NTNU – Trondheim
Norwegian University of
Science and Technology

Mesh Generation Techniques for Isogeometric Analysis

Kine Aurora Mothes Hansvold

Master of Science in Physics and Mathematics
Submission date: June 2015
Supervisor: Trond Kvamsdal, MATH
Co-supervisor: Kjetil André Johannessen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

In any method aimed at solving a boundary value problem using isogeometric analysis, it is imperative to find a high quality parameterization of the domain over which the partial differential equation is posed. The parameterization of the domain is an essential part of solving the problem, and the accuracy of the analysis to be performed rely heavily on the quality of the parameterization.

In this thesis we introduce four different methods for parameterization of planar geometries for applications within isogeometric analysis. The methods all rely on B-splines and the isogeometric framework. We describe B-splines and isogeometric analysis in detail, and we introduce several mesh metrics to be used to check the mesh quality in our pursuit of producing superior meshes.

Several illustrative examples are given, and the methods tested on several different geometries, all representing different parameterization challenges.

The methods are found to produce quite different parameterizations for the same geometry. We have found that the most complex methods in general show the best overall performance, both with respect to mesh quality and perseverance.

Sammendrag

I enhver metode som tar sikte på å løse et randverdiproblem ved hjelp av isogeometrisk analyse er det viktig å finne en høy-kvalitets parametrisering av det fysiske domenet der den partielle differensialligningen er formulert. Parametriseringen av domenet er en avgjørende del av løsningen av det aktuelle problemet, og analysens gyldighet og nøyaktighet er sterkt avhengig av kvaliteten på parametriseringen.

I denne oppgaven vil vi presentere fire ulike metoder for parametrisering av domener i planet med hensyn på bruk innen isogeometrisk analyse. De fire metodene vil alle involvere B-spliner og det isogeometriske rammeverket. Av den grunn vil vi presentere en detaljert beskrivelse av både B-spliner og isogeometrisk analyse. Vi vil også introdusere flere gitter-metrikker som vil bli brukt som mål på gitterkvalitet, og som et hjelpemiddel for å produsere optimale gitter.

Flere illustrerende eksempler vil bli gitt ved at metodene testes på flere forskjellige geometrier, alle med ulike utfordringer forbundet med domene-parametrisering.

Resultatene viser at når metodene anvendes på den samme geometrien, produserer de relativt ulike parametriseringer. Det har vist seg at de mest komplekse metodene generelt kan vise til totalt best prestasjon, både når det gjelder gitterkvalitet og utholdenhet på utfordrende geometrier.

Preface

This thesis completes my studies for the Master's Degree Programme in Applied Physics and Mathematics, with specialization in Industrial Mathematics, at the Norwegian University of Science and Technology (NTNU). The work has been carried out during the spring semester of 2015 at the Department of Mathematical Sciences. It has been supervised by Professor Trond Kvamsdal and Postdoctoral Fellow Kjetil André Johannessen, both from the Department of Mathematical Sciences at NTNU, who also developed the idea for the thesis.

This thesis is an extension of a preceding specialization project, carried out during the fall semester of 2014. Theory from this project has been included here for completeness.

Trondheim, 21st June, 2015

Kine A. M. Hansvold

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Professor Trond Kvamsdal and Postdoctoral Fellow Kjetil André Johannessen, for their valuable guidance and support. Their help is greatly appreciated; the hours of discussion, suggestions, and result interpretations as well as their continuous advice and dedication helped this work become coherent and to elevate its quality.

I also wish to take this opportunity to thank my fellow students, Hager, Bjørn, Sverre, Mats, Henrik, Ole, Hallvard, Christoffer, and Aleks for their input and many good discussions.

Lastly, a huge thanks is directed to my family for their constant support and understanding, without which this thesis would not have seen the light of day.

K.A.M.H

Contents

Chapter 1

Introduction

1.1 Background

Finite Element Analysis (FEA) has been used for a long time to find numerical approximations to solutions of boundary value problems. The Finite Element Method (FEM) has been a preferred discretization technique for design and analysis for engineers [1] and has a wide range of applications [2], including solid and structural mechanics, dynamics, and thermal analysis to name a few. Common to all applications of FEM is the need for a geometrical model representing the object being analyzed. These design models are typically made using computer aided design (CAD) techniques.

Although this is the common practice, it is not without complications. When using CAD it generally takes a long time to generate a mesh suitable for analysis. This is unfortunate if the structure being analyzed undergoes frequent design changes as is the case in many industrial applications. Furthermore, the CAD model is not particularly suitable for analysis, so a geometric representation of the model is used. However, the geometric representation is only an approximation to the CAD model, and therefore refinement and improvements are necessary. This process is both costly and time-consuming, and in the end the result is only an approximated model, resulting in geometry inaccuracies [3].

A relatively new technique aimed at mending some of the shortcomings of FEM was introduced by Hughes et al. in 2004 [3]. The idea is centered around building a common model to be used both in CAD and FEM. Since CAD is a much larger industry compared to FEA, and because CAD already can be used to make exact models, it followed logically to explore possibilities for a new way of conducting the analysis, while still being based on the familiar FEA. The result is now known as isogeometric analysis (IGA) [2].

The idea is quite similar to that of FEA, with an important difference regard-

ing the basis functions, and, by extension, the geometry. By replacing the shape functions in FEA with spline basis functions, the classical Galerkin approach is still valid, while at the same time making it possible to perform analysis directly on CAD models, and thus ensuring exact geometrical representation. Splines are already incorporated in most CAD techniques, since most geometries are graphically represented using an extension of B-splines called NURBS.

By the introduction of isogeometric analysis, it became possible to represent the geometry for analysis and the object itself using the same functions, and hence avoiding difficulties connected to inaccurate geometries. The new basis functions would also make refinements easier, since the geometry is constant under spline refinement, and therefore the exact representation is preserved in every refinement.

IGA is a relatively new field when compared to FEA, and much research is being done on improving techniques. Several papers on local refinement of splines have been presented in recent years. In 2013, Dokken et al. addressed local refinement of splines defined on box-partitions [4]. This introduction to locally refined B-splines, LR B-splines, was continued by Johannessen et al. and used in local refinement strategies for adaptive IGA in 2014 [5]. The work on local refinement was expanded in 2015 by Johannessen et al. [6] where they presented and compared three strategies involving Classical Hierarchical B-Splines, Truncated Hierarchical B-splines, and Locally Refined B-splines.

It is indisputable that parameterization, adaptation and refinement is an very important part of IGA. This due to the fact that in any problem aimed at solving a boundary value problem with IGA, it is essential to have an efficient way of generating meshes of high quality. This is because the IGA methods require a valid parameterization of the domain over which the PDE is posed, and the accuracy of the analysis is heavily affected by the quality of this parameterization [7]. The importance of generating a high-quality parameterization of the physical domain in IGA can be compared to the importance of mesh generation in the standard FEA [1].

While adaptation and local refinement is widely used, it is also possible to generate a parameterization of the physical domain through control point relocation. The need for a robust and efficient parameterization from the boundary of a domain onto its interior is especially important in the use of isogeometric analysis in shape optimization, where a domain parameterization is performed in every optimization iteration. The application of isogeometric analysis in shape optimization problems has been investigated among others by Manh et al. in [8], and Gravesen et al. in [7].

When an isogeometric framework is used for fluid flow simulations, the simulations will become inaccurate unless the parameterization is of very high quality. This was examined by Nordanger et. al in [9] and [10].

Good techniques for parameterization of a domain are also needed when dealing with systems undergoing large deformations. In such situations local refinement may not be enough, and it will be necessary to do a re-meshings. Another reason for looking at parameterization of the interior of a physical domain given its boundary, is that CAD systems typically provide information about the boundary of a domain only, while IGA requires a parameterization of the entire object. Thus, for IGA and CAD to be compatible, we need to be able to construct a domain parameterization from a given boundary.

Summarizing, being able to produce high quality parameterizations of a domain is imperative, but it is also time consuming. Consequently, we wish to automate the process of finding the best possible mesh for a given domain.

1.2 Aim and Outline

In this thesis we will examine several methods for parameterization of the interior of a physical domain given its boundary through control point relocation. The methods will be based on an isogeometric foundation.

Chapter 2 introduces several mesh metrics as a way of measuring the quality of a parameterization mesh.

Chapter 3 gives a detailed and thorough introduction to splines. Understanding of splines is essential for the understanding and development of isogeometric methods. Chapter 3 also give an thorough introduction to isogeometric analysis. However, it will be assumed that the reader is already familiar with some of the classical numerical methods, including the finite element method, so these will not be presented with the same level of detail.

Chapter 4 presents four different methods for domain parameterization. Each method will be tested on three different geometries, all representing different parameterization challenges.

The numerical results are presented in Chapter 5. This chapter also compares the methods regarding their ability to produce valid parameterizations for increasingly challenging geometries, and their dependence on the polynomial degree.

A summary and concluding remarks can be found in Chapter 6. Finally, the appendix presents the source code for the four different methods, as well as some geometry specifications.

Chapter 2

Mesh Metrics

Since the aim of this thesis is to investigate different mesh generation techniques, it is useful to have some measure of the quality of each mesh. That is, once a parameterization is obtained it is necessary to have a way of evaluating its quality and determine if it is suitable for computational analysis. Ultimately, the mesh quality must be connected with the final solution error. However, for practical purposes, it is useful to be able to evaluate the mesh quality before doing the computations.

There exists a number of different metrics for meshes [11]. Only a few will be considered here. All of the metrics tested will use the Jacobian matrix [12]. The Jacobian is convenient because it contains information on size, shape, and skew. These will each be evaluated as a metric. The metrics will be presented as explicit formulas for quadrilateral elements. The reader is referred to [12] and [13] for more theoretical background and metrics for other element types.

2.1 Quadrilateral Elements

Before we start introducing the metrics, some background material is presented. See [12] for more details. All the parameterization methods will be using spline representations. Each mesh element will consist of four nodes, but the edges connecting them are not required to be straight. That is, the elements are not necessarily quadrilaterals, even though they have four nodes. However, for simplicity we will assume that the elements may be considered as quadrilaterals when developing mesh metrics. Therefor, all the metrics being presented here are applied to quadrilateral elements defined by four nodes with coordinates (x_i, y_i), where $i = 1, 2, 3, 4$, see Figure 2.1. For each node i, the corresponding Jacobian matrix

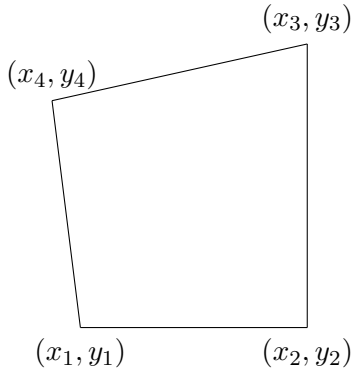


Figure 2.1: Quadrilateral element with nodes (x_i, y_i), $i = 1, 2, 3, 4$.

A_i can be formed as

$$A_i = \begin{bmatrix} x_{i+1} - x_i & x_{i+3} - x_i \\ y_{i+1} - y_i & y_{i+3} - y_i \end{bmatrix} \tag{2.1}$$

Note that the modulo operator is applied to the subscripts, so that if $i = 2$, then $i + 3 = 1$. The determinant of the Jacobian matrix, α_i, denotes the local area associated with node i [12], and thus the sum of α_i and α_{i+2} is twice the total area of the quadrilateral regardless of the index i. The product of the matrix $A_i^T A_i$ may be written as

$$A_i^T A_i = \begin{bmatrix} \lambda_{11}^i & \lambda_{12}^i \\ \lambda_{12}^i & \lambda_{22}^i \end{bmatrix} \tag{2.2}$$

where $\lambda_{11}^i = (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$ and $\lambda_{22}^i = (x_{i+3} - x_i)^2 + (y_{i+3} - y_i)^2$. This can be interpreted as the square of the length of the sides connecting node i with node $i + 1$ and $i + 3$ through the use of the Pythagorean theorem. The matrix is symmetrical, and the element λ_{12}^i relates to the angle between the two sides connected at node i.

If $\alpha_i = 0$ or $\alpha_i = \infty$, for any i, then the element is considered degenerate. This means that any element consisting of three or more collinear nodes will be degenerate.

Lastly, we note that any element possessing a negative local area, $\alpha_i < 0$ for any node i, is excluded from the domain of the metrics. This exclusion covers arrow shaped and bow-tie shaped elements. If such elements are encountered, the corresponding metrics will be set to zero.

2.2 Metrics

Now we are ready to introduce the metrics we will be using. We will present a relative size metric, a shape metric, and a skew metric. Additionally, we will use

combinations of these metrics. All the metrics compare each element in the mesh with a reference element in order to determine its quality.

2.2.1 Relative Size Metric

The first metric is the relative size metric. Mesh elements of relatively equal size are preferred, while elements that are much smaller or much larger than other elements are undesirable. One way to measure the relative size is to compare each element to a reference element with area a. Here we define the reference area to be the size of an element if the elements were uniform in the domain. That is, if we have N elements on a domain with total area A_{tot}, then $a = \frac{A_{tot}}{N}$.

Furthermore, we define $\sigma_i = (\alpha_i + \alpha_{i+2})/2a$ for each element. Notice that σ is node independent and can be used for an arbitrary i, since $\alpha_i + \alpha_{i+2} = \alpha_{i+1} + \alpha_{i+3}$. As we want to find both very small and very large elements, we use

$$\mathcal{M}_{Size} = \min\left(\sigma_i,\ \frac{1}{\sigma_i}\right) \tag{2.3}$$

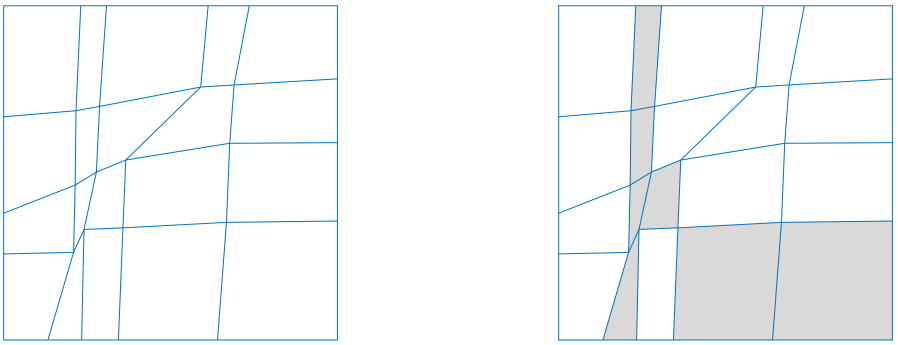
as the relative size metric for quadrilateral elements. The domain of this metric is all quadrilateral elements with positive, finite area, requiring that $0 \leq \alpha_i + \alpha_{i+2} < \infty$. If some elements in the mesh are inverted, the metric will pick up on this because these elements will result in a negative area.

The image of the metric is all real numbers between 0 and 1, with the special cases

$$\begin{aligned} \mathcal{M}_{Size} = 1 \quad &\Longleftrightarrow \quad (\alpha_i + \alpha_{i+2})/2 = a, \\ \mathcal{M}_{Size} = 0 \quad &\Longleftrightarrow \quad \text{element is degenerate.} \end{aligned}$$

Figure 2.2a shows a randomly generated mesh with 20 elements and Figure 2.2b shows in shading the elements for which $\mathcal{M}_{Size}$ is less than 0.5. The metric picks up both very small elements such as those forming the second leftmost column and large elements like those on the right side of the bottom row.

However, this metric alone is not very useful. Figure 2.2 illustrates that the metric allows for some rather poorly shaped elements, implying that it is not enough to rely solely on the relative size metric to identify all elements of poor quality.



(a) Mesh with 20 elements

(b) Elements with $\mathcal{M}_{Size} < 0.5$

Figure 2.2: Randomly generated mesh and elements varying in size.

2.2.2 Shape Metric

The shape metric $\mathcal{M}_{Shape}$ is designed to find poorly shaped elements. The idea behind this metric is to find how much each element deviates from a square reference element. In this metric it is necessary to include terms from all four nodes of the quadrilateral in order to ensure that the metric is nodal invariant.

For a square we have that $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 =$ element area, and as all the sides of a square by definition have the same length, $\lambda_{11}^i = \lambda_{22}^i$ and $\lambda_{11}^i = \lambda_{11}^j$, for $i, j = 1, 2, 3, 4$. Furthermore, because λ_{11}^i can be interpreted as the square of the length of the sides in the square, it follows that λ_{11}^i equals the total area of the square element. This means that a square element satisfies

$$\left(\lambda_{11}^i + \lambda_{22}^i\right) = 2\alpha_i, \quad i = 1, 2, 3, 4. \tag{2.4}$$

Using this result, we define the shape metric as

$$\mathcal{M}_{Shape} = \frac{8}{\sum_{i=1}^{4} \left(\lambda_{11}^i + \lambda_{22}^i\right)/\alpha_i}. \tag{2.5}$$

Here, we divide 8 by the sum of equation (2.4) for all i in order for the metric to lie between 0 and 1, since the maximum of (2.4) is 2. Some special cases also arise with this metric,

$$\begin{aligned} \mathcal{M}_{Shape} = 1 \quad &\Longleftrightarrow \quad \text{element is square,} \\ \mathcal{M}_{Shape} = 0 \quad &\Longleftrightarrow \quad \text{element is degenerate.} \end{aligned}$$

This metric makes it clear that some elements are excluded from the domain. For example, the local area corresponding to the node $(x, y) = (0.5, 0.2)$ in the

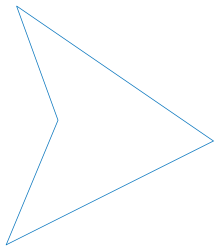


Figure 2.3: Arrow shaped element with nodes in $(0, -1)$, $(2, 0)$, $(0.1, 1.3)$ and $(0.5, 0.2)$.

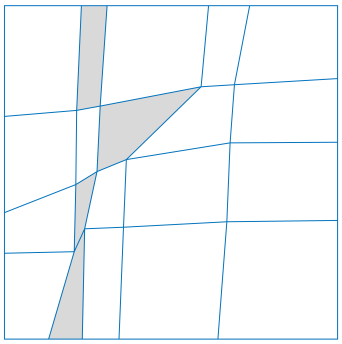


Figure 2.4: Elements with $\mathcal{M}_{Shape} < 0.5$

arrow-shaped element shown in Figure 2.3, is negative and the resulting shape metric would become larger than one, which is outside our domain.

Applying the shape metric to the mesh in Figure 2.2a to see which elements satisfy $\mathcal{M}_{Shape} < 0.5$ and thus can be considered poorly shaped, yields the results in Figure 2.4. The elements with $\mathcal{M}_{Shape} < 0.5$ are shaded.

We see that the shape metric has picked up on some of the same elements as the size metric, but also some that were not detected by the size metric. The large elements to the bottom right are not detected by the shape metric. In fact, the shape metric recognizes these elements as being very good because of their nearly square shape. On the other hand, the shape metric identifies the third element on the third row from the bottom to be rather bad, even though it was not identified as poor by the relative size metric.

Here, the necessity of metrics also become apparent by comparing this element to its neighbor on the right. The two elements may look similar, and one might

conclude that they have similar $\mathcal{M}_{Shape}$ values. However, the rightmost element has significantly better value in the shape metric than the left neighbor element.

2.2.3 Skew Metric

The last metric to be introduced here is the skew metric, $\mathcal{M}_{Skew}$. Although the shape metric detects elements of poor shape deviating from a square shape, it does not necessarily detect distortions arising from small or large angles in an element. The reference element of the skew metric is rectangular [12].

As for the square element, the local area α_i associated with node i equals the total element area independent of i, but contrary to the square, the λ^i's are not all the same, instead they come in fours. That is, $\lambda_{11}^i = \lambda_{11}^{i+2} = \lambda_{22}^{i+1} = \lambda_{22}^{i+3}$ and $\lambda_{22}^i = \lambda_{22}^{i+2} = \lambda_{11}^{i+1} = \lambda_{11}^{i+3}$. Since the λ's are interpreted as edge length squared, it follows that $\lambda_{11}^i \lambda_{22}^i = \alpha_i^2$. This is utilized when the skew metric is formulated. As we are looking for a metric that evaluates the skewness of an element compared with a rectangle, we define the skew metric as

$$\mathcal{M}_{Skew} = \frac{4}{\sum_{i=1}^{4} \left(\sqrt{\lambda_{11}^i \lambda_{22}^i} \right) / \alpha_i}. \tag{2.6}$$

Here, 4 is divided by the sum over all i's of $\sqrt{\lambda_{11}^i \lambda_{22}^i} = \alpha_i$ to get the metric to lie between 0 and 1. The skew metrics' special cases are similar to the special cases of the shape metric,

$$\begin{aligned} \mathcal{M}_{Skew} = 1 \quad &\Longleftrightarrow \quad \text{element is rectangular,} \\ \mathcal{M}_{Skew} = 0 \quad &\Longleftrightarrow \quad \text{element is degenerate.} \end{aligned}$$

The arrow shaped element in Figure 2.3 will also result in a skew metric larger than one, and hence elements of this type will be assigned a skew metric of zero.

Figure 2.5 shows in shading the elements from the mesh in Figure 2.2a identified by the skew metric to satisfy $\mathcal{M}_{Skew} < 0.5$. The figure shows that there are only two elements for which the skew metric is smaller than 0.5. Closer inspection shows that both of these elements consist of highly acute and obtuse angles.

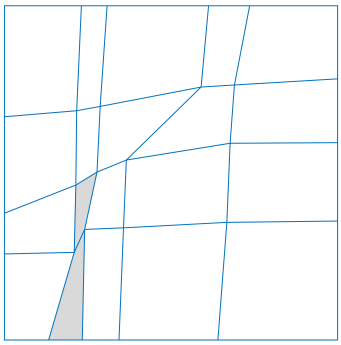


Figure 2.5: Elements with $\mathcal{M}_{Skew} < 0.5$

2.2.4 Combination Metrics

From the previous sections and examples, we have seen that the relative size, shape, and skew metrics identify some of the same elements, but also some different elements. It is possible to have elements that pass the quality threshold for each metric, but appear to possess relatively bad size, shape, and skew at the same time. In order to be able identify these elements we introduce combination metrics, where the relative size metric is combined with the shape metric and the skew metric respectively. This is unproblematic since the metrics (2.3), (2.5) and (2.6) all are dimensionless, node independent, and take on values in the interval $[0, 1]$. They do, however, have slightly different reference element, but this is easily rectified by defining new reference elements for each combination metric.

