NTNU – Trondheim
Norwegian University of
Science and Technology

Isogeometric Analysis Based Shape Optimization.

Kristin Solbakken

Master of Science in Mathematics
Submission date: May 2015
Supervisor: Anton Evgrafov, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

We solve shape optimization problems involving partial differential equations. The latter are discretized and solved using isogeometric analysis. B-splines and non uniform rational B-splines are introduced, along with both theory and implementation aspects of the numerical method. We solve a chemical reaction problem that is governed by a system of non-linearly coupled Laplace equations, which is only one of numerous possible application areas of isogeometric analysis based shape optimization.

Sammendrag

Vi løser formoptimeringsproblemer som involverer partielle differensialligninger. Sistnevnte er diskretisert og løst ved bruk av isogeometrisk analyse. B-splines og Non uniform rational B-splines introduseres, sammen med teori og implementasjonsaspekter for den numeriske metoden. Vi løser et kjemisk reaksjonsproblem med utgangspunkt i et system av ikkelineære koblete Laplacelignigner, som er kun ett av mange anvendelsesområder for isogeometrisk analyse basert formoptimering.

Acknowledgements

I would like to thank my supervisor Prof. Anton Evgrafov for his guidance and valuable feedback and for patiently answering my numerous questions. In addition I thank Post.doc Kjetil André Johannessen for helpfully giving advices to overcome certain programming obstacles. Thanks to Nikolai Ubostad and my fellow students Mathilde Skylstad and Guðrún Sigfúsdóttir for proof reading this thesis. At last I thank my parents for their continuous support during my years at NTNU .

Contents

Abbreviations and Notations

Abbreviations

For simplicity, throughout this thesis, we use the abbreviations listed in the following table:

IGA	Isogeometric Analysis
FEM	Finite Element Method
NURBS	Non Uniform Rational B-splines
CAD	Computer Aided Design
PDE	Partial Differential Equation
CFD	Computational Fluid Dynamics
AD	Automatic Differentiation
LR B-splines	Locally Refinable B-splines

Table 1: List of abbreviations.

Notation

We denote by Ω an open, bounded and simply connected subset of $\mathbb{R}^n$, which we from now on call the domain Ω with boundary $\partial\Omega$. In our case, the boundary $\partial\Omega$ will be piecewise C^1.

For a function $f : \Omega \mapsto \mathbb{R}$

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

denotes the gradient of the function f, and

$$\Delta f = \frac{\partial^2 f}{\partial x_1^2} + \ldots + \frac{\partial^2 f}{\partial x_n^2}$$

is the Laplacian of f.

For the normal derivative of the function f, $\frac{\partial f}{\partial n}$, we write either $\frac{\partial f}{\partial n} = \partial_\nu f$ or $\frac{\partial f}{\partial n} = n \cdot \nabla f$.

Chapter 1

Introduction

"There is no branch of mathematics, however abstract, which may not some day be applied to phenomena of the real world"

Nikolai Ivanovich Lobachevsky

1.1. IGA and Shape Optimization

Isogeometric analysis (IGA) has been relatively recently introduced by J.A. Cottrell, T.J.R. Hughes and Y. Bazilevs [8]. It represents a new method for the numerical approximation of solutions to partial differential equations (PDEs). Many similarities can be found between the classical finite element method (FEM) and IGA. The goal of the latter is to unify the fields of the FEM and computer aided design (CAD) [8]. CAD uses non uniform rational B-splines (NURBS) to represent complex geometries. The NURBS can be used to give exact representations of many geometries used in practice, which would otherwise only be approximated by polynomials [21]. Such geometries include circular and conic domains. By using the standard FEM for engineering purposes, a lot of time is wasted on approximating geometries before analysis can be carried out. Using the same basis functions for both design and analysis yields more precise results and increases efficiency, leaving more time for the actual analysis. In addition, for a PDE with smooth solutions, one can construct the basis functions utilized in IGA to be as smooth as required for the problem at hand. This is in contrast with the Lagrange polynomials that are typically used in a finite element discretization, that achieve only C^0 continuity at the interior elements. B-splines and NURBS have many desirable properties, and one can achieve high convergence rate by adjusting the polynomial degree of the basis functions. It should be mentioned that by using B-splines and NURBS as basis functions, complex shapes and geometries can be represented by using a relatively small number of degrees

of freedom. These piecewise smooth shapes can easily be passed on to CAD systems [15], making IGA a suitable tool for shape optimization. Additionally, the task of performing shape optimization consists of several important pieces. Essentially it is a union between mathematical sciences and computer science. A well done model of different geometries along with numerical analysis and programming must be carried out. However the integration between designers and analysts has proven difficult up until the recently introduced science of IGA, which provides a natural base to develop design optimization tools by a unification of the geometric design and analysis [19]. Shape optimization as a mathematical problem is the problem of finding the shape Ω, that minimizes/maximizes a specified objective functional. The objective functional will in all cases considered in this thesis be connected to the solution of a PDE solved using IGA on the domain Ω. Shape optimization applies to many fields of the engineering industry, e.g. [18] addresses optimal shape design applied to computational fluid dynamics (CFD), a field that has been given much attention in the recent years. Other types of applications can be found in [11] and [29] and the references therein. As a bonus, we include Section 1.2 to show two specific examples of how IGA and shape optimization can be applied to physical problems.

1.2. Applications of IGA Based Shape Optimization

1.2.1. An Application of Industrial Interest

An industrial application can be found in [14]. The authors apply an IGA based shape optimization procedure to optimize the shape of ferromagnetic covers of poles in a magnetic field generated by magnetic density separators (MDS). MDS exploits that different materials to be separated have different density, a technique increasingly used in the waste recycling industry.

1.2.2. Medical Modelling

IGA based shape optimization has also been applied to several studies within medical research. An example of high relevance involves the study of ventricular assist devices (VDAs) (in case of heart failures when no matching donors are available for transplantations). The interested reader is encouraged to consult [12] for a more detailed reading on a study for such a device for pediatric patients. In short, the objective was to use shape optimization as a tool for reducing thrombotic risk for patients with a VDA. A pulsatile VDA (PVDA) is modelled by a thin flexible membrane, separating air being pumped into the air chamber and blood that is injected into the aorta. NURBS-based IGA was used to model the structure of the membrane, and it was discovered that an IGA approach was beneficial both for efficiency and accuracy compared to modelling with the classical FEM. The authors found that the optimal shape increased the flow efficiency with a shape of significant difference from the original one.

1.3. Structure of this Thesis

Background theory on B-splines and NURBS is presented in Chapter 2. Having introduced the basis functions, we proceed by discussing important concepts of IGA and the main differences between IGA and FEM in Chapter 3. Primarily we will be working in three different spaces: a physical space, a parameter space and a parent space. We define the mapping that will take us back and forth between these spaces and discuss how to perform the numerical integration. Chapter 4 is dedicated to applying IGA to solve Poisson's equation in two dimensions. Though it is a simple PDE, it has several important applications, e.g. it plays an important role in the theory of gravitational, mechanical and electrical fields (which are all examples of conservative fields) [24]. In Chapter 5 we introduce a linear reparameterization algorithm, which will be used to update the inner control points in the context of shape optimization, that will be studied in Chapter 6. Having established all the above mentioned material, we have all the tools needed to study an application of IGA based shape optimization. The specific application considered in this thesis is found in Chapter 7, where a chemical reaction system is solved, involving three non-linearly coupled stationary diffusion equations defined over some domain Ω. To round it all up a discussion including conclusions and future work follows in Chapter 8.

Chapter 2

B-splines, Splines and NURBS

In this thesis both B-splines and NURBS are used as basis functions for the analysis. For simple geometries, e.g. squares/rectangles it is sufficient to use B-splines, whereas NURBS can be a great tool when dealing with more complex geometries. The NURBS are built up from B-splines, so a discussion of B-splines and splines is a natural starting point.

2.1. Background Theory and Definitions

B-splines, hence also NURBS are dependent on the polynomial degree p and a knot vector ξ, which is defined in the parameter space (see Section 3.1.1). The shape of a spline curve (sometimes referred to as a spline) is determined by a set of control points (the same goes for objects of higher dimension). Hence, starting from a polynomial degree, a knot vector and the control points, we can build B-splines, NURBS and splines.

Definition 2.1.1. A knot vector ξ is a non-decreasing sequence of numbers $\{\xi_i\}_{i=1}^{n+p+1}$ belonging to $\mathbb{R}$. A knot vector with $p+1$ identical knots at both ends is said to be $p+1$ regular.

By using $p+1$ regular knot vectors, the spline curve becomes clamped and interpolant to the first and the last control points.

2.1. Background Theory and Definitions

Now that the definition of a knot vector is given, we introduce the B-splines in the following definition:

Definition 2.1.2. The B-spline $N_{i,p}$ is defined recursively by Cox-de Boors formula, starting with piecewise constants when $p = 0$.

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi \in [\xi_i, \xi_{i+1}),\ i = 1, ..., n+p. \\ 0 & \text{otherwise.} \end{cases}$$

For $p > 0$, the B-spline $N_{i,p}(\xi)$ is defined by

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi). \tag{2.1.1}$$

Algorithm 1 Evaluation of the B-Splines

1: Give input $p, \xi_{\text{eval}}, \xi, ni$
2: Store B-spline $N \rightarrow$ `zeros`$(p+1, 1)$
3: Initiate the first basis function, $N(1) \rightarrow 1$
4: **For** $j = 2$ to $p+1$
5: **For** $k = 1$ to j
6: Compute B-splines recursively according to Cox-de Boor's formula
7: **End** loops
8: Output $\rightarrow N(1 : p+1)$

It should be noted immediately from Definition 2.1.2 that B-splines are compactly supported. Starting from the piecewise constants it should come as no surprise that each B-spline has small support. In fact, the B-spline $N_{i,p}(\xi)$ is non-zero only in the semi-open interval $[\xi_i, \xi_{i+p+1})$. This means that the maximum number of B-splines that are non-zero, when the spline is evaluated at a point ξ with the same knot vector, is $p+1$. This should be utilized when implementing such recursive formulas. If one is not observant of the compact support, the code for finding the basis functions will do a lot of redundant work and evaluate numerous expressions to zero, that is, expressions known a priori to evaluate to zero. Algorithm I takes care of this by only allocating memory for $p+1$ B-splines. In this way the algorithm becomes more efficient and uses less memory, as no basis functions known to be zero are stored.

The B-splines are easy to work with, and since they are piecewise polynomials by definition they can be efficiently differentiated. The first derivative of a B-spline of degree p, $\frac{\mathrm{d}N_{i,p}}{\mathrm{d}\xi}$, is calculated as follows:

$$\frac{\mathrm{d}N_{i,p}}{\mathrm{d}\xi}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi). \tag{2.1.2}$$

Similarly, higher order derivatives, $\frac{\mathrm{d}^k N_{i,p}}{\mathrm{d}\xi^k}$ are computed by following the same procedure,

$$\frac{\mathrm{d}^k N_{i,p}}{\mathrm{d}\xi^k} = \frac{p}{\xi_{i+p} - \xi_i}\left(\frac{\mathrm{d}^{k-1}}{\mathrm{d}\xi^{k-1}} N_{i,p-1}(\xi)\right) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}\left(\frac{\mathrm{d}^{k-1}}{\mathrm{d}\xi^{k-1}} N_{i+1,p-1}(\xi)\right). \tag{2.1.3}$$

The program evaluating the derivatives of the B-splines makes use of the B-spline evaluation program, hence only $p+1$ derivatives are stored.

Definition 2.1.3. Splines are piecewise continuous polynomials, where the different polynomial pieces join together at the knots. Given a knot vector ξ, a polynomial degree p and a set of control points $\{c_j\}_{j=1}^n$, a spline function C can be written as a linear combination of B-splines and these control points, i.e.

$$C(\xi) = \sum_{j=1}^{n} c_j N_{j,p}(\xi). \tag{2.1.4}$$

The spline in Definition 2.1.3 depends on a set of control points and has also got an associated control polygon defined as follows:.

Definition 2.1.4. The control polygon of the spline C in Equation (2.1.4) is the piecewise linear interpolant between the coordinates $\left(\xi_j^*, c_j\right)$ for $j = 1, \ldots, n$, where the c_j's are the control points and the ξ_j^*'s are the Greville abscissae:

$$\xi_j^* = \frac{\xi_{j+1} + \ldots + \xi_{j+p+1}}{p}.$$

It is shown in [13] that the control polygon converges to its associated spline as the size of the knot intervals tends towards zero.

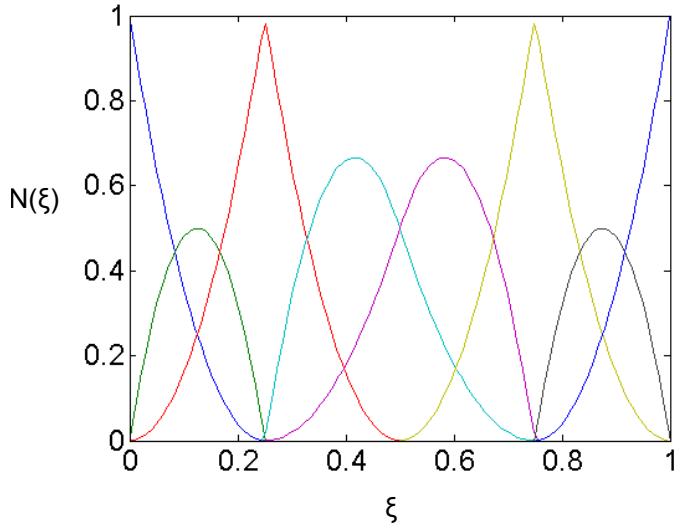


Figure 2.1: Quadratic B-splines defined on the $p+1$ regular, non-uniform knot vector $\xi = \{0, 0, 0, 0.25, 0.25, 0.5, 0.75, 0.75, 1, 1, 1\}$. Note how the multiplicity of the knots controls the continuity of the B-splines.

The knots may be uniform or non-uniform, meaning that they are either equally spaced or unequally spaced. Non-uniform knot vectors can be separated into two main cases: 1) Knot intervals of different sizes. Such knot vectors can be used to move the spline closer to, or further away from the control points associated with the knots. 2) Multiple knots, which reduces the continuity of the spline at the multiple repeated knot. A knot appearing m times in the knot vector is said to have multiplicity m. The B-spline is C^{p-m} continuous at this knot, whereas inside a knot interval it is C^{∞}. The B-spline is discontinuous at a knot appearing $p+1$ times, and simply continuous at a knot of multiplicity $m = p$. At such a knot, the spline itself will coincide with its control polygon. Throughout this thesis, $p+1$ regular knot vectors are always considered when solving PDEs, e.g. given $p = 2$, the knot vector ξ will be of the form, $\xi = \{\xi_1, \xi_1, \xi_1, \xi_2, \xi_3, \ldots, \xi_{n+1}, \xi_{n+1}, \xi_{n+1}\}$. The reason for choosing $p+1$ regular knot vectors when solving a PDE is that the PDE itself will be defined on some domain Ω. Without this specific structure of the knot vectors, the numerical solution will no longer be on the same domain Ω, leading to wrong solutions. Mainly, we consider normalized knot vectors, meaning that the vectors span the interval $[0, 1]$. Normalized knot vectors may be helpful when it comes to numerical accuracy. Notice also that by using $p+1$ regular knot vectors, the end points of the spline curves are forced to coincide with the control polygon. Thus the knot vectors equip us with great power to control the regularity of the spline functions and the B-splines. Figure 2.1 shows quadratic B-splines defined on the knot vector $\xi = \{0,\ 0,\ 0,\ 0.25,\ 0.25,\ 0.5,\ 0.75,\ 0.75,\ 1,\ 1,\ 1\}$. This example serves as an illustration of the continuity properties of B-splines. Note that the different polynomial pieces join together at the knots, and that the $p+1$ regular knot vector causes the B-splines at the ends to be discontinuous. The knots placed at $\xi = 0.25$ and $\xi = 0.75$ have double multiplicity, illustrating that the B-spline is C^0 at a knot of multiplicity p as mentioned above.

2.1.1. Tensor Product Splines

Consider the two knot vectors $\xi = (\xi_i)_{i=1}^{n+p+1}$ and $\eta = (\eta_j)_{j=1}^{m+q+1}$. Each of these knot vectors can be associated with a total of n and m B-splines, respectively. The spline dependent on ξ or η lives in one of the two spline spaces, $\mathbb{S}_{p,\xi}$ and $\mathbb{S}_{q,\eta}$.

Definition 2.1.5. Let ξ and η be of length given by $l(\xi) = n+p+1$ and $l(\eta) = m+q+1$. The space of all linear combinations of the total number of n B-splines defines the spline space $\mathbb{S}_{p,\xi}$. Similarly the space of all linear combinations of the total number of m B-splines defines the spline space $\mathbb{S}_{q,\eta}$, i.e.

$$\begin{aligned} \mathbb{S}_{p,\xi} &= \left\{ \sum_{i=1}^{n} c_i N_{i,p}(\xi) \mid c_i \in \mathbb{R}^d, \quad 1 \leq i \leq n \right\}, \\ \mathbb{S}_{q,\eta} &= \left\{ \sum_{j=1}^{m} c_j M_{j,q}(\eta) \mid c_j \in \mathbb{R}^d, \quad 1 \leq j \leq m \right\}. \end{aligned} \tag{2.1.5}$$

Due to B-splines being linearly independent, they form a basis for these two spaces. The linear independence of the B-splines of degree p follows from the fact than any polynomial of degree p can be represented as a linear combination of B-splines. The proof is outlined in [13].

2.1. Background Theory and Definitions

A tensor product spline space, $\mathbb{S}_{p,q}(\xi,\eta)$, is defined to be the space of products of all spline functions defined in $\mathbb{S}_{p,\xi}$ and $\mathbb{S}_{q,\eta}$ and is denoted by $\mathbb{S}_{p,\xi} \otimes \mathbb{S}_{q,\eta}$. This space is spanned by $\{N_{i,p}(\xi)M_{j,q}(\eta)\}_{i,j=1}^{n,m}$. By linear independence of the basis functions the tensor product spline space has dimension $\dim\{\mathbb{S}_{p,\xi} \otimes \mathbb{S}_{q,\eta}\} = n \times m$ [13].

Definition 2.1.6. Define $\mathbb{S}_{p,q}(\xi,\eta) = \mathbb{S}_{p,\xi} \otimes \mathbb{S}_{q,\eta}$. A tensor product spline surface is a function $S \in \mathbb{S}_{p,q}(\xi,\eta)$ defined by

$$S(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} \mathbf{B}_{i,j} N_{i,p}(\xi) M_{j,q}(\eta), \tag{2.1.6}$$

where the $\mathbf{B}_{i,j}$'s are the control coefficients.

2.1.2. Spline Curves and Surfaces

The spline $C(\xi)$ in Definition 2.1.3 is also referred to as a spline curve. A spline curve $C(\xi)$ in $\mathbb{R}^d$ is represented as a linear combination of B-splines, where the $\{c_i\}$'s $\in \mathbb{R}^d$ are the control points.

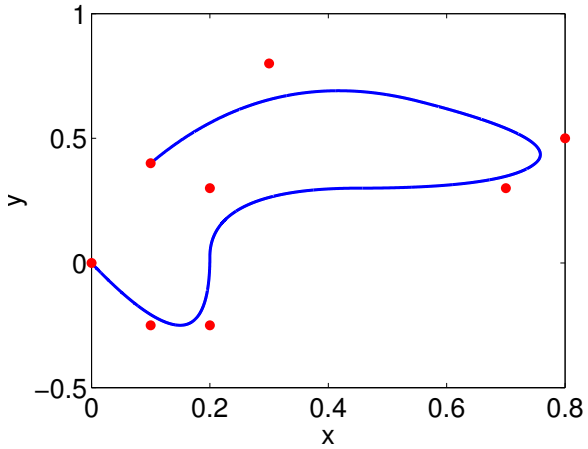
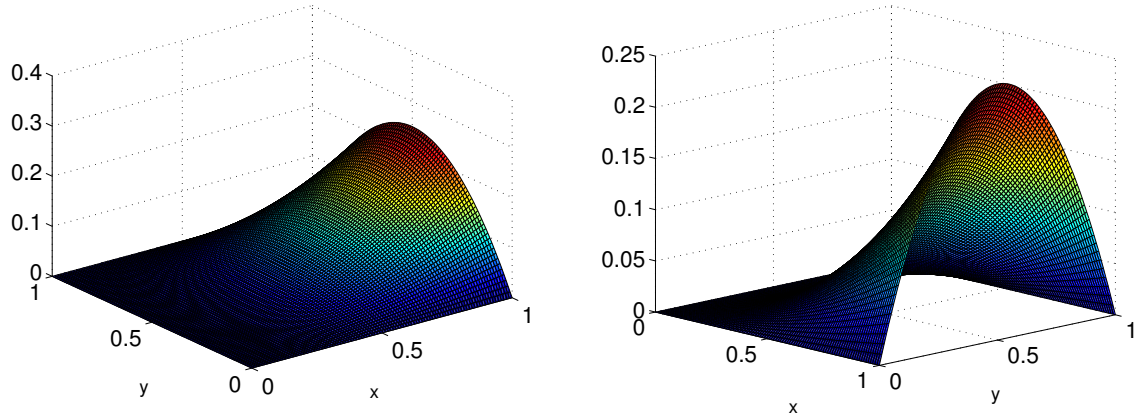


Figure 2.2: A spline curve together with its control points marked in red. The polynomial degree of the spline and the knot vector are as in Figure 2.1.

Figure 2.2 shows a spline plotted together with a set of control points. The knot vector is the same as in Figure 2.1.



(a) Surface plot of a biquadratic spline surface.

(b) The same spline surface shown from another angle.

Figure 2.3: Bi-quadratic spline surface, defined on the knot vectors $\xi = \{0\ 0\ 0\ 1\ 1\}$ and $\eta = \{0\ 1\ 2\ 2\ 3\}$.

A spline surface, $S(\xi, \eta)$, is dependent on two knot vectors ξ and η of degrees p and q respectively and a control net $B = B_{i,j}$. The equation for the surface is given by Equation (2.1.6). Figure 2.3 shows a spline surface with the control net B given in Table 2.1. The knot vectors used in this illustration are $\xi = \{0\ 0\ 0\ 1\ 1\}$ and $\eta = \{0\ 1\ 2\ 2\ 3\}$. All the B-splines are quadratic.

i	j	$B_{i,j}$
1	1	{0}
1	2	{-1}
2	1	{1}
2	2	{2}

Table 2.1: Control points for the biquadratic spline surface in Figure 2.3.

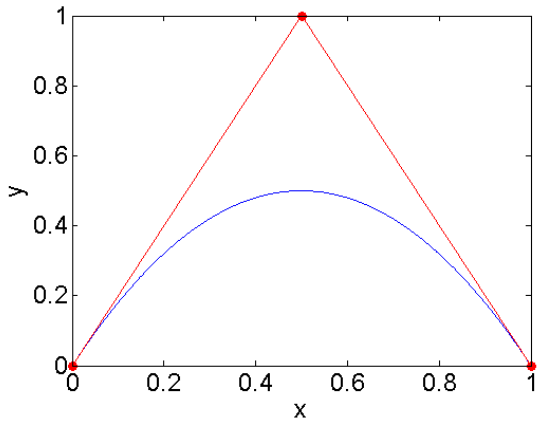
2.1.3. Knot Insertion

Knot insertion or knot refinement is the process of refining an already existing knot vector. For a degree p knot vector ξ of length $l(\xi) = n+p+1$, the new extended knot vector $\bar{\xi}$ of length $l(\bar{\xi}) = \bar{n} + p + 1$, where $\bar{n} = (n+$ number of inserted knots), is a refinement of ξ if it contains the original knot vector as a subsequence. If this is the case, then the original spline space is a subspace of the new spline space $\mathbb{S}_{p,\bar{\xi}}$ [13]. Knot insertion will not change the curve or the geometrical shape of the spline as long as the control coefficients are properly updated. However, knot insertion will reduce the distance between the spline and its control polygon. Knot insertion results in a richer basis, and the new spline space will contain more spline functions. As discussed above, a B-spline is $p-m$ continuous at a knot of multiplicity m, thus it is reasonable to think that the continuity of the spline decreases since the multiplicity of knots may increase. However, the continuity of the spline is preserved by relating the new $\bar{n}$ control points, $\bar{B}$, to the original control points, B, by letting $\bar{B} = \mathbf{T}^p B$, where $\mathbf{T}$ is calculated in accordance with the following expression:

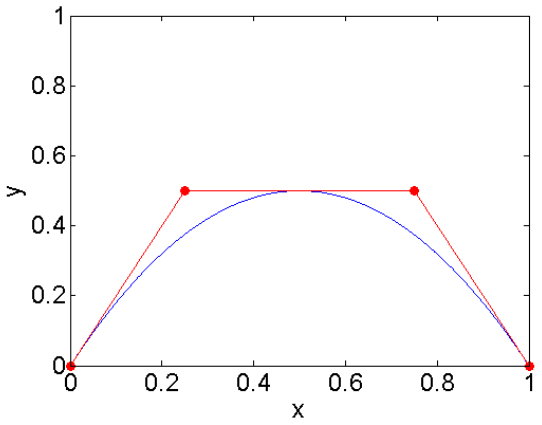
$$T_{i,j}^0 = \begin{cases} 1 & \text{if } \bar{\xi}_i \in [\xi_j, \xi_{j+1}) \\ 0 & \text{otherwise,} \end{cases}$$

$$T_{i,j}^{q+1} = \frac{\bar{\xi}_{i+q} - \xi_j}{\xi_{j+q} - \xi_j} T_{i,j}^q + \frac{\xi_{j+q+1} - \bar{\xi}_{i+q}}{\xi_{j+q+1} - \xi_{j+1}} T_{i,j+1}^q, \quad \text{for } q = 0, 1, ..., p-1. \tag{2.1.7}$$

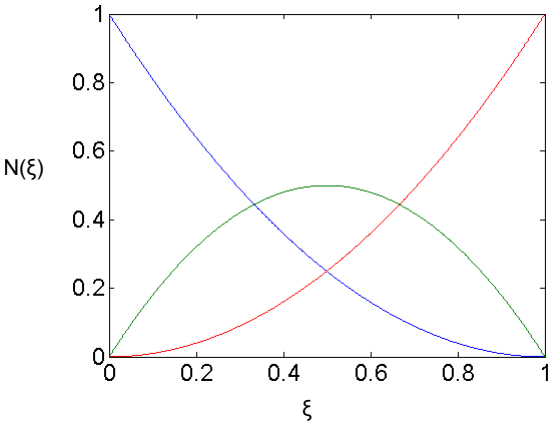
Knot insertion is used in this thesis for solving PDEs in two dimensions. Hence, two different matrices $\mathbf{T}_\xi$ and $\mathbf{T}_\eta$ are needed. The new control coordinates in the x- and y direction are dependent on both of these matrices. Knot insertion is a powerful tool for design purposes, as details can be described at a high level. It will be become clear later that it also allows one to increase the accuracy of the numerical approximation of solutions to PDEs. However, it is often the case when solving PDEs that one seeks to obtain a close enough numerical solution by using as few as possible degrees of freedom. In this sense, knot insertion will lead to systems of equations that are more computationally expensive to solve. A better strategy would be to apply local refinement (LR) in places known to need finer discretizations. For instance if one is solving a PDE with singularities, or in cases where the geometry involves sharp corners. Due to the tensor product structure of both B-splines and NURBS, these basis functions have proven inadequate for LR. As an alternative to B-splines and NURBS, several basis functions have been proposed. Amongst them are T-splines and LR B-splines, see [5, 26] for further details. LR is outside the scope of this thesis, but material written on the topic is available for the interested reader. See for example [9].



