)
```

```
    N=M(i,p+1);
else
    if (p>0)
            if (knot(i+p)-knot(i))<1e-10
                a=0;
            else
                a=((xi-knot(i))/(knot(i+p)-knot(i)));
            end
            if (knot(i+p+1)-knot(i+1))<1e-10
                b=0;
            else
                b=((knot(i+p+1)-xi)/(knot(i+p+1)-knot(i+1)));
            end
                N=a*calcBspline(knot,xi,i,p-1,pstatic)+b*
                    calcBspline(knot,xi,i+1,p-1,pstatic);
     elseif (p<1e-10)
            if((xi-knot(i)>-1e-15) && (knot(i+1)-xi>1e-15))
                N=1;
            elseif((xi-knot(i)>-1e-15) && (knot(i+1)-xi>-1e-15)
               && (length(knot)-pstatic-1)-i<1e-10)
                N=1;
            else
                N=0;
            end
    end
    M(i,p+1)=N;
end
else
N=0;
end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate all basis functions for all values of xi and eta
%-------------------------------------------------------------------------

B_xi=inf*ones(length(knot_xi)-p_xi-1,length(xi));
pstatic_xi=p_xi;
dB_xi=inf*ones(length(knot_xi)-p_xi-1,length(xi));
N=zeros(p_xi+1,1);
N_diff=zeros(p_xi+1,1);

for j=1:length(xi);
M=inf*ones(length(knot_xi)-p_xi-1,p_xi+1);
    for k=1:length(knot_xi)-p_xi-1
        B_xi(k,j)=calcBspline(knot_xi,xi(j),k,p_xi,pstatic_xi);
```

```
    end
    if(d==1)
        for k=1:length(knot_xi)-p_xi-1
            dB_xi(k,j)=FirstDerivBspline(knot_xi,k,p_xi,M,
                pstatic_xi);
        end
    else
        for k=1:length(knot_xi)-p_xi-1
            dB_xi(k,j)=DerivBspline(knot_xi,k,p_xi,M,pstatic_xi,
                d);
        end
    end
        N=B_xi;
        N_diff=dB_xi;
end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Function to calculate the first derivative of basis functions
%-------------------------------------------------------------------------
function dN=FirstDerivBspline(knot,i,p,M,pstatic)
if(i-(length(knot)-pstatic)<-1e-10)
        if(knot(i+p)-knot(i))<1e-10
            a=0;
        else
            a=(p/(knot(i+p)-knot(i)));
        end
        if (knot(i+p+1)-knot(i+1))<1e-10
            b=0;
        else
                b=(p/(knot(i+p+1)-knot(i+1)));
        end
        if(i-(length(knot)-pstatic-1)<-1e-10)
            dN=a*M(i,p)-b*M(i+1,p);
        elseif(i-(length(knot)-pstatic)<-1e-10)
                dN=a*M(i,p);
        end
end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Function to calculate the d-derivative of basis functions
%-------------------------------------------------------------------------
function dNk=DerivBspline(knot,i,p,M,pstatic,m)
sum=0;
```

```
fac=(factorial(p)/factorial(p-m));
    for l=0:m
    if(i+l-(length(knot)-pstatic)<-1e-10)
%           if A(m+1,l+1)<inf
%               sum=sum+fac*A(m+1,l+1)*M(i+l,p-m+1);
%           else
            sum=sum+fac*alpha(m,l,i,p,knot)*M(i+l,p-m+1);
%           end
    end
    end
dNk=sum;
end

function a=alpha(k,j,i,p,knot)
if(i+j-(length(knot)-p)<-1e-10)
    %if (A(k+1,j+1)<inf)
    %    a=A(k+1,j+1);
    %else
        if(k>-1e-10 && j>-1e-10)
            if (k<1e-10 && j<1e-10)
                a=1;
            elseif (k>1e-10 && j<1e-10)
                if(knot(i+p-k+1)-knot(i)<1e-10)
                    a=0;
                else
                    a=alpha(k-1,j,i,p,knot)/(knot(i+p-k+1)-knot(
                        i));
                end
            elseif (k>1e-10 && j>1e-10 && j-(k-1)<1e-10)
                if(knot(i+p+j-k+1)-knot(i+j)<1e-10)
                    a=0;
                else
                    a=(alpha(k-1,j,i,p,knot)-alpha(k-1,j-1,i,p,
                        knot))/(knot(i+p+j-k+1)-knot(i+j));
                end
            elseif (k>1e-10 && j>1e-10 && k-j<1e-10)
                    if(knot(i+p+1)-knot(i+k)<1e-10)
                    a=0;
                    else
                        a=(-alpha(k-1,k-1,i,p,knot))/(knot(i+p
                            +1)-knot(i+k));
                    end
            end

        end
       % A(k+1,j+1)=a;
    %end
```

```
end
end
```

A.3.2 Shape function routine

```
function [R dR_dx J_mat]=Shape_function(ni,nj,xi,eta,knot_xi,
    knot_eta,ContrPts,p_xi,p_eta,nen)