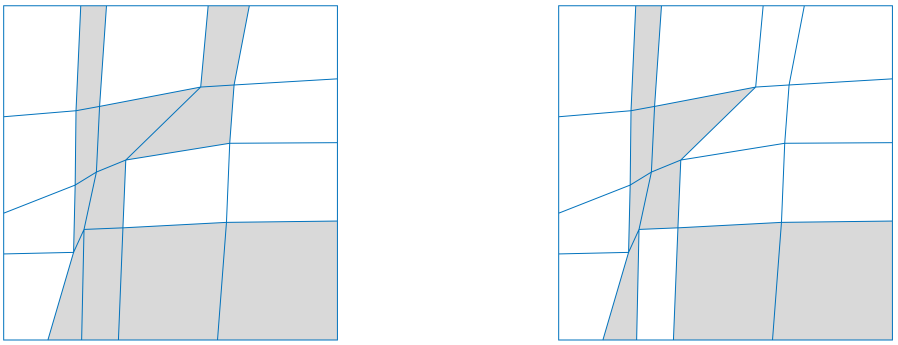
The size-shape metric identifies elements that show tendencies for both bad size and poor shape. With a square reference element with area a, the size-shape metric is the product of the relative size and shape metric,

$$\mathcal{M}_{SizeShape} = \mathcal{M}_{Size} \cdot \mathcal{M}_{Shape}. \tag{2.7}$$

It follows that the essential properties of this metric is a combination of the properties for the size and shape metric separately. Additionally, $\mathcal{M}_{SizeShape} = 1$ if and only if the element is a square with area a and it is zero if and only if the element is degenerate.

Similarly, the the size-skew metric is the product of the relative size metric and the skew metric, and hence it detects elements with both poor size and skewness. The metric,

$$\mathcal{M}_{SizeSkew} = \mathcal{M}_{Size} \cdot \mathcal{M}_{Skew}, \tag{2.8}$$



(a) Elements with $\mathcal{M}_{SizeShape} < 0.5$ (b) Elements with $\mathcal{M}_{SizeSkew} < 0.5$

Figure 2.6: Combination metrics between size-shape and size-skew

has a rectangular reference element with area a. Thus, it has the properties that $\mathcal{M}_{SizeSkew} = 1$ if and only if the element is a rectangle with area a, and $\mathcal{M}_{SizeSkew} = 0$ if and only if the element is degenerate.

Figures 2.6a and 2.6b show the elements identified as having $\mathcal{M}_{SizeShape} < 0.5$ and $\mathcal{M}_{SizeSkew} < 0.5$ respectively. By comparing the results in Figure 2.6a with Figure 2.2b and Figure 2.4 we see that the $\mathcal{M}_{SizeShape}$ metric has detected several elements not detected by either of $\mathcal{M}_{Size}$ or $\mathcal{M}_{Shape}$ alone. Similar observations can be made when comparing Figure 2.6b with Figure 2.2b and Figure 2.4.

2.3 Measure of Mesh Quality

The three metrics, relative size, shape, and skew, and the two combination metrics, size-shape and size-skew, give a measure of the quality of each element in the mesh under evaluation. Based on the local measure of each element, it is now possible to find a global measure for the entire mesh. There are several ways of combining the local element measures to a global mesh measure. Below we present the two global measures for each of the metrics presented in the previous sections that we will be using in this thesis.

2.3.1 Min-Max Measure

Each of the metrics presented gives a measure of element quality to each of the N elements in the mesh. That is, each metric $\mathcal{M}_k$ gives a list of measures, $m_1^k, m_2^k, m_3^k, \ldots, m_N^k$. The first global measure we will be using is the min-max

(MM) measure. It simply takes the smallest m^k and divides by the largest m^k,

$$MM(\mathcal{M}_k) = \frac{m^k_{min}}{m^k_{max}}, \tag{2.9}$$

where $max, min \in [1, 2, 3, \ldots, N]$. The largest value m^k_i can have is one, and the smallest is zero, thus $MM(\mathcal{M}_k) \in [0, 1]$. If $MM(\mathcal{M}_k)$ tends towards zero, we know that there are large variations between the elements of the mesh. If it is identically equal to zero we have at least one degenerate element, since this can only happen if $m^k_{min} = 0$. On the other hand, if the min-max measure is closer to one, we know that all the elements in the mesh are relatively similar in size, shape, or skew, depending on k. Hence, the closer $MM(\mathcal{M}_k)$ is to one, the better the mesh.

2.3.2 Root Mean Square Measure

The second global measure we are going to use is the root mean square (RMS). For each metric, k, the RMS is the square root of the mean of the squares of the values m^k_i,

$$RMS(\mathcal{M}_k) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(m^k_i\right)^2}. \tag{2.10}$$

This measure includes terms from all the mesh elements, not just the best and worst as the min-max measure. From the fact that $m^k_i \in [0, 1]$ for all $i = 1, 2, 3, \ldots N$ and metrics k, it follows that $RMS(\mathcal{M}_k) \in [0, 1]$. This means that a high value of $RMS(\mathcal{M}_k)$ implies a good mesh.

Both global measures are convenient in their own way. The min-max measure gives an impression of how bad the worst mesh elements are compared to the best, but it does not provide much information about the mesh overall. The RMS measure provides information about the mesh as a whole, but does reveal much about the very worst elements. Thus, the two measures complement each other by each compensating for the shortcomings of the other.

Chapter 3

Spline Theory and IGA

Before proceeding, we take some time to introduce splines and isogeometric analysis. IGA was first presented in 2004 by Hughes et al. [3]. It is based on classic Finite Element Analysis, but instead of the linear shape functions used in most Finite Element Methods [1] IGA uses splines [3], [2] as basis functions. Most of what follows in the next few sections of this chapter is part of the work done in the preceding specialization project, with some alterations and additions, and it is included here for completeness.

3.1 B-Splines

Like most splines, B-splines are piecewise polynomial functions defined over connected intervals. They are continuous, differentiable, and since they are polynomials, they inherit all polynomial properties. B-splines are a part of the very core of isogeometric analysis, and it is paramount to understand them thoroughly.

3.1.1 Knot Vectors

A B-spline typically consists of n piecewise polynomial basis functions, with polynomial degree p [14], [15]. A knot vector is used to define the set of basis functions. The knot vector is a set of non-decreasing real values $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, where $\xi_i \in \mathbb{R}$ is the i^{th} knot and p is the polynomial degree. The knot intervals define elements where the basis functions typically are C^∞-continuous [2], [15].

Since the knots need only be non-decreasing, it is possible to have several knots of the same value. If a knot is repeated m times in the knot vector it is said to have multiplicity m. Across a knot with multiplicity m, continuity is reduced to C^{p-m}. Thus, by repeating knots, the continuity of the basis functions can be reduced. When repeating a knot p times the spline will be C^0-continuous at the knot [14].

This property makes it is possible to create sharp corners in the spline curve [15] by controlling the continuity to the associated basis functions.

The knot vectors are often required to repeat the first and last knot values a total of $p+1$ times, making the spline discontinuous at these points. This ensures that the corresponding basis functions are interpolating at these points. When the end knots are repeated $p+1$ times, the knot vector is said to be open. Furthermore, a knot vector is said to be uniform if the knots are equally spaced. Alternatively, if the knots are not uniformly distributed, they give rise to a nonuniform knot vector, and hence nonuniform splines.

For any knot vector, it is only the multiplicity and relative distance between the knots that affect the resulting basis functions. That is, if a new knot vector $\widetilde{\Xi}$ is produced by adding a constant to every knot or scaling by a constant, the new knot vector will produce the same B-splines as the original knot vector Ξ.

3.1.2 B-Splines

The i^{th} B-spline basis function of degree p with knot vector $\Xi = [\xi_1, \ldots, \xi_i, \ldots, \xi_{n+p+1}]$ is defined recursively [15], [14] by

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+1+p} - \xi}{\xi_{i+1+p} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{3.1}$$

for $i = 1, 2, \ldots, n$. In the special case where $p = 0$ the i^{th} spline is simply

$$N_{i,0}(\xi) = \begin{cases} 1, & \text{if } \xi_i \leq \xi < \xi_{i+1} \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

It is important to note that in the case of open knot vectors or repeated knots, one might encounter a denominator evaluating to zero. This problem of possible zero division is resolved by defining that "anything divided by zero is zero" [15], implying that if $\xi_{i+p} - \xi_i = 0$ then $\frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} = 0$ and similarly, if $\xi_{i+p+1} - \xi_{i+1} = 0$ then $\frac{\xi_{i+1+p} - \xi}{\xi_{i+1+p} - \xi_{i+1}} = 0$. Figure 3.1 shows the dependencies between B-splines of different orders. Notice how any B-spline of order p depends on $p+1$ B-splines of order 0.

3.1.3 Example

To better understand the concepts presented thus far, consider the knot vector $\Xi = [0, 0, 0, 0.5, 1, 1, 1]$ for $p = 2$. This knot vector is open, since the knots 0 and 1 are repeated a total of $p + 1 = 3$ times each. Furthermore, the knot vector of 7 knots will give rise to $n = 7 - (p + 1) = 4$ basis functions. As there are no repeated interior knots, i.e. the multiplicity is $m = 1$, the basis functions will be $C^{p-m} = C^1$ continuous across the interior knot at $\xi = 0.5$.

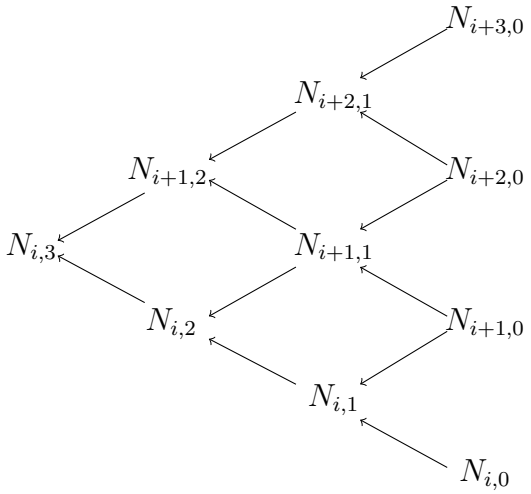


Figure 3.1: Dependencies when computing the B-spline $N_{i,p}$ for $p = 3$ using (3.1).

We will look at the second of these B-splines in more detail, namely $N_{2,2}(\xi)$. First of all, $N_{i,0}$, for $i = 1, 2, \ldots, 6$, is determined by the relation given in (3.2) as

$$\begin{aligned} N_{1,0} &= 0 \\ N_{2,0} &= 0 \\ N_{3,0} &= \begin{cases} 1 & \text{if } 0 \le \xi < 0.5 \\ 0 & \text{otherwise} \end{cases} \\ N_{4,0} &= \begin{cases} 1 & \text{if } 0.5 \le \xi < 1 \\ 0 & \text{otherwise.} \end{cases} \\ N_{5,0} &= 0 \\ N_{6,0} &= 0 \end{aligned} \tag{3.3}$$

Thus, by (3.1) and the zero division convention, the second B-spline of order 2 is

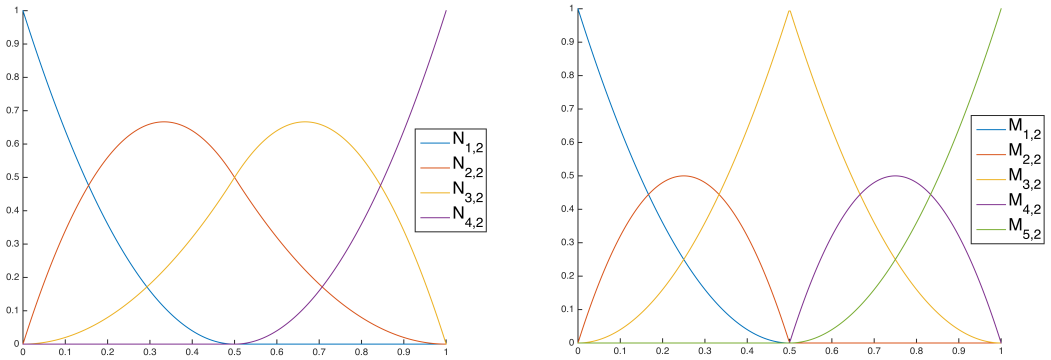
$$\begin{aligned} N_{2,2}(\xi) &= \frac{\xi}{0.5} N_{2,1}(\xi) + \frac{1-\xi}{1} N_{3,1}(\xi) \\ &= 2\xi \left(\frac{\xi}{0} N_{2,0}(\xi) + \frac{0.5-\xi}{0.5} N_{3,0}(\xi) \right) + (1-\xi) \left(\frac{\xi}{0.5} N_{3,0}(\xi) + \frac{1-\xi}{1-0.5} N_{4,0}(\xi) \right) \\ &= 2\xi(2-3\xi) N_{3,0}(\xi) + (1-\xi)(2-2\xi) N_{4,0}(\xi). \end{aligned} \tag{3.4}$$

In a similar fashion we can determine $N_{1,2}, N_{3,2}$ and $N_{4,2}$. By utilizing (3.3), we get

$$\begin{aligned}
N_{1,2}(\xi) &= \begin{cases} (1-2\xi)^2 & \text{if } 0 \leq \xi < 0.5 \\ 0 & \text{if } 0.5 \leq \xi < 1, \end{cases} \\
N_{2,2}(\xi) &= \begin{cases} 2\xi(2-3\xi) & \text{if } 0 \leq \xi < 0.5 \\ (1-\xi)(2-2\xi) & \text{if } 0.5 \leq \xi < 1, \end{cases} \\
N_{3,2}(\xi) &= \begin{cases} 2\xi^2 & \text{if } 0 \leq \xi < 0.5 \\ (2-2\xi)(3\xi-1) & \text{if } 0.5 \leq \xi < 1, \end{cases} \\
N_{4,2}(\xi) &= \begin{cases} 0 & \text{if } 0 \leq \xi < 0.5 \\ (2\xi-1)^2 & \text{if } 0.5 \leq \xi < 1. \end{cases}
\end{aligned} \tag{3.5}$$

Figure 3.2a gives a plot of the four basis functions from (3.5). The basis functions are clearly C^1-continuous across the interior knot.

The effect of adding an extra internal knot at 0.5, and thus elevating the multiplicity of this knot to $m = 2$, is shown in Figure 3.2b. The figure show how the continuity of the B-spline $N_{3,2}$ is reduced to C^0 across the knot. This coincides with what we know about the continuity depending on the order p as well as the multiplicity m of the corresponding knot. Since the new knot vector $\widetilde{\Xi}$ has one more element than Ξ, giving $n_{\widetilde{\Xi}} = n_{\Xi} + 1$, it also induces an additional basis function.



(a) Original Ξ (b) Refined $\widetilde{\Xi}$

Figure 3.2: Plot of basis functions corresponding to knot vector $\Xi = [0, 0, 0, \frac{1}{2}, 1, 1, 1]$, (a), and refined knot vector $\widetilde{\Xi} = [0, 0, 0, \frac{1}{2}, \frac{1}{2}, 1, 1, 1]$, (b), both with $p = 2$.

3.1.4 General Properties

B-splines possess a number of important properties that is frequently exploited in practical applications. If p is a nonnegative polynomial degree and $\Xi = (\xi_j)$ is a nondecreasing knot sequence, then the B-splines defined on Ξ will have the following properties [15], [3]:

Local knots The i^{th} B-spline $N_{i,p}$ depends only on the knots $\xi_i, \xi_{i+1}, \ldots, \xi_{i+p+1}$.

Local support
If $\xi \notin [\xi_i, \xi_{i+p+1})$ then $N_{i,p}(\xi) = 0$. In particular, if $\xi_i = \xi_{i+p+1}$ then $N_{i,p} \equiv 0$. If $x \in [\xi_\mu, \xi_{\mu+1})$ then $N_{i,p}(\xi) = 0$ if $i < \mu - p$ or $i > \mu$.

Positivity If $\xi \in (\xi_i, \xi_{i+p+1})$ then $N_{i,p}(\xi) > 0$. This closed interval $[\xi_i, \xi_{i+p+1}]$ is called the support of $N_{i,p}$.

Piecewise polynomial The B-spline $N_{i,p}$ can be written:

$$N_{i,p}(\xi) = \sum_{k=i}^{i+p} N_{i,p}^k(\xi) N_{k,0}(\xi) \tag{3.6}$$

where each $N_{i,p}^k(\xi)$ is a polynomial of degree p.

Special values If $z = \xi_{i+1} = \cdots = \xi_{i+p} < \xi_{i+p+1}$ then $N_{i,p}(z) = 1$ and $N_{i,p}(z) = 0$ for $i \neq j$.

Smoothness If the number z occurs m times among $\xi_i, \ldots \xi_{i+p+1}$ then the derivatives of $N_{i,p}$ of order $0, 1, \ldots, p - m$ are all continuous at z.

Basis The B-splines are all linearly independent and thus form a basis.

Partition of unity The basis forms a partition of unity, i.e.

$$\sum_{i=1}^{n} N_{i,p}(\xi) = 1, \quad \forall\, p \in \mathbb{Z},\ \xi \in \mathbb{R}. \tag{3.7}$$

Several of these properties are recognized in the example in Section 3.1.3.

3.1.5 Splines as a Basis for Curves

B-spline basis functions may be combined to make curve and surface representations in a physical space. This is achieved by multiplying each B-spline basis with a control point, $\mathbf{B}_i$, to make a linear combination of B-splines. The control points are usually points in $\mathbb{R}, \mathbb{R}^2$ or $\mathbb{R}^3$, implying that if $B_i^{(j)}$ is the j^{th} component of

control point i, then $B_i^{(j)} \in \mathbb{R}$. The mapping from the parameter space of basis functions to the physical space is given by

$$\mathbf{C} = \sum_{i=1}^{n} N_{i,p}(\xi)\, \mathbf{B}_i. \tag{3.8}$$

The linear space of all linear combinations of B-splines defined on a knot vector Ξ is the space

$$\mathcal{S}_{p,\Xi} = span\{N_{1,p}, N_{2,p}, \ldots, N_{n,p}\} = \left\{ \sum_{i=1}^{n} N_{i,p}\, \mathbf{B}_i \;\middle|\; B_i^{(j)} \in \mathbb{R} \right\}. \tag{3.9}$$

It is important to note that the curve does not in general interpolate the control points. It is also noteworthy that the control points and knots in general are uncorrelated. This is an important difference from the FEM. One may think of control point i as the weight put on the i^{th} basis function.

Figures 3.3 and 3.4 show two spline curves and the basis functions they consist of. Note that the control points are unchanged in the two figures. Only the knot vectors, and thus the basis functions, are different. In both cases the control points are given in $\mathbb{R}^2$ as

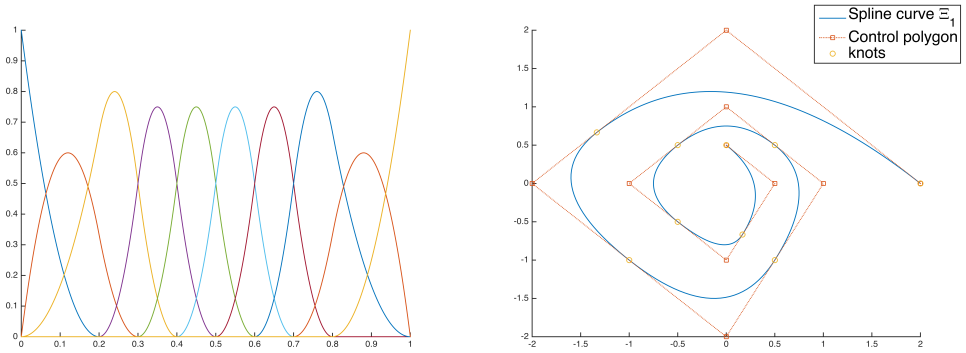
$$\mathbf{B} = \begin{bmatrix} B_x \\ B_y \end{bmatrix} = \begin{bmatrix} 2 & 0 & -2 & 0 & 1 & 0 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & 2 & 0 & -2 & 0 & 1 & 0 & -1 & 0 & \frac{1}{2} \end{bmatrix} \tag{3.10}$$

Since the control points already are points in the physical space, they can be plotted directly, and are shown in Figure 3.3 and 3.4 as red squares. To be able to show the knots in the same space, they need to be mapped to the physical space through (3.8). The knots are shown as yellow circles in the figures.

Figure 3.4 shows how a knot of multiplicity $m = p$ affects the curve. The basis is C^0-continuous and this results in sharp corners in the associated spline curve. The figures also show how the curve not necessarily passes through the control points. In Figure 3.3b the control points are only interpolated at the beginning and end of the curve. This is because the knot vector is open, i.e. $\xi_1 = \xi_2 = \ldots = \xi_{p+1}$ and $\xi_{n+1} = \xi_{n+2} = \ldots = \xi_{n+p+1}$.

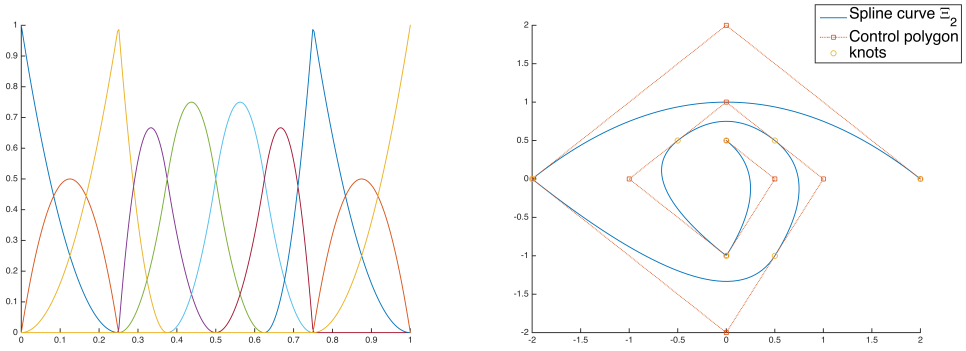
In Figure 3.4b two additional control points are being interpolated. This is due to the repeated knots in the knot vector. Since the knots $\frac{2}{8}$ and $\frac{6}{8}$ are repeated p times the curve has sharp corners. This is also reflected in the basis functions in Figure 3.4a, i.e. the curve interpolates control point i if spline i is C^0-continuous.

Another noteworthy characteristic is that if a curve is drawn based on the ordered set of control points, called the control polygon, then the spline curve $\mathbf{C}$ will always lie entirely within this polygon. In Figures 3.3 and 3.4 the control polygon is shown in red.



(a) Basis functions (b) Spline curve

Figure 3.3: Basis and corresponding curve when the knot vector is $\Xi_1 = [0, 0, 0, \frac{2}{10}, \frac{3}{10}, \frac{4}{10}, \frac{5}{10}, \frac{6}{10}, \frac{7}{10}, \frac{8}{10}, 1, 1, 1]$, polynomial degree is $p = 2$, and control points $\mathbf{B}$ given in (3.10).