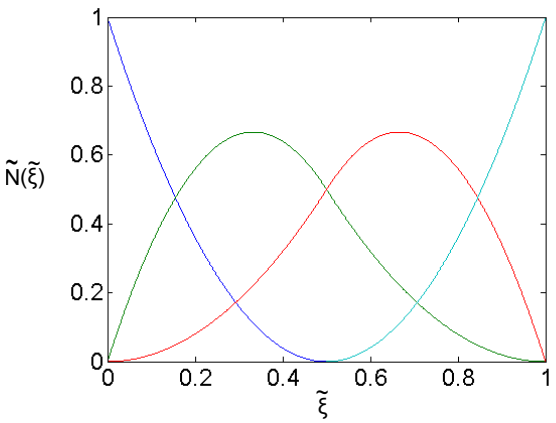
(a) Original control points and spline curve.



(b) Refined control points and spline curve.



(c) Original B-splines



(d) B-splines of the refined spline curve.

Figure 2.4: Example of knot insertion. One new knot has been inserted at $\xi = 0.5$. The basis functions of the original spline and the refined spline are shown in Figure 2.4c and Figure 2.4d respectively.

An example of knot insertion is shown in Figure 2.4. The original knot vector is $\xi = \{0,\ 0,\ 0,\ 1,\ 1,\ 1\}$ and the extended knot vector $\bar{\xi} = \{0,\ 0,\ 0,\ 0.5,\ 1,\ 1,\ 1\}$. The original B-splines and the B-splines of the refined spline are shown in Figure 2.4c and Figure 2.4d.

2.1.4. Properties of B-splines

B-splines have some very nice properties worth mentioning. The continuity properties have already been discussed in the previous section, but here we list some of the other useful properties.

- B-splines are piecewise non-negative, i.e. $N_{i,p}(\xi) \geq 0$ for all $i \in [1, n]$, where $n = \text{length}(\xi) - p - 1$.

- B-splines constitute a partition of unity, i.e. $\sum_{i=1}^{n} N_{i,p}(\xi) = 1$.

- B-splines have compact support. If the polynomial degree is p, there are at most $p+1$ non-zero B-splines in the knot interval $[\xi_i, \xi_{i+1}]$.

These properties are used frequently in connection with testing of the different parts of the IGA implementation in this thesis. Though they do not verify that the code is correct, checking that the B-splines and NURBS always sum to one and are piecewise non-negative gives an indication that at least the basis function routines are correct. It is a good practice to test every part of the program during the implementation, as any violation of these spline property criteria implies errors in the implementation.

2.1.5. NURBS

One flaw of B-splines is that they cannot be used to represent conic geometries such as circles, ellipsoids and hyperboloids exactly. NURBS on the other hand can be used as basis functions whenever the geometry to be represented is a conic geometry. In this case, the basis functions will be rational functions as opposed to polynomials. As the name indicates, they are rational B-splines, and thus many of the properties of B-splines carry on to NURBS. For example, the NURBS are piecewise non-negative, i.e. $R_{i,p}(\xi) \geq 0$, for all i. They constitute a partition of unity and are compactly supported. The equation for constructing a NURBS function $R_{i,p}(\xi)$ is:

$$R_{i,p}(\xi) = w_i \frac{N_{i,p}(\xi)}{W(\xi)}, \tag{2.1.8}$$

for a weight $w_i \in \mathbb{R}$. The first derivative, $\frac{\mathrm{d}R_{i,p}}{\mathrm{d}\xi}$, of a NURBS function $R_{i,p}$, is given by

$$\frac{\mathrm{d}}{\mathrm{d}\xi} R_{i,p}(\xi) = w_i \frac{W(\xi)\ N'_{i,p}(\xi) - W'(\xi)\ N_{i,p}(\xi)}{W(\xi)^2}, \tag{2.1.9}$$

where $W(\xi)$ is the weighting function

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

Geometrically, a NURBS curve can be viewed as a projection of a spline curve in $d+1$ dimensions onto d dimensions. The weight w_i geometrically corresponds to the control point c_i of the spline curve in dimension $d+1$. For implementation purposes we are more interested in the NURBS from an algebraic point of view, and this is where we focus our attention in this thesis. A more detailed discussion of the geometric point of view is outlined in [8]. As was the case for the spline curves, a NURBS curve $C(\xi)$ is constructed by a linear combination of NURBS:

i	j	$B_{i,j}$	$w_{i,j}$
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)	$\frac{}{\sqrt{2}}$
9	2	(1,0)	1

Table 2.2: Control points and weights for the NURBS surface in Figure 2.5b.

$$C(\xi) = \sum_{i=1}^{n} R_{i,p}(\xi)\ B_i, \tag{2.1.10}$$

where $c_i \in \mathbb{R}^d$ denotes the i'th control point. A bivariate NURBS function $R_{i,j}^{p,q}$, dependent on the two knot vectors ξ and η is given by

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_{i,p}(\xi)\ M_{j,q}(\eta)\ w_{i,j}}{\sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(\xi)\ M_{j,q}(\eta)\ w_{i,j}}, \tag{2.1.11}$$

and similar to a bivariate spline surface, a bivarite NURBS surface S is defined as follows:

$$S(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q}(\xi,\eta)\ B_{i,j}. \tag{2.1.12}$$

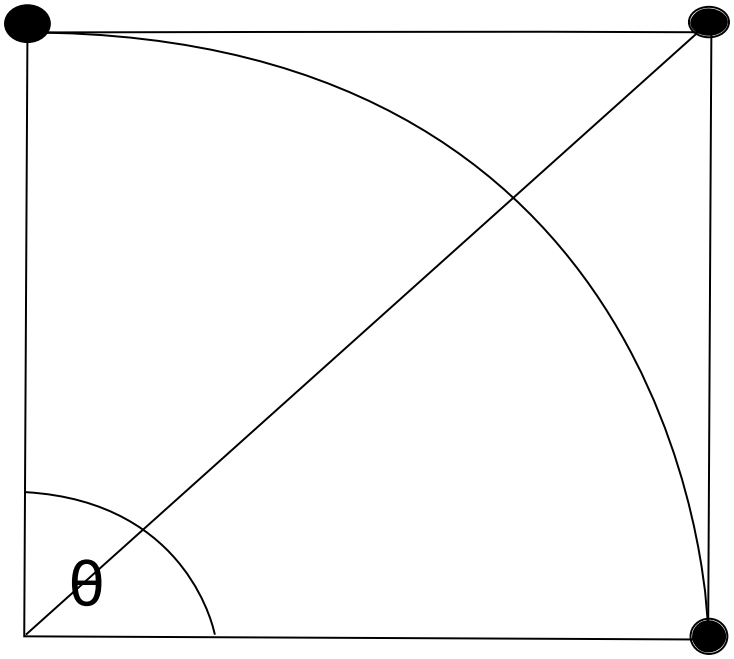
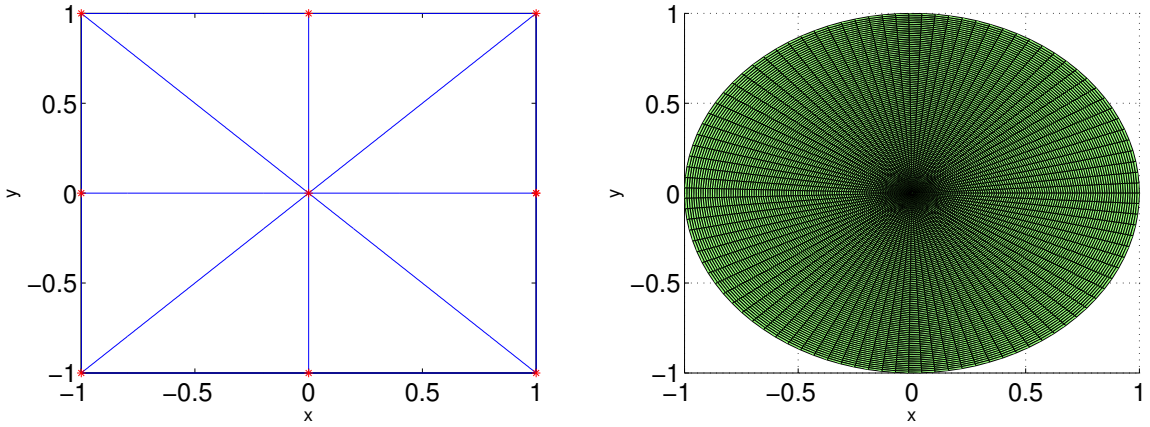


Figure 2.6: This figure is an illustration of how the weights for a circular arc is constructed.



(a) NURBS control mesh for the unit disk. The control points are shown in red.

(b) NURBS surface of the unit disk. Weights and control points are shown in Table 2.2.

Figure 2.5: Control mesh and NURBS surface. The point in (1,0) is repeated and 9 control points are placed in the origin.

An example of a NURBS surface is shown in Figure 2.5. This example shows that a conical geometry such as the unit disk can be represented exactly by assigning appropriate weights to the control points. The control points and weights for Figure 2.5 are shown in Table 2.2.

The specific weights in Table 2.2 can be explained as follows: The first and the third control point positioned on the circular arc in Figure 2.6 are assigned weights equal to one. These control points correspond to coordinates that are a part of the unit disk in the physical space. The second control point lies in the intersection of the tangent lines to the two other control points. The weight corresponding to the second control points is calculated as $\cos(\theta/2)$, where θ is the angle opposing the circular arc. This implies that the second weight for the circular arc is

$$w_2 = \cos(\theta/2) = \cos(\pi/4) = \frac{1}{\sqrt{2}}.$$

Chapter 3

Basics of Isogeometric Analysis

In this section the basics of IGA are explained. That is, FEM is extended by using B-splines or NURBS as basis functions. The geometry parameterization and the different spaces we work in, along with details for performing numerical integration with Gaussian quadratures are discussed in this chapter.

3.1. The Isogeometric Approach

In IGA, the same set of basis functions used to describe the geometry is also typically used as the basis for the solution field of the PDE. This is referred to as the "isogeometric concept". The basis functions throughout this thesis are either B-splines or NURBS.

The computer implementation of IGA has many similarities with a standard FEM solver, but there are theoretic and technical differences one must be aware of when going from the classical FEM to IGA.

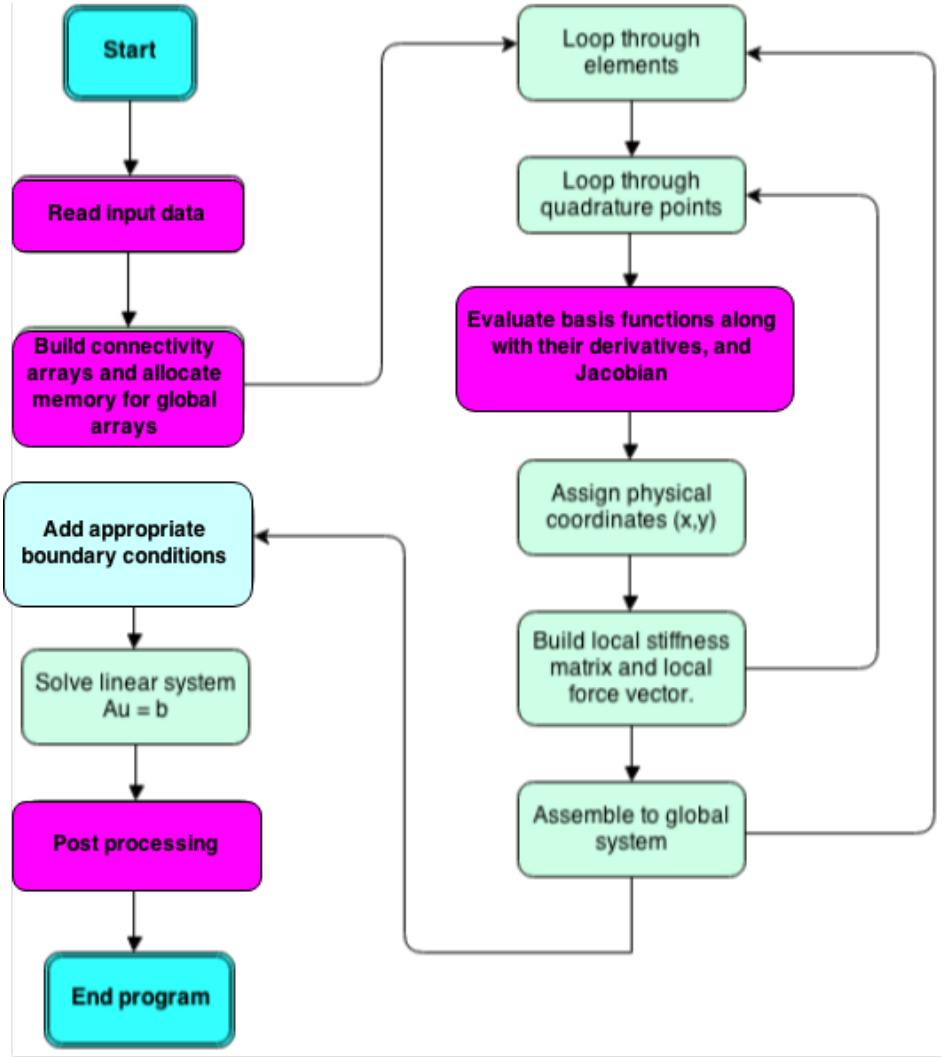


Figure 3.1: Code architecture for a typical FEM solver [8]. To convert to an IGA solver, the routines shown in pink must be changed.

A standard FEM solver architecture is shown in Figure 3.1. The main differences in the code architecture from FEM to IGA are the input data, the evaluation of the basis functions and the postprocessing part. The code can be converted to an IGA code by replacing the routines that are shown in pink.

To get a good grip of IGA it is important to be aware of the different spaces and parameterizations we work with. A more detailed discussion of these matters is given in the following sections.

3.1.1. Geometry Parameterization

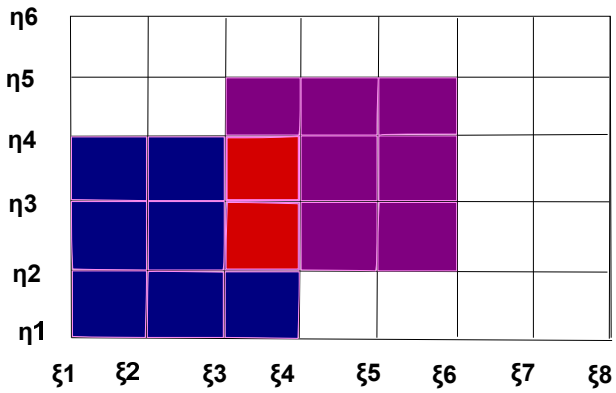
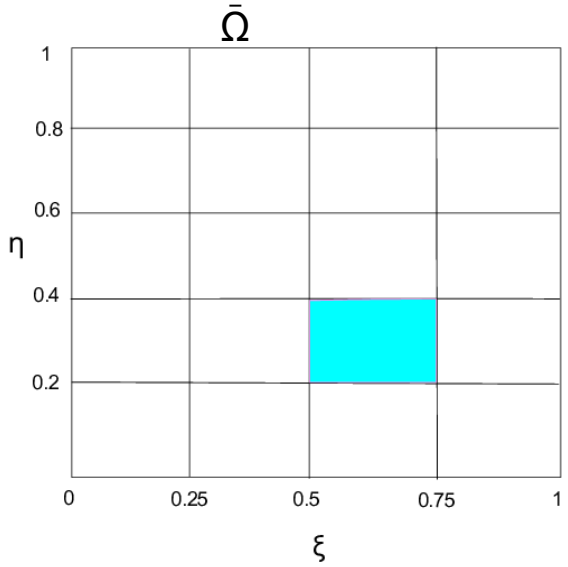


Figure 3.2: Example of an index space for the quadratic knot vectors defined by $\xi = \{0,\ 0,\ 0,\ 1/3,\ 2/3,\ 1,\ 1,\ 1\}$ and $\eta = \{0,\ 0,\ 0,\ 1,\ 1,\ 1\}$. The blue colored area represents the support of the B-spline $N_{1,1}(\xi,\eta)$, the purple colored area represents the support of the B-spline $N_{3,2}(\xi,\eta)$. The red colored area represents the overlap of the two B-splines.

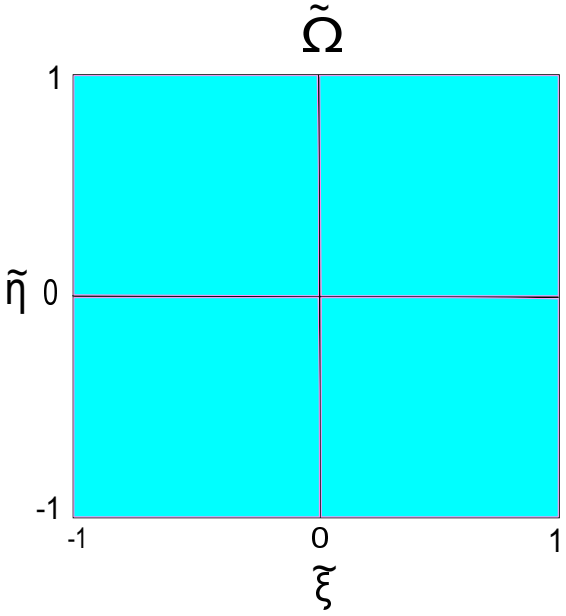
In FEM, the elements have one representation in the parent space (which is where the numerical integration takes place) and one representation in the physical space [8]. In IGA we additionally need a control mesh, an index space and a parameter space. The knot vectors, hence also the B-splines are defined in the parameter space. The index space is formed by giving each knot value (also the repeated ones) a coordinate, and gives a description of the support of the B-splines. Figure 3.2 shows the index space for the quadratic knot vectors $\xi = \{0,\ 0,\ 0,\ 1/3,\ 2/3,\ 1,\ 1,\ 1\}$ and $\eta = \{0,\ 0,\ 0,\ 1,\ 1,\ 1\}$. The highlighted blue area gives the support of the biquadratic tensor product B-spline $N_{1,1,2,2}(\xi,\eta)$, the highlighted purple area gives the support of the biquadratic tensor product B-spline $N_{3,2,2,2}(\xi,\eta)$. The red area is where their support overlap. The control mesh holds the control points, whereas the coordinates in the physical space x and y are built up by linear combinations of the basis functions and the control points. That is,

$$\begin{aligned}\begin{pmatrix} x \\ y \end{pmatrix} &= \sum_{i=1}^{n} R_i(\xi,\eta)\mathbf{B}_i \\ &= \sum_{i=1}^{nen} R_i(\xi,\eta)\mathbf{B}_i,\end{aligned} \tag{3.1.1}$$

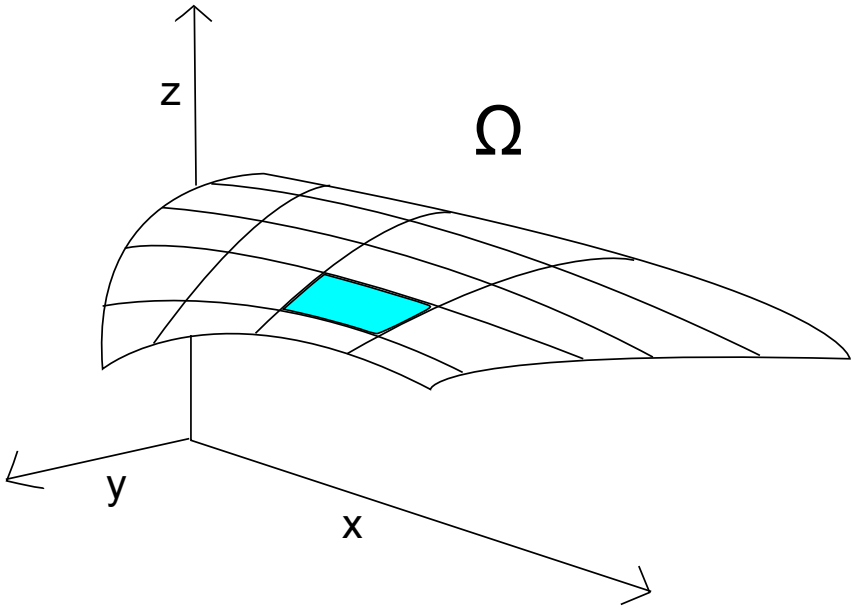
where the $\mathbf{B}_i's$ are the control variables corresponding to the basis functions with support on each tensor product knot interval. In two dimensions, a maximum of $nen = (p+1)\times(q+1)$ basis functions are non-zero on each tensor product knot interval, due to the B-splines and NURBS having compact support. The last sum in Equation (3.1.1) goes from $i = 1,\ \ldots\ nen$, in order to avoid redundant work. Figure 3.3c shows an example of a physical mesh.



(a) Parameter space for a linear spline defined by the two knot vectors $\xi = [0\ 0\ 0.25\ 0.5\ 0.75\ 1\ 1]$ and $\eta = [0\ 0\ 0.2\ 0.4\ 0.6\ 0.8\ 1\ 1]$.



(b) Parent space.



(c) Physical space

Figure 3.3: Parameter space, parent space and physical space.

A tensor product spline (in 2D) is dependent on two knot vectors, ξ and η. From these knot vectors the parameter space $\bar{\Omega}(\xi, \eta)$ can be created by a certain parameterization. Denote by $\tilde{\Omega}$ the parent space. The Gaussian quadrature points $\tilde{\xi}$ and $\tilde{\eta}$, used for numerical integration live in the parent space. The parameterization can now be defined dependent

only on the parent coordinates and the knot vectors in the following way:

$$\begin{aligned} \xi &= \xi_i + (\tilde{\xi} + 1)\,\frac{\xi_{i+1} - \xi_i}{2} = \frac{(\xi_{i+1} - \xi_i)\,\tilde{\xi} + (\xi_{i+1} + \xi_i)}{2}, \\ \eta &= \eta_i + (\tilde{\eta} + 1)\,\frac{\eta_{i+1} - \eta_i}{2} = \frac{(\eta_{i+1} - \eta_i)\,\tilde{\eta} + (\eta_{i+1} + \eta_i)}{2}. \end{aligned} \tag{3.1.2}$$

A parameter- and a parent space are shown in Figure 3.3a and Figure 3.3b respectively.