%Unvariate B-spline basis functions and derivatives wrt
    parametric coord
[N_xi dN_xi]=getBasis&Derivatives(p_xi,xi,knot_xi,1);
[N_eta dN_eta]=getBasis&Derivatives(p_eta,eta,knot_eta,1);
support_xi=zeros(p_xi+1,1);
if ni-p_xi>1e-10
     for i=0:p_xi
     support_xi(i+1)=ni-p_xi+i;
     end
end
support_eta=zeros(p_eta+1,1);
if nj-p_eta>1e-10
     for i=0:p_eta
     support_eta(i+1)=nj-p_eta+i;
     end
end
N_xi=N_xi(support_xi);
dN_xi=dN_xi(support_xi);
N_eta=N_eta(support_eta);
dN_eta=dN_eta(support_eta);

%trivariate NURBS basis functions
R=zeros(1,nen);
%trivariate NURBS functions derivatives wrt physical coord
dR_dx=zeros(2,nen);
%trivariate NURBS functions derivatives wrt parametric coord
dR_dxi=zeros(2,nen);
%Derivative of physical coord wrt parametric coord
dx_dxi=zeros(2,2);
%Inverse of dx_dxi
dxi_dx=zeros(2,2);

%Derivative of parametric coord wrt parent element coord
dxi_dtildexi=zeros(2,2);
%Jacobian matrix
J_mat=zeros(2,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
W=0;
dW_dxi=0;
dW_deta=0;
for j=0:p_eta
    for i=0:p_xi
        w=ContrPts(ni-i, nj-j, 3);
        W=W+N_xi(p_xi+1-i)*N_eta(p_eta+1-j)*w;
        dW_dxi=dW_dxi+dN_xi(p_xi+1-i)*N_eta(p_eta+1-j)*w;
        dW_deta=dW_deta+N_xi(p_xi+1-i)*dN_eta(p_eta+1-j)*w;
    end
end

        index_R=0;
        for j=0:p_eta
            for i=0:p_xi
                index_R=index_R+1;
                w=ContrPts(ni-i, nj-j, 3);
                R(index_R)=w*(N_xi(p_xi+1-i)*N_eta(p_eta+1-j))/W
                    ;
                dR_dxi(1,index_R)=w*(dN_xi(p_xi+1-i)*N_eta(p_eta
                    +1-j)*W-dW_dxi*N_xi(p_xi+1-i)*N_eta(p_eta+1-j
                    ))/(W^2);
                dR_dxi(2,index_R)=w*(N_xi(p_xi+1-i)*dN_eta(p_eta
                    +1-j)*W-dW_deta*N_xi(p_xi+1-i)*N_eta(p_eta+1-
                    j))/(W^2);
            end
        end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Gradient mapping from parameter space to physical space
loc_num=0;
for j=0:p_eta
    for i=0:p_xi
        loc_num=loc_num+1;
        for a=1:2
            for b=1:2
                dx_dxi(a,b)=dx_dxi(a,b)+ContrPts(ni-i,nj-j,a)*
                    dR_dxi(b,loc_num);
            end
        end

    end
end


%Compute inverse of gradient
```

```
dxi_dx=inv(dx_dxi);

%Compute derivatives of basis functions wrt physical coord
dR_dx=dxi_dx'*dR_dxi;

%Gradient mapping from parent element to parameter space
dxi_dtildexi(1,1)=(knot_xi(ni+1)-knot_xi(ni))/2;
dxi_dtildexi(2,2)=(knot_eta(nj+1)-knot_eta(nj))/2;

%Compute Jacobian matrix
J_mat=dx_dxi*dxi_dtildexi;

end
```

A.3.3 Calculate the Gaussian quadrature points and weights

```
function Gauss=GaussianQuadrature(ngp)
syms x real

p=vpa(zeros(ngp,1));
p(1)=1;
p(2)=x;
n=1;
if ngp>1
    for i=2:ngp
        p(i+1)=(1/(n+1))*((2*n+1)*x*p(i)-n*p(i-1));
        n=n+1;
    end
end