(a) Basis functions (b) Spline curve

Figure 3.4: Basis and corresponding curve when the knot vector is $\Xi_2 = [0, 0, 0, \frac{2}{8}, \frac{2}{8}, \frac{3}{8}, \frac{4}{8}, \frac{5}{8}, \frac{6}{8}, \frac{6}{8}, 1, 1, 1]$, polynomial degree is $p = 2$, and control points $\mathbf{B}$ given in (3.10).

It is also worth mentioning that due to the local support property of B-splines, only a small part of the curve, closest to the control point, will change if one control point is changed. This means that it is possible to manipulate the curve locally by changing the position of the control points.

3.1.6 2D Surfaces

Equipped with the properties presented in the previous sections it is now possible to extend to 2D surfaces. The extension is quite straightforward. In addition to the knot vector $\Xi = [\xi_1, \ldots, \xi_{n+p+1}]$ from the 1D case, we now need one more knot vector in the other parametric direction, $\mathcal{H} = [\eta_1, \ldots, \eta_{m+q+1}]$ with associated polynomial order q. The two knot vectors do not need to be equal, they are not even required to have the same polynomial degree, that is, we may have $n \neq m$ or $p \neq q$ or both.

The knot vectors Ξ and $\mathcal{H}$ give rise to a set of linearly independent basis functions, $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$, with $i = 1, \ldots, n$ and $j = 1, \ldots, m$ respectively, given recursively by (3.1). The two-dimensional B-spline basis functions can now be defined as the product of any pair (i, j) of these basis functions [15]. That is, the 2D basis functions are given as $N_{i,p}(\xi)M_{j,q}(\eta)$, where $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Now, the 2D basis functions can be used to represent a surface in the physical space by

$$\mathbf{S}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \mathbf{B}_{ij}, \tag{3.11}$$

where $\mathbf{B}_{ij}$ is a set of control points in $\mathbb{R}^2$ or $\mathbb{R}^3$. The parametric domain defining the surface will be the rectangle $(\xi, \eta) \in [\xi_1, \xi_{n+p+1}] \times [\eta_1, \eta_{m+q+1}]$.

Figure 3.5 shows the nine 2D basis functions when Ξ and $\mathcal{H}$ are identically [0, 0, 0, 1, 1, 1]. We can also recognize many of the spline properties presented in Section 3.1.4 for this case.

It is worth noting that the number of 2D B-splines are still determined by the knot vectors Ξ and $\mathcal{H}$. Each of the knot vectors induce three 1D B-splines, which when combined, generate $3 \cdot 3 = 9$ 2D basis functions.

Figure 3.6 shows an example of a spline representation of the paraboloid $\zeta = \xi^2 + \eta^2$. The control points are visualized in Figure 3.6b. Since the knot vectors used to render the surface in Figure 3.6a have one more internal knot compared to the knot vectors used in Figure 3.5, the surface is made up of $4^2 = 16$ basis functions.

3.1.7 Derivatives

When using B-splines for analysis we will often need the derivatives of the basis functions. The first derivative of the i^{th} B-spline basis function of order p is given by [16], [14], [15]

$$\frac{d}{d\xi} N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi). \tag{3.12}$$

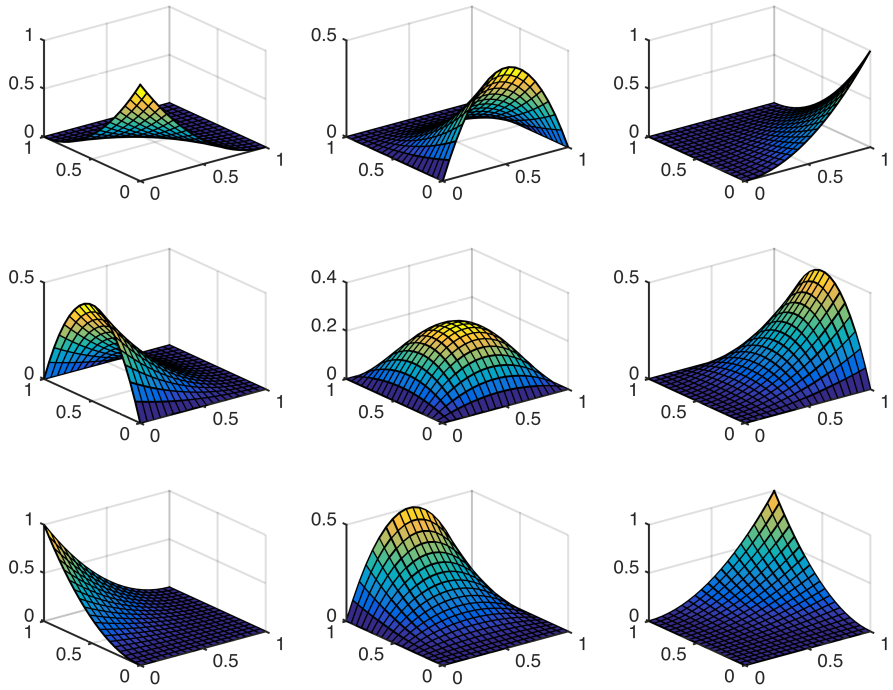
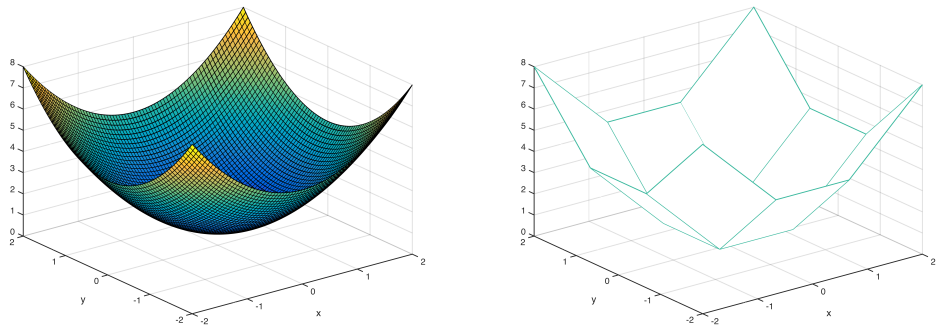


Figure 3.5: The nine 2D basis functions obtained from $\Xi = \mathcal{H} = [0, 0, 0, 1, 1, 1]$ with $p = q = 2$.



(a) 2D surface (b) Control points $\mathbf{B}_{ij}$ in $\mathbb{R}^3$

Figure 3.6: 2D surface with associated control points. Knot vectors $\Xi = \mathcal{H} = [0, 0, 0, \frac{1}{2}, 1, 1, 1]$, and with $p = q = 2$.

Higher order derivatives can be found by implicit differentiation [14],

$$\begin{aligned}\frac{d^k}{d^k\xi}N_{i,p}(\xi) &= \frac{p}{\xi_i+p-\xi_i}\left(\frac{d^{k-1}}{d^{k-1}\xi}N_{i,p-1}(\xi)\right) - \frac{p}{\xi_{i+p+1}-\xi_{i+1}}\left(\frac{d^{k-1}}{d^{k-1}\xi}N_{i+1,p-1}(\xi)\right) \\ &= \frac{p!}{(p-k)!}\sum_{j=0}^{k}\alpha_{k,j}N_{i+j,p-k}(\xi),\end{aligned} \tag{3.13}$$

where

$$\begin{aligned}\alpha_{0,0} &= 1, \\ \alpha_{k,0} &= \frac{\alpha_{k-1,0}}{\xi_{i+p-k+1}-\xi_i}, \\ \alpha_{k,j} &= \frac{\alpha_{k-1,j}-\alpha_{k-1,j-1}}{\xi_{i+p+j-k+1}-\xi_{i+j}} \quad \text{for } j=1,\ldots,k-1, \\ \alpha_{k,k} &= \frac{-\alpha_{k-1,k-1}}{\xi_{i+p+1}-\xi_{i+k}}.\end{aligned} \tag{3.14}$$

3.1.8 Refinement

In the standard finite element method there are two common ways of enriching the basis [2]. The first method, h-refinement, involves increasing the number of elements by decreasing the element size and thus enabling higher resolutions, while the second method, p-refinement, involves increasing the polynomial degree of the basis functions. Isogeometric analysis comprises methods similar to h- and p-refinement, as well as additional possibilities for refinement. Contrary to the FEM methods, the geometry remains unchanged under each refinement and the continuity across each element is more controllable in isogeometric analysis.

The isogeometric version of h-refinement is knot insertion [2]. This is done by inserting new knots in the knot vector. This enriches the basis because the knot vector now gives rise to additional basis functions. The value of the new knots may already be in the knot vector to elevate the multiplicity of an already existing knot, or it may be new, unique knots. Knot insertions are common to do in the interior of the knot vector, keeping the $p+1$ first and last knots unchanged in the enriched knot vector.

After a knot is inserted, the knot vector will render more B-splines and thus it is necessary to increase the number of control points accordingly. Several methods for obtaining new control points exist, see for instance [15], [14].

Figure 3.2a and Figure 3.2b illustrate how the basis is enriched when a knot is inserted and thus elevating the multiplicity of the internal knot to $m = 2 = p$.

The p-refinement equivalent in isogeometric analysis is order elevation, where the polynomial degree of the basis functions is increased from p to $\widetilde{p} > p$. Here

it becomes clear how we can control the continuity; in the classical p-refinement the basis has C^0-continuity. In order elevation the continuity across a knot is changed from C^{p-m} to $C^{\widetilde{p}-m}$. To preserve C^{p-m}-continuity, we simply increase the multiplicity of each knot by $\widetilde{p}-p$.

It is important to note that while knot insertion can be done locally, order elevation is a global operation since the polynomial order of a B-spline is a global property.

It is also possible to perform both knot insertion and order elevation in cascade. The sequence does not in general commute [2], and better results are obtained when performing the order elevation before knot insertion. In IGA this is called k-refinement, and this method does not have any equivalent refinement method in the standard FEM.

3.1.9 Matrix Representation

When dealing with recursively defined mathematical objects we quickly learn that the complexity of such objects rapidly becomes daunting, even if the recurrence relation itself is quite simple [15]. B-splines as defined by (3.1) are recursive, but luckily the splines can be represented as products of some relatively simple matrices.

As before we let $\Xi = [\xi_1, \ldots, \xi_{n+p+1}]$ be a knot vector with polynomial degree p. Furthermore, we let μ be an integer such that $p+1 \leq \mu \leq n$ and $\xi_\mu < \xi_{\mu+1}$. Then, for each positive integer $k \leq p$ we can define the B-spline matrix

$$\mathbf{R}_k(\xi) = \begin{pmatrix} \frac{\xi_{\mu+1}-\xi}{\xi_{\mu+1}-\xi_{\mu+1-k}} & \frac{\xi-\xi_{\mu+1-k}}{\xi_{\mu+1}-\xi_{\mu+1-k}} & 0 & \cdots & 0 \\ 0 & \frac{\xi_{\mu+2}-\xi}{\xi_{\mu+2}-\xi_{\mu+2-k}} & \frac{\xi-\xi_{\mu+2-k}}{\xi_{\mu+2}-\xi_{\mu+2-k}} & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\xi_{\mu+k}-\xi}{\xi_{\mu+k}-\xi_\mu} & \frac{\xi-\xi_\mu}{\xi_{\mu+k}-\xi_\mu} \end{pmatrix}. \tag{3.15}$$

Now, for $\xi \in [\xi_\mu, \xi_{\mu+1})$ and using (3.15) we can write the $p+1$ nonzero B-splines on the interval $[\xi_\mu, \xi_{\mu+1})$ as

$$\mathbf{N}_p = (N_{\mu-p,p}, N_{\mu-p+1,p}, \ldots, N_{\mu,p}) = \mathbf{R}_1(\xi)\mathbf{R}_2(\xi)\cdots\mathbf{R}_p(\xi). \tag{3.16}$$

In a similar fashion, the recursive relation for the derivative of a B-spline from (3.12) and (3.13) can be represented as a matrix product. Several algorithms for evaluating the B-spline matrix and its derivatives have been developed, see [15] for details.

When using B-splines for computations and analysis, we will use the matrix form.

3.2 Isogeometric Analysis

As mentioned briefly in the chapter introduction, splines are heavily incorporated in isogeometric analysis. IGA will be a vital part of our parameterization methods, and we will devote some time to get familiarized with IGA and the IGA problem formulation before presenting the parameterization methods.

Isogeometric analysis is, as the well-known finite element method, a technique for finding numerical approximations to boundary value problems for partial differential equations (PDEs). It is assumed that the reader is already familiar with the classic FEM, using for instance linear or bilinear polynomials as basis functions [1].

3.2.1 Strong Form Poisson Problem

To get better acquainted with isogeometric analysis as a method for numerically solving PDEs, we look at a simple, yet powerful example of how a PDE is actually solved using IGA. We choose the Poisson problem as an example, and it can be stated as follows [17]

$$\nabla^2 u(x,y) + f(x,y) = 0, \tag{3.17}$$

with boundary conditions

$$\begin{aligned} u(x,y) &= g \quad \text{on } \Gamma_D, \\ \nabla u(x,y) \cdot \mathbf{n} &= h \quad \text{on } \Gamma_N, \\ \alpha u + \nabla u(x,y) \cdot \mathbf{n} &= r \quad \text{on } \Gamma_R. \end{aligned} \tag{3.18}$$

The problem is defined on a two dimensional domain Ω with boundary $\partial\Omega = \Gamma_D \cap \Gamma_N \cap \Gamma_R$ and outward normal vector $\mathbf{n}$. The function $f(x,y)$ is given. The boundary is divided into Γ_D, Γ_N and Γ_R representing Dirichlet, Neumann and Robin boundary conditions respectively.

3.2.2 Weak formulation

To solve the strong form (3.17)-(3.18) numerically, we formulate the problem using the Galerkin approach, see [1], [18] [19] or [20] for details. The idea behind this method is to reformulate the strong form of the problem into a weak formulation by multiplying with a spline test function N on both sides and integrating over the domain Ω,

$$\int_\Omega \nabla^2 u \cdot N \, \mathrm{d}A = -\int_\Omega f \cdot N \, \mathrm{d}A. \tag{3.19}$$

Since the spline test function will fulfill $N|_{\Gamma_D} = 0$, integrating by parts yields

$$\int_\Omega \nabla u \cdot \nabla N \, \mathrm{d}A = \int_\Omega f \cdot N \, \mathrm{d}A + \int_{\Gamma_N} h \cdot N \, \mathrm{d}S + \int_{\Gamma_R} r \cdot N \, \mathrm{d}S - \beta \int_{\Gamma_R} u \cdot N \, \mathrm{d}S. \tag{3.20}$$

For simplicity it is assumed we only have Dirichlet boundary conditions, that is, the terms concerning Neumann and Robin boundaries vanish, and we arrive at

$$\int_\Omega \nabla u \cdot \nabla N \, \mathrm{d}A = \int_\Omega f \cdot N \, \mathrm{d}A. \tag{3.21}$$

This expression can be written as

$$a(u, N) = l(N), \tag{3.22}$$

which we recognize as the standard bilinear and linear form from the finite element method [1]. However, contrary to the FEM, isogeometric analysis uses splines as test functions, and hence we are searching for a solution in a subspace $\mathcal{S}_n$ of the space of all splines $\mathcal{S}$ instead of the Sobolev space $H^1(\Omega)$, which is common in FEM.

The subspace we will be searching for a solution in is the space spanned by the finite number of basis functions defined on a knot vector spanning the domain Ω, that is

$$\mathcal{S}_n = span \left\{ N_i \in \mathcal{S}_{p,\Xi} \,\middle|\, \mathbf{C}(\xi, \eta) = \sum_{i=1}^{n} N_i \mathbf{B}_i = \Omega \right\}. \tag{3.23}$$

Once the finite-dimensional approximation space in which we are going to search for the trial solution u and test function N have been chosen, we can proceed with the matrix representation of the weak form (3.22). This procedure is quite familiar for people acquainted with the FEM. We write the solution on the form

$$u_h = \sum_{i=1}^{n} u_h^i N_i, \tag{3.24}$$

which inserted into (3.22) gives

$$\sum_{i=1}^{n} a(N_i, N_j) u_h^i = l(N_j), \quad \forall N_j \in \mathcal{S}_n, \tag{3.25}$$

where we have exploited the bilinearity of (3.21). We immediately recognize (3.25) as a system of linear equations, and thus formulate it as

$$\mathbf{A}\mathbf{u} = \mathbf{f}, \tag{3.26}$$

where the elements of the stiffness matrix $\mathbf{A}$ are $a_{i,j} = a(N_i, N_j)$ and the elements of the force vector $\mathbf{f}$ are $f_j = l(N_j)$.

3.2.3 Elements

In the FEM, nodal functions are used to define the elements on which we operate [1]. When dealing with splines as basis functions, the understanding of elements are not as intuitive. By adopting the convention from [3] we define an element in one dimension as the span between two neighboring knots.

In Section 3.1.4 it was stated that the spline basis functions are defined on at most $p+1$ knot spans. This property of local support of the basis functions ensures that the stiffness matrix has a sparse nature. It is also worth noting that because a basis function can be defined on as many as $p+1$ knot spans, the basis functions can be contributing on several elements.

As briefly mentioned in Section 3.1, the continuity across elements is much easier to control when using splines compared to the standard basis functions used in FEM, where we typically have C^0 continuity across elements. As discussed in Section 3.1.1, the continuity across elements depends on the multiplicity of the corresponding knots when using spline basis functions. This essentially means that we are free to choose the continuity as we please.

3.2.4 Spaces and Mappings

The stiffness matrix $\mathbf{A}$ and the force vector $\mathbf{f}$ in the weak problem formulation (3.26) require integration. In this thesis we use Gaussian quadrature to evaluate these integrals. When using Gaussian quadrature in two dimensions we will need to map the functions being integrated to a unit square $(\widetilde{\xi}, \widetilde{\eta}) \in [-1,1]\times[-1,1]$. This is familiar from the FEM and will be explained in more detail in Section 3.2.5.

The unit square mapping comes in addition to the mapping from the parameter space to the physical space which was presented in (3.8) in Section 3.1.5.

Figure 3.7 shows the three spaces we will be working in together with the corresponding mappings between them. The mapping from the unit square to the parametric element $\widetilde{C} : \widetilde{\Omega}_K \mapsto \widehat{\Omega}_K$ is an affine mapping, that is, the mapping preserves points and lines, while the mapping $C : \widehat{\Omega}_K \mapsto \Omega_K$ is a polynomial mapping from the parametric element to the physical element.

It is important to be well acquainted with these spaces and mappings, as we will be working in all of them at the same time. The numerical integration will be done on the unit square $\widetilde{\Omega}$, all basis functions are defined over the parametric domain $\widehat{\Omega}$, and the differential equation is formulated and evaluated in the physical domain Ω. As the basis functions and the differentials are not defined in the same space, we need to formulate the equations using the Jacobian matrix of the geometry mapping between the parameter and physical space. In two dimensions this is

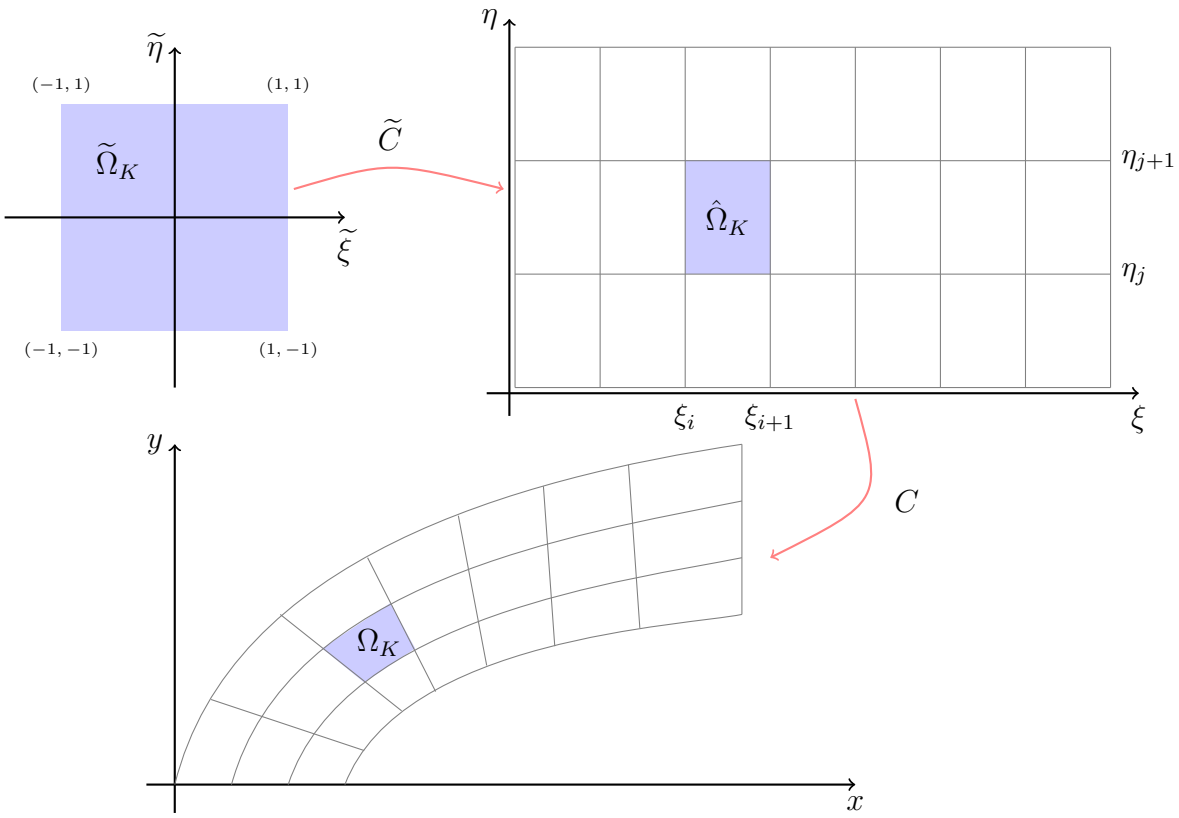


Figure 3.7: Illustration of the unit square, or parent element, $\widetilde{\Omega}$, the parametric space $\widehat{\Omega}$ and the physical space Ω, with the mappings $\widetilde{C} : \widetilde{\Omega} \mapsto \widehat{\Omega}$ and $C : \widehat{\Omega} \mapsto \Omega$.

stated as

$$\mathbf{C}(x, y) = \sum_{i=1}^{n} \mathbf{N}_i(\xi, \eta)\mathbf{B}_i. \tag{3.27}$$

We will also need the inverse of the Jacobian of this mapping before we can compute the spatial derivatives of the basis functions. The Jacobian and its inverse are defined as

$$J_\xi = \begin{bmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{bmatrix}, \quad J_\xi^{-1} = \begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix}. \tag{3.28}$$

with components $\frac{\partial x_I}{\partial \xi_J} = \frac{\partial N_i}{\partial \xi_J} B_{i,I}$, where $B_{i,I}$ is the I^{th} coordinate of control point i [16]. The Jacobian is defined as the determinant of the Jacobian matrix, $|J_\xi|$. The Jacobian represents the transformation from the physical space to the parameter space, and consequently the inverse represents the transformation from the parameter space back to the physical space.