3.1.2. Elements - From an IGA Point of View

The elements in the FEM are usually defined by nodal coordinates and one often performs a triangulation of the elements for representation of the physical mesh. The basis functions are evaluated at the nodal points and are typically interpolatory and can take on both positive and negative values [8]. When we mention elements in the context of IGA, we mean knot intervals (tensor product there of) defined by the knot vectors in the parameter space. The reader familiar with FEM knows that the resulting stiffness matrices for the PDEs are usually sparse. The basis functions used in IGA (here B-splines and NURBS) have support that spans over multiple elements (knot intervals). One might be tempted to think that this affects the sparsity of the stiffness matrices. However, we recall from Chapter 2 that both B-splines and NURBS are compactly supported, thus the stiffness matrices remain sparse by using B-splines and NURBS as basis functions. A B-spline of polynomial degree p will at each knot span have at most $p + 1$ non-zero functions. Such that for a tensor product spline or NURBS function, the number of non-zero basis functions on each element is at most $(p+1) \times (q+1)$, whereas the total number of global basis functions is $n \times m$. As long as n and m are greater than p and q we have a sparse system of equations.

3.1.3. Integration by Gaussian Quadratures

During the evaluation of both the stiffness matrix and the force vector used to solve any PDE with FEM or IGA, one needs to perform numerical integration. The approach used here is Gaussian quadrature. Gaussian quadratures with np points in 1D integrates exactly polynomials of degree $2np - 1$. This approach is sufficient for B-splines, but not optimal for NURBS, in the sense that the integrands are not polynomials, but rather rational functions. However, Gaussian quadrature seems to be effective for approximating the integrals [8] and it is the only method of numerical integration considered in this thesis. Typically, the number of quadrature points is chosen to be $np = (p+1)(q+1)$, where p and q are the polynomial degrees of the B-splines in ξ and η directions respectively. The integration will take place over the unit square, $[\tilde{\xi}, \tilde{\eta}] = [-1, 1] \times [-1, 1]$, whereas the basis functions are defined in the parameter space and the differential equation is defined in the physical space. In order to carry out the integration correctly, the functions must be mapped to the unit square. The physical domain parameterization will map the functions from the physical space to the parameter space, and the second mapping will be a map

from parameter space to the unit square.

In order for the parameterization to be valid i.e. a diffeomorphism between the physical space and the parameter space, one must ensure that the Jacobian determinant does not vanish on Ω [7]. Hence, it cannot be allowed to change sign. When used for integration we always use the absolute value of the determinant, so the Jacobian itself may be negative, if it always stays negative. The sign of the Jacobian depends on how the coordinate system is defined. By using the right hand rule for vectors when establishing the coordinates it will be positive, and if the axis are turned it will be negative. However, a good rule is to just define the coordinate system in such a way that the Jacobian mapping is always positive. We denote the Jacobian and its inverse by

$$J = \begin{bmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{bmatrix} = J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}, \quad J^{-1} = \begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix}.$$

The expression for J^{-1} can be found in any elementary text on linear algebra, e.g. [1], and is given by

$$J^{-1} = \frac{1}{x_\xi y_\eta - x_\eta y_\xi} \begin{bmatrix} y_\eta & -x_\eta \\ -y_\xi & x_\xi \end{bmatrix}. \tag{3.1.3}$$

The gradient in physical space is denoted by

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix},$$

but due to the NURBS and B-splines being defined in the parameter space, the gradients need to be computed with respect to ξ and η. We denote this gradient by $\bar{\nabla}$ and it follows by the chain rule for differentiation that it is given by

$$\bar{\nabla} = J^{-T} \cdot \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix}, \tag{3.1.4}$$

where J^{-T} means the transpose of J^{-1} that can be calculated according to identity (3.1.3).

Chapter 4

Analysis and Solutions for Poisson's Equation

This chapter is dedicated to solving the two dimensional Poisson equation using IGA. Section 4.1 introduces some basic theory on the equation and uniqueness of solutions. In addition we introduce the error estimates that will be used to check convergence for the numerical solutions. In Section 4.2 the variational formulation is derived. The main aspects of the assembly and implementation processes are introduced from an IGA point of view. Finally, in Section 4.3 the obtained numerical results are given. We include examples of homogeneous Dirichlet-, mixed Neumann and Dirichlet- and inhomogeneous Robin boundary conditions. In the end we use NURBS to solve Poisson's equation on a circular domain. Although two dimensions are considered here, everything can easily be extended to higher dimensions as well.

4.1. Preliminaries

Poisson's equation reads

$$\begin{aligned} -\Delta u &= f, \quad \text{in } \Omega \\ +\ &\text{boundary conditions}, \quad \text{on } \partial\Omega. \end{aligned} \tag{4.1.1}$$

The boundary conditions can be of the following types:

1)

$$u = g_D, \quad \text{on } \partial\Omega \quad \text{(Dirichlet)},$$

2)

$$\mathbf{n} \cdot \nabla u = g_N, \quad \text{on } \partial\Omega \quad \text{(Neumann)},$$

3)

$$\alpha u + \mathbf{n} \cdot \beta \nabla u = g_R, \qquad \text{on } \partial\Omega \qquad \text{(Robin)}\,,$$

or a mix of the three mentioned boundary conditions. The boundary conditions are said to be homogeneous whenever the right hand side $g = 0$, and otherwise inhomogeneous.

Definition 4.1.1 (**Well-Posedness**)**.** A PDE is said to be well-posed if it satisfies the following criteria:

1) There exists a solution.

2) The solution is unique.

3) The solution depends continuously on the data (for Poisson's equation, this means boundary conditions g and the right hand side f). This means that small changes in the data will only lead to small changes in the solution and is also known under the term stability.

Theorem 4.1.1 (**Uniqueness of solutions to Poisson's equation**)**.** *Assume Ω to be a piecewise smooth, bounded domain. If there exists a solution $u \in C^2(\Omega) \cap C^1(\bar{\Omega})$ to Poisson's equation, the solution is unique. In case of pure Neumann boundary conditions the solution is unique up to a constant.*

Proof. We prove uniqueness for Poisson's equation with Dirichlet and Neumann boundary conditions. The proofs of uniqueness in the case of Robin, or mixed boundary conditions are similar and are omitted here.

We first consider Poisson's equation with Dirichlet boundary conditions. Assume two solutions u and v exist and that they both satisfy Equation (4.1.1) with Dirichlet boundary conditions. Define a new solution $w := u - v$. Then w satisfies the homogeneous equation

$$\begin{aligned} -\Delta w &= 0, \qquad \text{in } \Omega \\ w &= 0, \qquad \text{on } \partial\Omega. \end{aligned} \tag{4.1.2}$$

We multiply equation (4.1.2) by w and apply Green's first identity to get

$$\int_\Omega w \Delta w \, \mathrm{d}\mathbf{x} = \int_{\partial\Omega} w \partial_\nu w \, \mathrm{d}S - \int_\Omega |\nabla w|^2 \, \mathrm{d}\mathbf{x}. \tag{4.1.3}$$

Exploiting Equation (4.1.2), all terms except for the last vanish, leaving us with

$$\int_\Omega |\nabla w|^2 \, \mathrm{d}\mathbf{x} = 0,$$

which can only happen if $|\nabla w|^2 = 0$, as Ω is a domain with positive measure. Hence, $\nabla w = 0 \implies w = \text{constant}$. This constant must be zero, since $w = 0$ on $\partial\Omega$. Thus, we arrive at the desired conclusion that $u = v$.

In case of pure Neumann conditions, the only information we have is that $\mathbf{n} \cdot \nabla w = 0$ on $\partial\Omega$, and hence we deduce that w is a constant, implying $u = v + k$, for some constant k. □

For Poisson's equation with force term f and pure Neumann boundary conditions g, it trivially follows that

$$\int_\Omega f \, \mathrm{d}\mathbf{x} = \int_{\partial\Omega} g \, \mathrm{d}S. \tag{4.1.4}$$

Equation (4.1.4) is known as the Neumann compatibility condition, and no solutions exist if this is not satisfied [24].

4.1.1. Error Estimates

Throughout this thesis, the distance between the obtained numerical solution of a PDE and the exact solution of the PDE is measured in the infinity norm. To ensure that the numerical solutions we find are correct, we need to establish an estimate for an acceptable convergence rate. According to [8], NURBS and hence also B-splines can converge at the same rate as the FEM basis functions, and we will base our error estimates on the well established error estimates for the FEM.

Let u^h_{num} be the FEM approximation of the exact solution u to Poisson's equation, where h is the size of the elements in the mesh. Then, as $h \to 0$, the L^∞ norm of the error $u^h_{\text{num}} - u$ will tend to zero as well. By an approximation with piecewise continuous polynomials of degree p, we expect the difference from the numerical solution to the exact solution to be of order $\mathcal{O}(h^{p+1})$, which we write by the following bound:

$$\|u^h_{\text{num}} - u\|_\infty \leq C \cdot h^{p+1}, \tag{4.1.5}$$

for $p \geq 2$. In the case of piecewise linear polynomials the error will decrease at least with rate $h^2|\log h|$ [25]. For further details and proof, see [25]. Our convergence tests are based on the error estimate in (4.1.5). In our case, h represents the size of the knot intervals and p represents the polynomial degree of the B-splines. Define the relative error e_{rel} by

$$e_{rel} = \frac{\|u^h_{\text{num}} - u_{\text{exact}}\|_\infty}{\|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty}. \tag{4.1.6}$$

Thus in the process of knot refinement (replacing h by $h/2$), we expect the relative error e_{rel} to be of order

$$e_{rel} \approx \mathcal{O}(2^{p+1}). \tag{4.1.7}$$

4.2. The Numerical Method - Poisson's Equation in $\mathbb{R}^2$

This section is devoted to explain the numerical method we use to solve the two dimensional Poisson equation: $-\Delta u = f$ on $\Omega := \{(x, y) \in [0, 1] \times [0, 1]\}$. We derive the variational formulation for the following problem:

$$\begin{aligned} -\Delta u &= f, \quad \text{in } \Omega, \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned} \tag{4.2.1}$$

In addition, we briefly discuss the variational formulation for Neumann and Robin boundary conditions. Some aspects of the computer implementation are explained in Section 4.2.2.

4.2.1. Variational Formulation

Equation (4.2.1) is referred to as the strong form of the PDE, meaning that the equation holds at every point of the domain. The approach used in FEM and IGA is to look at the weak formulation of the equation instead. A PDE in weak form satisfies the equation in the sense of distributions, and not every point in the domain has to satisfy the strong form. A benefit of the weak form is that the solution needs not be as regular as the strong form states. For example the solution of Poisson's equation in strong form is required to be at least two times differentiable, but when the equation is in weak form it is sufficient that the numerical solution is one time weakly differentiable. To get the equation over to the weak form, Equation (4.2.1) is multiplied with a test function $\phi \in W_0^{1,2}(\Omega)$ and integrated over the domain Ω. The space $W_0^{1,2}(\Omega)$ is a Sobolev space, that is, a complete vector space equipped with a norm measuring both length and regularity of the argument. The reader is encouraged to consult e.g. [6] for further reading and definitions.

$$\int_\Omega -\phi \Delta u \, \mathrm{d}\mathbf{x} = \int_\Omega \phi f \, \mathrm{d}\mathbf{x}. \tag{4.2.2}$$

By applying Green's first identity, Equation (4.2.2) can be expressed as follows:

$$\int_\Omega \nabla u \cdot \nabla \phi \, \mathrm{d}\mathbf{x} = \int_{\partial\Omega} \phi \partial_\nu u \, \mathrm{d}S \; + \int_\Omega \phi f \, \mathrm{d}\mathbf{x}. \tag{4.2.3}$$

Since ϕ belongs to $W_0^{1,2}(\Omega)$ it is zero on the the boundary. The weak form is obtained as

$$\int_\Omega \nabla u \cdot \nabla \phi \, \mathrm{d}\mathbf{x} \; = \int_\Omega \phi f \, \mathrm{d}\mathbf{x}, \qquad \forall \phi \in W_0^{1,2}(\Omega). \tag{4.2.4}$$

Thus, we seek to find $u \in \mathrm{W}_0^{1,2}(\Omega)$ such that

$$A(u, \phi) = l(\phi), \qquad \forall\ \phi \in\ W_0^{1,2}(\Omega), \tag{4.2.5}$$

where A is the bilinear form corresponding to the left hand side in Equation (4.2.4) and l is the linear form corresponding to the right hand side in Equation (4.2.4). The bilinear form A is a function for which the following hold:

$$\begin{aligned} A(u+v, w) &= A(u, w) + A(v, w), \\ A(u, v+w) &= A(u, v) + A(u, w), \\ A(\lambda u, v) &= A(u, \lambda v) = \lambda\ A(u, v), \end{aligned} \tag{4.2.6}$$

and the linear l satisfies:

$$\begin{aligned} l(u+v) &= l(u) + l(v), \\ l(\gamma u) &= \gamma l(u). \end{aligned}$$

The test functions ϕ will in our case be NURBS or B-splines that we denote by $R(\xi, \eta)$. We will now search for a solution in a finite dimensional subspace $S^h \subset S := \{u | u \in W^{1,2}(\Omega),\ u = g \text{ on } \partial\Omega_D\}$. The space S^h is defined by

$$S^h = \{R_i \in S\ : \sum_{i=1}^{n} R_i(\xi, \eta) B_i \in \Omega\}.$$

The final solution $u \in \mathrm{S}^h$ can be written as

$$u = \sum_{i=1}^{n} u_i R_i, \tag{4.2.7}$$

and by inserting (4.2.7) into (4.2.5) and exploiting the bilinearity of A it is clear that (4.2.5) can be written as

$$\sum_{i=1}^{n} A(R_i, R_j) u_i = l(R_j), \quad \forall R_j \in S^h. \tag{4.2.8}$$

Note that (4.2.8) is a system of n linear equations, that can be written

$$\mathbf{K}\mathbf{u} = \mathbf{b}.$$

We refer to $\mathbf{K}$ as the stiffness matrix and to $\mathbf{b}$ as the force vector. Their entries are given by

$$K_{i,j} = \int_\Omega \nabla R_i(\xi,\eta)' \nabla R_j(\xi,\eta)\ \mathrm{d}\mathbf{x},$$
$$b_i = \int_\Omega R_i(\xi,\eta) f(x,y)\ \mathrm{d}\mathbf{x}.$$

Recall from the discussion in Chapter 3 that the integrals must be mapped to the unit square to perform the numerical integration. This is done in the following steps:

$$\begin{aligned} K_{i,j} &= \iint_\Omega (\nabla R_i)'(\nabla R_j)\ \mathrm{d}x\mathrm{d}y \\ &= \iint_{\bar{\Omega}} (\bar{\nabla} R_i)'(\bar{\nabla} R_j)|J|\ \mathrm{d}\xi\mathrm{d}\eta \\ &= \iint_{\tilde{\Omega}} (\tilde{\nabla} R_i)'(\tilde{\nabla} R_j)|J||\tilde{J}|\ \mathrm{d}\tilde{\xi}\mathrm{d}\tilde{\eta} \\ &\approx \sum_\alpha \sum_\beta (\tilde{\nabla} R_i)'(\tilde{\nabla} R_j) w(\alpha) w(\beta) |J||\tilde{J}|. \end{aligned}$$

The force term b_i is constructed similarly,

$$\begin{aligned} b_i &= \iint_\Omega R_i f(x,y)\ \mathrm{d}x\mathrm{d}y \\ &\approx \sum_\alpha \sum_\beta R_i f(x,y) w(\alpha) w(\beta) |J||\tilde{J}|, \end{aligned}$$

where α and β are the indices of the quadrature points and w is the corresponding quadrature weight. The stiffness matrix $\mathbf{K}$ is symmetric and at this stage singular. This is due to the fact that the system will not have a unique solution until the desired boundary conditions are prescribed. For homogeneous Dirichlet boundary conditions, we remove the rows and columns of $\mathbf{K}$, that correspond to the Dirichlet edge. Likewise, the rows of $\mathbf{f}$ are removed before the system of linear equations is solved. When we are dealing with Neumann or Robin boundary conditions, the boundary terms must be taken care of before the system is being solved.

4.2.1.1. *The Case of Neumann Boundary Conditions*

For Neumann boundary conditions, that is when

$$\partial_\nu u = g, \quad \text{on } \partial\Omega,$$

the weak formulation takes form as:

$$\int_\Omega \nabla u \cdot \nabla v\ \mathrm{d}\mathbf{x} = \int_{\partial\Omega} v \partial_\nu u\ \mathrm{d}S + \int_\Omega v f\ \mathrm{d}\mathbf{x}, \tag{4.2.9}$$

which can be written as

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x} = \int_\Omega vf \, \mathrm{d}\mathbf{x} + \int_{\partial\Omega} gv \, \mathrm{d}S. \tag{4.2.10}$$

When implementing Neumann boundary conditions, the boundary integral is added to the global force vector $\mathbf{b}$. We stress once again that Neumann boundary conditions may only be used in the case of mixed boundary conditions for Poisson's equation whenever a unique solution is desired.

4.2.1.2. *The Case of Robin Boundary Conditions*

For Robin boundary conditions, that is when

$$\alpha(x, y)u + \partial_\nu u = g, \quad \text{on } \partial\Omega,$$

the weak formulation takes form as:

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x} = \int_{\partial\Omega} v\partial_\nu u \, \mathrm{d}S + \int_\Omega vf \, \mathrm{d}\mathbf{x}, \tag{4.2.11}$$

which can be written as

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x} + \int_{\partial\Omega} \alpha(x, y)uv \, \mathrm{d}S = \int_\Omega vf \, \mathrm{d}\mathbf{x} + \int_{\partial\Omega} gv \, \mathrm{d}S. \tag{4.2.12}$$

In this case the test functions $v \in W^{1,2}(\Omega)$. The first line integral in (4.2.12) is added to the stiffness matrix $\mathbf{K}$, whereas the line integral on the right hand side is added to right hand side $\mathbf{b}$. As long as $\alpha(x, y) > 0$, the solution to the Poisson equation is unique when Robin boundary conditions are applied to the entire boundary.

4.2.2. Assembly Process

The computation of the global stiffness matrix $\mathbf{K}$ and the global force vector $\mathbf{b}$ is known under the term assembling. Due to the compact support of the basis functions the global stiffness matrix will be sparse. If p is the degree of the knot vector ξ and q is the degree of the knot vector η, then a maximum of $nen = (p + 1) \times (q + 1)$ basis functions will be non-zero on any element. Instead of looping over all the basis functions, we will loop over the elements and for each element construct local stiffness matrices of size $(nen \times nen)$. The local matrices will now, of course, be dense. We find out where the support of each basis function function starts through a NURBS coordinate array, hereafter called the INC array. This array takes a global index number, and gives as output the NURBS coordinates in both ξ and η directions. A NURBS coordinate gives the index of where a basis function starts in the index space, and we follow [8] and refer to these coordinates as n_i or n_j. Once the local matrices and vectors are constructed it remains to assemble these into a global system of equations. Thus, we need a connectivity array, that connects the local and global indices. This connectivity array is usually denoted by the IEN array. Given a local index and an element number, the IEN array outputs the corresponding global index number. Hence, these two connectivity arrays let us place each entry of the local stiffness matrices and the local force vectors to its correct place in the global system.

Local basis function number	1	2	3	4	5	6	7	8	9
Element 1	11	10	9	7	6	5	3	2	1
Element 2	12	11	10	8	7	6	4	3	2

Table 4.1: Example of IEN array, for $n = 4$, $m = 3$, $p, q = 2$.

Global basis function number	1	2	3	4	5	6	7	8	9	10	11	12
ξ	1	2	3	4	1	2	3	4	1	2	3	4
η	1	1	1	1	2	2	2	2	3	3	3	3

Table 4.2: Example of INC array, for $n = 4$, $m = 3$, $p, q = 2$.

Examples of the IEN and INC arrays are shown in Table 4.1 and Table 4.2 respectively. In these examples the number of basis functions in ξ and η directions are 4 and 3 respectively, whereas the polynomial degrees of the basis functions are chosen to be $p = q = 2$.

The algorithms for the IEN aray and the INC array can be found in [8]. This goes also for the routine that computes the NURBS and the Jacobian. The script that computes the stiffness matrix and the force vector for the two dimensional Poisson equation with homogeneous Dirichlet boundary conditions is attached in Appendix A.1.

4.3. Numerical Results

In this section a selection of numerical examples is studied. The objective of this study is to verify that the IGA solver is implemented correctly. The equations solved are all versions of Poisson's equation, with different source terms and different boundary conditions. The domain Ω is the unit square and B-splines are used as basis functions throughout this section with exception of Section 4.3.4.

4.3.1. Homogeneous Dirichlet Boundary Conditions

We start by considering Poisson's equation with homogeneous Dirichlet boundary conditions. The problem goes as follows:

$$\begin{aligned} -\Delta u &= f, \quad \text{in } \Omega \\ u &= 0, \quad \text{on } \partial\Omega, \end{aligned} \tag{4.3.1}$$

with f given by

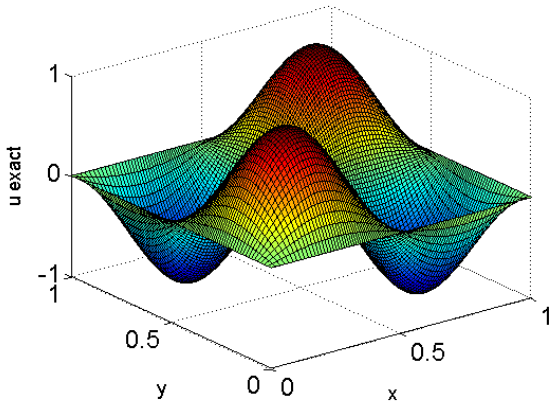
$$f(x, y) = 8\pi^2 \sin(2\pi x) \sin(2\pi y). \tag{4.3.2}$$

4.3. Numerical Results

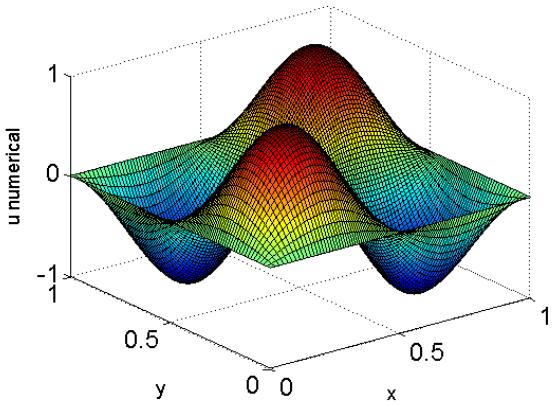
It is easy to verify that

$$\hat{u}(x, y) = \sin(2\pi x)\sin(2\pi y),$$

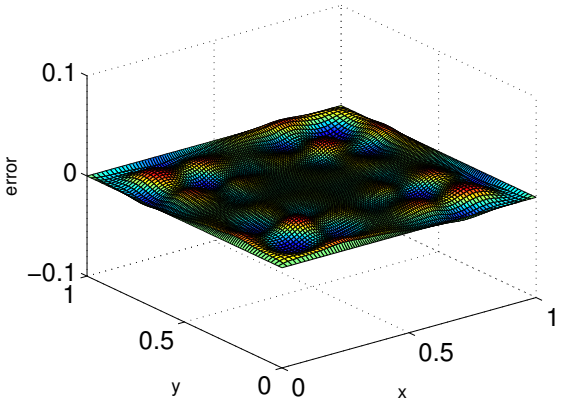
is the exact solution to Equation (4.3.1).



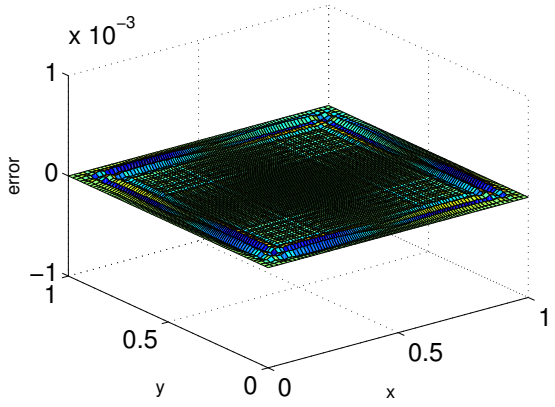
(a) Exact solution



(b) Numerical solution with cubic B-splines and a total number of 25 degrees of freedom.



(c) Difference between exact and numerical solution for a total of 25 degrees of freedom.



(d) Difference between exact and numerical solution for a total number of 1089 degrees of freedom.

Figure 4.1: The two dimensional Poisson equation solved on the unit square with force term f given by (4.3.2).

Figure 4.1 shows the exact solution along with the numerical solution with cubic B-splines and a total number of 25 degrees of freedom. In addition two plots of the error are shown. Figure 4.1c shows the difference between the two solutions with 25 degrees of freedom, whereas Figure 4.1d shows the difference between the two solutions for a total number of 1089 degrees of freedom.