Xt=solve(p(ngp+1));
X=real(double(Xt));
dp=diff(p(ngp+1));
wi=zeros(length(X),1);
w=2/((1-x^2)*dp^2);

for i=1:length(X)
    x=X(i);
    wi(i)=real(eval(w));
end

Gauss=[X wi];
end
```

A.3.4 Computations

```
function [A,F,DBGausspt,ngp,Mass]=computation(INC,IEN,nel,nnp,
    nen,dof,ud,ndof,knot_xi,knot_eta, p_xi,p_eta,ContrPts,D,
```

```
    tractionboundary,tpre,boundary_west,boundary_north,Fbody,n,m,
    Support_element_boundary_west,Support_element_boundary_north)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
F = zeros(ndof,1);
A = sparse(ndof,ndof);
Mass = sparse(ndof,ndof);

%number of Gauss points
ngp=5;
Gauss=GaussianQuadrature(ngp);

DBGausspt=zeros(3,ngp*ngp*nen*2,2);

%Expand matrix to include both x- and y-coordinate
function M=ExpandMatrix(Mtemp)
        Size=size(Mtemp);
        M=sparse(Size(1)*2,Size(2)*2);
        i1=1:2:Size(1)*2;
        i2=2:2:Size(1)*2;
        j1=1:2:Size(2)*2;
        j2=2:2:Size(2)*2;
        h=1:Size(1);
        kk=1:Size(2);
        M(i1,j1)=Mtemp(h,kk);
        M(i2,j2)=Mtemp(h,kk);
end

%Mapping from parent element to parameter space
function xi=parameter(knot_xi, xitilde, ni)
        xi=((knot_xi(ni+ones(length(ni),1))-knot_xi(ni))*xitilde
           +(knot_xi(ni+ones(length(ni),1))+knot_xi(ni)))/2;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k = 1:nel  % loop through elements
    indexGP=1:2*nen;

    %NURBS coordinates
    ni=INC(IEN(1,k),1);
    nj=INC(IEN(1,k),2);

    %Check if element has zero measure
    if(knot_xi(ni+1)==knot_xi(ni) || knot_eta(nj+1)==knot_eta(nj
       ))
        continue;
    end
```

```
%global numbering
globalnodestemp=IEN(:,k);
    for i=1:nen
    globalnodes(i*2)=globalnodestemp(i)*2;
    globalnodes(2*i-1)=globalnodestemp(i)*2-1;
end

    %Body forces on element k
    FEbody=Fbody(globalnodes);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Computing Ak, FEbody and FEtraction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Ak=zeros(2*nen,2*nen);
    Mk=zeros(2*nen,2*nen);
    tk=zeros(2*nen,1);

for alfa=1:ngp %for all gauss points
    for beta=1:ngp
        %Parametric coord
        xi=parameter(knot_xi, Gauss(alfa,1), ni);
        eta=parameter(knot_eta, Gauss(beta,1), nj);

        %Calculate basis functions and derivatives
        [R dR_dx J]=Shape_function(ni,nj,xi,eta,knot_xi,
            knot_eta,ContrPts,p_xi,p_eta,nen);

        Jmod=Gauss(alfa,2)*Gauss(beta,2)*det(J);

        %Find the B matrix
        Btemp=dR_dx;
        B=zeros(3,nen*2);
        i=2:2:2*nen;
        j=1:2:2*nen-1;
        B(1,j)=Btemp(1,:);
        B(2,i)=Btemp(2,:);
        B(3,j)=Btemp(2,:);
        B(3,i)=Btemp(1,:);

        %Computing D*B in the gauss points
        DBGausspt(:,indexGP,k)=D*B;
        indexGP=indexGP+2*nen*ones(1,2*nen);

        %Compute the element stiffness matrix by gaussian
            quadrature
```

```
        Ak=Ak+B'*D*B*Jmod;

        %Find the mass matrix
        Mtemp=zeros(nen,nen);
        for i=1:nen
            for j=1:nen
                Mtemp(i,j)=R(i)*R(j)*Jmod;
            end
        end
        M=ExpandMatrix(Mtemp);
        Mk=Mk+M;