The second mapping, from the unit square, $(\widetilde{\xi}, \widetilde{\eta}) \in \widetilde{\Omega}$, to a parametric element

domain, $(\xi, \eta) \in [\xi_i, \xi_{i+1}] \times [\eta_j, \eta_{j+1}]$, is easier to deal with because the mapping is affine. The transformation is given by

$$\begin{aligned} \xi &= \frac{1}{2}\big((\xi_{i+1} - \xi_i)\widetilde{\xi} + (\xi_{i+1} + \xi_i)\big) \\ \eta &= \frac{1}{2}\big((\eta_{j+1} - \eta_j)\widetilde{\eta} + (\eta_{j+1} + \eta_j)\big), \end{aligned} \tag{3.29}$$

and the Jacobian of this mapping becomes

$$|J_{\widetilde{\xi}}| = \frac{1}{4}(\xi_{i+1} - \xi_i)(\eta_{j+1} - \eta_j). \tag{3.30}$$

Now that we are equipped with three different spaces with appropriate mappings and Jacobians, we are ready to take on the numerical integration of the elements in the stiffness matrix and force vector. But first, let us see how the integral of a general function of two variables $g(x, y)$ taken over the entire domain Ω may be written when working with the spaces and mappings presented here,

$$\begin{aligned} \int_\Omega g(x,y)\,\mathrm{d}\Omega &= \sum_K \int_{\Omega_K} g(x,y)\,\mathrm{d}\Omega_K \\ &= \sum_K \int_{\widehat{\Omega}_K} g(x(\xi), y(\eta))\,|J_\eta|\,\mathrm{d}\widehat{\Omega}_K \\ &= \sum_K \int_{\widetilde{\Omega}_K} g(\widetilde{\xi}, \widetilde{\eta})\,|J_\xi|\,|J_{\widetilde{\xi}}|\,\mathrm{d}\,\widetilde{\Omega}_K. \end{aligned} \tag{3.31}$$

In addition to this, we need to be precise when applying the gradient in (3.25). The gradient ∇ is applied with respect to x and y, but the objects it is applied to, the spline basis functions N_i's, are expressed in terms of ξ and η. It is intuitive, and quite easy to check, that the relation

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \begin{bmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = J_\xi^{-T}\,\widehat{\nabla} \tag{3.32}$$

holds. Now, applying this to the bilinear form in (3.25) yields

$$a(N_i, N_j) = \int_\Omega (\nabla N_i)^T (\nabla N_j)\mathrm{d}x\mathrm{d}y \tag{3.33}$$

$$= \int_\Omega (J_\xi^{-T}\widehat{\nabla} N_i)^T (J_\xi^{-T}\widehat{\nabla} N_j)\mathrm{d}x\mathrm{d}y \tag{3.34}$$

$$= \int_{\widehat{\Omega}} (\widehat{\nabla} N_i)^T J_\xi^{-1} J_\xi^{-T} (\widehat{\nabla} N_j)\,|J_\xi|\,\mathrm{d}\xi\mathrm{d}\eta \tag{3.35}$$

$$= \int_{\widetilde{\Omega}} (\widehat{\nabla} N_i)^T J_\xi^{-1} J_\xi^{-T} (\widehat{\nabla} N_j)\,|J_\xi|\,|J_{\widetilde{\xi}}|\,\mathrm{d}\widetilde{\xi}\mathrm{d}\widetilde{\eta}. \tag{3.36}$$

The same process is applied to $l(N_j)$ in (3.25) to obtain

$$\begin{aligned} l(N_j) &= \int_{\Omega} f \cdot N_j \,\mathrm{d}x\mathrm{d}y \\ &= \int_{\widehat{\Omega}} f \cdot N_j \,|J_\xi| \,\mathrm{d}\xi\mathrm{d}\eta \\ &= \int_{\widetilde{\Omega}} f \cdot N_j \,|J_\xi| \,|J_{\widetilde{\xi}}| \,\mathrm{d}\widetilde{\xi}\mathrm{d}\widetilde{\eta}. \end{aligned} \tag{3.37}$$

3.2.5 Numerical Integration

As mentioned in the previous section, we need to perform numerical integration in order to evaluate the integrals that are present in the stiffness matrix and force vector. There exists a number of methods for numerical integration [17], [20], [21] We have chosen to use Gaussian quadrature in this thesis. The idea behind Gaussian quadrature in two dimensions is to approximate an integral over the domain $[-1,\,1] \times [-1,\,1]$ with finite sums on the form

$$\int_{-1}^{1}\int_{-1}^{1} g(x,y)\,\mathrm{d}x\mathrm{d}y \approx \sum_{\alpha=1}^{c_1}\sum_{\beta=1}^{c_2} g(\bar{x}_\alpha, \bar{y}_\beta) w_{\alpha,\beta}. \tag{3.38}$$

The Gauss points $(\bar{x}_\alpha, \bar{y}_\beta)$ and the weights $w_{\alpha,\beta}$ can be found in [22] and [23]. Naturally, it is possible to do the integration over a different domain, say $[a,b] \times [c,d]$, but for simplicity and generality, we use the unit square here. If we choose another domain, the mapping $\widetilde{C}$ in Section 3.2.4 will also change. In (3.38) c_1 and c_2 represent the number of Gauss integration points in each spatial direction.

In order for the approximation (3.38) to be reliable, the function g needs to be sufficiently smooth. Generally speaking, a C^{2c} continuous function needs c quadrature points in the approximation. When using splines as integrands, this might give rise to issues since the continuity of splines might be limited across knot spans, cf. Section 3.1.1. Luckily, we avoid the potential issue by performing integration element-wise as each element is defined as a knot span, on which the basis functions typically are C^∞-continuous.

Applying Gauss quadrature for numerical integration means that the exact integral given in (3.36) is approximated by

$$a(N_i, N_j) \approx \sum_{\alpha}\sum_{\beta} \left(\widehat{\nabla} N_i(\xi_\alpha, \eta_\beta)\right)^T J_\xi^{-1} J_\xi^{-T} \left(\widehat{\nabla} N_j(\xi_\alpha, \eta_\beta)\right) \,|J_\xi| \,|J_{\widetilde{\xi}}| \, w_{\alpha,\beta} \tag{3.39}$$

The exact same procedure is used to approximate (3.37).

3.2.6 Imposing Boundary Conditions

We have so far described what is a powerful method for solving boundary value problems such as (3.17). However, the enforcement of the boundary conditions needs some special handling when using IGA compared with the FEM. Many different approaches to the enforcement of boundary conditions have been suggested, such as least squares, Lagrange multipliers, and penalty methods. The reader is referred to [24], [25], [3], [2], and [16] for further details.

In this thesis we will only be considering Dirichlet boundary conditions and the easiest way to impose this type of boundary condition is to apply them to the control variables corresponding to the boundary.

Homogeneous Dirichlet boundary conditions, $u = 0$ on Γ_D, are imposed by finding the basis functions located at the boundary and modify the columns and rows corresponding to these basis functions in the stiffness matrix and force vector for these basis variables to be set to zero. In the case of homogeneous Dirichlet conditions this approach results in exact pointwise satisfaction [3]. When dealing with inhomogeneous Dirichlet boundary conditions, $u = g$ on Γ_D, this method will not be accurate, and it is necessary to approximate the boundary values with splines [3].

Here, we have chosen to go for a least squares approximation approach in which we seek to minimize the error at the boundary data points. The basic idea of the least squares method is to find the parameters of the boundary control points that minimize $\|\mathbf{G}\mathbf{c}^* - \mathbf{b}\|^2$. The matrix $\mathbf{G}$ and vector $\mathbf{b}$ have components $g_{i,j} = N_j(x_i)$ and y_i, respectively, where (x_i, y_i) are data points along the boundary. Solving the minimization problem for $\mathbf{c}^*$ is equivalent to solving the set of equations [16]

$$\mathbf{G}^T\mathbf{G}\mathbf{c}^* = \mathbf{G}^T\mathbf{b}. \tag{3.40}$$

This method is limited to sufficiently smooth boundary conditions and it will in general only be interpolatory at the endpoints.

We will only deal with Dirichlet conditions when IGA is applied to the domain parameterization problem. Therefore, we will only briefly mention that Neumann conditions are satisfied in the same way as in the standard FEM. That is, by including the terms which arise naturally from the variational problem formulation (3.20), see [2] and [3] for details.

3.3 Validating a B-spline Parameterization

When B-splines are used to find a parameterization of a domain Ω it is necessary to have a way to validate the parameterization. In a parameterization mesh, all elements must be non-inverted. If some elements are inverted, the mesh is not

suitable for computational analysis. It turns out that a parameterization

$$\mathbf{F}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} \mathbf{u}_{i,j} N_{i,p}(\xi) M_{j,q}(\eta) \tag{3.41}$$

of a domain $\Omega = \{\mathbf{x} \in \mathbb{R}^2 | \, \mathbf{x} = \mathbf{F}(\xi,\eta)\}$ is valid if the Jacobian of $\mathbf{F}$ has a positive determinant [8], [7]. If the determinant of the Jacobian is positive, we also know that the elements in question is non-inverted. The inner control points are here denoted by $\mathbf{u} = [u^x, u^y]$. The determinant of the Jacobian $\mathbf{J}$ can be written as

$$\det(\mathbf{J}) = \sum_{i,j}^{n,m}\sum_{k,l}^{n,m} \det[\mathbf{u}_{i,j}\mathbf{u}_{k,l}]^T \frac{\mathrm{d}N_{i,p}(\xi)}{\mathrm{d}\xi} M_{j,q}(\eta) N_{k,p}(\xi) \frac{\mathrm{d}M_{l,q}(\eta)}{\mathrm{d}\eta} \tag{3.42}$$

It is possible to determine if the determinant is positive by simply evaluating (3.42) at numerous points (ξ,η), provided we evaluate at enough points. This way of evaluating the sign of the determinant is not necessarily sufficient, as we may be evaluating at too few or at the wrong points, and thus missing some areas where $\det(\mathbf{J}) < 0$. However, it is sufficient for our needs as we will be evaluating at numerous points and evaluating the results in combination with the parameterization.

Chapter 4

Parameterization Methods

It is now time to introduce the methods we will be using to parameterize a given physical domain Ω. The parameterization will be given in terms of points as $\mathbf{u} = [u_x, u_y]^T$. Here u_x is the x-coordinate and u_y is the y-coordinate of the parameterization points.

Four methods will be presented. All the methods are based on the framework presented in the preceding chapter and utilizes the spline infrastructure and IGA approach. The methods will be presented in order of increasing complexity.

4.1 Gordon-Hall Algorithm

One of the first methods for mesh generation intended for use in finite element analysis was proposed by W. J. Gordon and C. A. Hall in 1973 [26]. The method is a form of bilinear blending, and is today known as the Gordon-Hall algorithm [27], transfinite interpolation, or Coons patch.

We will first present the general technique, before introducing splines into it. We start by considering the mapping $\mathcal{F} : \hat{\Omega} \mapsto \Omega$ from the parameter domain to the physical domain, see Figure 4.1. The mapping $\mathcal{F}$ is a continuous, vector-valued function taking the independent variables ξ and η as arguments, so that $\mathcal{F}$ maps the parametric boundary $\partial\hat{\Omega}$ into the physical boundary $\partial\Omega$ [27]. In order to generate a parameterization of Ω we need to use $\mathcal{F}$. However, the mapping is unknown to us, we only know the boundary $\partial\Omega$.

Gordon and Hall found a method of obtaining the mapping $\mathcal{F}$. The first step towards finding $\mathcal{F}$ is representing the domain boundary as curves,

$$\begin{aligned} \partial\Omega_1 &: \underline{x}(\xi, -1) \qquad \partial\Omega_2 : \underline{x}(1, \eta) \\ \partial\Omega_3 &: \underline{x}(\xi, 1) \qquad \partial\Omega_4 : \underline{x}(-1, \eta). \end{aligned} \tag{4.1}$$

We may now choose any point (ξ, η) along the boundary $\partial\hat{\Omega}$ and get a correspond-

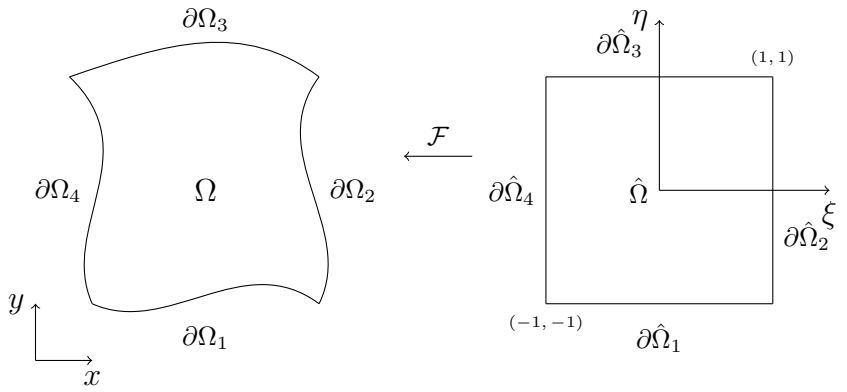


Figure 4.1: Physical domain Ω defined by the four curves $\partial\Omega_1$, $\partial\Omega_2$, $\partial\Omega_3$ and $\partial\Omega_4$, parametric space $\hat{\Omega}$, and the mapping $\mathcal{F}$.

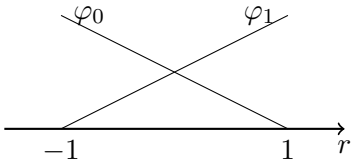


Figure 4.2: Linear polynomials φ_0 and φ_1 for linear interpolation.

ing point (x, y) on $\partial\Omega$. However, since there exists many mappings from $\partial\hat{\Omega}$ to $\partial\Omega$ with different mappings of the interior, this is not enough to uniquely define the mapping $\mathcal{F}$. To determine $\mathcal{F}$ we use linear interpolation with linear polynomials satisfying

$$\begin{aligned} \varphi_i(r) &\in \mathbb{P}_1(-1,1), \qquad i = 0,1, \\ \varphi_i(r_j) &= \delta_{ij}, \qquad 0 \le i,j \le 1, \end{aligned} \tag{4.2}$$

where $r_0 = -1$ and $r_1 = 1$. See Figure 4.2 for a simple illustration of the conditions (4.2). The polynomials $\varphi_0(r) = \frac{r-r_1}{r_0-r_1} = \frac{1-r}{2}$ and $\varphi_1(r) = \frac{r-r_0}{r_1-r_0} = \frac{1+r}{2}$ satisfy the conditions (4.2). Then, linear interpolation may be applied to obtain [17]

$$g(r) = \varphi_0(r)g_0 + \varphi_1(r)g_1, \tag{4.3}$$

where g_0 and g_1 are known values of the function at points r_0 and r_1 respectively. By setting these points to be coordinates along opposite edges of Ω, we can use linear interpolation once more to interpolate between the opposite sides $(\partial\Omega_1, \partial\Omega_3)$ and $(\partial\Omega_2, \partial\Omega_4)$. That is, the mapping

$$\mathcal{F}_\xi(\xi,\eta) \equiv \varphi_0(\xi)\,\underline{x}(-1,\eta) + \varphi_1(\xi)\,\underline{x}(1,\eta) \tag{4.4}$$

exactly preserves the edges $\partial\Omega_2$ and $\partial\Omega_4$, while the mapping

$$\mathcal{F}_\eta(\xi,\eta) \equiv \varphi_0(\eta)\,\underline{x}(\xi,-1) + \varphi_1(\eta)\,\underline{x}(\xi,1) \tag{4.5}$$

exactly preserves $\partial\Omega_1$ and $\partial\Omega_3$. Now all the boundary curves are preserved, but we also need to preserve the corner points of Ω. This is obtained through

$$\mathcal{F}_{\xi\eta}(\xi,\eta) = \sum_{i=0}^{1}\sum_{j=0}^{1} \varphi_i(\xi)\varphi_j(\eta)\,\underline{x}(r_i, r_j). \tag{4.6}$$

Combining the mappings (4.4), (4.5) and (4.6) yields the mapping $\mathcal{F}$ from the parameter space to the physical space,

$$\mathcal{F} \equiv \mathcal{F}_\eta + \mathcal{F}_\xi - \mathcal{F}_{\xi\eta}. \tag{4.7}$$

This mapping will map $\partial\hat{\Omega}$ exactly to $\partial\Omega$, and the interior points of Ω will be defined by an affine mapping [27].

Now, we introduce splines as basis functions in the expressions for $\mathcal{F}_\xi$, $\mathcal{F}_\eta$, and $\mathcal{F}_{\xi\eta}$. The linear polynomials φ_i are replaced by linear splines defined on the knot vector [0, 0, 1, 1] with polynomial degree $q = 1$. We use the mapping from the parametric space to the physical space given by equation (3.8) with n splines defined from a knot vector with polynomial degree p to represent the domain boundary curves. Thus, $\mathcal{F}_\xi$, $\mathcal{F}_\eta$, and $\mathcal{F}_{\xi\eta}$ can be expressed as

$$\begin{aligned}
\mathcal{F}_\eta &= \sum_{i=1}^{n}\sum_{j=1}^{2} N_i^p(\xi)\widetilde{M}_j^{q=1}(\eta)\,\widetilde{\mathbf{x}}_{ij}^{\eta} \\
\mathcal{F}_\xi &= \sum_{i=1}^{2}\sum_{j=1}^{n} \widetilde{M}_i^{q=1}(\xi)N_j^p(\eta)\,\widetilde{\mathbf{x}}_{ij}^{\xi} \\
\mathcal{F}_{\xi\eta} &= \sum_{i=1}^{2}\sum_{j=1}^{2} \widetilde{N}_i^{q=1}(\xi)\widetilde{M}_j^{q=1}(\eta)\,\widetilde{\mathbf{x}}_{ij}^{\xi\eta}
\end{aligned} \tag{4.8}$$

where $\widetilde{\mathbf{x}}_{ij}^{\eta}$ are points along Ω_1 and Ω_3, $\widetilde{\mathbf{x}}_{ij}^{\xi}$ are points along Ω_2 and Ω_4, and $\widetilde{\mathbf{x}}_{ij}^{\xi\eta}$ are the points where the four boundary curves meet.

Next, we wish to get all the the mappings on the same basis. This is done in several steps: first the linear basis undergoes order elevation and then knots are inserted into the knot vector to enrich the basis, that is, the linear basis undergoes

k-refinement. Then, the mapping $\mathcal{F}_\eta$ from (4.8) becomes

$$\mathcal{F}_\eta = \sum_{i=1}^{n}\sum_{j=1}^{2} N_i^p(\xi)\widetilde{M}_j^{q=1}(\eta)\,\widetilde{\mathbf{x}}_{ij}^{\eta} \tag{4.9}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{p+1} N_i^p(\xi)\widehat{M}_j^{p}(\eta)\,\widehat{\mathbf{x}}_{ij}^{\eta} \tag{4.10}$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} N_i^p(\xi)M_j^{p}(\eta)\,\mathbf{x}_{ij}^{\eta}. \tag{4.11}$$

The step from (4.9) to (4.10) involves order elevating the knot vector used to define the splines $\widetilde{M}_j^{q=1}(\eta)$. In the process the control points are also altered, so that we end up with more control points $\widehat{\mathbf{x}}_{ij}^{\eta}$. Now the basis $N(\xi)$ and $\widehat{M}(\eta)$ have the same polynomial degree, but the basis in the ξ-direction is still richer than in the η-direction.

This is fixed in the next step, from (4.10) to (4.11), by inserting knots into the knot vector giving rise to the basis in the η-direction. Once again, the control points are altered in the process to fit the new basis.

The exact same procedure is applied to $\mathcal{F}_\xi$ and $\mathcal{F}_{\xi\eta}$, but in the latter case, the process is repeated twice, once for $\widetilde{M}_i^{q=1}(\xi)$ and once for $\widetilde{N}_j^{q=1}(\eta)$.

Since we are free to choose the polynomial order and number of basis functions as we please, we chose the same p and n in all the mappings. Hence, when the mappings are added together, the control points can be added separately, i.e.

$$\begin{aligned}\mathcal{F} =& \sum_{i=1}^{n}\sum_{j=1}^{n} N_i^p(\xi)N_j^p(\eta)\,\mathbf{x}_{ij}^{\eta} + \sum_{i=1}^{n}\sum_{j=1}^{n} N_i^p(\xi)N_j^p(\eta)\,\mathbf{x}_{ij}^{\xi} \\ &- \sum_{i=1}^{n}\sum_{j=1}^{n} N_i^p(\xi)N_j^p(\eta)\,\mathbf{x}_{ij}^{\xi\eta} \\ =& \sum_{i=1}^{n}\sum_{j=1}^{n} N_i^p(\xi)N_j^p(\eta)\,\left(\mathbf{x}_{ij}^{\eta} + \mathbf{x}_{ij}^{\xi} - \mathbf{x}_{ij}^{\xi\eta}\right).\end{aligned} \tag{4.12}$$

The domain parameterization $\mathbf{u}$ is given by the mapping $\mathcal{F}$. The code implementation of this domain parameterization method can be found in Appendix A.1.

4.2 Uncoupled Poisson using IGA

The next method to be introduced involves the solving of an uncoupled Poisson problem for the parameterization $\mathbf{u}$. The Poisson problem will be solved by the isogeometric analysis approach presented in Section 3.2.