To verify that the IGA solver is implemented correctly, the difference between the exact and the numerical solution is measured in the infinity norm, i.e.

$$\text{dist}(u_{\text{num}}, u_{\text{exact}}) = \|u_{\text{num}} - u_{\text{exact}}\|_{\infty}. \tag{4.3.3}$$

p	h	$\|u_{\text{num}} - u_{\text{exact}}\|_\infty$
1	1/4	14.893
1	1/8	4.552
1	1/16	1.192
1	1/32	0.297
1	1/64	0.076
2	1/4	1.959
2	1/8	0.953
2	1/16	0.104
2	1/32	0.0124
2	1/64	0.0013
3	1/4	0.4902
3	1/8	0.385
3	1/16	0.029
3	1/32	0.002
3	1/64	0.00015

Table 4.3: Infinity norm of $u_{\text{num}} - u_{\text{exact}}$. The degree of the B-splines used are as usually denoted by p, and h denotes the size of the knot intervals.

p	h	$\frac{\|u^h_{\text{num}} - u_{\text{exact}}\|_\infty}{\|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty}$	$C \cdot 2^{p+1}$
1	1/4	3.272	$0.818 \cdot 2^2$
1	1/8	3.820	$0.955 \cdot 2^2$
1	1/16	4.014	$1.004 \cdot 2^2$
1	1/32	3.908	$0.977 \cdot 2^2$
2	1/4	2.056	$0.260 \cdot 2^3$
2	1/8	9.164	$1.150 \cdot 2^3$
2	1/16	8.390	$1.05 \cdot 2^3$
2	1/32	9.540	$1.193 \cdot 2^3$
3	1/4	1.273	$0.080 \cdot 2^4$
3	1/8	13.276	$0.830 \cdot 2^4$
3	1/16	12.610	$0.788 \cdot 2^4$
3	1/32	15.541	$0.971 \cdot 2^4$

Table 4.4: Table of $\|u^h_{\text{num}} - u_{\text{exact}}\|_\infty / \|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty$.

4.3. Numerical Results

Table 4.3 shows the infinity norm of the difference from numerical to exact solution for different number of knots and different polynomial degrees of the B-splines. To begin with, let

$$\xi = \eta = \{0 \ \ldots \ 0 \ 0.25 \ 0.5 \ 0.75 \ 1 \ \ldots \ 1\}. \tag{4.3.4}$$

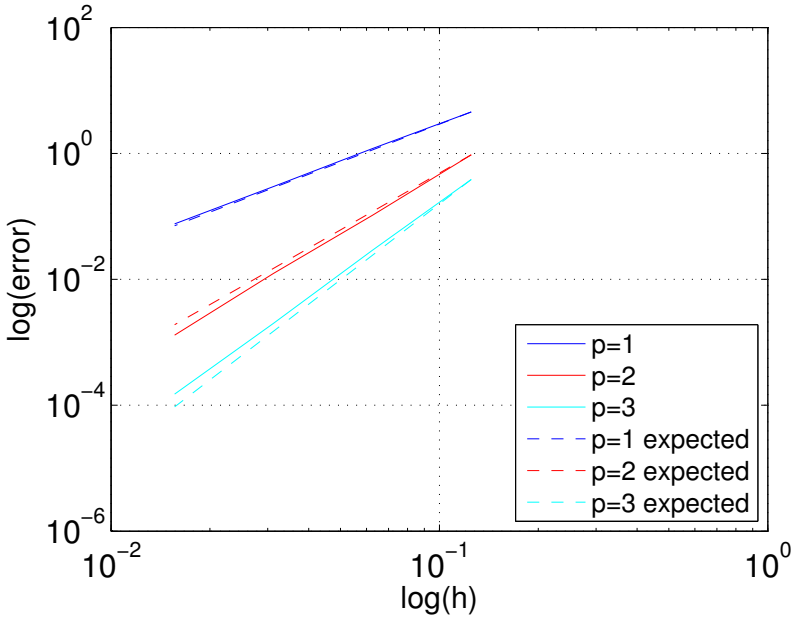


Figure 4.2: Logarithm of the size of the knot intervals h plotted against the logarithm of the error. Results are taken from Table 4.3. The dashed lines show how the error is expected to decrease.

Knot refinement is performed, where the new control points are calculated as explained in Section 2.1.3. When inserting new knots, it is important that the new knot vectors include all of the original knots as well. Thus all the new knot vectors will include the original knot vectors as subsequences. The relative norms after decreasing the knot intervals by a factor 2 are shown in Table 4.4. We see from Table 4.4 that the error decreases sufficiently for the relative error to satisfy (4.1.7).

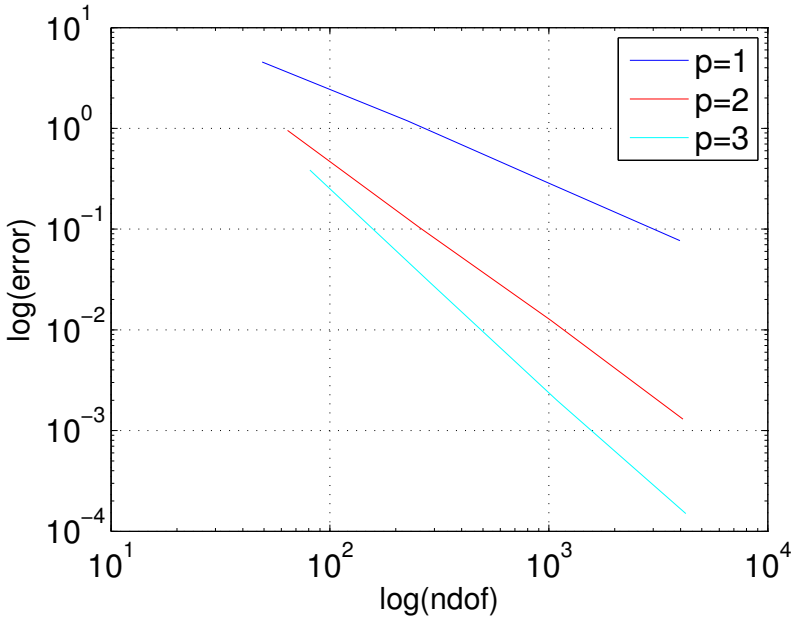


Figure 4.3: Logarithm of the number of degrees of freedom (ndof) plotted against the logarithm of the error.

Figure 4.2 visualizes the results from Table 4.3. The dashed lines show how the error is expected to drop by refining the size of the knot intervals by a factor of 2. Some deviations between the two plots are observed. However, this plot together with the results from Table 4.4 indicate that the computer program gives correct results, as the results are asymptotically within the expected results of the relative error (4.1.7).

4.3.2. Mixed Boundary Conditions

Both Neumann, Robin and mixed boundary conditions are needed to solve the problem that we will introduce in Chapter 7. This section is devoted to Poisson's equation in 2D with mixed homogeneous Dirichlet and inhomogeneous Neumann boundary conditions, before we proceed with Robin boundary conditions in the next section. Let Ω be the unit square: $\Omega := [0,1] \times [0,1]$ and consider the exact solution $\hat{u}$ given by

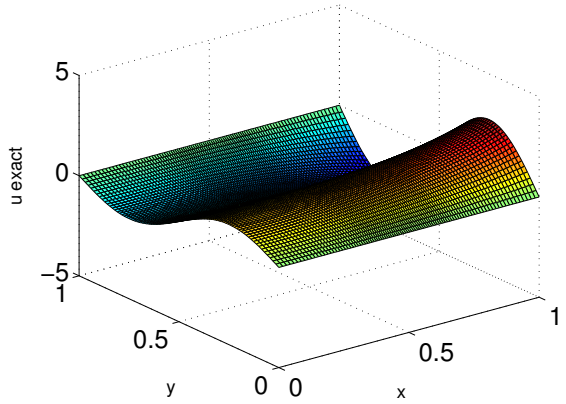
$$\hat{u}(x,y) = \sin(2\pi y)\mathrm{e}^{x^2}. \tag{4.3.5}$$

We impose homogeneous Dirichlet boundary conditions along the x-axis when $y = 0$, and implement inhomogeneous Neumann conditions on the three other edges. The problem goes as follows:

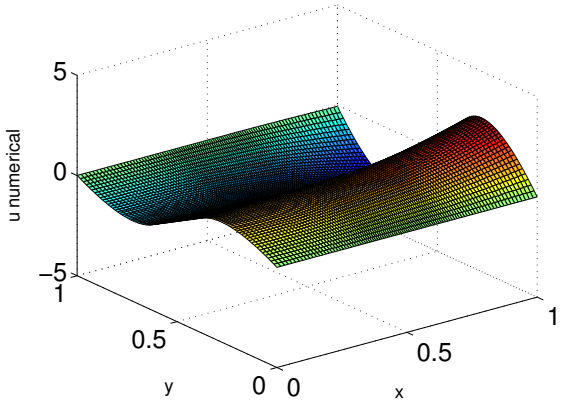
$$\begin{aligned} -\Delta u &= f, \quad \text{in } \Omega, \\ u &= 0, \quad \text{on } \{(x,0)|\ x \in [0,1]\}, \\ \partial_\nu u &= \partial_\nu \hat{u}, \quad \text{on } \{\partial\Omega \setminus (x,0)|\ x \in [0,1]\}, \end{aligned} \tag{4.3.6}$$

with f given by

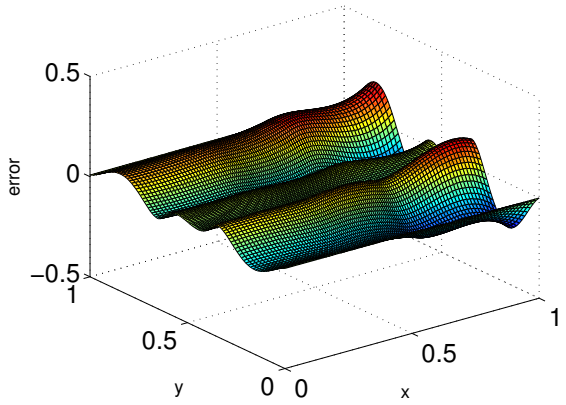
$$f(x,y) = \left(4\pi^2 - 4x^2 - 2\right)\sin(2\pi y)\mathrm{e}^{x^2}.$$



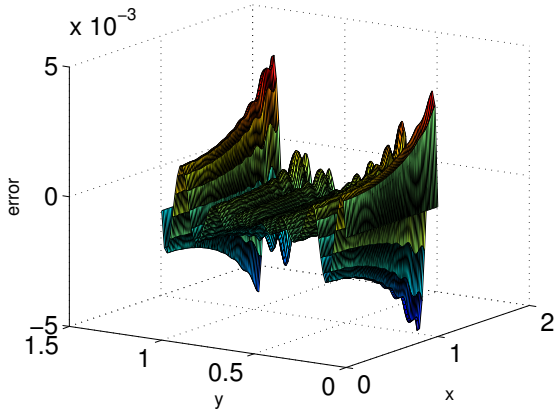
(a) Exact solution



(b) Numerical solution with quadratic B-splines and 25 degrees of freedom.



(c) Difference between Figure 4.4a and Figure 4.4b.



(d) Difference between the exact and the numerical solution solved with cubic B-splines and 121 degrees of freedom.

Figure 4.4: Exact and numerical solution of Equation (4.3.6) together with error plots, showing the difference between the exact and the numerical solution. The axis where $y = 0$ is imposed with homogeneous Dirichlet boundary conditions, whereas the three other edges have inhomogeneous Neumann boundary conditions.

Figure 4.4 shows the exact solution of Equation (4.3.5) together with the numerical solution of Equation (4.3.6). Quadratic B-splines are used for the solution in Figure 4.4b with 25 degrees of freedom. The deviation from the exact to the numerical solution is shown for two cases. For the error plot in Figure 4.4c we have used quadratic B-splines and a number of 25 degrees of freedom, whereas the error plot in Figure 4.4d shows the error when cubic B-splines are used and a total number of 121 degrees of freedom.

p	h	$\|u_{\text{num}} - u_{\text{exact}}\|_\infty$
1	1/4	32.561
1	1/8	7.629
1	1/16	1.879
1	1/32	0.471
1	1/64	0.118
1	1/128	0.0293
2	1/3	8.516
2	1/6	2.636
2	1/12	0.254
2	1/24	0.026
2	1/48	0.003
2	1/96	$3.97 \cdot 10^{-4}$
3	1/2	4.914
3	1/4	3.383
3	1/8	0.132
3	1/16	0.007
3	1/32	$5.12 \cdot 10^{-4}$
3	1/64	$3.15 \cdot 10^{-5}$

Table 4.5: Infinity norm of $u_{\text{num}} - u_{\text{exact}}$.

4.3. Numerical Results

p	h	$\frac{\|u^h_{\text{num}} - u_{\text{exact}}\|_\infty}{\|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty}$	$C \cdot 2^{p+1}$
1	1/4	4.268	$1.067 \cdot 2^2$
1	1/8	4.059	$1.015 \cdot 2^2$
1	1/16	3.984	$0.996 \cdot 2^2$
1	1/32	3.992	$0.998 \cdot 2^2$
1	1/64	4.069	$1.017 \cdot 2^2$
2	1/3	3.231	$0.404 \cdot 2^3$
2	1/6	6.593	$0.824 \cdot 2^3$
2	1/12	10.378	$1.297 \cdot 2^3$
2	1/24	9.770	$1.221 \cdot 2^3$
2	1/48	8.667	$1.083 \cdot 2^3$
3	1/2	1.453	$0.091 \cdot 2^4$
3	1/4	25.630	$1.602 \cdot 2^4$
3	1/8	18.857	$1.179 \cdot 2^4$
3	1/16	13.672	$0.855 \cdot 2^4$
3	1/32	16.254	$1.016 \cdot 2^4$

Table 4.6: Table of $\|u^h_{\text{num}} - u_{\text{exact}}\|_\infty / \|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty$.

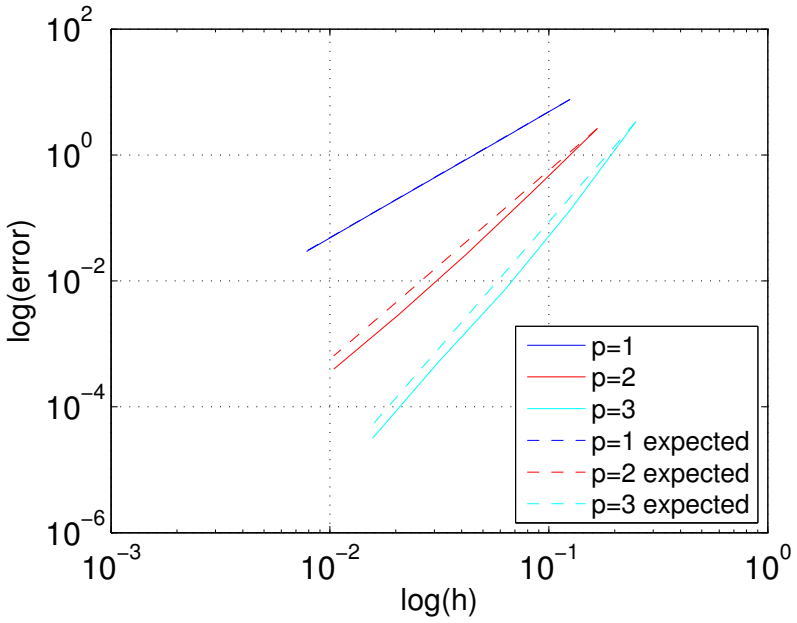


Figure 4.5: Logarithm of the size of the knot intervals h plotted against the logarithm of the error that are listed in Table 4.5. The dashed lines show how the error is expected to decrease by refining the knot intervals by a factor of 2. For $p = 1$, we note that the actual convergence rate is almost identical to the expected convergence rate.

The error plots, showing the difference between the exact and the numerical solutions are

meant to give a rough image of the error. However, a more detailed error analysis is shown in Table 4.5 and Table 4.6. We use the same procedure as we did in Section 4.3.1. Table 4.5 shows the difference between the exact solution and the numerical solution measured in the infinity norm. From these results it is observed that the error decreases as more knots are inserted, and the error becomes close to zero, at least in the cases where we use quadratic and cubic B-splines. However pleasing the results in Table 4.5 are, they are not enough to verify that the solutions are indeed correct. Table 4.6 serves to ensure us that the error decreases with an appropriate rate in order for the relative error to satisfy Equation (4.1.7). Observe from Table 4.6 that the step size between the knots is decreased by a factor of 2 for each iteration of solving Equation (4.3.6). Both when quadratic- and cubic B-splines are used, there is not much reduction in the error between the first iteration and the second iteration. Eventually, however, the relative error seems to stabilize as more knots are inserted.

The results from Table 4.5 are visualized in the log-log plot in Figure 4.5. The logarithm of the size of the knot intervals is plotted against the logarithm of the error for the different degrees of B-splines. The dashed lines show how the error is expected to drop, by refining the size of the knot intervals by a factor of 2. The results shown in Figure 4.5 together with the results from Table 4.6 lead us to conclude that the error decreases within an acceptable rate, according to the expected expression for the relative error (4.1.7).

4.3.3. Robin Boundary Conditions

As Robin boundary conditions are to be applied for a more complex problem later in this thesis, a simple example is considered in this section for verification of correct implemented boundary conditions. The considered equation is

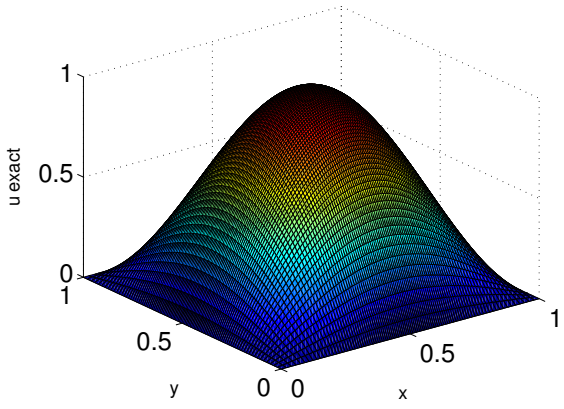
$$\hat{u} = \sin(\pi x)\sin(\pi y), \tag{4.3.7}$$

on the domain $\Omega := [0,1] \times [0,1]$. Poisson's equation is then constructed to obtain comparable solutions by the following:

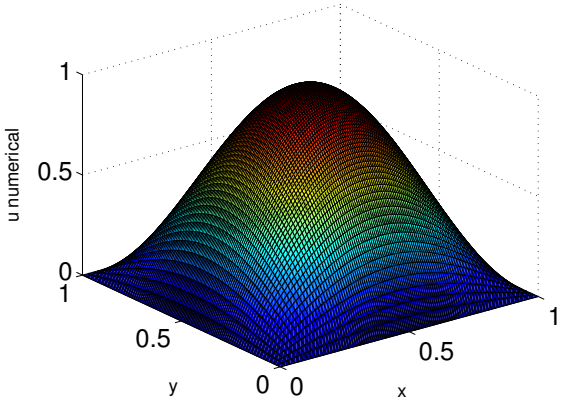
$$\begin{aligned} -\Delta u &= f, \quad \text{in } \Omega \\ \alpha u + \partial_\nu u &= \alpha \hat{u} + \partial_\nu \hat{u}, \quad \text{on } \partial\Omega, \end{aligned} \tag{4.3.8}$$

with f given by

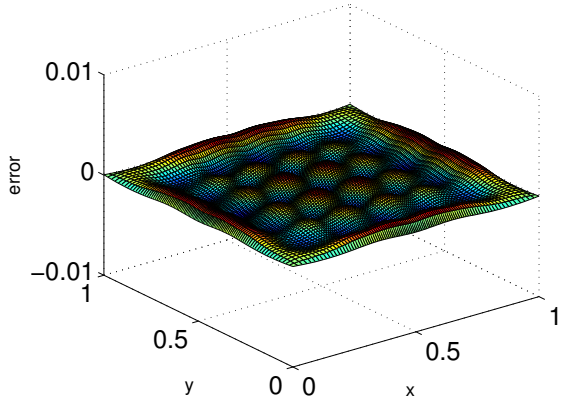
$$f(x,y) = 2\pi^2 \sin(\pi x)\sin(\pi y). \tag{4.3.9}$$



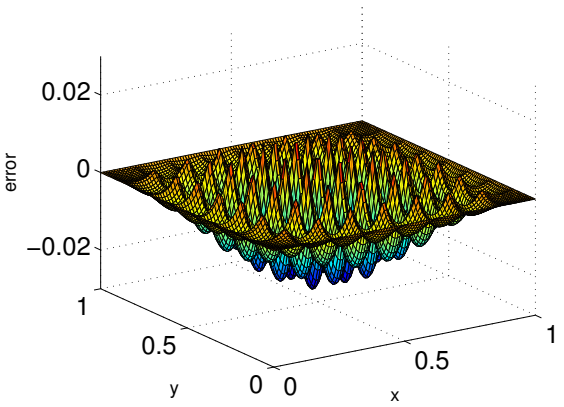
(a) Exact solution.



(b) Plot of numerical solution to Equation (4.3.8) with cubic basis functions and 49 degrees of freedom.



(c) Plot of error of the solution in Figure 4.6b.



(d) Plot of error with linear basis functions and 81 degrees of freedom.

Figure 4.6: Plot of Equation (4.3.7) and the numerical solution of Equation (4.3.8) together with plots of the error for cubic and linear basis functions.

p	h	$\|u_{\text{num}} - u_{\text{exact}}\|_\infty$
1	1/4	4.544
1	1/8	1.192
1	1/16	0.297
1	1/32	0.076
1	1/64	0.019
2	1/4	0.3445
2	1/8	0.1452
2	1/16	0.0141
2	1/32	0.002
2	1/64	$1.86 \cdot 10^{-4}$
3	1/4	0.076
3	1/8	0.057
3	1/16	0.004
3	1/32	$3.00 \cdot 10^{-4}$
3	1/64	$2.00 \cdot 10^{-5}$

Table 4.7: Infinity norm of $u_{\text{num}} - u_{\text{exact}}$.

p	h	$\frac{\|u^h_{num} - u_{\text{exact}}\|_\infty}{\|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty}$	$C \cdot 2^{p+1}$
1	1/4	3.812	$0.953 \cdot 2^2$
1	1/8	4.013	$1.003 \cdot 2^2$
1	1/16	3.908	$0.98 \cdot 2^2$
1	1/32	4.000	$1.00 \cdot 2^2$
2	1/4	2.371	$0.296 \cdot 2^3$
2	1/8	10.298	$1.287 \cdot 2^3$
2	1/16	7.05	$0.881 \cdot 2^3$
2	1/32	10.75	$1.344 \cdot 2^3$
3	1/4	1.333	$0.0833 \cdot 2^4$
3	1/8	14.25	$0.891 \cdot 2^4$
3	1/16	13.333	$0.833 \cdot 2^4$
3	1/32	15	$0.938 \cdot 2^4$

Table 4.8: Table of $\|u^h_{\text{num}} - u_{\text{exact}}\|_\infty / \|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty$.

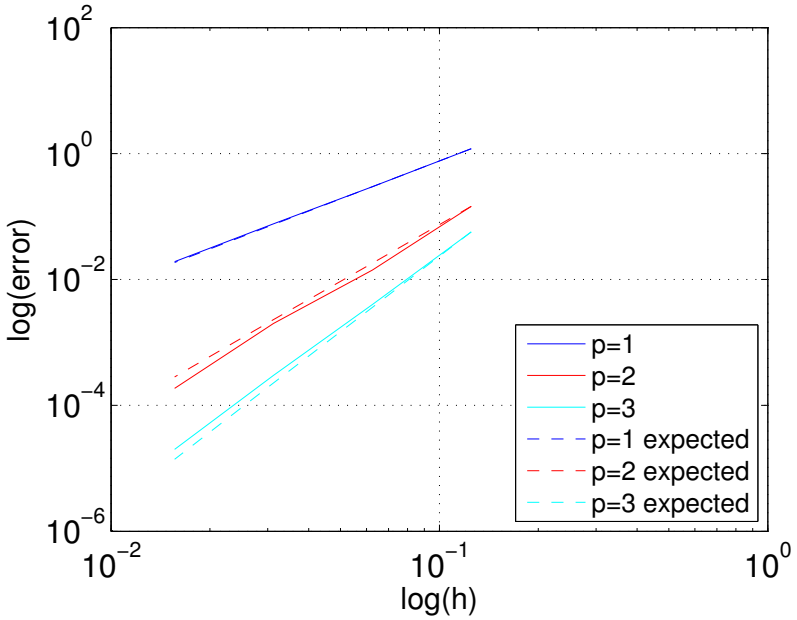


Figure 4.7: Logarithm of the size of the knot intervals h plotted against the logarithm of the error. Results are taken from Table 4.7. The dashed lines show how we expect the error to drop by reducing the size of the knot intervals by a factor of 2.

Figure 4.6 shows the exact solution, the numerical solution of Equation (4.3.8) together with plots of the error. Cubic B-splines are used for the solution in Figure 4.6b which is solved for 49 degrees of freedom. Figure 4.6c shows the corresponding error, while Figure 4.6d shows the error for the numerical solution with linear B-splines and 81 degrees of freedom. As a verification that the error decreases at expected speed, knot insertion is applied and the error is measured in the infinity norm. Knots are inserted such as the step length h between the knots is halved. This means that for $p = 1,\ 2$ and 3, the error is expected to drop by a factor of 4, 8 and 16 respectively for each time new knots are inserted according to the strategy explained above.