    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %Find contribution from traction;
    eta_boundary=1;%=parameter(knot_eta, 1, nj);
    for i=1:length(tractionboundary)
        tktemp=zeros(nen,2);
        if tractionboundary(i)==1
            gamma=0;
            for j=1:2*nen
                for l=1:length(boundary_west)
                    if globalnodes(j)==boundary_west(l)
                    gamma=gamma+1;
                    end
                end
            end
            if gamma==Support_element_boundary_west
                [Rtrb1 dR_dxtrb1 Jtrb1]=Shape_function(ni,nj
                    ,xi,eta_boundary,knot_xi,knot_eta,
                    ContrPts,p_xi,p_eta,nen);
                for j=0:nen-1
                    tkt=Rtrb1(j+1);
                    tktemp(1+2*j,1)=tkt;
                    tktemp(2+2*j,2)=tkt;
                end
                LengthJTrb1=sqrt(Jtrb1(1,1)^2+Jtrb1(2,1)^2);
                %%%%%%%%%%
                x1=-4;
                y1=0;
                ri=1;
                for b=0:p_eta
                    for a=0:p_xi
                        y1=y1+Rtrb1(ri)*ContrPts(ni-a,nj-b
                            ,2);
                        ri=ri+1;
```

```
                    end
                end
                stress1=g_neumann(x1,y1,[-1;0]);
                tk=tk+tktemp*stress1*Gauss(alfa,2)*
                    LengthJTrb1;
                %%%%%%%%%%
            end
        elseif tractionboundary(i)==2
            gamma=0;
            for j=1:2*nen
                for l=1:length(boundary_north)
                    if globalnodes(j)==boundary_north(l)
                    gamma=gamma+1;
                    end
                end
            end
            if gamma==Support_element_boundary_north
                [Rtrb2 dR_dxtrb2 Jtrb2]=Shape_function(ni,nj
                    ,xi,eta_boundary,knot_xi,knot_eta,
                    ContrPts,p_xi,p_eta,nen);
                for j=0:nen-1
                    tkt=Rtrb2(j+1);
                    tktemp(1+2*j,1)=tkt;
                    tktemp(2+2*j,2)=tkt;
                end
                LengthJTrb2=sqrt(Jtrb2(1,1)^2+Jtrb2(2,1)^2);
                %%%%%%%%%%
                x2=0;
                y2=4;
                ri=1;
                for b=0:p_eta
                    for a=0:p_xi
                        x2=x2+Rtrb2(ri)*ContrPts(ni-a,nj-b
                            ,1);
                        ri=ri+1;
                    end
                end
                stress2=g_neumann(x2,y2,[0;1]);
                tk=tk+tktemp*stress2*Gauss(alfa,2)*
                    LengthJTrb2;
                %%%%%%%%%%
            end
        end
    end
end

    FEtraction=tk;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Find the contribution to F and A from element k
        Fk=-Ak*(ud(globalnodes))+Mk*FEbody+FEtraction;
        for alfa=1:2*nen
        i=dof(globalnodes(alfa));
        if i>0
            F(i)=F(i)+Fk(alfa);
        for beta=1:2*nen
            j=dof(globalnodes(beta));
            if j>0
                A(i,j)=A(i,j)+Ak(alfa,beta);
                Mass(i,j)=Mass(i,j)+Mk(alfa,beta);
            end
        end
        end
    end
 end %end loop element k
 A=(A+A')/2;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate prescribed traction
%-------------------------------------------------------------------------

function ret_val = g_neumann(x,y,n)

epsilon = 1e-10;
grad_u = grad_u_exact(x,y);
n = n/norm(n);

if abs(n(1)+1)<epsilon,                    % gamma1
        ret_val = [grad_u(1)*n(1); grad_u(3)*n(2)]; % s_xx*n(1);
            s_xy*n(2)
elseif abs(n(2)-1)<epsilon,                % gamma2
        ret_val = [grad_u(3)*n(1); grad_u(2)*n(2)]; % s_yx*n(1);
            s_yy*n(2)
else
        ret_val = [0;0];
end


function grad_u = grad_u_exact(x, y)
T =10.0;
r2    = x.^2+y.^2;
theta = atan2(y,x);
R2 = 1;%=R^2, where R is the radius of the hole
```