The idea behind this method is to solve two uncoupled Poisson problems, one for the x-coordinate and one for the y-coordinate of the parameterization points. That is, the method yield a parameterization mesh with coordinates given as $\mathbf{u} = [u_x, u_y]$, where u_x is obtained by solving one Poisson problem, and u_y is obtained by solving another Poisson problem. The two Poisson problems are solved independently from each other. The boundary conditions are set to be the parametric representation of the domain boundary. This is formulated mathematically as

$$\begin{aligned} \nabla^2 u_x &= 0 \qquad \text{in } \Omega \\ u_x &= \hat{x} \qquad \text{on } \partial\Omega \end{aligned} \tag{X'}$$

and

$$\begin{aligned} \nabla^2 u_y &= 0 \qquad \text{in } \Omega \\ u_y &= \hat{y} \qquad \text{on } \partial\Omega \end{aligned} \tag{Y'}$$

where $\hat{x}$ and $\hat{y}$ are known values for u_x and u_y along the boundary $\partial\Omega$. To prepare the problems (X') and (Y') for isogeometric analysis we rewrite them into the weak form, cf. 3.2.2.Thus, to find the parameterization $\mathbf{u} = [u_x, u_y]$ of a domain Ω with boundary $\partial\Omega$, the two problems

$$\mathbf{A}\mathbf{u}_x = \mathbf{f}_x, \tag{X}$$
$$\mathbf{A}\mathbf{u}_y = \mathbf{f}_y \tag{Y}$$

are solved separately. The elements of the stiffness matrix $\mathbf{A}$, are as in Section 3.2 $a_{i,j} = a(N_i, N_j)$. The elements of the right hand side vectors $\mathbf{f}_x$ and $\mathbf{f}_y$ are $(f_x)_j = l_x(N_j)$ and $(f_y)_j = l_y(N_j)$, where $l_i(\cdot)$, $i = \{x, y\}$ are given by (3.21). However, since the right hand side in both (X') and (Y') are zero, $\mathbf{f}_x = \mathbf{0}$ and $\mathbf{f}_y = \mathbf{0}$.

4.2.1 Implementation and Verification

The method is implemented in MATLAB. When performing finite element analysis and, by extension, isogeometric analysis, the procedure can generally be divided into three parts: the pre-processing, the processing, and the post-processing.

The pre-processing initializes the problem with the right hand side function f, knot vectors Ξ and $\mathcal{H}$, control points, boundary conditions, refinement and any other necessary features. The connectivity between the basis functions on each element is important for the assembly of the stiffness matrix, so they are also included in the pre-processing.

The processing part of the program is where the system is assembled. The stiffness matrix $\mathbf{A}$ and the right hand side $\mathbf{f}$ are assembled by looping over each element, and adding the contribution from all the nonzero basis functions on each element. After the assembly of the system, the boundary conditions are enforced

through the least squares approach, before the system $\mathbf{AU} = \mathbf{f}$ is solved using MATLAB's solver for systems of linear equations. The solution $\mathbf{U}$ is the value of the control variables.

In the post-processing, the control variables are used to find the solution $\mathbf{u}$ in the physical space. This whole procedure has to be carried out twice in order to solve the problems (X') and (Y').

An outline for this may be seen in Algorithm 4.1. The `main` solver function consists of a number of other functions running in sequence, each contributing to setting up and solving the Poisson problem. The main sequence is explained schematically in Appendix A.2 together with the code in its entirety.

Before the program may be used to parameterize an arbitrary domain, it must be verified. That is, we have to confirm that the program produces the right output. This is done by applying the solver to a problem with a known solution. The inhomogeneous Poisson problem

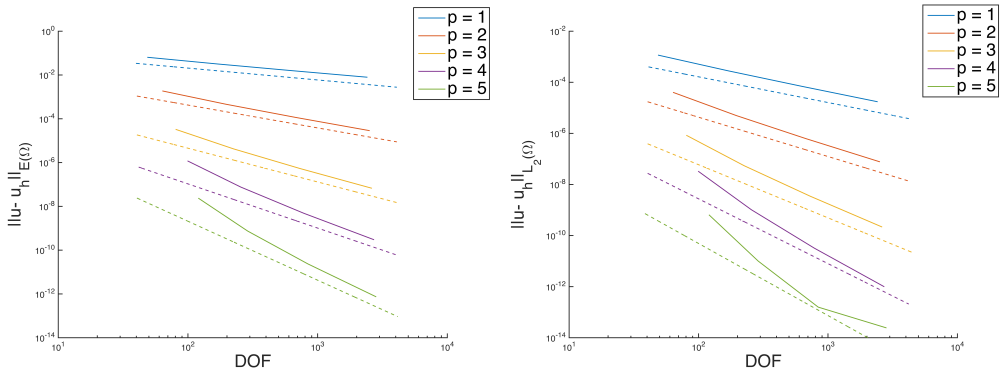
$$\begin{aligned} \nabla^2 u(x,y) &= 0 && \text{in } \Omega \\ u(x,y) &= e^x \sin(y) && \text{on } \partial\Omega \end{aligned} \tag{4.13}$$

on the square $\Omega = [0,\, 1] \times [0,\, 1]$ has exact solution $u = e^x \sin(y)$.

In order to evaluate the performance of the program, it is run several times on the problem in (4.13) with different values for the polynomial order p and number of degrees of freedom (DOFs). For each run, the error e between the exact solution u and the numerical solution u_h is evaluated in the energy norm and in the L_2 norm.

Algorithm 4.1 Progam outline for the uncoupled Poisson parameterization method. The `Main`-sequence can be found in Appendix A.2

```
1:  procedure UncoupledPoisson
2:      initialize f = [0, 0], p, q, Ξ, H
3:  Solve for x-coordinate:
4:      initialize x-parameterization of boundary ∂Ω
5:      main
6:      u_x ← u
7:  Solve for y-coordinate:
8:      initialize y-parameterization of boundary ∂Ω
9:      main
10:     u_y ← u
11: Parameterization u = [u_x, u_y]
```



(a) Convergence rates in energy norm (b) Convergence rates in L_2 norm

Figure 4.3: Convergence rates for the Poisson solver. Error as a function of degrees of freedom for order $p = 1, 2, 3, 4$ and 5 in the energy and L_2 norm.

The energy norm for the Poisson problem is defined as

$$||e||_{E(\Omega)} = \left(\int_{\Omega} (\nabla e)^T (\nabla e) \, \mathrm{d}\Omega \right)^{1/2} \tag{4.14}$$

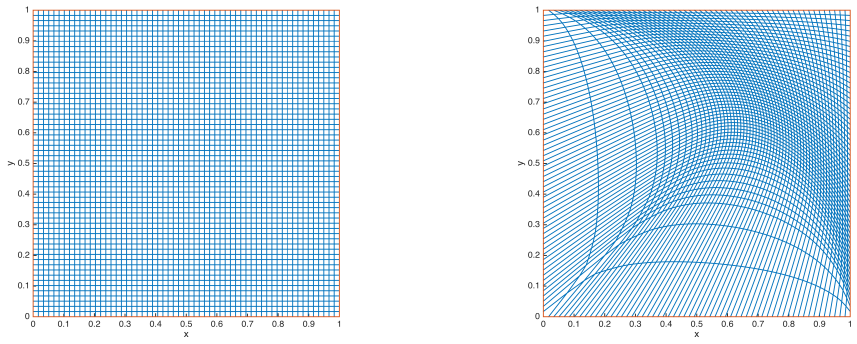
while the L_2 norm is

$$||e||_{L_2(\Omega)} = \left(\int_{\Omega} e^T e \, \mathrm{d}\Omega \right)^{1/2}. \tag{4.15}$$

Figure 4.3 shows how the solution converges as a function of DOFs with the error both in the energy norm and in the L_2 norm. Under each convergence plot is a dashed line. The dashed lines in Figure 4.3a have a slope of $-p/2$, while the dashed lines in Figure 4.3b have a slope of $-(p+1)/2$. That is, the rate of convergence is $p/2$ in the energy norm and $(p+1)/2$ in the L_2 norm. In Figure 4.3b, the lower part of the convergence rate for $p = 5$ no longer follows the dashed line as closely. This is because the error has reached machine precision. The convergence rates are as expected [3], and verifies that the solver works correctly.

4.2.2 Gamma Function

It turns out that the uncoupled Poisson approach has a recurrent problem with producing invalid parameterizations in which parameterization points are placed outside of the domain, leading to mesh lines crossing the domain boundary and inverted elements. In an attempt to remedy this shortcoming, we introduce a function γ, inspired by the elasticity matrix from linear elasticity theory which will be introduced more thoroughly in Section 4.3. The function is incorporated



(a) $\gamma = 1$ (b) $\gamma(x, y) = x \cdot y$

Figure 4.4: Effect of introducing $\gamma(x, y)$ into the uncoupled Poisson method.

in the stiffness matrix $\mathbf{A}$, so that the matrix elements $a_{i,j}$ become

$$a_{i,j} = \gamma(x, y) \cdot a(N_i, N_j), \tag{4.16}$$

where $a(N_i, N_j)$ is the usual bilinear form defined in (3.21). The function varies in both spatial directions and thus makes it possible to adapt the parameterization to better fit the domain.

The idea is to use a relatively low γ-value in the geometry areas experiencing problems with the straight forward uncoupled Poisson method, and a relatively high γ-value in the areas not experiencing these problems. This way we make the challenging areas more elastic, and the performance of the algorithm may be improved.

Figure 4.4 shows the effect of the γ-function for a mesh on the square $(x, y) \in [0, 1]^2$. In Figure 4.4a the mesh is uniform as there is no γ-function pulling the mesh lines. In Figure 4.4b, the mesh lines are distorted compared to the previous mesh. The function $\gamma(x, y) = x \cdot y$ makes the mesh lines for high values of x and y less elastic than those for lower x and y coordinates, and hence the mesh is pulled away from the lower left corner, while the upper right corner is less affected by the introduction of γ.

The gamma function must be chosen to the specific geometry in question, i.e. it is not a general function that works for all geometries.

4.3 Linear Elasticity using IGA

In the third parameterization method we examine the potential of a linear elasticity approach using IGA to find a domain parameterization. Linear elasticity will, contrary to the uncoupled Poisson approach, connect the x- and y-coordinates in

a more direct manner. It will also, to some degree, avoid the difficulty of finding an appropriate γ-function for the geometry in question. We start by providing an introduction to linear elasticity before looking at the isogeometric formulation of the parameterization problem.

4.3.1 Introduction to Linear Elasticity

Linear elasticity can be used to study mathematically how solid materials undergo small deformations when subjected to loading conditions. It is assumed that the material behaves in a linear manner, that it is only subject to small deformations, and that no overlaps occur during the deformation [28]. Linear elasticity is governed by the relationship between different quantities such as stress and strain. Even though our parameterization problem is not directly related to these physical quantities, they are heavily incorporated into the linear elasticity problem. It is therefore important to be somewhat familiar with the concepts before attempting to develop a parameterization method based on linear elasticity.

Strain

Strain is a measure describing the relative deformation of a body. Under the assumption of small displacements, the deformation of a body can be described by three independent variables, corresponding to extensional and shear strain, denoted by ε_{xx}, ε_{yy}, and ε_{xy} respectively. The extensional strains can be thought of as the relative displacement in the material compared to a rigid body. That is, by writing the displacement $\mathbf{d}$ in component form $[d_x,\, d_y]$, the extensional strains can be expressed as

$$\begin{aligned} \varepsilon_{xx} &= \lim_{\Delta x \to 0} \frac{d_x(x+\Delta x,\, y) - d_x(x,\, y)}{\Delta x} = \frac{\partial d_x}{\partial x} \\ \varepsilon_{yy} &= \lim_{\Delta y \to 0} \frac{d_y(x,\, y+\Delta y) - d_y(x,\, y)}{\Delta y} = \frac{\partial d_y}{\partial y}. \end{aligned} \tag{4.17}$$

The shear strain is a measure of the change in angle between unit vectors in the x- and y-directions, and can be expressed as [28]

$$\varepsilon_{xy} = \frac{\partial d_y}{\partial x} + \frac{\partial d_x}{\partial y}. \tag{4.18}$$

The extensional strain in (4.17) and the shear strain in (4.18) can be combined to get a total strain vector,

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = \nabla_S \, \mathbf{d} \tag{4.19}$$

Stress

Stress is used to measure the strength of the forces acting on the body causing the deformation. In vector form, stress is expressed as

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} \, \sigma_{yy} \, \sigma_{xy} \end{bmatrix}^T . \tag{4.20}$$

Here, σ_{xx} and σ_{yy} are normal stresses, and σ_{xy} is shear stress. The first subscript denotes the direction of the normal of the plane on which the stress is acting, and the second subscript denotes the direction of the force. Traction is also normally involved in linear elasticity, but traction is omitted here as it will not be needed for the method to be developed.

Hooke's Law

Strain and stress are related to each other through Hooke's law. When studying linear elasticity, the generalized Hooke's law is given by [28]

$$\boldsymbol{\sigma} = \mathbf{D} \, \boldsymbol{\varepsilon}. \tag{4.21}$$

The elasticity matrix $\mathbf{D}$ depends on whether plane stress or plane strain is assumed, but it is always a 3×3 symmetric positive definite matrix. The problem to be considered here assumes plane stress, i.e. it is assumed that the body in question is a thin plate with stresses action along its plane, with no stress acting perpendicular to the plane. Then, the matrix $\mathbf{D}$ is given as

$$\mathbf{D} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}, \tag{4.22}$$

where E is Young's modulus and ν is Poisson's ratio [29].

Equilibrium Equation

One more aspect is needed before we are equipped to tackle problems regarding linear elasticity, namely the equilibrium equation. When considering deformation of a body, there are forces acting on the body. These body forces, $\mathbf{b} = [b_x, \, b_y]^T$, can for example be gravity or magnetic forces. In order for the body in question to be in equilibrium, it is required that [29]

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \mathbf{b} = 0. \tag{4.23}$$

In vector form (4.23) becomes

$$\nabla \sigma_x + b_x = 0, \quad \nabla \sigma_y + b_y = 0, \tag{4.24}$$

which by the use of ∇_S can be written as

$$\nabla_S^T \boldsymbol{\sigma} + \mathbf{b} = 0. \tag{4.25}$$

Thus, for the body to be in equilibrium, equation (4.25) must be satisfied.

4.3.2 Linear Elasticity IGA Formulation

The isogeometric formulation of the linear elasticity problem is based on the same approach as presented in Section 3.2 for the Poisson problem. The strong form of the problem is stated before the weak form is derived. Then adjustments are made to make the problem suitable for IGA. We will attempt to parameterize a given domain, by treating it as a displacement problem. That is, we solve the parametereization problem by the linear elasticity approach by setting $\mathbf{u} = \mathbf{d}$.

Strong Form

The strong form for the linear elasticity problem arises when the results from the previous section are combined with a boundary condition. In linear elasticity there are two kinds of boundary conditions to consider: traction boundary condition and displacement boundary conditions. As we are using the basis of linear elasticity to solve a parameterization problem where the domain boundary is given, we will only be considering the displacement boundary condition. That is, the domain boundaries $\partial\Omega_1$, $\partial\Omega_2$, $\partial\Omega_3$ and $\partial\Omega_4$ will be used as the prescribed displacement when the boundary condition is enforced, and the strong form of the problem can be written as [28]

$$\nabla \sigma_x + b_x = 0 \text{ and } \nabla \sigma_y + b_y = 0 \qquad \text{in } \Omega \tag{4.26a}$$

$$\boldsymbol{\sigma} = \mathbf{D}\, \nabla_S\, \mathbf{u} \tag{4.26b}$$

$$\mathbf{u} = \bar{\mathbf{u}} \qquad \text{on } \partial\Omega \tag{4.26c}$$

where the relation $\boldsymbol{\sigma} = \mathbf{D}\,\boldsymbol{\varepsilon} = \mathbf{D}\,\nabla_S \mathbf{u}$ has been used to relate strain and stress in (4.26b).

Weak Form

Following the approach from Section 3.2.2, the strong form of the linear elasticity problem (4.26a) is multiplied by a test function before we integrate over the domain Ω. Once again we use spline functions as test functions, but this time in a vector

form $\mathbf{N} = [N_x,\, N_y]$, such that the expressions in equation (4.26a) becomes

$$\begin{aligned} \int_\Omega N_x \, \nabla \sigma_x \, \mathrm{d}A + \int_\Omega N_x \, b_x \, \mathrm{d}A = 0 \\ \int_\Omega N_y \, \nabla \sigma_y \, \mathrm{d}A + \int_\Omega N_y \, b_y \, \mathrm{d}A = 0 \end{aligned} \tag{4.27}$$

Here, N_x and N_y are the spline test functions used to find $\mathbf{u} = [u_x,\, u_y]$. Then, Green's Theorem is applied to (4.27) and the two equations are added together,

$$\int_\Omega \left(\nabla N_x \, \sigma_x + \nabla N_y \, \sigma_y \right) \, \mathrm{d}A = \int_\Omega \left(N_x b_x + N_y b_y \right) \, \mathrm{d}A. \tag{4.28}$$

By utilizing the relations $(\nabla_S \, \mathbf{N})^T \boldsymbol{\sigma} = \nabla N_x \, \sigma_x + \nabla N_y \, \sigma_y$ and (4.26b) this can be written more compactly as

$$\int_\Omega \left(\nabla_S \, \mathbf{N} \right)^T \, \mathbf{D} \, \nabla_S \mathbf{u} \, \mathrm{d}A \, = \int_\Omega \mathbf{N}^T \mathbf{b} \, \mathrm{d}A. \tag{4.29}$$

This can be written as

$$a(\mathbf{u}, \mathbf{N}) = l(\mathbf{N}), \tag{4.30}$$

where

$$\begin{aligned} a(\mathbf{u}, \mathbf{N}) &= \int_\Omega \left(\nabla_S \, \mathbf{N} \right)^T \, \mathbf{D} \, \nabla_S \, \mathbf{u} \, \mathrm{d}A \\ l(\mathbf{N}) &= \int_\Omega \mathbf{N}^T \mathbf{b} \, \mathrm{d}A. \end{aligned} \tag{4.31}$$

The observant reader will recognize some similarities between the weak formulation of the Poisson problem in equation (3.22) and the weak formulation of the linear elasticity problem in (4.30), with the main difference being that the displacement $\mathbf{u}$ is a vector in the latter case.

It turns out that the assembly of the stiffness matrix can be done in very similar ways in both cases, but in linear elasticity one must account for the vector nature of the displacement $\mathbf{u}$. This is done by introducing spline basis functions $\mathbf{R}_i^d = N_i(\xi,\, \eta) \, \mathbf{e}^d$, where $\mathbf{e}^d$ are the unit vectors and $N_i(\xi,\, \eta)$ are spline functions as defined in Chapter 3. Defining

$$\mathbf{u}_h = \sum_{i=1}^{n} \sum_{d=1}^{2} u_{h,i}^d \mathbf{R}_i(\xi,\, \eta). \tag{4.32}$$

makes it possible to rewrite the weak form as a system of linear equations,

$$\mathbf{A}\mathbf{u} = \mathbf{f}. \tag{4.33}$$

The elements of $\mathbf{A}$ and $\mathbf{f}$ are

$$\begin{aligned} a_{i,j} &= a(R_i, R_j) = \int_\Omega (\nabla_S \mathbf{R}_i)^T \mathbf{D} (\nabla_S \mathbf{R}_j) \, \mathrm{d}A \\ f_j &= l(R_j) = \int_\Omega \mathbf{R}_j^T \mathbf{b} \, \mathrm{d}A, \end{aligned} \tag{4.34}$$

where ∇_S is given in (4.19).

4.3.3 Implementation and Verification

When using linear elasticity for parameterization of a domain, the parameterization is given by the deformation $\mathbf{u}$ found by solving (4.33). The matrix $\mathbf{A}$ and right hand side $\mathbf{f}$ with components given by (4.34), is assembled in a similar manner as in the uncoupled Poisson case, by looping over all elements and evaluating (4.34) for all the nonzero basis functions.

The boundaries of the domain being parameterized will be used as displacement boundary conditions. That is, the prescribed displacement on the boundary $\bar{\mathbf{u}}$ will be given by $\partial\Omega$. The boundary condition is enforced through a least squares approach similar to the one used on the Poisson problem, but with extensions to fit the new, vectorized problem.

The set-up and structure of the procedure have quite a few similarities to the program solving the Poisson problem presented in Section 4.2.1, but with modifications in the core of the problem. The full software program for solving linear elasticity problems can be found in Appendix A.3.

Before using the implemented program to find parameterizations, it is important to run it on some problems with known solutions to validate the model and benchmark the performance. In order to do this, the program is tested on a problem with known solution

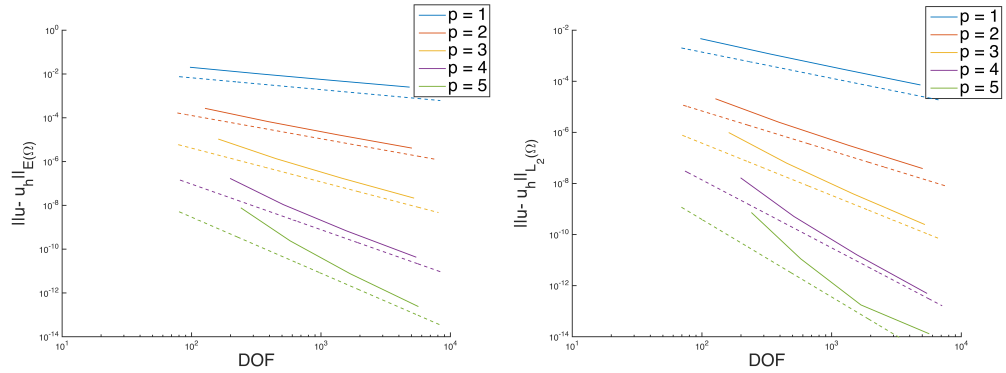
$$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} \cos(x)\cos(y) \\ \cos(x)\cos(y) \end{bmatrix}, \quad (x, y) \in [0, 1] \times [0, 1]. \tag{4.35}$$

By differentiating the exact solution, the right hand side of the elasticity problem

$$\begin{aligned} \nabla \boldsymbol{\sigma}(\mathbf{u}) &= -\mathbf{f} && \text{in } \Omega \\ \mathbf{u} &= \mathbf{0} && \text{on } \partial\Omega, \end{aligned} \tag{4.36}$$

becomes

$$\mathbf{f} = -\frac{E}{2(1-\nu^2)} \begin{bmatrix} (\nu - 3)\cos(x)\cos(y) + (1+\nu)\sin(x)\sin(y) \\ (\nu - 3)\cos(x)\cos(y) + (1+\nu)\sin(x)\sin(y) \end{bmatrix}. \tag{4.37}$$



(a) Error in energy norm (b) Error in L_2 norm

Figure 4.5: Convergence of the linear elasticity solver. Error as a function of degrees of freedom for order $p = 1, 2, 3, 4$ and 5 in the energy and L_2 norm.