Table 4.7 shows the error measured in the infinity norm between the exact solution and the numerical solution for knot intervals of size h. We observe that the error decreases noticeably while the knot intervals are reduced. To ensure that the convergence rate is indeed correct, we calculate the relative error. These results can be seen in Table 4.8. The results in Table 4.7 are visualized in Figure 4.7, where the logarithm of the size of the knot intervals is plotted against the logarithm of the error, and as the results from Table 4.8 show as well, the log-log plots seem to decrease at an acceptable rate, recall (4.1.7). For $p = 3$ the error decreases almost from 10^{-2} to 10^{-6} when the step length goes from 10^{-1} to 10^{-2}, indicating that the Robin boundary conditions are implemented correctly. We can see directly from Table 4.8 that everything works nicely for $p = 1$. The tendency for $p = 2$ is that the error drops a little too fast. However, once again, all the results are asymptotically as expected.

4.3.4. A NURBS Based Example

In the previous examples the physical domain Ω was the unit square. As no conic section was to be represented we used B-splines as basis functions. In fact, Equation (2.1.11) was used with all weights $w_{i,j}$ equal to one, resulting in B-splines. In this section we modify the weights, and represent the geometry with NURBS, which are also used as basis functions for the numerical analysis.

We consider the domain Ω to be a quart disk in the first quadrant with a circular hole centered in the origin. The inner circular arc is of radius $r_1 = 1$ and the outer circular arc is of radius $r_2 = 2$. We construct a solution of Poisson's equation on Ω with homogeneous Dirichlet boundary conditions given by:

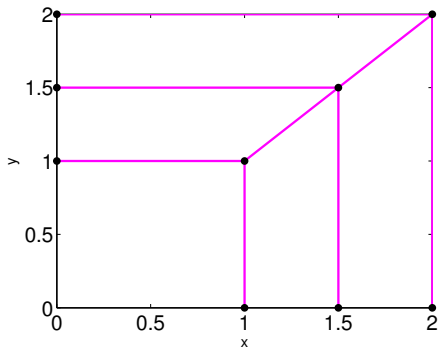
$$u(x, y) = \sin(xy)\left(x^2 + y^2 - r_1^2\right)\left(x^2 + y^2 - r_2^2\right). \tag{4.3.10}$$

Our numerical solution will be the solution of

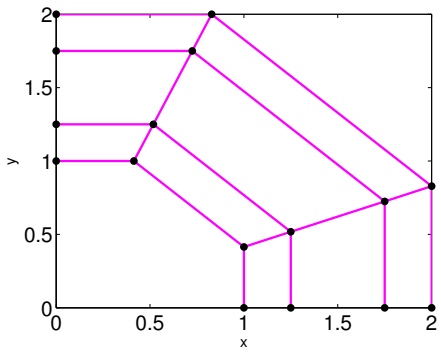
$$\begin{aligned} -\Delta u &= f, && \text{in } \Omega \\ u &= 0, && \text{on } \partial\Omega, \end{aligned} \tag{4.3.11}$$

with f given by

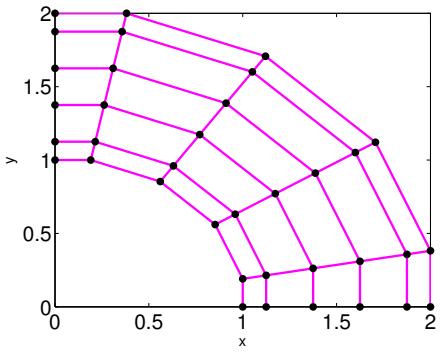
$$\begin{aligned} f(x, y) = 8xy\cos(xy)\left(2x^2 + 2y^2 - r_1^2 - r_2^2\right) \\ + \sin(xy)8\left(x^2 + y^2\right) + 4\sin(xy)\left(2x^2 + 2y^2 - r_1^2 - r_2^2\right) \\ - \sin(xy)\left(x^2 + y^2\right)\left(x^2 + y^2 - r_1^2\right)\left(x^2 + y^2 - r_2^2\right). \end{aligned}$$



(a) Initial control net. No internal knots.



(b) Refined mesh with one extra knot in both directions at $\xi = \eta = 0.5$.



(c) Refined mesh. Three new knots are inserted and linearly placed between 0 and 1.

Figure 4.8: Initial and refined control nets. By finding appropriate weights corresponding to each control net, conical domains can be represented exactly. Both control points and weights are updated during the knot refinement.

Different control meshes are shown in Figure 4.8. Figure 4.8a represents the initial control mesh. The corresponding knot vectors ξ and η have three knots placed in $\xi = \eta = 0$ and three knots in $\xi = \eta = 1$, with no internal knots. For the control mesh in Figure 4.8b, one knot is inserted at $\xi = \eta = 0.5$. Three linearly spaced knots between 0 and 1 are inserted in Figure 4.8c. We observe that the shape of the control mesh takes on the shape of the circular domain as we insert more knots.

For the mesh in Figure 4.8a, the control points on the diagonal are associated with the weights $w_2 = w_5 = w_8 = \frac{1}{\sqrt{2}}$. All other weights equal one. As we refine the mesh, both the control points and the weights are updated. The only fixed weights are the ones not lying on the circular arc. These equal one at all times. The initial weights are found by the same method as described in Section 2.1.5. The following figure illustrates the numbering of the different control points.

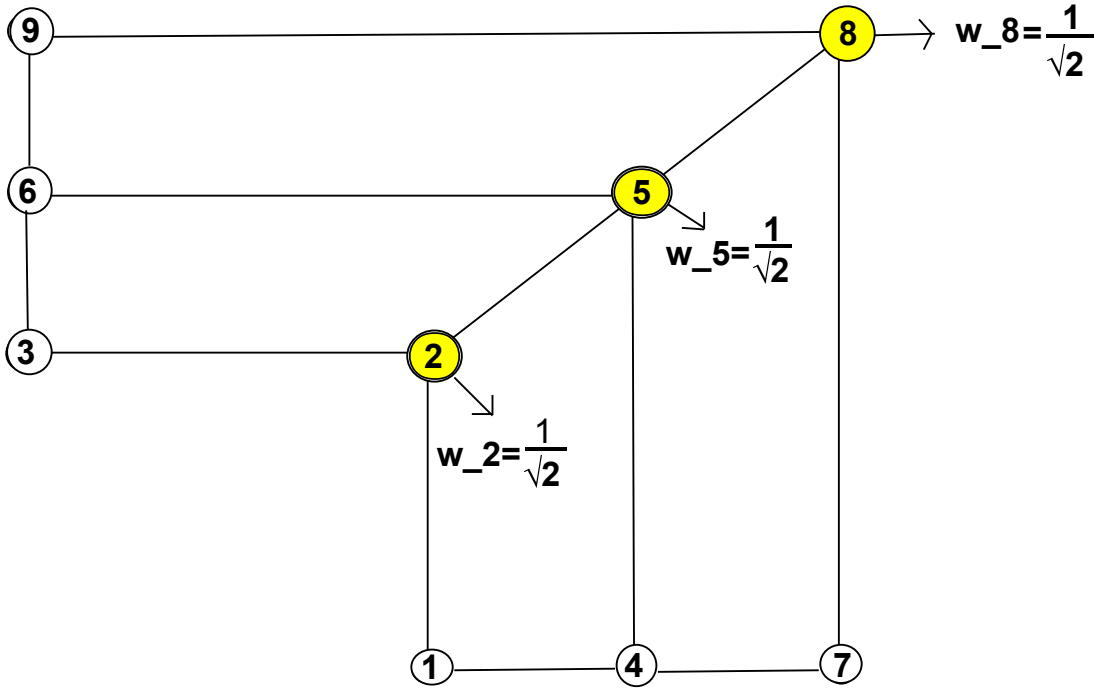
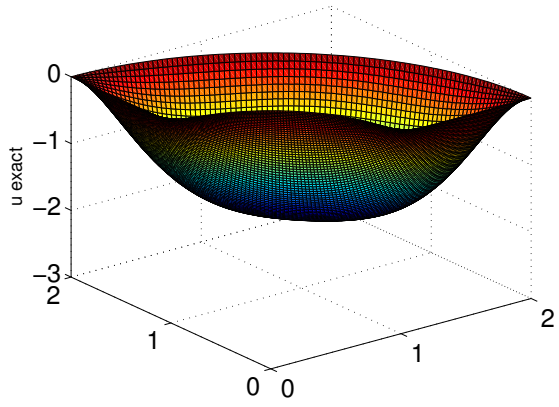
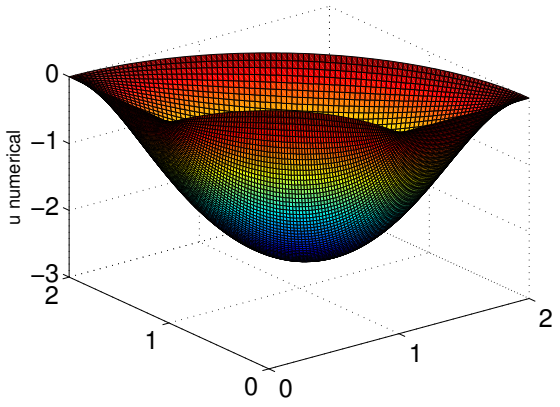


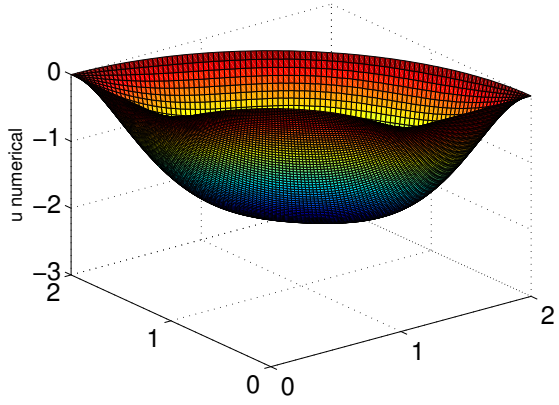
Figure 4.9: The numbering of the control points for the initial mesh. The points marked in yellow represent the points whose weights differ from 1.



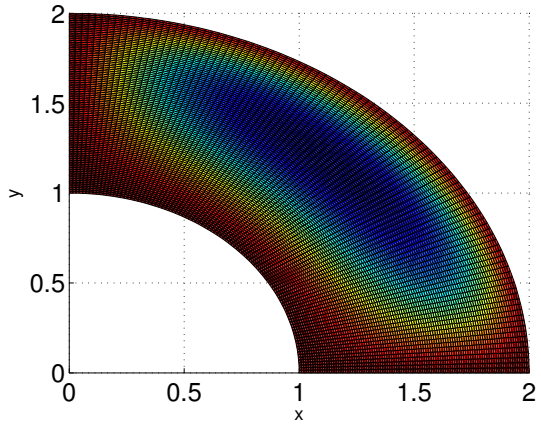
(a) Exact solution.



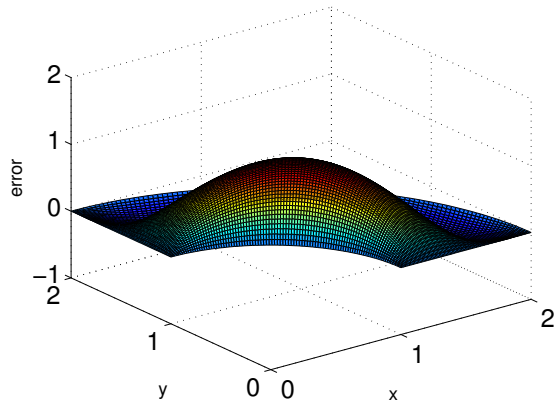
(b) Initial numerical solution.



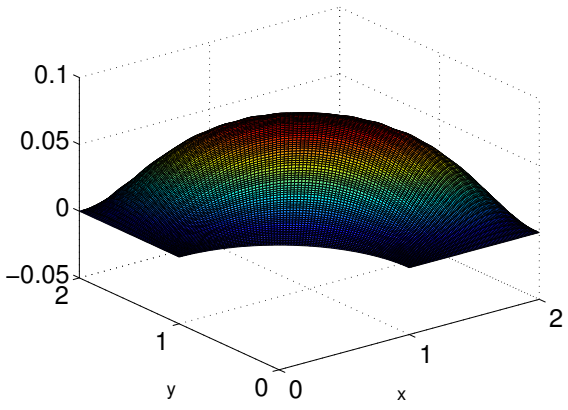
(c) Numerical solution after three rounds of knot refinement.



(d) Physical mesh.



(e) Initial error.



(f) Error after three rounds of knot refinement.

Figure 4.10: Solution of Poisson's equation on a quart disk with a circular hole. Both initial solution and solutions after knot refinement are shown. A plot of the physical mesh can be seen from above in Figure 4.10d in addition to two error plots.

All the numerical solutions considered here to this problem are constructed with use of quadratic NURBS. That being said, the solver can easily be extended to other polynomial degrees as well.

Figure 4.10 shows the numerical results for this problem. The domain for which the PDE is defined on is visualized in Figure 4.10d. The exact solution , Equation (4.3.10) on Ω is shown in Figure 4.10a. When comparing the initial numerical solution of Equation (4.3.11) with the exact solution, we observe that that the bottom in Figure 4.3.11 has a lower value than the bottom for the exact solution. However, Figure 4.10c shows that the numerical solution approaches the desired shape as we refine the knots. The knot intervals used for the solution corresponding to Figure 4.10c are of size $h = 0.125$ with a total of 64 degrees of freedom. The error of this solution is shown in Figure 4.10f. Note that the error after refining the knot intervals is significantly decreased compared to the initial error in Figure 4.10e. This is expected, as the initial numerical solution is computed with no internal knots and only one degree of freedom.

p	h	$\|u_{\text{num}} - u_{\text{exact}}\|_\infty$
2	1	66.642
2	1/2	7.680
2	1/4	0.963
2	1/8	0.110

Table 4.9: Infinity norm of $u_{\text{num}} - u_{\text{exact}}$.

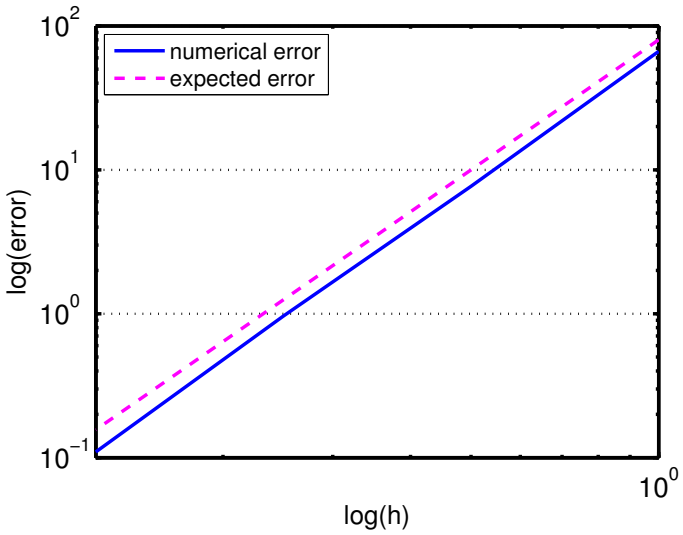


Figure 4.11: Logarithm of the size of the knot intervals h plotted against the logarithm of the error. The dashed line shows how the error is expected to decrease.

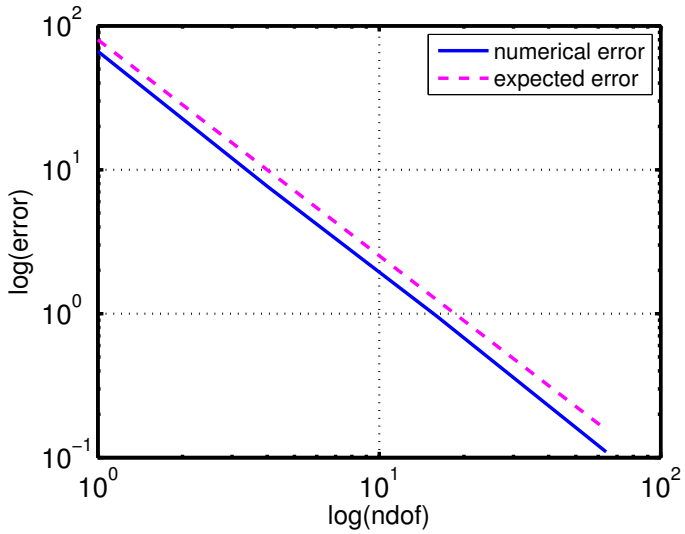


Figure 4.12: Logarithm of the number of degrees of freedom, ndof, plotted against the logarithm of the error. The dashed line shows how the error is expected to decrease.

Yet again, we want to check the convergence of the numerical solution. We follow the same strategy as we did for the problems involving B-splines and measure the distance from the numerical solution to the exact solution in the infinity norm. Assuming still, that the NURBS converge at the same rate as the FEM polynomials [8], we expect the relative error to be of order $e_{rel} \approx \mathcal{O}(2^{p+1}) = \mathcal{O}(2^3)$ for quadratic NURBS. The error for the different sizes of knot intervals can be read off of Table 4.9. The error results are visualized in Figure 4.11 and Figure 4.12, plotted against the logarithm of the size of the knot intervals and the logarithm of the degrees of freedom respectively. In both plots, the blue line shows the numerically achieved decrease of the error, whereas the dashed line shows how the error is expected to decrease in accordance with the estimate for the relative error (4.1.7). We observe that our numerical results follow the slope of the expected error, and conclude that the NURBS based solver is implemented correctly.

Chapter 5

Geometry Parameterization

The aim of this section is to create a geometry reparameterization for non-trivial geometries. We need an algorithm capable of updating the geometry parameterization automatically when the control points are perturbed. Several linear and non-linear parameterization methods can be found in [7], amongst them is *The Spring Model* that has been implemented in this thesis.

5.1. The Spring Model

The spring model is a linear parameterization method, generating the inner control points given only the boundary parameterization. Every internal control point is assigned the average value of its four neighboring control points. The situation is similar to a five point discretization of the Laplacian in two dimensions. Consider a grid with constant step size. That is, $x_i = x_0 + ih$ and $y_j = y_0 + jh$ for some constant h. The Laplacian $\Delta U(x_i, y_j)$ can then be discretized as follows:

$$\Delta U(i,j) = \frac{U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}}{h^2}, \tag{5.1.1}$$

see [27] for derivation. Instead of discretizing the numerical solution, the same procedure is used to update the internal control points.

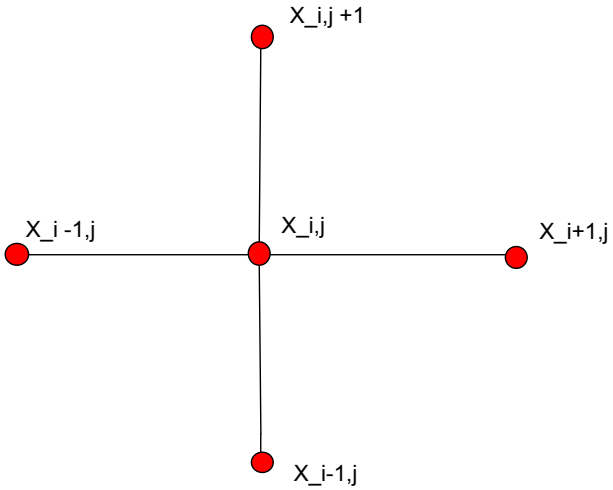


Figure 5.1: Five point stencil. Each point is assigned the average value of its four neighboring points.

$$x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} - 4x_{i,j} = 0. \tag{5.1.2}$$

Figure 5.1 and Equation (5.1.2) describe the situation. This leads to a linear system of equations,

$$\mathbf{Ax} = \mathbf{f}, \tag{5.1.3}$$

where $\mathbf{f}$ is the vector containing the contributions from the boundary and $\mathbf{A}$ is the two dimensional discrete Laplacian.

5.1.1. Applying The Spring Model on Poisson's Equation

In this section the two dimensional Poisson equation is solved by applying the Spring Model for the geometry parameterization. That is, the boundary control points are given and the internal control points are updated according to (5.1.2). The program evaluating the internal control points for the unit disk is listed in Appendix A.2.

Let Ω be the unit disk, i.e. $\Omega = \{(x, y) : x^2 + y^2 \leq 1\}$ and consider

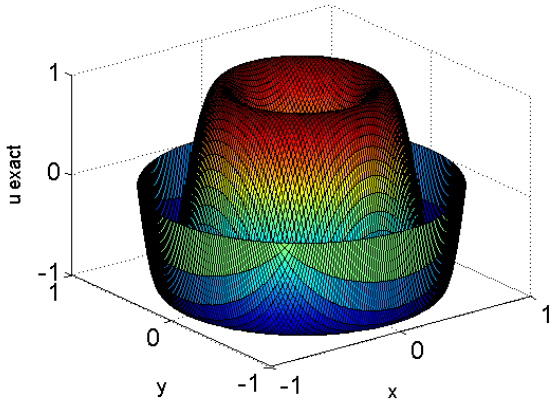
$$\begin{aligned} \Delta u &= -f, \quad \text{in } \Omega \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned} \tag{5.1.4}$$

By letting the exact solution be

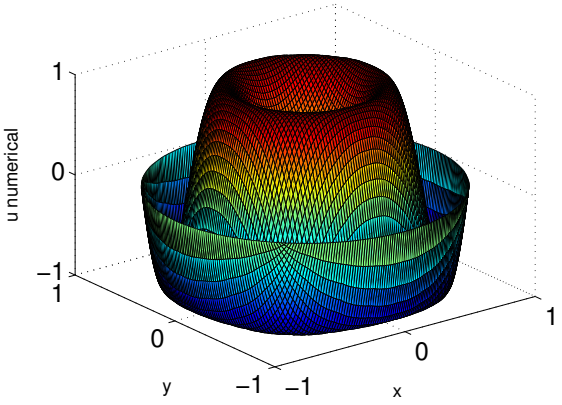
$$\hat{u}(x, y) = \sin\left(2\pi(x^2 + y^2)\right), \tag{5.1.5}$$

it is easily verified that the loading term f is given in polar coordinates by

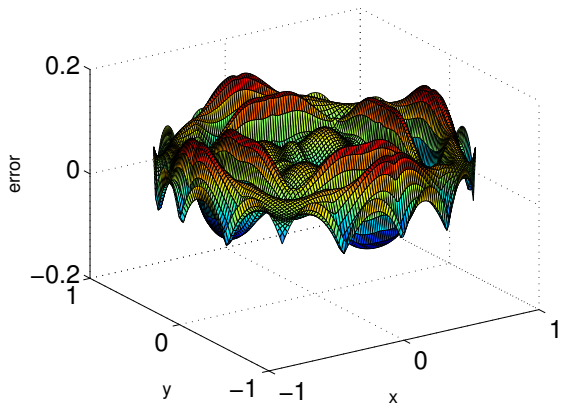
$$f(r,\theta) = -8\pi\cos\left(2\pi r^2\right) + 16\pi^2 r^2 \sin\left(2\pi r^2\right). \tag{5.1.6}$$



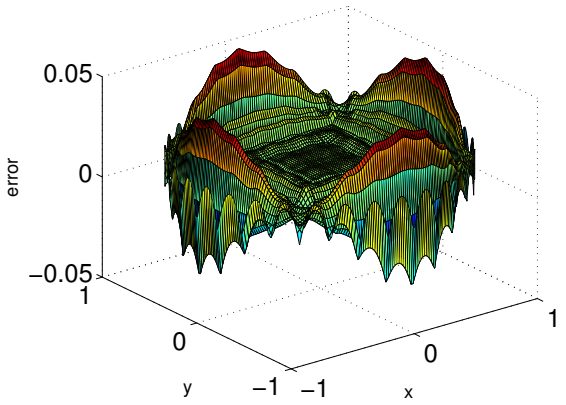
(a) Exact solution.



(b) Numerical solution with cubic B-splines. The knot intervals are of size $h = 1/8$.



(c) Difference between the exact and the numerical solution with cubic B-splines and knot intervals of size $h = 1/4$.



(d) Difference between the exact and the numerical solution in Figure 5.2b.

Figure 5.2: Exact and numerical solution for equation (5.1.4). Figure 5.2c and Figure 5.2d show the difference between the exact and the numerical solution for different sizes of the knot intervals.

Figure 5.2 shows the exact solution to Equation (5.1.4) together with the numerical solution. The knot vectors used in Figure 5.2b are both cubic and the knot intervals are of size $h = 1/4$. The differences of the exact and numerical solutions are shown in Figure 5.2c and Figure 5.2d with $h = 1/4$ and $1/8$, respectively. The error plots show that the difference between the exact and the numerical solution decreases significantly even after just one round of refinement. A more detailed analysis is done to check how the convergence rate behaves. As was done in Chapter 4, we measure the distance from the numerical to the exact solution in the infinity norm, and look at the relative error as we refine the knot intervals further. Recall that the value of the relative error is expected to be according to (4.1.7), by reducing the size of the knot intervals by a factor of 2.

p	h	$\|u_{\text{num}} - u_{\text{exact}}\|_\infty$
1	1/4	102.589
1	1/8	54.669
1	1/16	14.697
1	1/32	3.939
2	1/4	40.486
2	1/8	4.694
2	1/16	1.128
2	1/32	0.282
3	1/4	8.874
3	1/8	3.123
3	1/16	0.553
3	1/32	0.093

Table 5.1: Infinity norm of $u_{\text{num}} - u_{\text{exact}}$.

p	h	$\frac{\|u^h_{\text{num}} - u_{\text{exact}}\|_\infty}{\|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty}$	$C \cdot 2^{p+1}$
1	1/4	1.877	$0.469 \cdot 2^2$
1	1/8	3.720	$0.930 \cdot 2^2$
1	1/16	3.731	$0.933 \cdot 2^2$
2	1/4	8.625	$1.078 \cdot 2^3$
2	1/8	4.161	$0.520 \cdot 2^3$
2	1/16	4.000	$0.500 \cdot 2^3$
3	1/4	2.841	$0.1776 \cdot 2^4$
3	1/8	5.647	$0.353 \cdot 2^4$
3	1/16	5.946	$0.372 \cdot 2^4$

Table 5.2: Table of $\|u^h_{\text{num}} - u_{\text{exact}}\|_\infty / \|u^{h/2}_{\text{num}} - u_{\text{exact}}\|_\infty$.