```
sigma_dxdx = T*(1 -R2./r2.*(3/2*cos(2*theta)+cos(4*theta)) +
   3/2*(R2./r2).^2.*cos(4*theta));
sigma_dydy = T*(-R2./r2.*(1/2*cos(2*theta)-cos(4*theta)) - 3/2*(
   R2./r2).^2.*cos(4*theta));
sigma_dxdy = T*(-R2./r2.*(1/2*sin(2*theta)+sin(4*theta)) + 3/2*(
   R2./r2).^2.*sin(4*theta));

if length(x)==1 && length(y)==1,
        grad_u = [sigma_dxdx; sigma_dydy; sigma_dxdy];
else
        [width bredth] = size(x); % == size(y)
        grad_u = zeros(width, bredth, 3);
        grad_u(:,:,1) = sigma_dxdx;
        grad_u(:,:,2) = sigma_dydy;
        grad_u(:,:,3) = sigma_dxdy;
end
end
```

A.4 Post-processor

```
function postprocessor(D,solution,IEN,nen,nel,nnp,DBGausspt,
   ContrPts,n,m, p_xi,p_eta,knot_xi,knot_eta)

nnpxy=nnp*2;
sx=1:2:nnpxy-1;
sy=2:2:nnpxy;
solx=zeros(length(knot_xi)-p_xi-1,length(knot_eta)-p_eta-1);
soly=zeros(length(knot_xi)-p_xi-1,length(knot_eta)-p_eta-1);
index=1;

for j=1:m
    for i=1:n
        solx(i,j)=solution(sx(index));
        soly(i,j)=solution(sy(index));
        index=index+1;
    end
end


%visualization points
xi_eval=[0:0.05:1];
eta_eval=[0:0.05:1];

u=zeros(length(xi_eval),length(eta_eval),2);
xy=zeros(length(xi_eval),length(eta_eval),2);
R_eval=zeros(length(xi_eval),length(eta_eval),n,m);
```

```
dR_dx_eval=zeros(length(xi_eval),length(eta_eval),n,m,2);
TotalStresseval=zeros(length(xi_eval),length(eta_eval),3);

 for alfa=1:length(xi_eval)% loop through evaluation points
     for beta=1:length(eta_eval)

         dR_dxi=zeros(1,nnp);
             [N_xi dN_xi]= getBasisandDerivatives(p_xi, xi_eval(
                alfa), knot_xi,1);
             [N_eta dN_eta]= getBasisandDerivatives(p_eta,
                eta_eval(beta), knot_eta,1);
            W=N_xi'*ContrPts(:,:,3)*N_eta;
         dW_dxi=dN_xi'*ContrPts(:,:,3)*N_eta;
         dR_deta=zeros(1,nnp);
         dx_dxi=zeros(2,2);
         dR_dx=zeros(2,nnp);

         dW_deta=N_xi'*ContrPts(:,:,3)*dN_eta;
         index_R=1;
         for j=1:m
             for i=1:n
                 R=(N_xi(i)*N_eta(j)*ContrPts(i,j,3))/W;
                 R_eval(alfa,beta,i,j)=R;
                 u(alfa,beta,1)=u(alfa,beta,1)+R*solx(i,j);
                 u(alfa,beta,2)=u(alfa,beta,2)+R*soly(i,j);
                 dR_dxi(index_R)=ContrPts(i,j,3)*(dN_xi(i)*N_eta(
                    j)*W-dW_dxi*N_xi(i)*N_eta(j))/(W^2);
                 dR_deta(index_R)=ContrPts(i,j,3)*(N_xi(i)*dN_eta
                    (j)*W-dW_deta*N_xi(i)*N_eta(j))/(W^2);
                 xy(alfa,beta,1)=xy(alfa,beta,1)+R*ContrPts(i,j
                    ,1);
                 xy(alfa,beta,2)=xy(alfa,beta,2)+R*ContrPts(i,j
                    ,2);
                 index_R=index_R+1;
             end
         end
         dR_dxiv=[dR_dxi;dR_deta];

         %Gradient mapping from parameter space to physical space
         loc_num=0;
         for j=1:m
             for i=1:n
                 loc_num=loc_num+1;
                 for a=1:2
                     for b=1:2
                         dx_dxi(a,b)=dx_dxi(a,b)+ContrPts(i,j,a)*
                            dR_dxiv(b,loc_num);
```

```
                end
            end
        end
    end

    %Compute inverse of gradient
    dxi_dx=inv(dx_dxi);