As a measure of the performance of the program, the error $\mathbf{e}$ is evaluated in the L_2 norm and the energy norm, which in the linear elasticity case takes on the forms [30],

$$\begin{aligned} ||\mathbf{e}||_{L_2(\Omega)} &= \left(\int_\Omega \mathbf{e}^T \mathbf{e} \, \mathrm{d}\Omega \right)^{1/2} \\ ||\mathbf{e}||_{E(\Omega)} &= \left(\int_\Omega (\mathbf{e}_{\boldsymbol{\sigma}})^T \mathbf{D}^{-1} (\mathbf{e}_{\boldsymbol{\sigma}}) \, \mathrm{d}\Omega \right)^{1/2}, \end{aligned} \tag{4.38}$$

where $\mathbf{e} = \mathbf{u} - \mathbf{u}_h$ and $\mathbf{e}_{\boldsymbol{\sigma}} = \boldsymbol{\sigma} - \boldsymbol{\sigma}_h$.

In order to gain understanding on the performance of the program, it is run with several polynomial degrees, $p = \{1, 2, 3, 4, 5\}$, and with increasing number of DOFs. The results are presented in Figure 4.5.

Figure 4.5a shows the convergence rates in the energy norm, $||\mathbf{e}||_{E(\Omega)}$, as a function of DOF. The dashed lines beneath the convergence rate for each polynomial degree has a slope of $-p/2$. It is clear that the convergence rates follow the dashed lines closely. Figure 4.5b shows the rate of convergence in the L_2 norm, and the slopes of the dashed lines are $-(p+1)/2$. Once again the convergence rates closely follow the dashed lines, except the L_2-norm of $p = 5$ for high number of degrees of freedom, where the error reaches machine precision.

This confirms that the rate of convergence is $p/2$ in the energy norm and $(p + 1)/2$ in the L_2 norm, which is as expected in isogeometric analysis [3].

4.3.4 Modified Elasticity Matrix

In regular linear elasticity, the elasticity matrix $\mathbf{D}$ in (4.22) is constant as it is related to the material undergoing deformation. However, in our approach towards

geometry parameterization, this matrix gets a slightly different role. It needs no longer be constant, but rather it can be dependent on domain location. That is, $\mathbf{D} = \mathbf{D}(x, y)$. This is an expansion of the gamma function introduced in Section 4.2.2. Similarly to the gamma function, $\mathbf{D}(x, y)$ is set to have low values in challenging areas of the domain and higher values in less challenging areas. In a linear elasticity context this corresponds to having more elastic materials in the challenging areas and more rigid materials in the more manageable areas. The matrix has to be tailored for each domain to be parameterized.

4.4 Quasistatic Method

The fourth and final method we are going to look at for parameterizing a domain is a quasistatic approach. The idea behind this method is, as for the linear elasticity, to treat the problem as a displacement problem, but instead of solving the displacement simultaneously, it is solved gradually. This is illustrated in Figure 4.6, where the domain we wish to parameterize is the circle shown in the lower right corner of the figure. But instead of going straight for this geometry, as in the case of linear elasticity, the method starts with a domain that is easy to parameterize, namely the square. Then the domain is gradually transformed into the circle. In each step the method uses the result from the last step as initial control points. Thus, the method moves gradually towards the specified geometry, solving several parameterization problems along the way to improve the final parameterization.

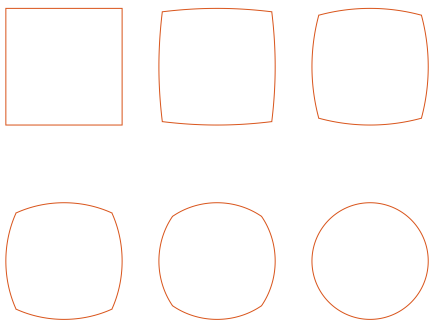


Figure 4.6: Illustration of gradually changing the boundary from a square to a circle as it is done in the quasistatic method.

4.4.1 Implementation and Verification

The program for finding a parameterization of a domain using the quasistatic approach utilizes the fact that a method based on linear elasticity has already been implemented: the linear elasticity solver presented in Section 4.3 is used to find a parameterization in each step of the quasistatic method. Verification of the method has therefore already been done. Algorithm 4.2 shows an outline for the quasistatic procedure. The full code implementation can be found in Appendix A.4.

In the second line in Algorithm 4.2 a square is initialized as the domain, and then a new boundary is set in each step i of the while-loop. The new boundary $[b_x, b_y]$ is found in each step by performing the procedure

$$\begin{aligned} b_x(i) &= b_x(0) + \frac{i}{n}\left(b_x(n) - b_x(0)\right) \\ b_y(i) &= b_y(0) + \frac{i}{n}\left(b_y(n) - b_y(0)\right). \end{aligned} \tag{4.39}$$

Here $[b_x(0), b_y(0)]$ is the boundary of the square, and $[b_x(n), b_y(n)]$ is the boundary of the final domain for which we want to find a parameterization, both of which are known.

Algorithm 4.2 Program outline of the quasistatic parameterization method.

```
1: procedure QUASISTATICS
2:     Initialize square boundary
3:     Set number of steps n
4:     Run linear elasticity solver to get Ũ = [Ũ_x, Ũ_y]
5:     while i < n do
6:         Control points ← [Ũ_x, Ũ_y]
7:         Set boundary for step i/n
8:         Run linear elasticity solver to get new Ũ = [Ũ_x, Ũ_y]
9:     Parameterization given by final U = [U_x, U_y]
```

Chapter 5

Numerical Results

In this chapter we will present numerical results from the four methods described in the previous chapter. We will look at several geometries with different parameterization challenges and compare the performance of each method. The comparison between the methods will be based on the mesh metrics presented in Chapter 2, and the methods' ability to produce valid parameterizations.

For each main geometry under consideration, we will define a parameter κ as a measure of how challenging the geometry is to parameterize. Then, the parameterization methods will be applied to see how well they perform on the given geometry, and compared with each other.

The results will be presented with geometries of increasing difficulty. We start with a geometry that is close to being square, with the exception of one boundary curve which will be a sine function. Then, the difficulty is increased in the second geometry, where all the boundaries are defined as sine functions. The third geometry is even more difficult to parameterize, and looks like a piece from a jigsaw puzzle. We will present all the methods on one geometry before moving on to the next geometry.

The chapter is concluded with a section on method comparison. We will investigate how the four parameterization methods perform on the geometries with different values of the difficulty parameter κ on each geometry, and how the polynomial degree p influences the results.

5.1 Bottom Sine Geometry

The first geometry we are going to consider has a sine function on the bottom boundary, while the three other edges are parallel to either the x- or y-axis. That

is, the domain Ω is defined by the curves

$$\begin{aligned} \partial\Omega_1 &= \begin{bmatrix} x \\ \kappa \sin(2\pi x) \end{bmatrix}, & \partial\Omega_3 &= \begin{bmatrix} x \\ 1 \end{bmatrix}, \quad x \in [0,1] \\ \partial\Omega_2 &= \begin{bmatrix} 1 \\ y \end{bmatrix}, & \partial\Omega_4 &= \begin{bmatrix} 0 \\ y \end{bmatrix}, \quad y \in [0,1]. \end{aligned} \tag{5.1}$$

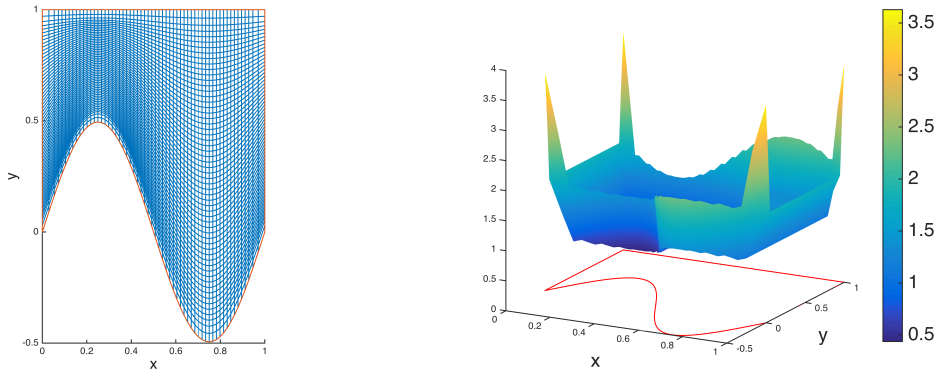
The difficulty parameter κ is included in the expression for $\partial\Omega_1$, and corresponds to the amplitude of the sine function on the bottom boundary. If $\kappa = 0$ the domain is simply a square, while if $\kappa = 1$ the bottom boundary will intersect the top boundary. Hence, we will be working with κ's in the half-open interval $[0, 1)$, with special emphasis on $\kappa = 0.5$.

To make comparisons easier, all the methods are set to produce meshes with equal number of elements. Furthermore, the polynomial degree is set to $p = q = 2$, the knot vectors $\Xi = \mathcal{H}$ are equal and uniform with $n = 20$, and the material properties used in the linear elasticity and quasistatic methods are set to correspond to an elastic material with $E = 0.1$ and $\nu = 0.499$.

5.1.1 Gordon-Hall

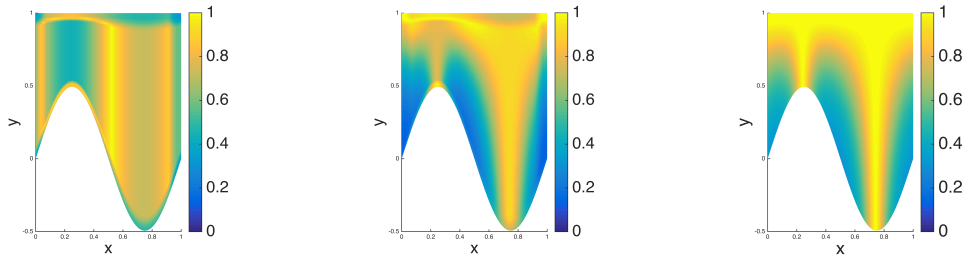
The Gordon Hall technique generates the parameterization mesh shown in Figure 5.1a for the sine geometry with $\kappa = 0.5$. The blue lines show the grid lines, while the red lines show the curves defining the physical domain Ω.

This mesh is valid in the sense that no neighboring grid lines intersect and all parameterization points are inside the domain. This is also confirmed by the Jacobian in Figure 5.1b, which is strictly positive throughout the entire mesh.



(a) Parameterization mesh (b) Jacobian

Figure 5.1: **Gordon-Hall:** Mesh and corresponding Jacobian obtained with the Gordon-Hall algorithm on the bottom sine geometry.



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.2: **Gordon-Hall:** Size, shape, and skew metrics of the mesh generated by the Gordon-Hall algorithm on the bottom sine geometry.

Table 5.1: **Gordon-Hall:** Measure of mesh quality for the mesh obtained by the Gordon-Hall algorithm on the bottom sine geometry.

k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0.3212	0.1081	0.3078	0.0590	0.1012
$RMS(\mathcal{M}_k)$	0.7115	0.6787	0.7779	0.4828	0.5351

The size of each grid element varies a throughout the mesh, as does the shape and skew of the elements. This is illustrated in Figure 5.2. The figure shows the value of the size, shape, and skew metric of the mesh on the sine geometry.

Figure 5.2a shows how the size metric, $\mathcal{M}_{Size}$, varies over the elements in the mesh. The metric picks up on the small elements in the area around $x = 0.25$ and the bigger elements around $x = 0.75$, where the sine function on the bottom boundary experience the highest and lowest y-values respectively.

Figure 5.2b shows that the shape metric, $\mathcal{M}_{Shape}$, picks up on poorly shaped elements along the bottom sine boundary. At the same time we see that the elements around $x = 0.25$ and $x = 0.75$ generally have a good shape, and that elements with high y-values also have a good shape regardless of the x-value.

Finally, the skew metric, $\mathcal{M}_{Skew}$, is shown in Figure 5.2c. It shows similar tendencies as the shape metric, but considers the elements along the bottom boundary to be slightly better, and it also deems the elements around $x = 0.25$, $x = 0.75$ and high y-values as better than the shape metric.

Table 5.1 shows the min-max measure and the root mean square measure of each of the five mesh metrics presented in Chapter 2 for the mesh in Figure 5.1a.

The min-max measure suggests that there is a variety between the best and worst elements in all the metrics. Especially the shape metric finds the relative shape between the best and worst element to be quite different. This is also

reflected in the combination metrics between size and shape, which is particularly low.

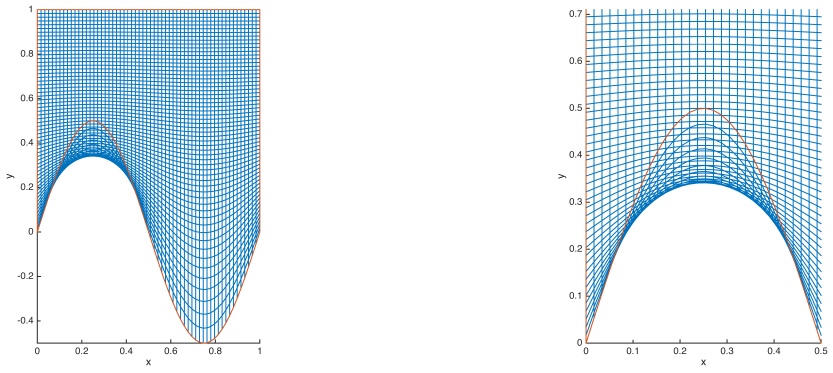
The root mean square measure of the pure metrics are all relatively high, but once again, the shape metric shows less optimistic results than the other metrics. The high values in the root mean square measure means that, although the parameterization resulted in some poor elements, the overall mesh is relatively good.

It is worth noticing that the combination metrics turn out far worse than the metrics they are comprised of. This is not surprising since more elements will consist of poor size or shape or skew or a combination of these, than just poor size or shape or skew.

5.1.2 Uncoupled Poisson

Next, we apply the uncoupled Poisson solver to the bottom sine geometry defined in (5.1). The parameter κ signifying the amplitude of the sine curve is once again set to 0.5, while the γ-function is set to be constantly equal to one. The result is shown in Figure 5.3.

The domain parameterization is clearly invalid because of the many mesh lines going outside the domain, that is, there are parameterization points not contained in the domain and the mesh is inverted. Figure 5.3b shows an enlarged portion of the mesh around the maximum value of the sine function making up the bottom boundary, and makes it very clear that the parameterization is invalid. This is also confirmed by the Jacobian of the parameterization which is shown in Figure 5.4, where the distance to the z-axis corresponds to the value of the Jacobian.



(a) Parameterization mesh (b) Magnified section

Figure 5.3: **Uncoupled Poisson:** Mesh obtained by solving the uncoupled Poisson problem on the bottom sine geometry with $\gamma(x, y) = 1$.

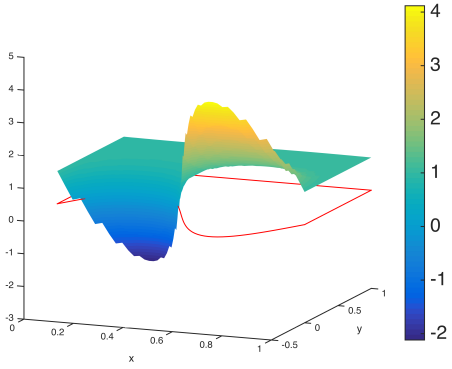


Figure 5.4: **Uncoupled Poisson:** Jacobian of the mesh obtained by solving the uncoupled Poisson problem on the bottom sine geometry with $\gamma(x, y) = 1$.

The red line in this figure shows the outline of the domain at $z = 0$.

The areas in Figure 5.3a that lies outside the domain correspond to the areas where the Jacobian becomes negative, and we recall from Section 3.3 that a negative Jacobian implies that the parameterization is invalid.

In an attempt to remedy the unsatisfactory results, a non-constant γ-function taking x and y as argument is introduced. The best $\gamma(x, y)$ we have been able to find for this geometry is the function

$$\gamma(x, y) = y^{\frac{7}{4}} \cdot \left(1 - \frac{1}{4}x\right). \tag{5.2}$$

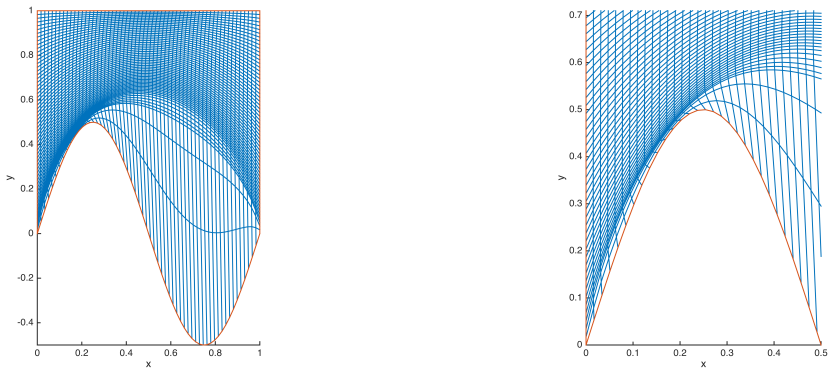
The mesh resulting from running the uncoupled Poisson solver with the new γ-function is shown in Figure 5.5a. The parameterization is valid in the sense that no mesh lines leave the domain, i.e. all parameterization points are contained in Ω. This is more clearly seen in Figure 5.5b where a section has been magnified.

The Jacobian of the parameterization is shown in Figure 5.6. Although the Jacobian varies greatly in magnitude, it is strictly positive for the parameterization.

Even though the parameterization is valid, it it not necessarily a good parameterization, and the mesh metrics highlight several elements as being rather poor.

Figures 5.7a, 5.7b and 5.7c shows the size, shape and skew metric respectively.

The size metric reveals that the parameterization yields some very small and very large elements along the bottom boundary. For low x-values, the elements along the sine boundary are terribly small, while the elements for higher x-values are terribly large. The metric also reveals some areas where the elements have a satisfactory size, highlighted by a more yellow color.



(a) Parameterization mesh (b) Magnified section

Figure 5.5: **Uncoupled Poisson:** Mesh obtained by solving the uncoupled Poisson problem on the bottom sine geometry with $\gamma(x, y) = y^{\frac{7}{4}} \cdot (1 - \frac{1}{4}x)$

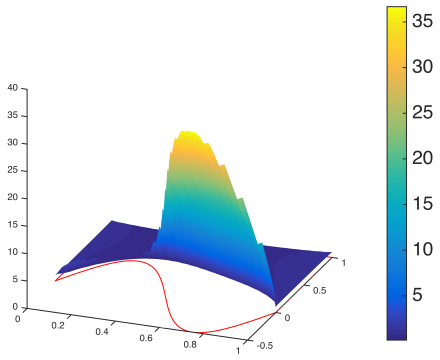
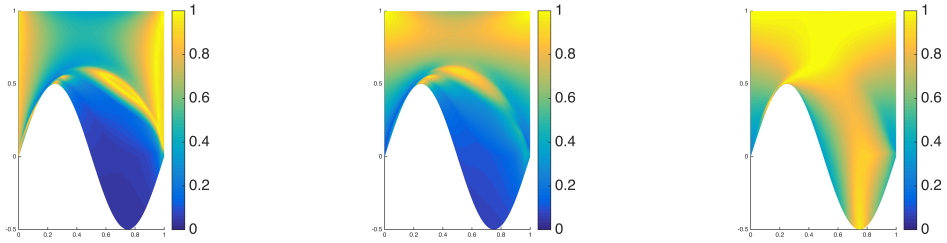


Figure 5.6: **Uncoupled Poisson:** Jacobian of mesh obtained by the uncoupled Poisson method when γ is no longer constant.

The shape metric identifies some of the same elements as the size metric to be poorly shaped. The elements on the lower part of the geometry are generally emphasized as being ill-shaped, while the elements on the upper part of the geometry are viewed as better shaped by the metric.

The skew metric is slightly more optimistic towards the mesh quality compared with the size and shape metrics. That is, even though many elements are of both poor size and shape, they are not necessarily bad with regards to skewness.

Table 5.2 accentuates the findings from Figure 5.7. The min-max measure of the size and shape metrics are particularly low, suggesting that there is a vast difference in the elements. This is not a surprising result given the larger elements



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.7: **Uncoupled Poisson:** Metrics of the mesh generated by the uncoupled Poisson method with $\gamma = \gamma(x, y)$.

Table 5.2: **Uncoupled Poisson:** Measure of mesh quality for the parameterization obtained with the uncoupled Poisson method.

k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0.0325	0.0641	0.2463	0.0021	0.0168
$RMS(\mathcal{M}_k)$	0.6171	0.6945	0.8748	0.4420	0.5198

on the bottom part of mesh and the very small elements visible on the top part of the sine boundary, shown in Figure 5.5b.

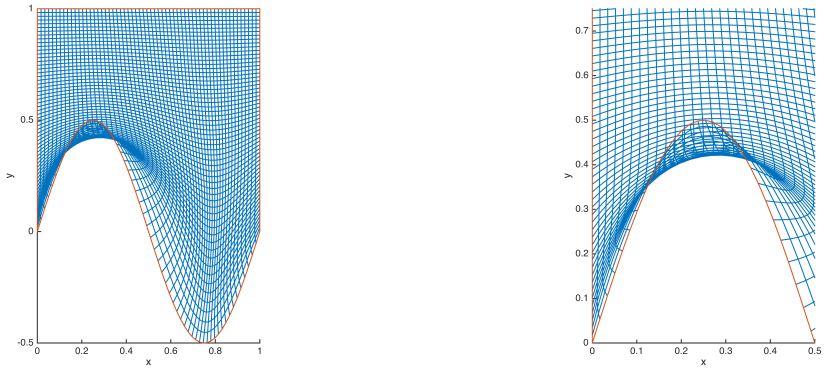
It is also revealed that several elements possess some degree of bad size, skew, or a combination of the two. That is, the combination metric $\mathcal{M}_{SizeSkew}$ shows a worse outcome in both the min-max measure and the root mean square measure compared to the metrics of pure size or pure skew. This also holds true for the $\mathcal{M}_{SizeShape}$ combination metric, and this metric shows even less optimistic results than the size-skew combination.

Regardless of the poor elements detected by the metrics and the difference in element quality between the best and the worst element, the root mean square measure still finds the mesh to be adequate in some metrics. The skew metric shows an especially high value of the root mean square measure, while the combination size-shape has the lowest value.

5.1.3 Linear Elasticity

We will now attempt to find a parameterization of the bottom sine geometry with the linear elasticity method. We start by letting the elasticity matrix $\mathbf{D}$ be as defined in (4.22), constant over the entire domain. Then the linear elasticity methods yields the mesh in Figure 5.8a.

The parameterization is not valid because parameterization points are placed outside the domain, as clearly seen in Figure 5.8b. This results in a negative



(a) Parameterization mesh (b) Magnified section

Figure 5.8: **Linear Elasticity:** Mesh obtained with the linear elasticity method on the bottom sine geometry with constant elasticity matrix.