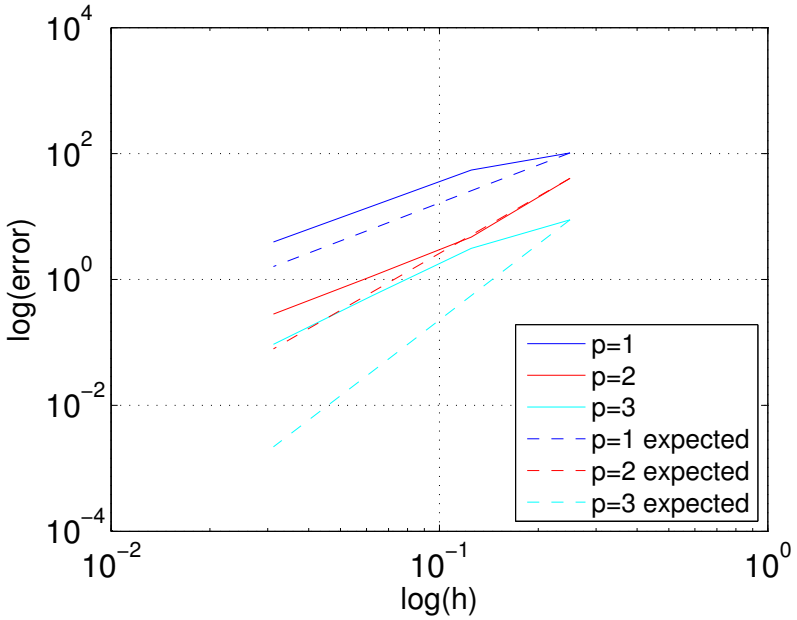


Figure 5.3: Visualization of the results from Table 5.1. The logarithm of the size of the knot intervals h, plotted against the logarithm of the error. The dashed lines show the expected decrease of error. Observe how the numerically achieved convergence rate for both $p = 1,\ 2$ and 3 essentially all follow the expected convergence rate for $p = 1$.

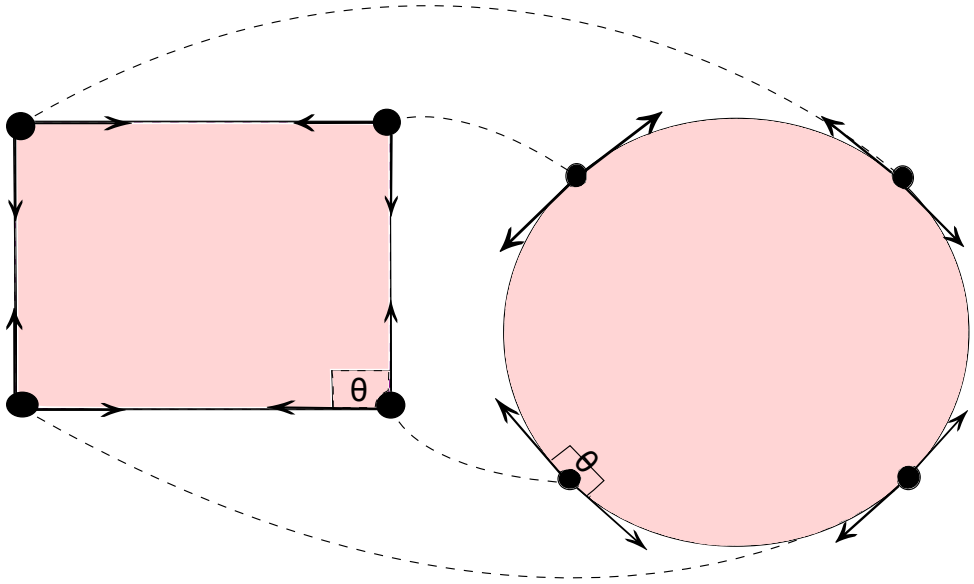
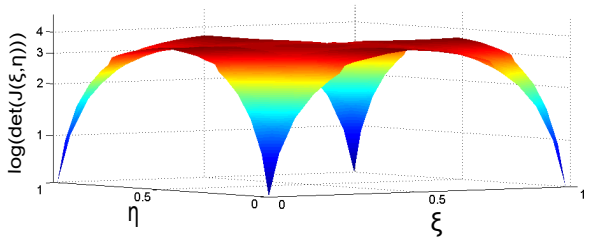
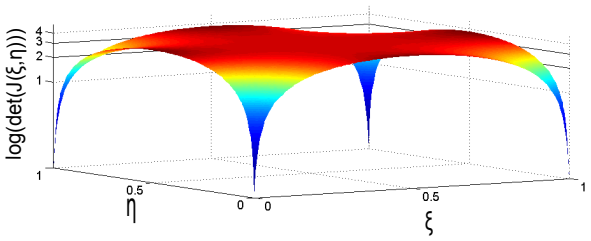


Figure 5.4: Illustration of what happens when the unit square is mapped to the unit disk when B-splines are used as basis functions instead of NURBS. There are singularities (Jacobian close to or equal to zero) that cause the behavior of the numerical method observed in Figure 5.3. Two linearly independent vectors forming the corners of the unit square maps to two linearly dependent vectors, forming the tangents of the unit disk.



(a) The logarithm of the Jacobian determinant plotted as a function of ξ and η. Knot intervals are of size $h = 0.25$.



(b) The logarithm of the Jacobian determinant plotted as a function of ξ and η. Knot intervals are of size $h = 0.03125$.

Figure 5.5: Plots of the Jacobian determinant as a function of ξ and η. By reducing the knot intervals, the value of $\det(J)$ in the four corners decreases towards zero. The z-axis is in log-scale, while the x- and y-axis are both linearly scaled. Both plots show the result when cubic B-splines are used.

p	h	$\det(J(\xi,\eta))$
1	1/4	1.023
1	1/8	0.553
1	1/16	0.293
1	1/32	0.154
2	1/4	0.655
2	1/8	0.360
2	1/16	0.194
2	1/32	0.104
3	1/4	0.539
3	1/8	3.299
3	1/16	0.164
3	1/32	0.089

Table 5.3: Value of $\det(J)$ in the four corners for $p = 1, 2$ and 3, and different knot interval spacings h.

Table 5.1 shows the distance measured in the infinity norm for linear, quadratic and cubic B-splines. We observe that there is a big difference from the exact to the numerical solutions when the knot intervals are large. When the knot intervals are refined, the error is reduced as well. These results are visualized in Figure 5.3, and it can be seen that the error does not decrease at an optimal rate. Recall from Section 4.1 that by reducing the knot intervals by a factor of 2, we would expect the relative error to go as $e_{rel} = \mathcal{O}(2^{p+1})$. The relative errors are shown in Table 5.2, and we observe that the expected reduction of the error is not achieved. The logical explanation for this phenomenon is that we have used B-Splines as basis functions. From previous sections we know that NURBS have the advantage of being able to represent conical geometries exactly, whereas B-Splines do not have this property. This is happening because B-splines are used for the parameterization of the unit disk, causing the Jacobian to be zero at the four corner boundary points. Figure 5.4 illustrates what happens when we map the corner points from the unit square to the unit disk. Two linearly independent vectors (corners) maps to two linearly dependent vectors (tangent to the unit disk). Figure 5.5 visualizes the logarithm of the Jacobian determinant as a function of ξ and η. For Figure 5.5a, the knot intervals are

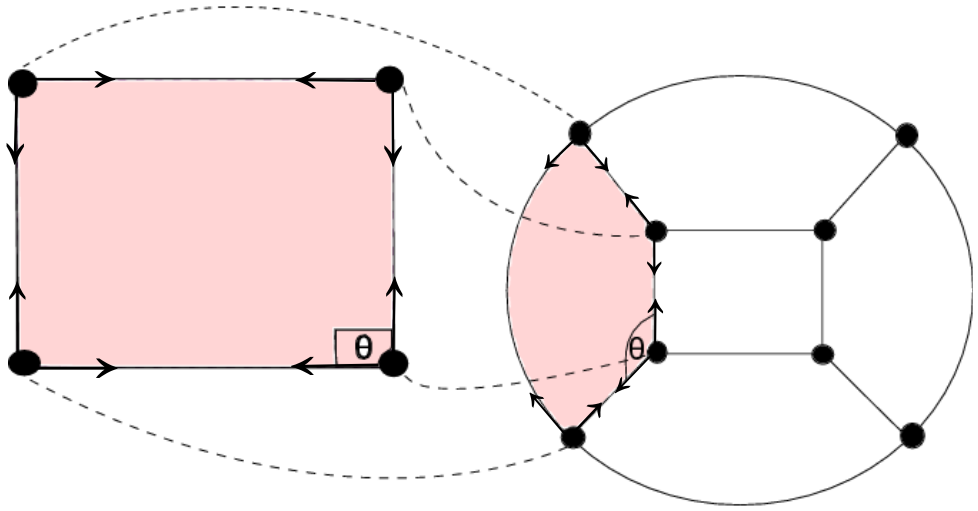


Figure 5.6: Unit disk described by five patches. Each patch has its own map to the unit square. By using multiple patches for the unit disk one avoids the problem of singularities in the mapping between the unit disk and the unit square.

of size $h = 0.25$. We include figures only for $p = 3$, but the situation is similar both for $p = 1$ and $p = 2$. In Figure 5.5b, knot refinement is performed. The knot intervals are now of size $h = 0.03125$, and we observe that the value of the Jacobian determinant decreases when the knot intervals are refined. The different values for the Jacobian determinant in the four corners are shown in Table 5.3 for $p = 1, 2$ and 3. Though we expect the Jacobian determinant to be zero (or very close to zero) in the corners, the fact that the value of the determinant in the corners tends more and more towards zero as the knot intervals are refined, supports the explanation of why we do not achieve the much wanted convergence rate when B-splines are used as basis functions for this problem.

We note that as we refine the knot intervals, the relative errors are asymptotically the same independent of whether we use linear, quadratic or cubic B-splines. See how all three slopes in Figure 5.3 essentially follow the slope for $p = 1$. This illustrates that it is more efficient to use knot refinement than order elevation when we have a mapping with singularities. In other words: for a parameterization that is not sufficiently smooth, it makes little sense to elevate the polynomial order of the B-splines. See [8] or [23] and the error estimates therein. Additionally, the greater the polynomial degree p is, the less sparse the system of equations to be solved becomes. Thus, the computational cost of solving the equations can be reduced by choosing a low polynomial degree of the B-splines.

5.1.2. An Alternative Approach

We have now seen that describing geometries such as the unit disk by a single patch leads to singularities in the mapping between the unit square and the unit disk. An alternative approach is to instead describe the geometry by multiple patches. Each patch will then have its own mapping back and forth to the unit square. Using this approach, the problem with singular mappings is avoided. We know that the angle θ between the vectors forming

the corners in the unit square is 90 degrees. As long as these vectors are mapped to a patch of the unit disk in such a way that the span between them is strictly less than 180 degrees, they preserve their linear independence. Figure 5.6 shows a division of the unit disk, which avoids the problem of singular mappings while using B-splines as basis functions. The disk is divided into five patches, where all the angles involved in each patch are strictly less than 180 degrees.

Chapter 6

Shape Optimization

In shape optimization, one typically seeks to find the optimal shape of a domain $\Omega(p)$, in order to minimize or maximize an objective functional, satisfying a number of given constraints. The objective functional is dependent on the solution of a PDE, which itself is dependent on the domain Ω. An example of such an objective functional, J, can for instance be

$$J(\mathbf{p}) = \int_{\Omega(\mathbf{p})} U\left(\mathbf{p}\right)^2 \, \mathrm{d}\mathbf{x}, \tag{6.0.1}$$

where U is the IGA based numerical solution to the PDE with control points $\mathbf{p}$. We will use `MATLAB`'s optimization toolbox `fmincon` to study the shape optimization problems. To begin with, we optimize without providing gradients to `fmincon`. We will hereafter refer to this as "gradient-free optimization". This terminology is introduced here in order to distinguish between the method we use in Section 6.1 and the section on automatic differentiation (AD) 6.2. The gradients are approximated by finite differences when they are not provided by the user, hence this method should not be mistaken to be what is normally referred to as a gradient-free method. Examples of such optimization algorithms can be found e.g. in [22], but are not considered in this thesis.

6.1. Gradient-free Optimization (Finite Differences)

We use `MATLAB`s optimization toolbox for constrained optimization problems, `fmincon`. This toolbox can be used without further computation of gradients. Say we want to find the control points for the geometry such that expression (6.0.1) is minimized. The constraints might for instance be that the control points are allowed to move no more than 10 % from their original positions, i.e.

$$\begin{aligned} &\min_{p} \; J\left(\mathbf{x}(\xi(\mathbf{p}))\right) \\ &\text{subject to} \\ &p(i)_{\text{new}} \geq p(i)_{\text{old}} - p(i)_{\text{old}} \cdot 0.1, \\ &p(i)_{\text{new}} \leq p(i)_{\text{old}} + p(i)_{\text{old}} \cdot 0.1, \end{aligned} \tag{6.1.1}$$

where the subscript i indicates the control point being perturbed. So we may try to change the control points one by one, but it is also possible to say that all the control points can be perturbed at the same time. If we are in $\mathbb{R}^2$ we may choose to let both the x and y components change, or choose one of them. The program evaluating the objective function J, `Objective.m`, is attached in Appendix A.3. Modifications of this program must be done depending on how the objective function is defined, but this is typically the program that we pass to `fmincon` as a function handle input. The first thing to do is to specify the lower and upper bounds, and to choose which control points we wish to optimize with respect to. The lower and upper bounds, along with initial control points and the program evaluating the objective function must also be given as input. The output will be the new control point(s), X, that minimizes/maximizes J.

$$X \;=\; \text{fmincon}\left(@(X_0)\ \texttt{Objective}(), X_0, [\,], [\,], [\,], [\,], \text{LB}, \text{UB}\right) \tag{6.1.2}$$

The line included in (6.1.2) shows a typical call to `fmincon`. The brackets that are empty are so due to the only constraints for our examples are the lower and upper bounds.

We consider the PDE we solved in Chapter 5. We know that th the exact solution is given by

$$u(x, y) = \sin\left(2\pi(x^2 + y^2)\right), \tag{6.1.3}$$

but instead of using the exact solution, we define J to be the difference between the numerical solution of Equation (6.1.3) solved on the parameterized unit disk and the approximated solution with perturbed boundary control points. That is,

$$J = \int_{\Omega} |u - U^*|^2 \, \mathrm{d}\mathbf{x}, \tag{6.1.4}$$

where u is the numerical solution to Equation (6.1.3) and U^* is the numerical solution with perturbed boundary. Needless to say, we expect that this objective function will reach its minimum when the domain Ω approaches the unit disk. To start, we let one control point vary over a prescribed interval. In this test the bounds correspond to the positions of control points for the unit disk. By running `fmincon` with one arbitrary initial control point and the rest of them fixed, we expect the program to iterate until it has reached a minimum, and that the control point giving the optimal value is indeed positioned on the unit disk.

6.1. Gradient-free Optimization (Finite Differences)

6.1.1. Test 1: One perturbed Boundary Control Point

We use quadratic B-splines and let the knot vectors ξ and η be defined to have length $l(\xi) = l(\eta) = n + p + 1 = 14$. In the original geometry the point $p_1 = (x_1, y_1) = (-0.7071, -0.7071)$. As any point within the bounds will do as initial point, we try with $p_1 = (x_1, y_1) = (-0.75, -0.75)$.

iteration	function value
0	0.767
1	0.022
2	0.008
3	0.0002
4	$2.7{\cdot}10^{-6}$

Table 6.1: Result of `fmincon` with one perturbed control point.

The results of the algorithm are shown in Table 6.1, and the produced output for which values of p_1 giving the optimal function value, $fval = 2.7 \cdot 10^{-6}$, in this test was $x_1 = -0.7071$ and $y_1 = -0.7071$. This is the point from the original parameterization of the unit disk.

6.1.2. Test 2: Two Perturbed Boundary Control Points

After having tested the code with variation in only one boundary control point it is now time to see what happens when several boundary control points differ from the boundary of the unit disk. Knot vectors and basis functions are the same as in Test 1.

At first we increase the number of varying control points to two. We know that the first and last boundary control points in the parameterization of the unit disk are

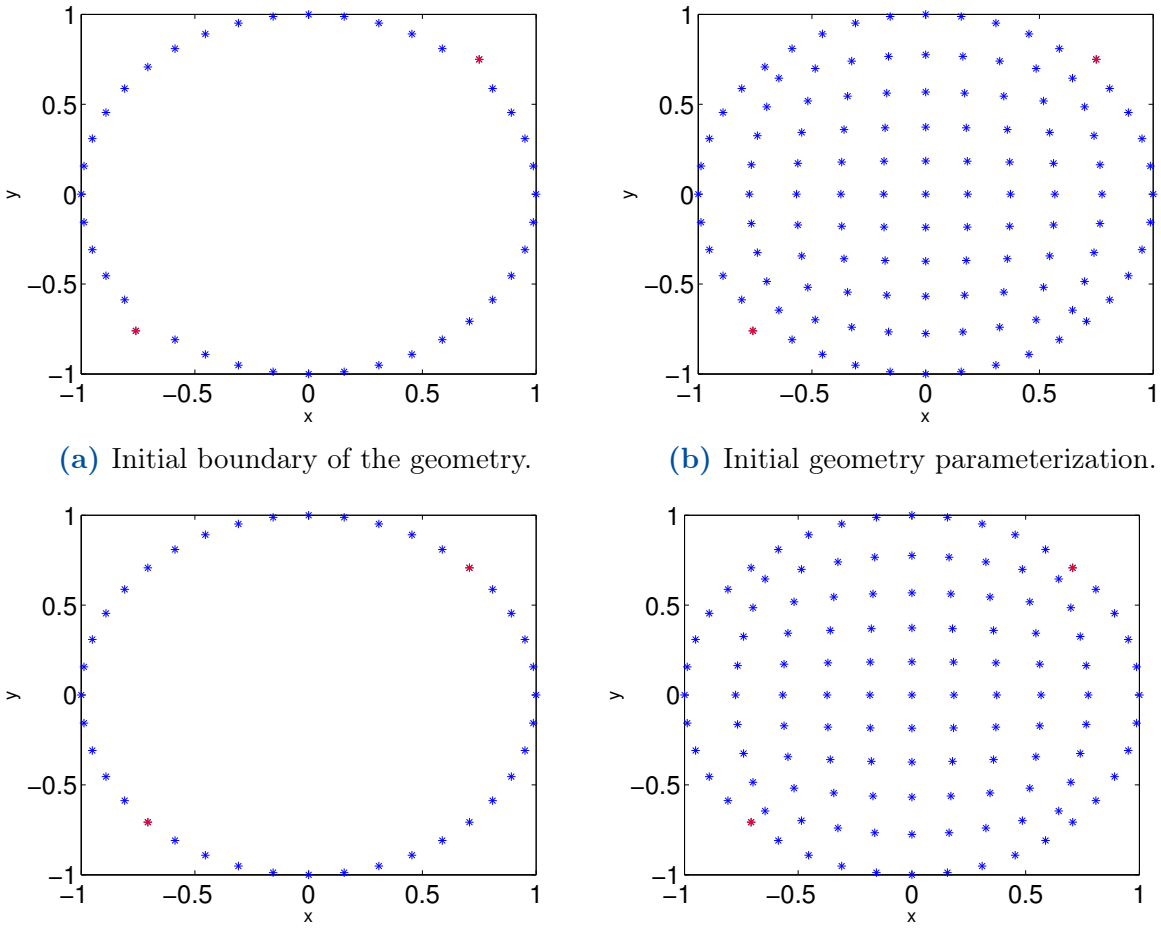
$$p_1 = (x_1, y_1) = (-0.7071, -0.7071),$$
$$p_{\text{end}} = (x_{\text{end}}, y_{\text{end}}) = (0.7071, 0.7071).$$

By letting these two points initially deviate from the unit disk and running `fmincon`, we expect them moving towards the desired boundary as the minimum of the objective function is approached.
Table 6.2 shows the result of the function value after each iteration- The new boundary control points for p_1 and p_{end} are denoted by $\mathbf{p}_{\text{opt}}$.

$$\mathbf{p}_{\text{init}} = \begin{pmatrix} -0.76 & -0.76 \\ 0.75 & 0.75 \end{pmatrix} \xrightarrow{\texttt{fmincon}} \begin{pmatrix} -0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{pmatrix} = \mathbf{p}_{\text{opt}}.$$

iteration	function value
0	1.850
1	0.384
2	0.307
3	0.060
4	0.017
5	0.001
6	0.0003
7	$8.5{\cdot}10^{-5}$
8	$3.5{\cdot}10^{-9}$

Table 6.2: Result of `fmincon` with two perturbed control points.



(a) Initial boundary of the geometry.

(b) Initial geometry parameterization.

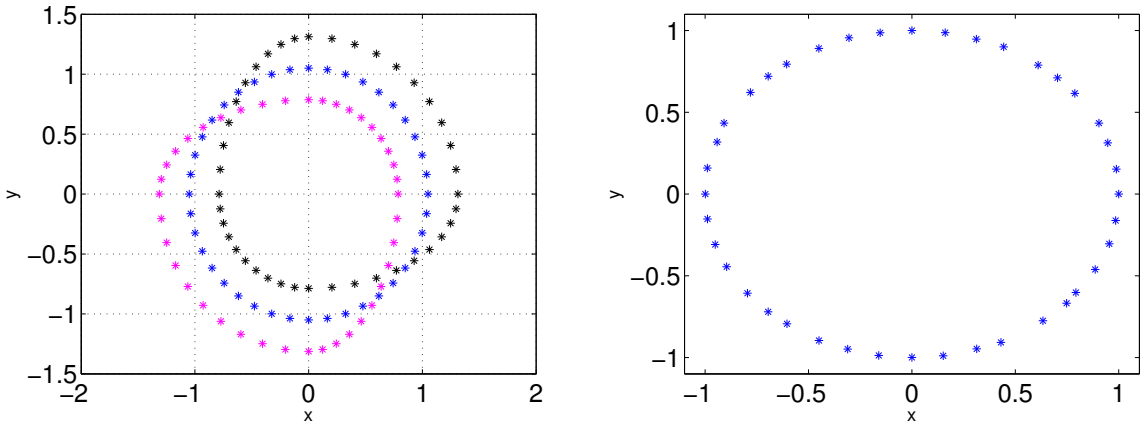
(c) The final boundary.

(d) Optimal geometry parameterization.

Figure 6.1: Different parameterizations of the unit disk. The red control points are the ones we perturb. Figure 6.1a shows the initial parameterization of the boundary. These figures show the results after running `fmincon` on the problem. The output result of the best found parameterization is shown in Figure 6.1d.

6.1.3. Test 3: Several Perturbed Boundary Points

In this test, `fmincon` seeks to find the minimized value of the objective function J, by optimizing with regards to all the boundary control points. The initial boundary control points for the test case are set to be the original ones from the parameterization found from `fivePointPoisson`, plus a small variation $\epsilon = 0.05$ of these. For lower and upper bounds we tolerate that the edge points move with -25% and $+25\%$ respectively.



(a) Lower and upper bounds together with the initial edge.

(b) Plot of the final output boundary control points for $n = m = 11$.

Figure 6.2: Figure 6.2a shows the initial bondary together with the lower and upper bounds. The blue points correspond to the initial control points. The magenta- and black colored boundaries represent the lower bounds and the upper bounds respectively. Output result of the boundary control points leading to minimized objective function. This time, all the boundary points were allowed to move within the given bounds. The initial value of the objective function was $fval = 163.14$ which decreased to $fval = 0.0353$ during simulation.

In Figure 6.2a the lower and upper bounds are shown in magenta and black and the initial edge is shown in blue for $n = m = 11$. The initial value of the objective function is $fval = 163.14$. The final functional value decreases to $fval = 0.0353$ after 23 iterations.

6.1.4. Discussion

We observe that we in all cases manage to find the desired shapes we are looking for, and that the final functional value in each case decreases towards a factor 0. The final solution when optimizing with respect to only one control point in Section 6.1.1 is found after only 4 iterations. In contrast, when we perturb all the boundary control points, it takes `fmincon` 23 iterations and $2792.66s$ to find the optimal solution. In practice however, we are more interested in finding the optimal shape of a domain, for which we may not have enough information to let just one or two boundary control points be the optimization points. That means, if by starting with a more or less random shape, the algorithm itself should be able to find the optimal shape for the problem at hand. Considering the computational time spent on the test in Section 6.1.3, we now wish to see if it can help

to provide `fmincon` with more accurately evaluated gradients, to make the process more efficient.