    %Compute derivatives of basis functions wrt physical
       coord
    dR_dx=dxi_dx'*dR_dxiv;

    index=1;
    for j=1:m
        for i=1:n
            dR_dx_eval(alfa,beta,i,j,1)=dR_dx(1,index);
            dR_dx_eval(alfa,beta,i,j,2)=dR_dx(2,index);
            index=index+1;
        end
    end


    %Find the B matrix
    Btempeval=dR_dx;
    Beval=zeros(3,nnp*2);
    i=2:2:2*nnp;
    j=1:2:2*nnp-1;
    Beval(1,j)=Btempeval(1,:);
    Beval(2,i)=Btempeval(2,:);
    Beval(3,j)=Btempeval(2,:);
    Beval(3,i)=Btempeval(1,:);

    TotalStresseval(alfa,beta,:)=D*Beval*solution;

  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculating stress in Gauss points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ke=2*nen*ones(2*nen,1);
globalnodes=[];
index=1;

for k=1:nel
    je=[1:2*nen]';
    globalnodestemp=IEN(:,k);
```

```
        for i=1:nen
        globalnodes(i*2)=globalnodestemp(i)*2;
        globalnodes(2*i-1)=globalnodestemp(i)*2-1;
    end
    h=size(DBGausspt(:,:,k));
        while je(2*nen)<h(2)+1
        TotalStress(:,index)=DBGausspt(:,je,k)*solution(
            globalnodes);
        je=je+ke;
        index=index+1;
    end
end
max_stress=max(TotalStress(1,:))
min_stress=min(TotalStress(1,:))
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Visualization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(1)
plot(ContrPts(:,:,1), ContrPts(:,:,2), 'bo-');
hold on
plot(ContrPts(:,:,1)', ContrPts(:,:,2)', 'bo-');
hold off
  title('Control net','fontsize', 14)
  xlabel('x','fontsize', 14)
  ylabel('y','fontsize', 14)

figure(2)
plot(xy(:,:,1), xy(:,:,2), 'bo-');
hold on
plot(xy(:,:,1)', xy(:,:,2)', 'bo-');
hold off
  title('Physical mapping','fontsize', 14)
  xlabel('x','fontsize', 14)
  ylabel('y','fontsize', 14)



 figure(3)
 surf(xy(:,:,1),xy(:,:,2),u(:,:,1))
 colormap jet
 shading interp
 colorbar
 title('Displacement in x-direction','fontsize', 14)
 xlabel('x','fontsize', 14)
 ylabel('y','fontsize', 14)
```

```
 figure(4)
 surf(xy(:,:,1),xy(:,:,2),u(:,:,2))
 colormap jet
 shading interp
 colorbar
 title('Displacement in y-direction','fontsize', 14)
 xlabel('x','fontsize', 14)
 ylabel('y','fontsize', 14)

 figure(5)
 surf(xy(:,:,1),xy(:,:,2),TotalStresseval(:,:,1))
 colormap jet
 shading interp
 colorbar
 title('Stress xx','fontsize', 14)
 xlabel('x','fontsize', 14)
 ylabel('y','fontsize', 14)

 figure(6)
 surf(xy(:,:,1),xy(:,:,2),TotalStresseval(:,:,2))
 colormap jet
 shading interp
 colorbar
 title('Stress yy','fontsize', 14)
 xlabel('x','fontsize', 14)
 ylabel('y','fontsize', 14)

 figure(7)
 surf(xy(:,:,1),xy(:,:,2),TotalStresseval(:,:,3))
 colormap jet
 shading interp
 colorbar
 title('Stress xy','fontsize', 14)
 xlabel('x','fontsize', 14)
 ylabel('y','fontsize', 14)

fig=10;
for figm=m
 for fign=1:n
 figure(10+fig)
 surf(xy(:,:,1),xy(:,:,2),R_eval(:,:,fign,figm))
 colormap jet
 shading interp
 colorbar
 title(['R(',num2str(fign),',',num2str(figm),')'],'fontsize',
    14)
 xlabel('x','fontsize', 14)
```

```
 ylabel('y','fontsize', 14)
 fig=fig+1;
    end
end

fig=30;
for figm=m
    for fign=1:n
 figure(30+fig)
 surf(xy(:,:,1),xy(:,:,2),dR_dx_eval(:,:,fign,figm,1))
 colormap jet
 shading interp
 colorbar
 grid on
 title(['dRdx(',num2str(fign),',',num2str(figm),')'],'fontsize',
     14)
 xlabel('x','fontsize', 14)
 ylabel('y','fontsize', 14)
 fig=fig+1;
    end
end
end
```