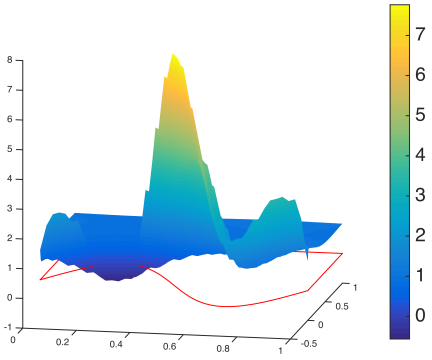


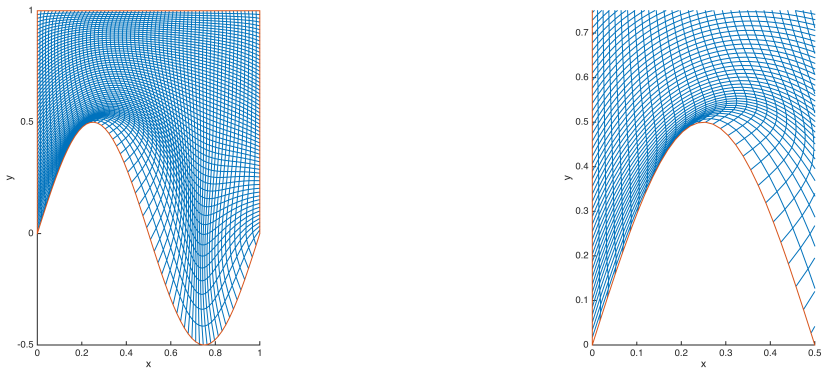
Figure 5.9: **Linear Elasticity:** Jacobian of the mesh obtained with the linear elasticity method on the bottom sine geometry with constant elasticity matrix.

Jacobian, as shown in Figure 5.9.

In an attempt to make the method produce a valid parameterization, the elasticity matrix is modified. The best $\mathbf{D}(x, y)$-matrix we have been able to find for this geometry is

$$\mathbf{D}(x,y) = \frac{E\left(y^{\frac{5}{8}} + 0.3\right)\left(1 - \frac{1}{4}x\right)}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}. \tag{5.3}$$

With $\mathbf{D}(x, y)$ as in (5.3), the results in Figure 5.10 are obtained. Figure 5.10b shows that all the mesh lines are contained inside Ω, and the Jacobian of the



(a) Parameterization mesh (b) Magnified section

Figure 5.10: **Linear Elasticity:** Mesh obtained with the linear elasticity method on the bottom sine geometry with $\mathbf{D}(x, y)$ given by (5.3).

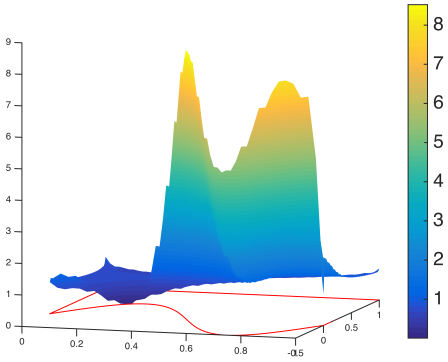
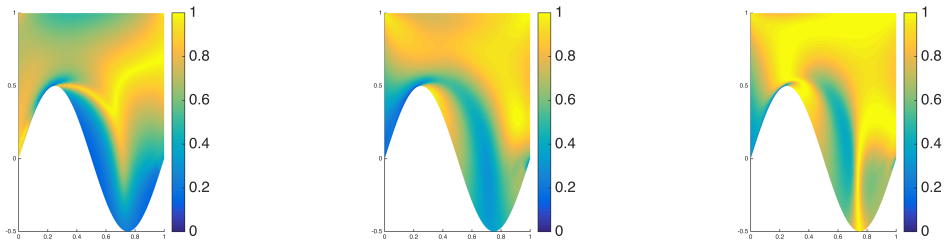


Figure 5.11: **Linear Elasticity:** Jacobian of the mesh obtained with the linear elasticity method with $\mathbf{D}(x, y)$ given by (5.3).

parameterization, shown in Figure 5.11, is strictly positive. Hence we can conclude that the parameterization is valid.

The magnified section of the parameterization in Figure 5.10b also show that the mesh lines intersect the boundary perpendicularly. This was not obtained with neither the Gordon-Hall method nor the Uncoupled Poisson method.

Figure 5.12 shows the size, shape, and skew metrics of the obtained parameterization. They all show a mix of both good and bad elements. Upon closer inspection, it is revealed that all the metrics picks up on elements with almost perfect size, shape, and skew, that is, elements for which $\mathcal{M}_{Size} \approx 1$, $\mathcal{M}_{Shape} \approx 1$ or $\mathcal{M}_{Skew} \approx 1$. However, it is also revealed that the metrics find elements for



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.12: **Linear elasticity:** Metrics of the mesh generated by the linear elasticity method.

Table 5.3: **Linear elasticity:** Metric measures for the mesh generated by the linear elasticity method on the sine geometry.

k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0.0363	0.0059	0.0081	0.0002	0.0003
$RMS(\mathcal{M}_k)$	0.7195	0.7619	0.8670	0.5628	0.6234

which $\mathcal{M}_{Size}, \mathcal{M}_{Shape}, \mathcal{M}_{Skew} \approx 0$. It turns out that they all highlight the same element as being the worst, and that this element is very close to being degenerate.

This nearly degenerate element makes an impact on the mesh metric measures in Table 5.3. Particularly the min-max measure is affected by having elements with such low values in the different metrics.

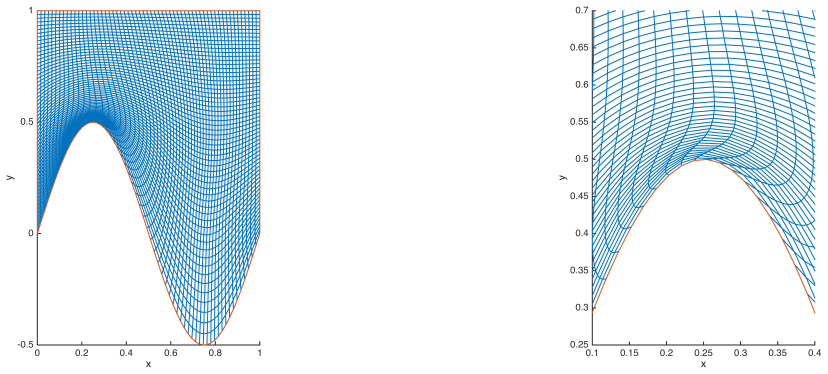
The root mean square measure is quite high for all metrics despite some poor elements. This suggest that even though the mesh contains some rather poor elements, the overall mesh quality is still high.

5.1.4 Quasistatic

Finally, the quasistatic method is applied to the bottom sine geometry. As the quasistatic method consists of solving the parameterization problem in stages, we also have to set the number of steps used to arrive at the final geometry. We set the number of iterations to be 15 and let the elasticity matrix $\mathbf{D}$ be constant. The resulting parameterization is shown in Figure 5.13.

All the parameterization points are contained in Ω, and the Jacobian, shown in Figure 5.14, is strictly positive.

It is not surprising that the quasistatic approach is able to solve the parameterization problem with a constant elasticity matrix even if the linear elasticity method failed on the same. The linear elasticity method assumes that we only have small deformations, and it is in general unable to accurately model the prob-



(a) Parameterization mesh (b) Magnified section

Figure 5.13: **Quasistatic:** Mesh obtained with the quasistatic method on the bottom sine geometry.

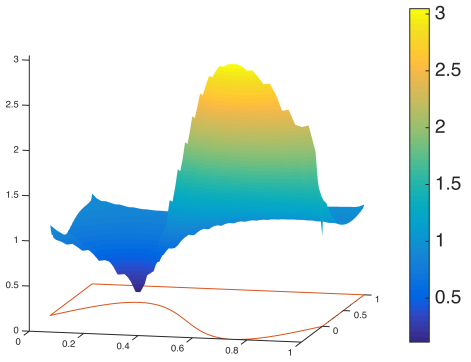
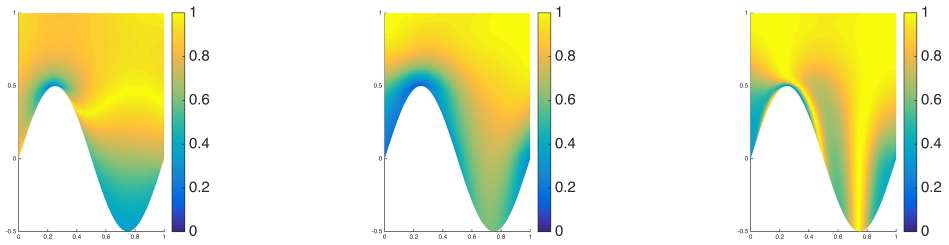


Figure 5.14: **Quasistatic:** Jacobian of the parameterization obtained with the quasistatic method.

lem when the deformations become too large. The quasistatic approach eliminates this problem by solving the deformation in steps. By increasing the number of iterations, we can control that the deformations always are small enough for the method to be accurate for each iteration.

Figure 5.15 shows illustrations of the metrics of the mesh. The figures show that some elements are of a lesser quality, especially along the bottom boundary, and some elements are of good quality regardless of which metric is being considered.

When comparing the metrics with the metrics for the Gordon-Hall algorithm, Figure 5.2, the uncoupled Poisson method, Figure 5.7, and the linear elasticity approach, Figure 5.12, we see that the quasistatic method produces more elements



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.15: **Quasistatic:** Metrics of the mesh generated by the quasistatic method.

Table 5.4: **Quasistatic:** Measure of mesh quality for the mesh obtained through the quasistatic method.

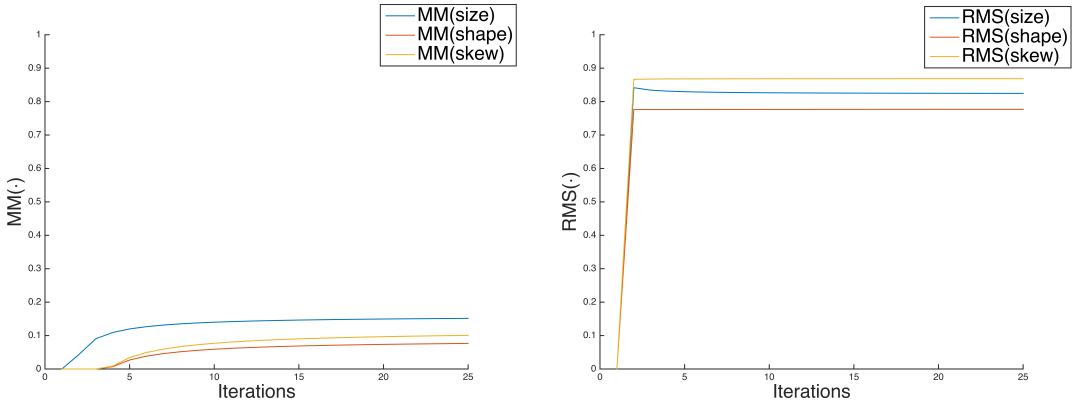
k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0.1464	0.0689	0.0902	0.0102	0.0132
$RMS(\mathcal{M}_k)$	0.8254	0.7767	0.8684	0.6801	0.7383

of a higher quality than any of the other methods.

This is also reflected in the metric measures given in Table 5.4. By comparing the min-max and root mean square measure for the metrics in the quasistatic method with the measures for the Gordon-Hall, uncoupled Poisson and linear elasticity, cf. Table 5.1, 5.2 and 5.3 respectively, we see that the quasistatic approach produces a parameterization with better mesh quality in all metric measures, with the only exception being the min-max measure of the parameterization obtained with the Gordon-Hall algorithm. Even so, the quasistatic method performs better in the root mean square measure than any other method. Hence we can say that on this problem, with $\kappa = 0.5$, the quasistatic method produces the best parameterization overall.

As opposed to the uncoupled Poisson and the linear elasticity method, the quasistatic approach avoids the effort and uncertainty of having to find a suitable $\gamma(x, y)$ or $\mathbf{D}(x, y)$. Instead we have to decide in how many iterations we wish to solve the parameterization problem. We do not wish to use too few iterations, as this may compromise the final result, and at the same time we wish to avoid using more iterations than necessary, as this is time consuming and computationally intensive.

Figure 5.16 shows how the min-max and root mean square measures are affected by the number of iterations used to find a parameterization of the bottom sine geometry with $\kappa = 0.5$. Figure 5.16a shows that the min-max measure improves



(a) Min-max measure (b) Root mean square measure

Figure 5.16: **Quasistatic:** Min-max and root mean square measure as functions of iterations used to find the parameterization of the sine geometry using the quasistatic approach. The size metric is shown in blue, while the shape and skew metric is shown in red and yellow respectively.

as the number of iterations is increased. The steepest improvement is found in the interval between 1 and 10 iterations, and then a plateau is reached. The figure also shows that if the geometry is parameterized with three iterations or less, the resulting mesh contains degenerate elements.

The dependency on number of iterations for the root mean square measure in Figure 5.16b shows less sign of improvement with increasing number of iterations. It holds a high value regardless of metric and number of iterations.

Both measures suggest that the quasistatic approach is unable to find a valid parameterization of the bottom sine geometry with only one iteration. This is not surprising since the quasistatic method with one iteration essentially is the linear elasticity method with constant elasticity matrix.

Furthermore, the relatively constant root mean square measures and the flattening of the improvements of the min-max measure suggest that it is unnecessary to do more than about 15 iterations. That is, our choice of 15 iterations to solve the parameterization problem on the bottom sine geometry seems optimal.

5.2 Clover Geometry

The next geometry we are exploring is defined by the curves

$$\begin{aligned} \partial\Omega_1 &= \begin{bmatrix} x \\ -\kappa\sin(\pi x) \end{bmatrix}, \quad \partial\Omega_3 = \begin{bmatrix} x \\ 1+\kappa\sin(\pi x) \end{bmatrix}, \quad x \in [0,1] \\ \partial\Omega_2 &= \begin{bmatrix} 1+\kappa\sin(\pi y) \\ y \end{bmatrix}, \quad \partial\Omega_4 = \begin{bmatrix} -\kappa\sin(\pi y) \\ y \end{bmatrix}, \quad y \in [0,1]. \end{aligned} \tag{5.4}$$

For this geometry the difficulty parameter κ does not have the same natural range limitations as for the bottom sine geometry. The boundary curves will never intersect regardless of the value of κ, as long as it is larger than 0. However, we will mainly focus on κ in the range $[0,\ 1]$. If $\kappa = 0$, the domain is simply the square $(x, y) = [0,\ 1]^2$. For higher values of κ, the angle between two neighboring domain defining curves become sharper, and the geometry becomes more difficult to parameterize.

Figure 5.17 shows the geometry when $\kappa = 0.5$. Once again, we will be using the same polynomial order, $p = q = 2$ and uniform knot vectors $\Xi = \mathcal{H}$ with $n = 20$ in all the methods. The material constants are set to $E = 1200$ and $\nu = 0.2$ to reflect a more rigid material compared with the previous geometry. This setting proved to give better results.

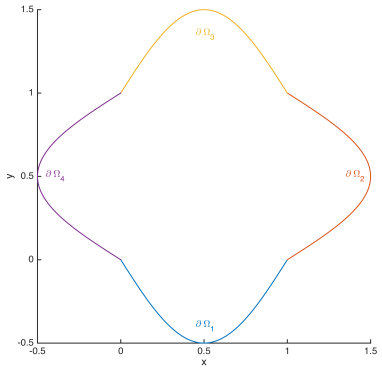
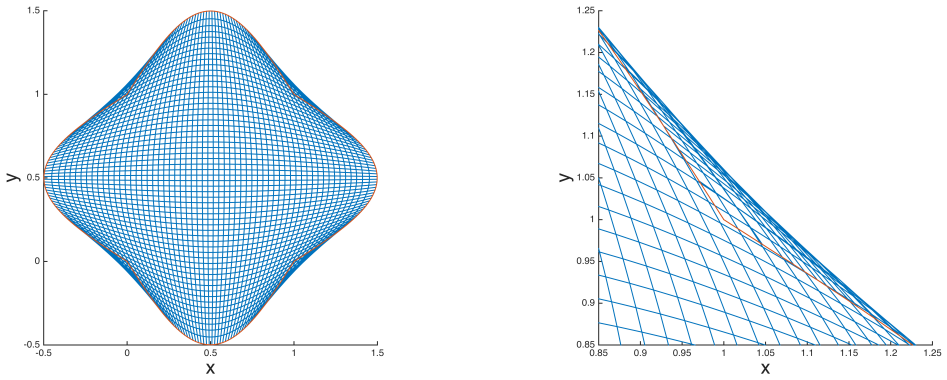


Figure 5.17: Curves defining the physical domain Ω of the clover geometry with $\kappa = 0.5$.

5.2.1 Gordon-Hall

When the Gordon-Hall algorithm is applied to the clover geometry with $\kappa = 0.5$ it generates the parameterization shown in Figure 5.18.



(a) Parameterization mesh (b) Magnified upper right corner

Figure 5.18: **Gordon-Hall:** Parameterization obtained with the Gordon-Hall algorithm on the clover geometry when $\kappa = 0.5$.

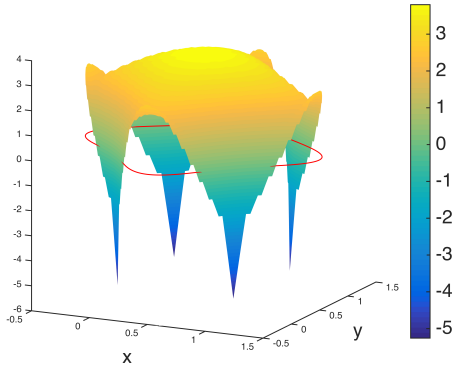
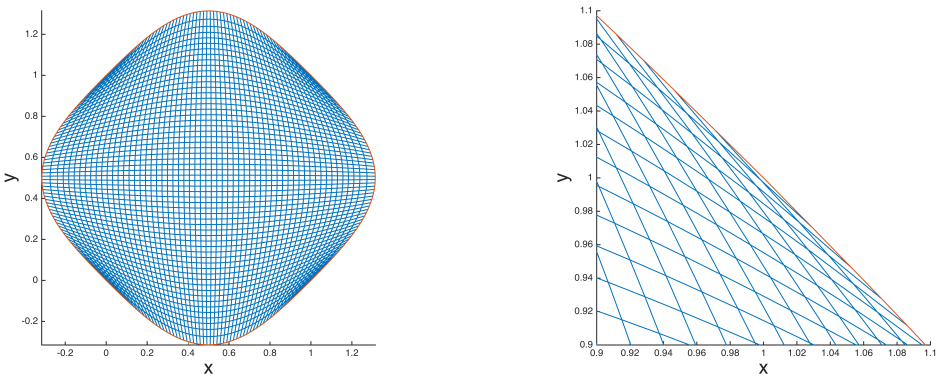


Figure 5.19: **Gordon-Hall:** Jacobian of the mesh obtained with the Gordon-Hall algorithm on the clover geometry when $\kappa = 0.5$.

The parameterization is invalid because of the several parameterization points that lie outside the domain given by the red lines. Figure 5.18b shows an enlarged portion of the geometry where the parameterization is invalid, namely the upper right corner where the curves $\partial\Omega_2$ and $\partial\Omega_3$ meet. This is also clearly confirmed by the Jacobian in Figure 5.19, which is negative in all the four corners of the geometry.

In fact, it turns out that the Gordon-Hall algorithm is only capable of producing valid meshes on this geometry for κ in the range $[0, 0.315]$. Figure 5.20 shows the results when Gordon-Hall is applied to the problem with $\kappa = 0.315$, with the corresponding Jacobian in Figure 5.21.



(a) Parameterization mesh

(b) Magnified upper right corner

Figure 5.20: **Gordon-Hall:** Parameterization obtained through the Gordon-Hall algorithm on the clover geometry when $\kappa = 0.315$.

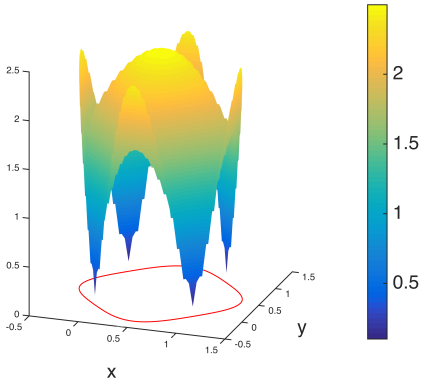
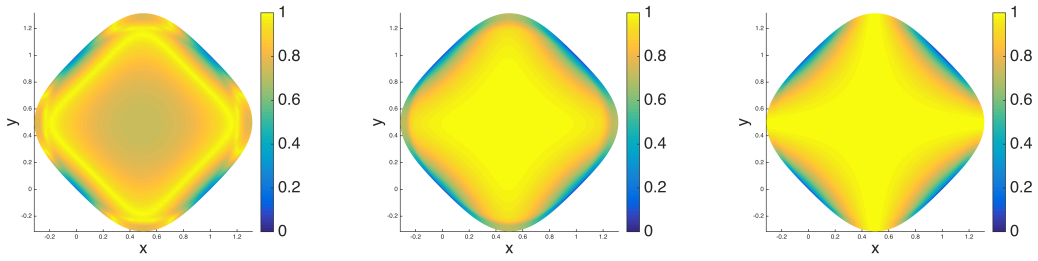


Figure 5.21: **Gordon-Hall:** Jacobian of the mesh obtained with the Gordon-Hall algorithm on the clover geometry when $\kappa = 0.315$.

Although the parameterization in this case is valid in the sense that all parameterization points are inside the domain and the Jacobian is positive, it still has some problem areas.

In the magnified version of the parameterization in Figure 5.20b we see that several of the elements are quite poor and some are even close to being degenerate.

Figure 5.22 shows illustrations of the distribution of the value of the size, shape and skew metric on the parameterization. It is not surprising to see that the elements around the areas where the boundary curves meet are particularly poor, while the elements in the middle of the geometry are of a much higher quality.



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.22: **Gordon-Hall:** Metrics of the parameterization generated by the Gordon-Hall algorithm on the clover geometry with $\kappa = 0.315$.

Table 5.5: **Gordon-Hall:** Measure of mesh quality for the Gordon-Hall parameterization on the clover geometry with $\kappa = 0.315$.

k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0.1656	0.0352	0.0352	0.0061	0.0058
$RMS(\mathcal{M}_k)$	0.8251	0.8464	0.8699	0.7227	0.7443

This is emphasized in Table 5.5, which shows the value of the min-max and root mean square measures of each metric.