6.2. Gradient-based Optimization - Automatic Differentiation

In order to apply a gradient based optimization algorithm we need to be able to calculate the gradients of the objective function when the control points change. That is, we optimize with regards to the boundary control points while the inner control points are still found according to The Spring Model. With an objective function like (6.0.1) for $\mathbf{p} \in \mathbb{R}^n$ we calculate

$$\nabla_{\mathbf{p}} J = \left(\frac{\partial J}{\partial \mathbf{p}_1}, \ldots, \frac{\partial J}{\partial \mathbf{p}_n} \right). \tag{6.2.1}$$

These gradients can be found in several ways. One way is to use finite differences, which is the method used by `fmincon` when no gradients are provided by the user. Of course the gradient can also be calculated symbolically, or by a third alternative, Automatic Differentiation (AD). In this thesis the latter alternative is implemented. AD is a handy tool that comes with several advantages. Derivatives are calculated accurately, and one avoids round off errors as opposed to numerical differentiation. For simple expressions symbolic differentiation may be the most efficient choice. However, symbolic differentiation can in many cases be cumbersome, thus leading to AD as the most practical method of differentiation. For example in the process of computing sensitivities in shape optimization.

The idea of sticking to `MATLAB` as a scripting language is taken from R.D. Neidingers article on the subject [20]. Some of the necessary software is found from [17], leaving mainly the adjustments of the self written programs to the author. Rewriting the computer programs to object oriented codes is considered to be worth the extra effort as it comes along with gained knowledge on a new discipline, and additionally refreshes the author's object oriented programming skills.

6.2. Gradient-based Optimization - Automatic Differentiation

AD with forward accumulation is done by traversing the chain rule for differentiation from the inside and out. For a function $F(x,y) = f(g(x,y))$, the chain rule gives

$$\begin{aligned}\frac{\partial F}{\partial x} &= \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}\\ \frac{\partial F}{\partial y} &= \frac{\partial f}{\partial g}\frac{\partial g}{\partial y}.\end{aligned}$$

A forward accumulating AD tool would compute the terms $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y}$ first and then proceed by computing $\frac{\partial f}{\partial g}$. Another alternative is reverse accumulation, that is the method that traverses the chain rule for differentiation the other way around.

AD is implemented using object oriented programming in `MATLAB`. We rewrite the entire framework of the IGA solver, such that the boundary points coming from the parameterization are now objects holding both a value for themselves and their derivatives. When using AD for the problems we wish to solve in this thesis, it is the parameterization algorithm that needs to be differentiated (The Spring Model). We pick out the points that correspond to the boundary and differentiate them, before we pass the values of the parameterization points and their corresponding derivatives to `fmincon`.

6.2.1. An illustrative example

We illustrate the idea of AD through a simple example. Consider the function

$$f(x,y) = x^2y + ye^x. \tag{6.2.2}$$

We want to find the gradient of f, and do so by forward accumulation.

$$\begin{aligned} w_1 &= x & w_1' &= 1\\ w_2 &= y & w_2' &= 0\\ w_3 &= w_1^2 w_2 & w_3' &= 2w_1w_1'w_2 + w_1^2w_2' = 2xy\\ w_4 &= w_2e^{w_1} & w_4' &= w_2'e^{w_1} + w_2(e^{w_1})'w_1' = ye^x\\ w_5 &= w_3 + w_4 & w_5' &= w_3' + w_4' = 2xy + ye^x. \end{aligned} \tag{6.2.3}$$

Thus we have found the partial derivative $\frac{\partial f}{\partial x} = 2xy + ye^x$. The procedure for finding $\frac{\partial f}{\partial y}$ is similar, and is found by changing the seeding point, that is set $w_1' = 0$ and $w_2' = 1$.

The results after applying automatic differentiation in `MATLAB` for equation (6.2.2) are,

$$\begin{aligned} f(x,y) &= x^2y + ye^x\\ \frac{\partial f}{\partial x} &= \{0,\ 0.446,\ 1.3244,\ 2.7128,\ 4.7183\}\\ \frac{\partial f}{\partial y} &= \{1,\ 1.3465,\ 1.8987,\ 2.6795,\ 3.7183\}, \end{aligned} \tag{6.2.4}$$

for $\mathbf{x}$ and $\mathbf{y}$ defined by,

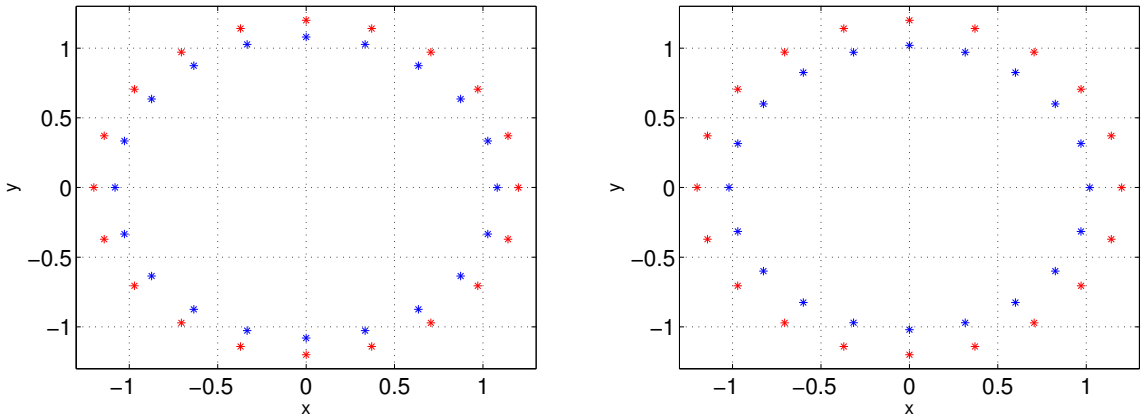
$$\mathbf{x} = \{0,\ 0.25,\ 0.5,\ 0.75,\ 1\},$$
$$\mathbf{y} = \{0,\ 0.25,\ 0.5,\ 0.75,\ 1\}.$$

It is easy to check that these are the correct values for the partial derivatives for this example.

6.2.2. Numerical Results

It is now time to provide gradients to `fmincon` by applying our new differentiation tool. The optimization algorithm used is Sequential Quadratic Programming (SQP). See [22] for further details of this algorithm. The problem is essentially the same as in the section on gradient free optimization. Equation (5.1.4) is solved on the unit disk using IGA. Thus the unit disk is the geometry we wish to obtain from the optimization algorithm. The objective function J to be minimized is again the expression in (6.1.4). As we saw from the results from the gradient-free based optimization technique, `fmincon` was able to find minimas that satisfied the bounds, and the optimal results for the boundaries were only a factor of $\epsilon \approx 0$ away from the original parameterized unit disk.

So what is gained from using a gradient-based optimization algorithm? First of all, AD provides us with exact derivatives, whereas numerical differentiation such as finite differences, only provides approximate derivatives. Other aspects to be considered are the running time of the program and the number of iterations it takes to reach a solution.



(a) Initial- and final boundary points plotted in red and blue respectively.

(b) Initial- and final boundary points plotted in red and blue respectively.

Figure 6.3: Output results of the geometry leading to local minimas of the objective function. The only difference of the two examples are the lower and upper bounds. Lower and upper bounds in Figure 6.3a are set to the initial boundary ±10%. Lower and upper bounds for Figure 6.3b are set to the initial boundary ±15%. In both cases quadratic B-splines are used, and a total number of degrees of freedom $ndof = n \times m = 6 \times 6 = 36$. The choice of optimization algorithm is SQP. In both figures we see that the final output boundary is closer to the desired unit disk.

Some results can be seen from Figure 6.3. In both Figure 6.3a and Figure 6.3b the total number of degrees of freedom is $ndof = n \times m = 6 \times 6 = 36$. The initial boundary is the same in both cases, a disk centred at the origin, with radius $r = 1.2$. These boundary points are shown in red. The only input difference is in the definitions of the lower and upper bounds. In both cases the final boundary is closer to the parameterized unit disk than the initial boundary. For the results corresponding to Figure 6.3a the boundary control point x_i satisfies

$$x_{init_i} \pm 0.1 \cdot x_{init_i} \leq x_i \leq x_{init_i} \mp 0.1 \cdot x_{init_i}, \tag{6.2.5}$$

where $\pm$ and $\mp$ are used in order to get the bounds correct according to whether the x_{init_i}'s are negative or positive. The objective function value decreases from $fval = 417.85$ to $fval = 150.34$ after two iterations. The boundary points corresponding to this solution are shown in blue in Figure 6.3a.

For the results corresponding to Figure 6.3b the boundary control point x_i satisfies

$$x_{init_i} \pm 0.15 \cdot x_{init_i} \leq x_i \leq x_{init_i} \mp 0.15 \cdot x_{init_i}. \tag{6.2.6}$$

In this case the final solution is again found after two iterations. The value of the objective function $fval$ decreases to $fval = 9.48$, a noticeable better result. The boundary points corresponding to this solution are shown in blue in Figure 6.3b.

	final value	nr. of iterations	runtime
gradient based - bounds in (6.2.5)	150.34	2	63.03 s
gradient free - bounds in (6.2.5)	167.68	8	88.07 s
gradient based - bounds in (6.2.6)	9.48	2	63.06 s
gradient free - bounds in (6.2.6)	26.30	8	120.173 s

Table 6.3: A comparison of gradient based- and gradient free shape optimization for the same problems. We compare final function value, number of iterations and the runtime of the two methods.

To compare the gradient based shape optimization with the gradient free method, the exact same examples are redone without the use of AD. Table 6.3 shows difference in final function value, number of iterations and runtime between the gradient based and gradient free shape optimization. We see that the gradient based shape optimization finds better solutions using less iterations in shorter time in both cases. The runtime difference is typically caused by the need for two function evaluations in the finite difference approximation, whereas this is not required for AD. A logical explanation for why `fmincon` uses a higher number of iterations with finite differences than with AD can be that the gradients computed with finite differences are inaccurate, perhaps leading to slower convergence. Another reason for slow convergence can be that the ideal size of the finite difference step may depend on the application [4].

With the differentiation in place, we now have all the tools we need to proceed with an application that will utilize the IGA solver, the reparameterization algorithm and AD for gradient based shape optimization.

Chapter 7

A Chemical Reaction System

In this chapter, different topics discussed so far are assembled to solve a composed problem. A chemical reaction system is considered, as a system of PDEs, defined on some domain Ω and coupled through non-linear boundary conditions. During the reaction, a substance is produced and a measure of the produced substance is maximized by performing shape optimization on Ω. To solve the chemical system, all the IGA ingredients are needed. That is, a solver that can solve PDEs for different kinds of boundary conditions and geometries. A Gauss Seidel type program is used as an iterative solver, as we are dealing with non-linear equations. For the shape optimization, `fmincon` is used, both as a gradient-free and gradient-based optimization tool. Recall from Section 5.1 that The Spring Model can be used to find the inner control points when the boundary points are given. Thus, we follow this procedure to update the inner control points when the boundary is perturbed. This chapter is split into three sections. To begin with, the mathematical model of the system is set up, followed by a section on the iterative solver. The last section is on shape optimization on Ω, and the numerical results are found there.

7.1. Mathematical Model

In this section the mathematical model for the chemical system is set up. We consider two reacting substances, R_1 and R_2, creating a new produced substance P. Denote the concentration of substance i by C_i and consider the chemical reaction

$$R_1 + R_2 \xrightarrow{\mathrm{r}} P, \tag{7.1.1}$$

where r is the reaction rate defined by

$$r = r(R_1, R_2) = \alpha C_{R_1} C_{R_2}, \qquad \alpha \in \mathbb{R}. \tag{7.1.2}$$

The chemical reaction rate (7.1.2) follows by the law of mass action [3], and α is the reaction rate constant.

To derive the differential equation for the concentration of substance i, we consider a control volume Ω_{contr}. In the derivation we follow the law of conservation of mass. For a system with no sink or source, the rate of change for C_i depends only on the flux $J(x,t)$ going through the boundary of the control volume $\partial\Omega_{\text{contr}}$. That is

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega_{\text{contr}}} C_i(\mathbf{x},t)\ \mathrm{d}V \ = \ -\int_{\partial\Omega_{\text{contr}}} J(\mathbf{x},t)\ \mathrm{d}S, \tag{7.1.3}$$

where $\mathrm{d}S$ is a surface integral over the boundary of Ω. By Gauss' divergence theorem,

$$\int_{\partial\Omega_{\text{contr}}} J(\mathbf{x},t)\ \mathrm{d}S \ = \ \int_{\Omega_{\text{contr}}} \mathrm{div} J(\mathbf{x},t)\ \mathrm{d}V.$$

By assuming that C_i belongs to the space of continuous differentiable functions C^1, it follows that the differentiation $\frac{\mathrm{d}}{\mathrm{d}t}$ can be moved under the integral sign, see [16]. Hence, the equation for conservation becomes

$$\int_{\Omega_{\text{contr}}} \left(\frac{\partial}{\partial t}C_i \ + \ \mathrm{div} J\right)\ \mathrm{d}V \ = \ 0. \tag{7.1.4}$$

In addition the integral signs can be dropped, since (7.1.4) holds for all control volumes and C_i is assumed to be continuous. This leads us to

$$\frac{\partial}{\partial t}C_i \ + \ \mathrm{div} J \ = \ 0. \tag{7.1.5}$$

By Fick's law of diffusion [2], it is known that

$$J = -D_i \nabla C_i, \tag{7.1.6}$$

where D_i is the diffusion coefficient corresponding to C_i. Hence

$$\frac{\partial}{\partial t}C_i - D_i \Delta C_i = 0. \tag{7.1.7}$$

The chemical system we consider is stationary, i.e. it is independent of time. Hence $\frac{\partial}{\partial t}C_i = 0$, leading to the differential equation for the concentration, C_i, given by:

$$-D_i \Delta C_i = 0, \quad \text{in } \Omega. \tag{7.1.8}$$

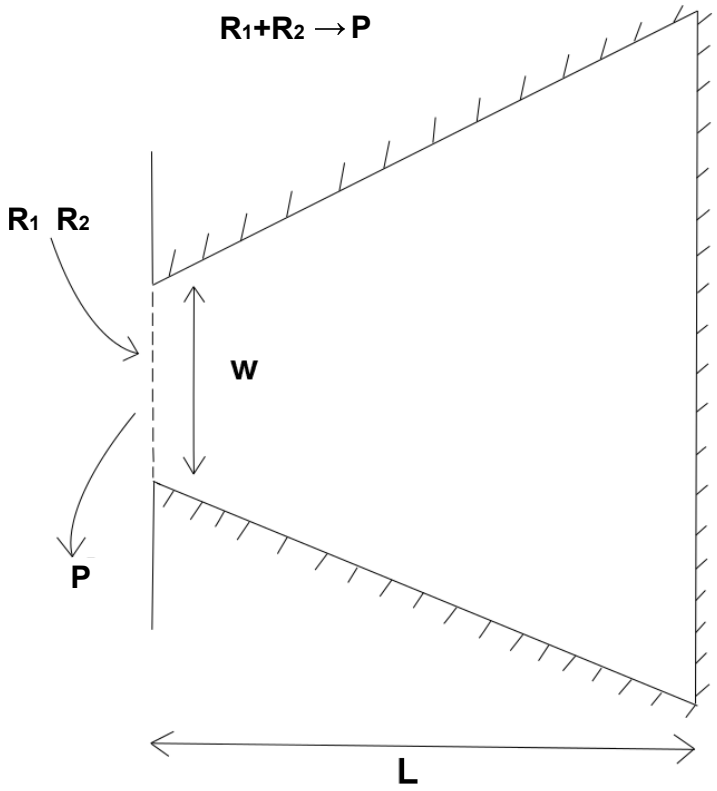


Figure 7.1: Mathematical model of the chemical system. Substance R_1 and R_2 reacts, producing substance P. The equations for the system are coupled through non-linear Robin boundary conditions on three of the edges, whereas linear Robin boundary conditions are used on the dashed line, where no chemical reaction happens.

The situation considered is described in Figure 7.1. There is a chemical reaction between substance R_1 and R_2 on all edges except for the edge with the dashed line. The system of PDEs can thus be described as follows:

$$\begin{aligned} -D_{R_1}\Delta C_{R_1} &= 0, \quad \text{in } \Omega, \\ -D_{R_2}\Delta C_{R_2} &= 0, \quad \text{in } \Omega, \\ -D_P\Delta C_P &= 0, \quad \text{in } \Omega, \end{aligned} \tag{7.1.9}$$

$$\begin{aligned} \mathbf{n}\cdot(-\nabla D_i C_i) &= \pm\alpha C_{R_1}C_{R_2}, \quad \text{on } \partial\Omega_r, \\ \mathbf{n}\cdot(-\nabla D_i C_i) &= \beta\left(C_i - \hat{C}_i\right), \quad \text{on } \partial\Omega \setminus \partial\Omega_r, \end{aligned} \tag{7.1.10}$$

where $\partial\Omega_r$ denotes the part of the boundary where the chemical reaction takes place and $\hat{C}_i$ is the given concentration of substance i exterior to the domain Ω. The equations (7.1.9) are coupled through non-linear boundary conditions. Along the edge of the domain described by a dashed line in Figure 7.1, all equations have linear Robin boundary conditions. Non-linear Robin boundary conditions are used for substance R_1 and R_2 on the reaction boundary, whereas non-linear Neumann boundary conditions are used for substance P on the reaction boundary. This is described in Equation (7.1.10).

The imposition of the specific boundary conditions can be explained as follows: The flux of the concentration C_i, through the reaction boundary depends on the reaction rate, which again depends on the concentration of the substances R_1 and R_2. When the reaction takes place, the new substance P is created, leaving less concentration of R_1 and R_2, which again also leads to a decrease in the reaction rate. A situation where the flux of a concentration depends on the same concentration is described by Robin boundary conditions. For substance P, the flux of concentration depends only on the reaction rate of the two other substances, and we are in the situation of Neumann boundary conditions. On the edge with no reaction, the flux of concentration for each of the substances, is defined as a linear relationship between the concentration of substance i on this edge and the given concentration of substance i on the outside of Ω. Hence, also substance P has Robin boundary conditions along this edge. This suits us well, due to the non-uniqueness of the Laplace equation with pure Neumann conditions that was proved in Section 4.1.

Note that the sign of α on the right hand side in Equation (7.1.10) varies. We let it be negative for the consumed substances R_1 and R_2 and positive for the produced substance P.

7.2. Solution by an Iterative Method

The coupled system is solved by an iterative method, more spesifically a block Gauss Seidel scheme. For the non-linear Robin boundary conditions, a splitting is used. That is, for iteration $k+1$, the boundary conditions are prescribed by

$$\alpha C_{R_1}^{k+1} + \mathbf{n} \cdot \nabla C_{R_1}^{k+1} = -\alpha \left(C_{R_1}^{k} C_{R_2}^{k} + C_{R_1}^{k} \right), \tag{7.2.1}$$

if $C_{R_2}^k \geq \epsilon > 0$ and if not, the boundary conditions go directly into the stiffness matrix. That is,

$$\alpha = \alpha \cdot C_{R_2}^k, \tag{7.2.2}$$

and similarly for $C_{R_2}^{k+1}$.

Algorithm 2 Block Gauss Seidel

1: Set initial guess $C_{R_1}^0$ and $C_{R_2}^0$
2: **For** $k = 0,\ 1, \ldots$ until reached convergence
3: $C_{R_1}^{k+1} = \texttt{SolveLaplace}\left(C_{R_1}^k,\ C_{R_2}^k\right)$
4: $C_{R_1}^{k+1} = \omega C_{R_1}^{k+1} + (1-\omega) C_{R_1}^k$
5: $C_{R_2}^{k+1} = \texttt{SolveLaplace}\left(C_{R_1}^{k+1},\ C_{R_2}^k\right)$
6: $C_{R_2}^{k+1} = \omega C_{R_2}^{k+1} + (1-\omega) C_{R_2}^k$
7: $C_P^k = \texttt{SolveLaplace}\left(C_{R_1}^{k+1},\ C_{R_2}^{k+1}\right)$
8: **End For**

Algorithm II shows the essential steps of the iterative method. To be precise, this is an implementation of the Successive Overrelaxation Method (SOR). By letting ω equal 1, this method is reduced to the ordinary Gauss Seidel method. The relaxation parameter ω can be chosen such that the convergence speed is increased. In fact it is called overrelaxation as long as $\omega > 1$.

In order to know when to stop iterating, a stopping criterium must be decided. Stopping criteria can be defined in several ways. One alternative is to consider

$$\mathbf{K}_i C_i^{k+1} = \mathbf{f}_i^k, \tag{7.2.3}$$

and require that

$$\|\mathbf{K}_i C_i^{k+1} - \mathbf{f}_i^k\|_\infty \leq \epsilon \to 0. \tag{7.2.4}$$

The program breaks when the stopping criterium is fulfilled. Table 7.2 shows an example of an iteration process, starting with $C_{R_1}^0 = C_{R_2}^0 = \texttt{zeros}(nnp, 1)$. For this example the program was told to stop iterating when ϵ became strictly smaller than $1.0 \cdot 10^{-6}$.

k	$\|\mathbf{K}_{R_1} C_{R_1}^{k+1} - \mathbf{f}_{R_1}^k\|_\infty$	$\|\mathbf{K}_{R_2} C_{R_2}^{k+1} - \mathbf{f}_{R_2}^k\|_\infty$	$\|\mathbf{K}_P C_P^{k+1} - \mathbf{f}_P^k\|_\infty$
0	5.833	4.520	0.367
1	3.617	0.796	0.548
2	1.368	0.899	0.270
3	1.075	0.086	0.130
4	0.370	0.167	0.065
5	0.306	0.029	0.031
⋮	⋮	⋮	⋮
26	$8.195 \cdot 10^{-7}$	$1.590 \cdot 10^{-7}$	$2.610 \cdot 10^{-8}$

Table 7.1: Iterating process, calculation of Equation (7.2.4).

7.3. Shape Optimization

The objective of this section is to perform shape optimization on the chemical system. As we did in Chapter 6, the optimization is done with respect to the boundary control points. The inner control points are updated according to The Spring Model. For the chemical system, the dashed edge in Figure 7.1 will be the varying part of the boundary. Several approaches can be used as objective function, but the intention will be to maximize some measure of the production of substance P.

7.3.1. Maximize $\int_\Omega C_P \, \mathrm{d}\mathbf{x}$

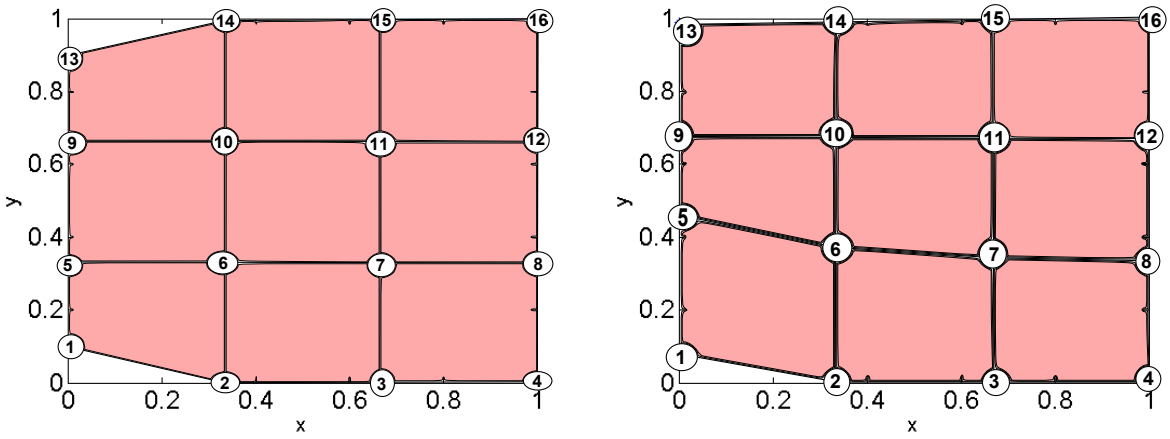
The system of equations to be solved is given in equations (7.1.9) and (7.1.10). We use the iterative method discussed in Section 7.2 and use `fmincon` to maximize the objective functional $J = \int_\Omega C_P \, \mathrm{d}\mathbf{x}$, where C_P is the concentration of substance P that is produced by substance R_1 and R_2. We start with fixed concentrations of all three substances on the exterior of Ω. For substance P, we fix the exterior parameter $\hat{C_P} = 100 \ mol/m^2$, whereas the other exterior concentrations are fixed to lower values. Though `fmincon` is a minimization tool, we can still use it for maximization purposes by considering

$$\min_\Omega -f(\mathbf{x}) = \max_\Omega f(\mathbf{x}). \tag{7.3.1}$$

7.3. Shape Optimization

Iteration nr.	$\int_\Omega C_P \, \mathrm{d}\mathbf{x}$
0	5.956 *mol*
1	6.157 *mol*
2	6.167 *mol*
3	6.441 *mol*
4	7.733 *mol*
5	8.107 *mol*

Table 7.2: Values for the iterations performed by `fmincon` in order to maximize the objective functional: $J = \int_\Omega C_P \, \mathrm{d}\mathbf{x}$. Note that the value of the final surface integral is increased with approximately 36% compared to the initial surface integral.



(a) Initial domain Ω. **(b)** Final domain Ω.

Figure 7.2: Initial domain and resulting domain after the shape optimization process. Figure 7.2b shows the shape of Ω giving $max \int_\Omega C_p$.

Table 7.2 shows the value of $J = \int_\Omega C_P \, \mathrm{d}\mathbf{x}$ after each iteration of `fmincon`. Both knot vectors ξ and η are quadratic, normalized with two internal knots at $\xi = \eta = 1/3$ and $2/3$. Several values for the relaxation parameter ω have been tested, but for these results we used $\omega = 1.78$. A maximum is found after 5 iterations with maximum value:

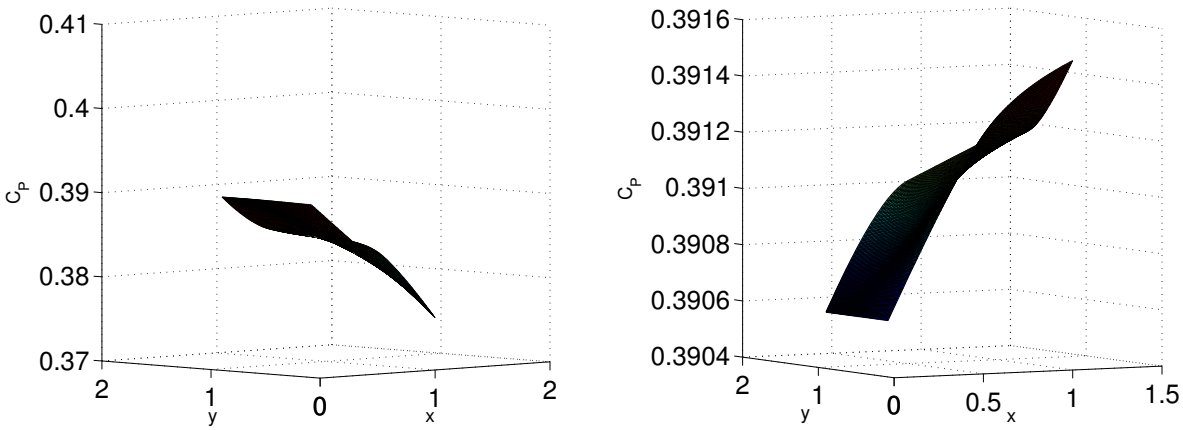
$$\max_\Omega \int_\Omega C_P \, \mathrm{d}\mathbf{x} = 8.107. \tag{7.3.2}$$

The optimization is done with respect to the left edge with linear Robin boundary conditions, where there is no reaction between R_1 and R_2. Though both the initial and the final value of our objective functional are low in total, we observe a 36.1% increase of the evaluated surface integral during the optimization process. The variation from the initial to the final perturbed control points is given by the map:

$$\begin{bmatrix} X_0 & Y_0 \\ 0 & 0.1 \\ 0 & 1/3 \\ 0 & 2/3 \\ 0 & 0.9 \end{bmatrix} \mapsto \begin{bmatrix} X_{\text{final}} & Y_{\text{final}} \\ 0 & 0.072 \\ 0 & 0.450 \\ 0 & 0.675 \\ 0 & 0.990 \end{bmatrix}.$$

7.3. Shape Optimization

Figure 7.2 shows both the initial and the resulting domain. The control points are marked in white, and in Figure 7.2b the inner control points are also slightly perturbed. We observe that the edge with no chemical reaction in Figure 7.2b is stretched compared to the edge with no chemical reaction in the initial domain in Figure 7.2a. From a physical point of view this makes sense, as we started with a high concentration $\hat{C}_p$ on the outside of Ω. The boundary conditions on the non-reacting edge for substance P are dependent on the exterior concentration of substance P. When this exterior concentration is high, it will be beneficial to increase the span of this edge, in order to achieve as high as possible concentration of substance P also in the interior domain.



(a) Initial concentration profile C_P **(b)** Final concentration profile C_P.

Figure 7.3: Figure 7.3a and 7.3b show the initial- and final concentration profiles for substance P respectively. We observe that the initial concentration profile is lower than the final concentration profile. Also note that the area where it seems to be lowest concentration initially, becomes the area with highest concentration in the final solution.

Table 7.3 shows the initial concentration of substance P in the centered column, and the final concentration in the right column at each of the control points. The final concentration is a result after 5 iterations of shape optimization. Knot vectors, polynomial degrees and initial control points are as described above. The results in Table 7.3 are visualized in Figure 7.3. It seems that the area of lowest initial concentration in Figure 7.3a becomes the area of highest final concentration in Figure 7.3b. This supports that the final shape of the domain is improved compared to the initial domain. We mentioned above that the concentration exterior to the domain Ω of substance P, $\hat{C}_P$, was set to $\hat{C}_P = 100\ mol/m^2$. Since substance P is dependent on this high exterior concentration on the edge of which we optimize, it seems reasonable that a stretching of this edge will lead to higher internal concentration of substance P.

Control point nr.	C_P - initial	C_P - final
1	0.3905 mol/m^2	0.3906 mol/m^2
2	0.3852 mol/m^2	0.3903 mol/m^2
3	0.3812 mol/m^2	0.3902 mol/m^2
4	0.3762 mol/m^2	0.3902 mol/m^2
5	0.3906 mol/m^2	0.3906 mol/m^2
6	0.3872 mol/m^2	0.3904 mol/m^2
7	0.3834 mol/m^2	0.3902 mol/m^2
8	0.3817 mol/m^2	0.3901 mol/m^2
9	0.3906 mol/m^2	0.3906 mol/m^2
10	0.3876 mol/m^2	0.3903 mol/m^2
11	0.3851 mol/m^2	0.3900 mol/m^2
12	0.3848 mol/m^2	0.3898 mol/m^2
13	0.3906 mol/m^2	0.3906 mol/m^2
14	0.3870 mol/m^2	0.3903 mol/m^2
15	0.3855 mol/m^2	0.3900 mol/m^2
16	0.3846 mol/m^2	0.3894 mol/m^2

Table 7.3: This table shows the values of the concentratiton of substance P evaluated at the different control points. The column in the middle shows the initial concentration, whereas the column located to the right shows the final concentration after 5 iterations of shape optimization. We observe that the final concentration in total, is slightly higher than the initial concentration.

Chapter 8

Concluding Remarks

8.1. Conclusions

In this thesis we have implemented a NURBS based isogeometric finite element solver in `MATLAB` for Poisson's equation. The solver has been tested on equations assigned either Dirichlet, Neumann, Robin or mixed boundary conditions over different types of physical geometries. In the first examples, we let the domain Ω be the unit square and found in all cases that the expected convergence according to (4.1.7) was asymptotically achieved. Additionally we tested the solver for a circular domain, by utilizing NURBS as basis functions and achieved expected convergence in the FEM sense also for this example.

In the study of shape optimization, a reparameterization algorithm was needed to update the inner control points when we perturbed the boundary control points. We have utilized a linear parameterization method to parameterize the control meshes. The choice of method fell on The Spring Model, a method applicable also for non-trivial geometries. As a verification that our parameterization algorithm was correctly applied to the IGA solver, we solved Poisson's equation on the unit disk. That the optimal convergence in this example was not achieved is due to the use of B-splines. They lack the ability to represent conic geometries. The problems arose in the mapping from the parameter space to the physical space. That is, linearly independent vectors were mapped to linearly dependent vectors. The result was that the mappings introduced four singularities in the parameterization, as was confirmed by noting that the Jacobian mapping was singular in the points corresponding to the four corners of the parameter space. Both Figure 5.5 and Table 5.3 show that $\det(J)$ tends more and more to zero by refining the knot intervals.

Further, we have studied what we referred to as gradient free and gradient based shape optimization, by utilizing `MATLAB`'s optimization toolbox `fmincon`. For the latter we have rewritten the IGA solver to an object oriented code, in order to apply automatic differentiation. Both approaches gave satisfying results and we could have settled for gradient free methods. However, `fmincon` uses finite differences by default to approximate

the derivatives when they are not provided by the user. Numerical derivatives can lead to inaccurate results due to round off errors, so we wanted to compare the obtained results by the two methods. We found that by supplying gradient to `fmincon`, the program used less iterations to find better solutions.

As an application of IGA based shape optimization we have studied a chemical reaction system. The system consists of three stationary diffusion equations coupled through non-linear boundary conditions. Due to the non-linearity of the equations we had to use an iterative method to solve the system, and the choice fell on Gauss Seidel with relaxation (SOR). The shape optimization problem was to find the shape of the domain Ω in order to find

$$\max_{\Omega} \int_{\Omega} C_P \, \mathrm{d}x,$$

where C_P is the concentration of the produced substance. By starting with a high exterior concentration C_P, we found that the objective functional J increased from $J = 5.956\ mol$ to $J = 8.107\ mol$, that is a 36.1% improvement. We also observed that the optimal shape for this problem with the specific used initial data, corresponded to a stretching of the edge we optimized with respect to. We conclude that this result makes sense also from a physical point of view, due to the Robin boundary conditions for substance P, that are dependent on the high exterior concentration, C_P.

8.2. Future Work

In this thesis, all the considered domains have been described by a single patch. When we used B-splines and described the unit disk with only one patch, we faced convergence issues with the numerical method. These issues could have been fixed, had we instead described the geometry with multiple patches. By introducing sufficiently many patches, the problem with singular mappings is avoided and we expect that the we would have observed a similar convergence rate as one does when using FEM polynomials or NURBS. An interesting problem is to validate the convergence rate for circular domains while using B-splines and introducing multiple patches. Another advantage of multiple patches is that it opens the possibility for local refinement. We would like to study shape optimization problems by using LR B-splines, e.g. to see how these basis functions would affect boundaries with singularities. IGA with use of LR B-splines is an already established research area, see e.g. [9]. However, we have found it difficult to find much existing literature on the combination of shape optimization and LR B-splines.

During the writing of this thesis, the main focus has been to acquire knowledge on shape optimization and IGA, both of which were relatively new branches of mathematics to the author. This left little time to optimize the efficiency of the written `MATLAB` programs. In particular, we would like to optimize the programs for executing automatic differentiation. We observed that the runtime between the shape optimization with automatic differentiation was only half the runtime of shape optimization with use of finite differences. We believe that the runtime would have been significantly decreased with AD, had the computer programs been implemented with more focus on efficiency.

At last, we mention that we now have a solid code library, that is able to solve systems of PDEs arising from physics. A natural continuation of the work done on the chemical reaction system in Chapter 7 is to consider more complex geometries and to do more research and testing on the different parameters we pass on to the solver. Further development of the IGA code framework is an extension to solve other types of PDEs, for instance to make it applicable to problems within fluid dynamics, an area of high interest within modern research and technology.

Appendices

Appendix A

A Selection of Included MATLAB Programs

A.1. Evaluate Global Stiffness Matrix A and Force Vector F

The following program evaluates the global stiffness matrix **A** and the global force vector **F** for the two dimensional Poisson equation with homogeneous Dirichlet boundary conditions.

```
function [A,F_glob,R] = stiffnessGlobal(Txi,Teta,p,q,n,m,...
    INCarray,IENarray,nel,nen,nnp,Xout,Yout,B)

% stiffnessGlobal computes the global stiffness matrix and right hand
% side for the 2D Poisson
% equation on the unit disk.
% B is holding the weights, which are all 1 in case of B-splines. ...
    Xout and
% Yout are the control points in X and Y direction, stored in column
% vectors. Txi and Teta are the knot vectors.

f = @(x,y)  -8*pi*cos(2*pi*(x^2+y^2))...
    +16*pi^2*(x^2+y^2)*sin(2*pi*(x^2+y^2));  % Right hand side
ngp = 4;                                     % Number of Gauss ...
    quadrature points.
[X,wi]= gaussQuad(ngp);                     % Find Gauss points and ...
    weights.
A = zeros(nnp,nnp);                          % Allocate global ...
    stiffness matrix
F_glob = zeros(nnp,1);                       % Allocate global Force ...
    vector
ff = @(x,y) x^2+y^2;                         % function used to test .....
    .
                                             % that numerical ...
                                                integration is correct.
```

```
fft = 0;
  for el = 1:nel
      ni = INCarray(IENarray(1,el),1);      % find the support...
      nj = INCarray(IENarray(1,el),2);      %(NURBS coordinates for ...
         each element).
      ihatt = IENarray(:,el); temp = fliplr(ihatt');ihatt =temp';
                                            % indices for which ...
                                               control points...
                                            % to pick out on each element.
      Bx = Xout(ihatt);
      By = Yout(ihatt);

     if (Txi(ni+1)==Txi(ni) || Teta(nj+1)==Teta(nj))
                                            % Check if element has ...
                                               zero measure.
         continue;
     end

     F_loc = zeros(nen,1);                  % initialize local matrix ...
        and vector.
     A_el = zeros(nen);

    for alpha = 1:ngp
        for beta = 1:ngp

            [R,dR_dx,J,Jtilde] = ...
               shape_function1(ni,nj,X(alpha),X(beta),...
                Txi,Teta,p,q,nen,n,m,Xout,Yout,B,ihatt);

            x = R*Bx;                       % physical coordinates
            y = R*By;

            fft = fft + wi(alpha)*wi(beta)*ff(x,y)*abs(J)*Jtilde;
                                            %test to check if ...
                                               integration is correct
            for a = 1:nen
                F_loc(a) = F_loc(a) + ...
                    wi(alpha)*wi(beta)*f(x,y)*R(a)*abs(J)*Jtilde;

                for b = 1:nen
                    A_el(a,b) = A_el(a,b) + [dR_dx(1,a) dR_dx(2,a)]*
                        [dR_dx(1,b) ...
                           dR_dx(2,b)]*wi(alpha)*wi(beta)*abs(J)*Jtilde;

                end
            end
        end
    end
        A(ihatt,ihatt) = A(ihatt,ihatt) + A_el;      % Assemble to ...
           global system
        F_glob(ihatt) = F_glob(ihatt) + F_loc;

  end
end
```

A.2. Program for Updating Internal Control Points

The function `fivePointPoisson.m` evaluates internal control points, when the boundary control points are given. It does so by exploiting a five point stencil, as described in the section on The Spring Model. This procedure must be adjusted to the problem at hand, as it depends on the parameterization of the domain for which the PDE is solved on. Though the procedures for other domains are similar, this particular program applies for problems where the domain Ω is the unit disk. Adjustments must also be made depending on the number of basis functions (n and m) in the ξ- and η directions.

```
1
2  function [Xout,Yout] = fivePointPoisson(n,m)
3  close all;
4  % The discretization is a five point stencil, as in finite differences,
5  % s.t. each internal control point becomes
6  % the average of its 4 neighbour values, though we only know the control
7  % points on the edge.
8  % Xout and Yout are column vectors holding both boundary and internal
9  % control points in X- and Y direction respectively.
10
11 [Edge,interior] = Find_edge_and_interior1(n,m);
12 interiorLength = length(interior);
13 lengthEdge = length(Edge);
14 Xout = zeros(n*m,1);            %The new internal control points in x-dir
15 Yout = zeros(n*m,1);            %The new internal control points in y-dir
16
17 %% allocate memory
18 rhsxL = zeros(interiorLength,1);
19 rhsyL = zeros(interiorLength,1);
20 rhsxB = zeros(interiorLength,1);
21 rhsyB = zeros(interiorLength,1);
22 rhsxR = zeros(interiorLength,1);
23 rhsyR = zeros(interiorLength,1);
24 rhsxT = zeros(interiorLength,1);
25 rhsyT = zeros(interiorLength,1);
26
27 A = -full(gallery('poisson',n)); %Discrete laplacian matrix.
28 A(Edge,:) = [];
29 A(:,Edge) = [];
30
31
32 vinkel = (360/(lengthEdge))*pi/180;
33
34 indexedgle = n+1:2:lengthEdge-n ;          % Index of the edge.
35 indexedgbo = 1:n;
36 indexedgri = n+2:2:lengthEdge-n;
37 indexedgto = lengthEdge-n+1:lengthEdge;
38
39 xedge = zeros(lengthEdge,1);
40 yedge = zeros(lengthEdge,1);
41 length(xedge);
42 indexvstartl = lengthEdge/2+floor(n/2)-1;
43
44 indexvl = indexvstartl:-1:indexvstartl-length(indexedgle)+1;
```

A.2. Program for Updating Internal Control Points

```
indexvb = indexvstartl+1:indexvstartl+n;
indexvr = [indexvstartl+n+1:lengthEdge,1:floor(n/2)-1];
indexvt = indexvstartl-length(indexedgle):-1:floor(n/2);
xedge(indexedgle) = cos(indexvl*vinkel);
yedge(indexedgle) = sin(indexvl*vinkel);
xedge(indexedgbo) = cos(indexvb*vinkel);
yedge(indexedgbo) = sin(indexvb*vinkel);
xedge(indexedgri) = cos(indexvr*vinkel);
yedge(indexedgri) = sin(indexvr*vinkel);
xedge(indexedgto) = cos(indexvt*vinkel);
yedge(indexedgto) = sin(indexvt*vinkel);

indexintL = 1:n-2:interiorLength;
indexedgL = n+1:n:n*m-(n+1);
rhsxL(indexintL) = -xedge(indexedgle);
rhsyL(indexintL) = -yedge(indexedgle);
indexintB = 1:n-2;
indexedgB = 2:n-1;
rhsxB(indexintB) = -xedge(indexedgB);
rhsyB(indexintB) = -yedge(indexedgB);
indexintR = (n-2):(n-2):interiorLength;
indexedgR = 2*n:n:n*m-n;
rhsxR(indexintR) = -xedge(indexedgri);
rhsyR(indexintR) = -yedge(indexedgri);
indexintT = interiorLength-(n-3):interiorLength;
indexedgT = n*n-(n-1):n*m;
rhsxT(indexintT) = -xedge(indexedgto(1)+1:lengthEdge-1);
rhsyT(indexintT) = -yedge(indexedgto(1)+1:lengthEdge-1);
rhsx = rhsxL+rhsxB+rhsxR+rhsxT;
rhsy = rhsyL+rhsyB+rhsyR+rhsyT;


%%solving linear system to give new internal control points.

X = A\rhsx;
Y = A\rhsy;

Xout(indexedgbo) = xedge(indexedgbo);
Yout(indexedgbo) = yedge(indexedgbo);
Xout(indexedgR) = xedge(indexedgri);
Yout(indexedgR) = yedge(indexedgri);
Xout(indexedgL) = xedge(indexedgle);
Yout(indexedgL) = yedge(indexedgle);
Xout(indexedgT) = xedge(indexedgto);
Yout(indexedgT) = yedge(indexedgto);
Xout(interior) = X;
Yout(interior) = Y;


end
```

A.3. Evaluation of the Objective Functional J

Here comes the program evaluating the objective function used in sections 6.1 and 6.2. The exact solution is given as input along with the initial boundary control points. The program `fivePointPoisson` takes the boundary points as input and then computes the internal control points as discussed in section 5.1. The reason for why the edge is removed in the stiffness matrix and the force vector coming from `stiffnessGlobal.m`, is that homogeneous Dirichlet conditions are used in Equation (5.1.4) which was the reference equation in sections 6.1 and 6.2.

```
function objective = ...
    Objective(u_eks,Txi,Teta,p,q,n,m,INCarray,IENarray,nel,nen,X0Y0)

ngp = 4;                          % Number of Gauss quadrature points.
[X,wi]= gaussQuad2(ngp);          % Find Gauss points and weights.
objective = 0;
nnp = n*m;

[Xout,Yout]= fivePointPoisson(n,m,X0Y0);
[rows2remove,interior] = Find_edge_and_interior(n,m) ;
[K,F_glob,R] = ...
    stiffnessGlobal(Txi,Teta,p,q,n,m,INCarray,IENarray,nel,nen,nnp,X0Y0);
cols2remove = rows2remove;
K(rows2remove,:)=[];
K(:,cols2remove)=[];
F_glob(rows2remove) = [];
U_num = K\F_glob;
U_h = zeros(n*m,1);
U_h(interior) = U_num;
U_numer = zeros(n,m);
U_numer(:) = U_h;

  for el = 1:nel
      ni = INCarray(IENarray(1,el),1);
      nj = INCarray(IENarray(1,el),2);
      ihatt = IENarray(:,el); temp = fliplr(ihatt');ihatt =temp';

     if (Txi(ni+1)==Txi(ni) || Teta(nj+1)==Teta(nj))
         continue;
     end


    for alpha = 1:ngp
        for beta = 1:ngp

            [R,dR_dx,J,Jtilde] = ...
                shape_function2(ni,nj,X(alpha),X(beta),...
                 Txi,Teta,p,q,nen,n,m,Xout,Yout,B,ihatt);


            for a = 1:n
                for b = 1:m

                    objective = objective + wi(alpha)*wi(beta)...
```

```
                        *((U_numer(a,b)-u_eks(a,b))^2)*abs(J)*Jtilde;
                end
            end

        end
    end

  end


end
```

Bibliography

[1] Howard Anton and Chris Rorres. *Elementary Linear Algebra With Supplementary Applications*. Wiley, 10th edition, 2010.

[2] Lee A.Segel. *Mathematical models in molecular and cellular Biology*. Cambridge University Press, 1980.

[3] E. Blūms, Yu.A. Mikhailov, and R. Ozols. *Heat and Mass Transfer in MHD Flows*. World Scientific, 1987.

[4] Andre R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction To Derivative-Free Optimization*. SIAM, 2009.

[5] Tor Dokken, Tom Lyche, and Kjell Fredrik Pettersen. Polynomial splines over locally refined box-partitions. *Computer Aided Geometric Design*, 30(3):331–356, 2013.

[6] Lawrence C. Evans. *Partial Differential Equations*. The American Mathematical Society, 2nd edition, 2010.

[7] Jens Gravesen, Anton Evgrafov, Dang-Manh Nguyen, and Peter Nørtoft. Planar Parametrization in Isogeometric Analysis.

[8] Yuri Bazilevs J. Austin Cottrell, Thomas J.R Hughes. *Isogeometric Analysis: Toward Integration of CAD and FEA*. WILEY, 1th edition, 2009.

[9] Kjetil André Johannessen, Trond Kvamsdal, and Tor Dokken. *Isogeometric Analysis Using LR B-splines*. Elsevier, 2014.

[10] W. Kahan. *Gauss-Seidel methods of solving large systems of linear equations. Doctoral thesis, University of Toronto, Toronto, Canada.* 1958.

[11] Günter Leugering, Peter Benner, Sebastian Engell, Andreas Griewank, Helmut Harbrecht, Michael Hinze, Rolf Rannacher, and Stefan Ulbrich. *Trends in PDE Constrained Optimization*. Springer, 2014.

[12] C.C. Long, A.L. Marsden, and Y. Bazilevs. Shape optimization of pulsatile ventricular assist devices using FSI to minimize thrombotic risk. 2014.

Bibliography

[13] Tom Lyche and Knut Mørken. Spline methods draft. `http://www.uio.no/studier/emner/matnat/ifi/INF-MAT5340/v11/undervisningsmateriale/book.pdf`, 2011.

[14] Nguyen Dang Mahn, Anton Evgrafov, Jens Gravesen, and Domenico Lahaye. Isogeometric shape optimization of magnetic density separators. 2014.

[15] Nguyen Dang Manh. *Isogeometric Analysis and Shape Optimization in Electromagnetism, Ph.D. thesis- Technical University of Denmark*. 2012.

[16] John N. McDonald and Neil A. Weiss. *A course in REAL ANALYSIS*.

[17] William McIlhagga. autodiff library. `http://www.mathworks.com/matlabcentral/fileexchange/26807-automatic-differentiation-with-matlab-objects/content/automatic%20differentiation/autodiff.m`, 2010.

[18] Bijan Mohammadi and Olivier Pironneau. *Applied Shape Optimization for Fluids*. Oxford Science Publications, 2nd edition, 2009.

[19] Attila Péter Nagy. Isogeometric Design Optimisation. 2011.

[20] Richard D. Neidinger. Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming.

[21] Vinh Phu Nguyen, Stéphane P.A. Bordas, and Timon Rabczuk. Isogeometric analysis: an overview and computer implementation aspects.

[22] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[23] Alfio Quarteroni. *Numerical Models for Differential Problems*. Springer, 2nd edition, 2009.

[24] Sandro Salsa. *Partial Differential Equations in Action*. Springer, 2nd edition, 2009.

[25] Ridgway Scott. Optimal L^∞ Estimates for the Finite Element Method on irregular Meshes. 1976.

[26] Thomas W. Sederberg, Jianmin Zheng, Almaz Bakenov, and Ahmad Nasri. T-splines and T-NURCCs. 2003.

[27] John C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Siam, 2nd edition, 2004.

[28] Wolfgang A. Wall, Moritz A. Frenzel, and Christian Cyron. Isogeometric structural shape optimization.

[29] R.J. Yang, A. Lee, and D.T. McGeen. Application of basis function conceptsto practical shape optimization problems. 1992.