The measures reveal that even though the parameterization mostly consists of elements of a high quality, the elements around each of the four corners are poor. This is especially evident in the min-max measure, which takes on a low value for all the metrics. The root mean square measure are quite high for all the metrics, implying that the mesh overall holds a high quality.

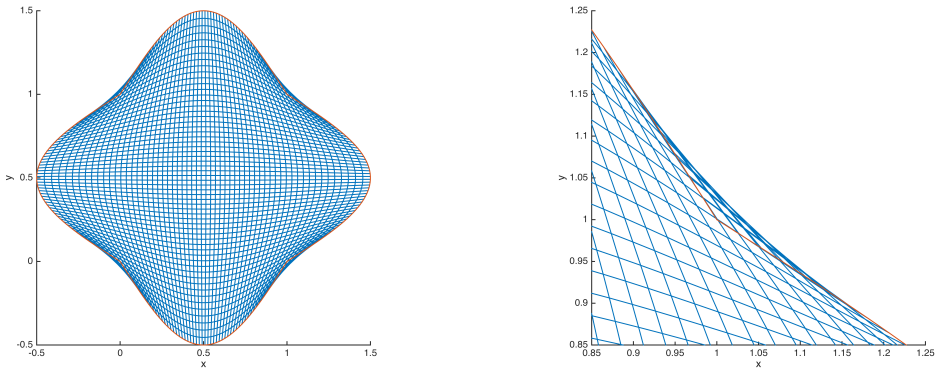
5.2.2 Uncoupled Poisson

Next, we use the uncoupled Poisson method with constant gamma function on the clover geometry with $\kappa = 0.5$. The results are shown in Figure 5.23.

It can be seen clearly in the magnified version of the parameterization in Figure 5.23b that the parameterization is invalid. This is also confirmed by the Jacobian, shown in Figure 5.24, which becomes negative around the areas where the boundary curves meet.

We modify the function $\gamma(x, y)$ in the assembly of the stiffness matrix to see if this can help produce a valid parameterization of the domain. The best version of γ we have been able to find is

$$\gamma(x, y) = (y - y^2)(x - x^2). \tag{5.5}$$



(a) Parameterization mesh (b) Magnified upper right corner

Figure 5.23: **Uncoupled Poisson:** Parameterization obtained with the uncoupled Poisson method on the clover geometry when $\gamma = 1$ and $\kappa = 0.5$.

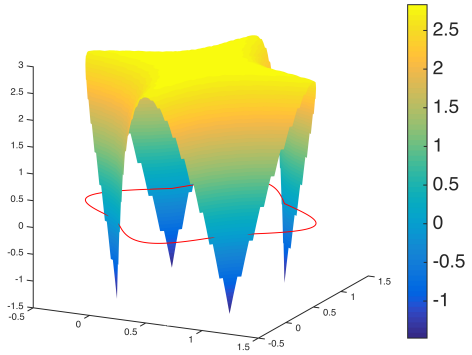
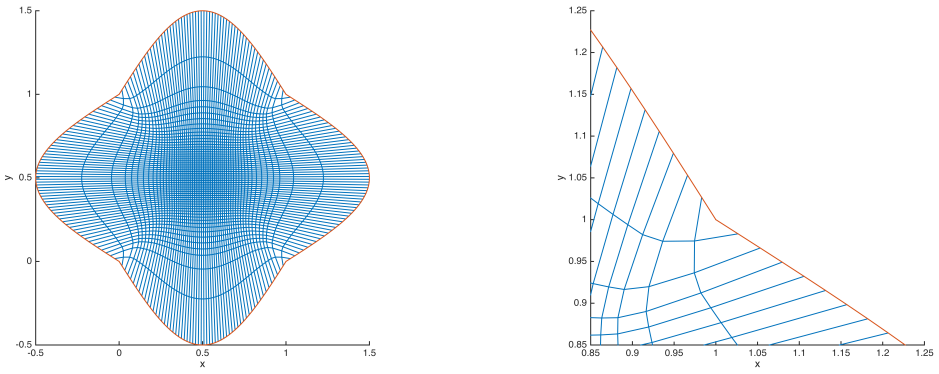


Figure 5.24: **Uncoupled Poisson:** Jacobian of the parameterization obtained with the uncoupled Poisson method on the clover geometry using a constant γ.

When this modification is incorporated in the parameterization procedure the results illustrated in Figure 5.25 are obtained. The Jacobian of this parameterization is shown in Figure 5.26.

The figure shows that although there are no longer parameterization points located outside the domain, the Jacobian is not strictly positive for all points in the domain. The Jacobian is mainly positive, except it becomes negative in the four points $(x, y) = \{(0, 0), (0, 1), (1, 0)(1, 1)\}$, i.e. at the points in which the domain boundary curves meet. One of the four elements containing such a point can be seen in Figure 5.25b. The element is arrow-shaped, and such elements will always result in a negative Jacobian.



(a) Mesh (b) Magnified upper right corner

Figure 5.25: **Uncoupled Poisson:** Parameterization obtained with the uncoupled Poisson method on the clover geometry when γ is given by (5.5) and $\kappa = 0.5$.

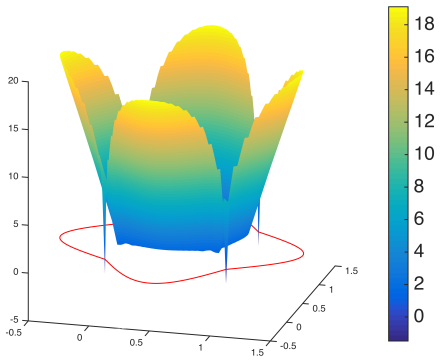
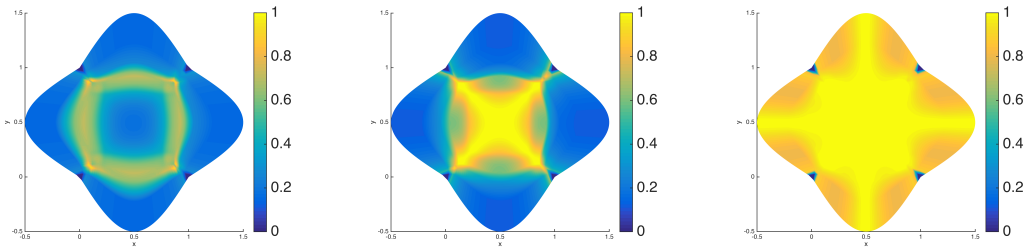


Figure 5.26: **Uncoupled Poisson:** Jacobian of the parameterization obtained with the modified uncoupled Poisson method.

As discussed in Chapter 2, arrow-shaped elements are not contained in the domain of the mesh metrics we are operating with. Elements of this kind need special handling. For techniques on how to handle such elements, the reader is referred to [31]. Handling of special elements is beyond the scope of this thesis, and we will simply define the metrics of such elements to be zero, that is, they are considered degenerate. The metrics of the parameterization are shown in Figure 5.27.

Figure 5.27a shows that a large amount of the elements are considered too large or too small by the size metric. The shape metric in Figure 5.27b shows that the elements in the middle of the geometry have a good shape, while the



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.27: **Uncoupled Poisson:** Metrics of the parameterization generated by the uncoupled Poisson method.

Table 5.6: **Uncoupled Poisson:** Measure of mesh quality for the mesh resulting from running the uncoupled Poisson method on the clover geometry with $\kappa = 0.5$ and γ as given in (5.5).

k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0	0	0	0	0
$RMS(\mathcal{M}_k)$	0.4252	0.8267	0.9858	0.3263	0.4211

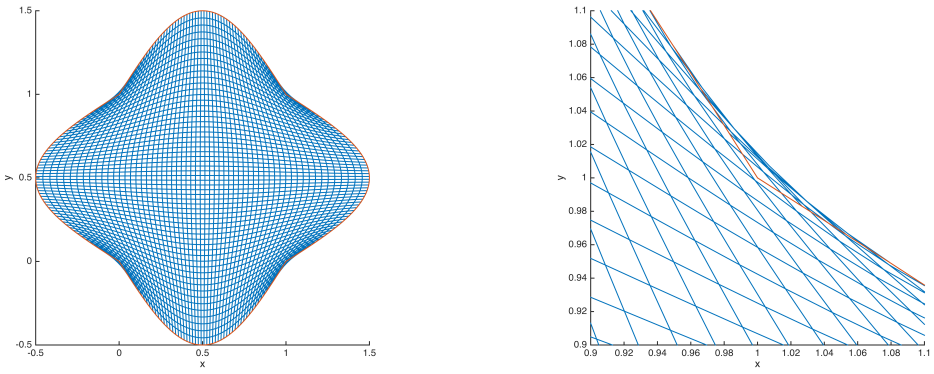
elements along the boundaries are considered badly shaped. Finally, the skew metric in Figure 5.27c reveals that the parameterization produces elements with a good skew. The only elements showing bad skew are the ones at the boundary corners, which by definition are degenerate.

Table 5.6 shows the measures of the different metrics. Logically, the min-max measure is zero for all the metrics because of the four degenerate elements in the corners. The root mean square measure corresponds nicely with what we have just observed from Figure 5.27, that is, the parameterization is quite poor from a size perspective, but is better from a shape perspective and very good when considering the skew.

5.2.3 Linear Elasticity

We are now ready to test how the linear elasticity approach perform on the clover geometry. We start by letting $\kappa = 0.5$, and once again leaving the elasticity matrix constant as given in (4.22), to yield the results illustrated in Figure 5.28.

The magnified version of the parameterization in Figure 5.28b and the Jacobian in Figure 5.29 leave no doubt that the parameterization is invalid. We therefore modify the elasticity matrix to be a function of x and y in order to try to force the parameterization to stay inside the domain.



(a) Parameterization mesh (b) Magnified upper right corner

Figure 5.28: **Linear elasticity:** Parameterization obtained with the linear elasticity methodon the clover geometry with constant elasticity matrix.

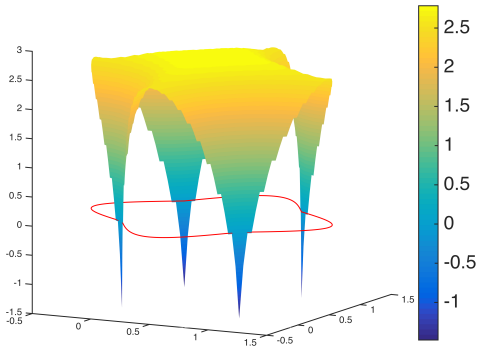


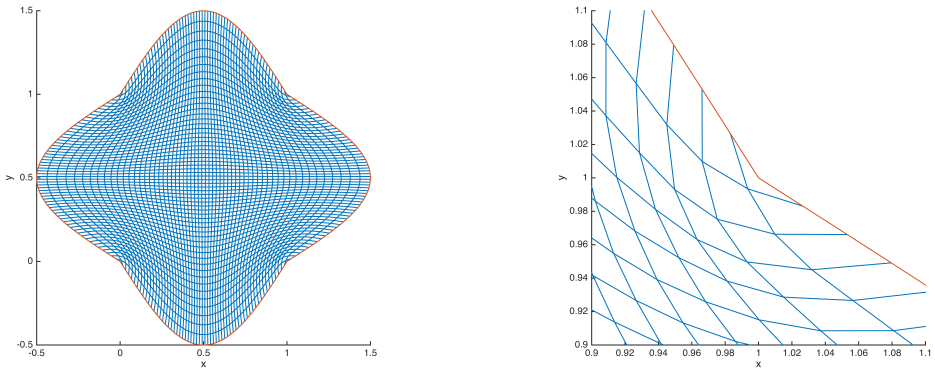
Figure 5.29: **Linear elasticity:** Jacobian of the parameterization obtained with the linear elasticity method on the clover geometry with constant elasticity matrix.

The best expression of the elasticity matrix for the clover geometry is given by

$$\mathbf{D}(x,y) = \frac{E\big((x-x^2)+(y-y^2)\big)}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}. \tag{5.6}$$

With this modification, the linear elasticity method produces the mesh in Figure 5.30. The Jacobian of this parameterization is shown in Figure 5.31.

From the figure we make the same observation as in the previous section about the positivity of the Jacobian, namely that it is generally positive, with the sole exception of the four corner points where the boundary curves meet. This problem is rooted in the geometry itself, and therefore it occurs also in this parameterization.



(a) Parameterization mesh (b) Magnified upper right corner

Figure 5.30: **Linear elasticity:** Parameterization obtained with the linear elasticity method on the clover geometry when $\mathbf{D}(x, y)$ is given by (5.6).

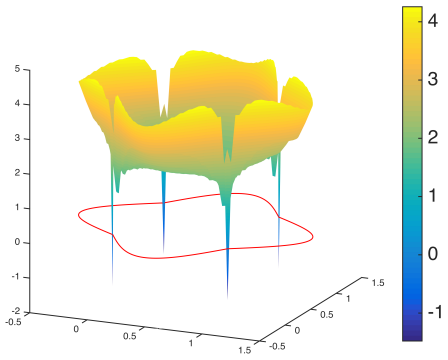
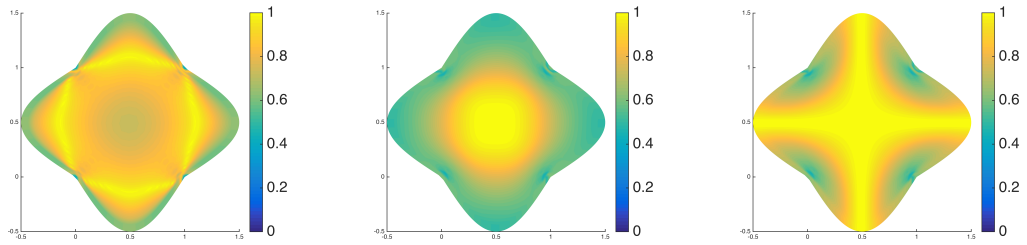


Figure 5.31: **Linear elasticity:** Jacobian of the parameterization obtained with the linear elasticity method with non-constant elasticity matrix.

We handle it in the same way as before by defining the arrow shaped element containing the point with the negative Jacobian value to be degenerate and in requirement of special handling.

We proceed to find the metrics of the parameterization. These are shown in Figure 5.32. The figures show that the four corner elements are degenerate in all metrics. The size metric finds most elements to be of a satisfactory size, with exception of the elements along the boundary. The shape metric reveals that the elements at the very center of the domain has a good shape, with decreasing quality outward towards the boundary. The skew metric shows that, aside from the elements around the corner points, the mesh consists of elements of a satisfactory



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.32: **Linear elasticity:** Metrics of the parameterization generated by the linear elasticity method with modified elasticity matrix.

Table 5.7: **Linear elasticity:** Measure of mesh quality for the mesh given by the linear elasticity method with non-constant elasticity matrix.

k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0	0	0	0	0
$RMS(\mathcal{M}_k)$	0.8344	0.7455	0.8659	0.6184	0.7199

skew.

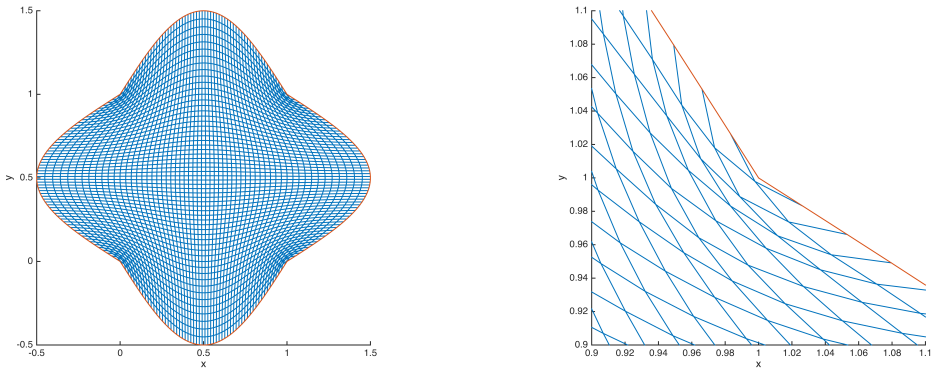
These results are also reflected in the measures presented in Table 5.7. The root mean square measure show that even though some elements are degenerate, the mesh as a whole is of a quite high quality in all metrics. The root mean square measure holds a high value for all the metrics, and especially for the size and skew metric. Since the pure metrics contain high values, this is transferred to the combination root mean square measures, which also show high values.

Since we have four degenerate elements, the min-max measure becomes zero for all the metrics.

5.2.4 Quasistatic

Finally, we test the performance of the quasistatic method on the clover geometry. The results when the quasistatic method is applied with 15 steps and $\kappa = 0.5$ are shown in Figure 5.33. Figure 5.33a shows the parameterization of the entire geometry, while Figure 5.33b shows the parameterization enlarged around the upper right corner of the geometry.

We can see that the parameterization does not experience any of the problems that occurred in the previous methods with parameterization points on the outside of the domain. However, as shown in Figure 5.34, the parameterization includes elements for which the Jacobian is negative. Based on the results from the previous methods, it is not surprising that these elements are in the corners where two



(a) Parameterization mesh (b) Magnified upper right corner

Figure 5.33: **Quasistatic:** Parameterization obtained with the quasistatic method on the clover geometry through 15 iterations.

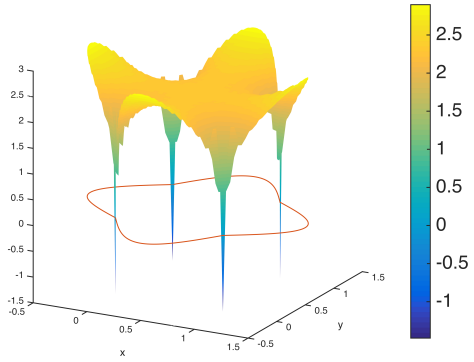


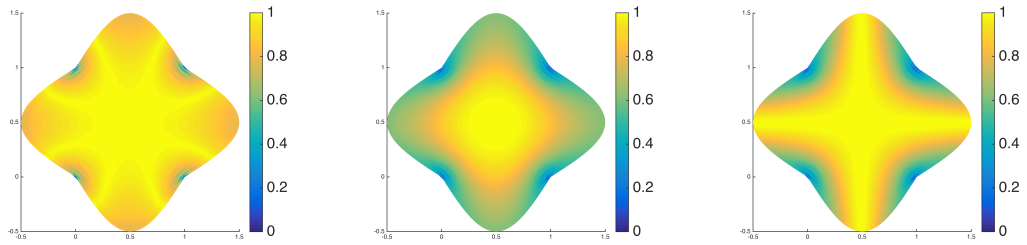
Figure 5.34: **Quasistatic:** Jacobian of the parameterization obtained with the quasistatic method on the clover geometry.

boundaries meet.

Proceeding in the same fashion as before, we define the four corner elements as degenerate, and find the metrics of the mesh. These are presented in Figure 5.35.

The size metric in Figure 5.35a shows that the parameterization results in a mesh where the vast majority of the elements have a satisfactory size. The only poor elements are located at or in the immediate vicinity of the four corner points.

The shape metric in 5.35b implies that a relatively large amount of elements have a good shape, although the overall profile is not as good as that of the size metric. The elements of best shape are located in the middle of the geometry, while the worst elements once again are located around the corner points.



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.35: **Quasistatic:** Metrics of the parameterization generated by the quasistatic method on the clover geometry.

Table 5.8: **Quasistatic:** Measure of mesh quality for the mesh generated by the quasistatic method on the clover geometry.

k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0	0	0	0	0
$RMS(\mathcal{M}_k)$	0.9304	0.7552	0.8429	0.7211	0.7980

Lastly, the skew metric in Figure 5.35c also gives good results for most of the elements, excluding the corner points and their closest surroundings.

Table 5.8 displays the min-max measure and the root mean square measure of the parameterization. The root mean square measures show that the parameterization generally consists of high quality elements. The min-max measures become zero due to the degeneracy of the elements located in each of the four boundary corners.

The parameterization generated by the quasistatic method was obtained through 15 iterations. In order to evaluate what the optimal number of iterations is, the problem is solved several times with different number of iterations. Figure 5.36 shows how the root mean square measures change with the number of iterations used to make the parameterization.

We can see that the root mean square measure holds a relatively steady value, independent of the number of iterations, as long as we perform more than three iterations. This means that if we attempt to parameterize the clover geometry in less than three iterations, we will end up with an invalid parameterization, i.e. with mesh lines outside the domain.

The min-max measures of the parameterizations remained zero for all the metrics regardless of the number of iterations performed. This is a result of the degeneracy of the arrow shaped elements in the boundary corners.

Based on the iteration results in Figure 5.36 we conclude that we could have

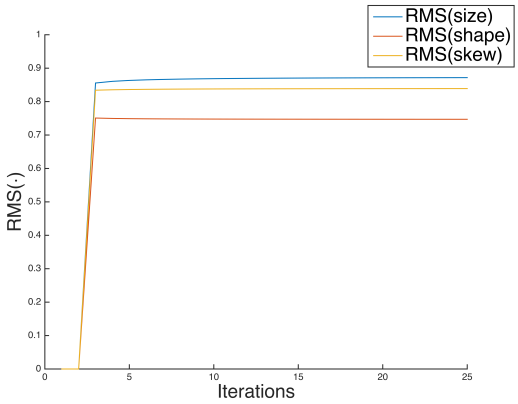


Figure 5.36: **Quasistatic:** Root mean square measures as functions of iterations used to find the parameterization of the clover geometry with $\kappa = 0.5$ using the quasistatic method. The size metric is shown in blue, while the shape and skew metric are shown in red and yellow respectively.

used far less than 15 iterations to obtain the parameterization in Figure 5.33a.

5.3 Jigsaw Geometry

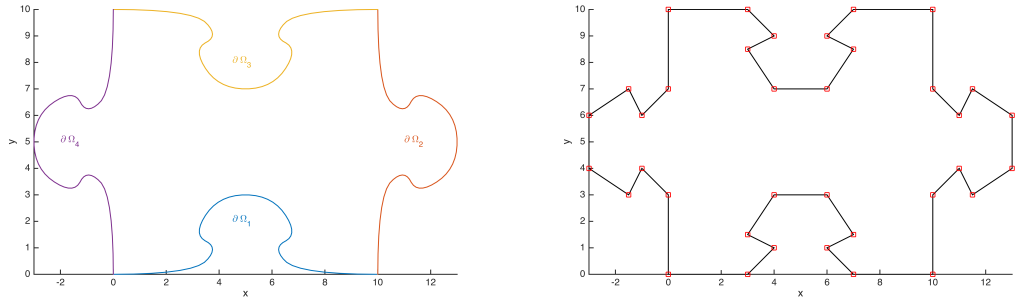
The final geometry we are going to test is shaped as a piece from a jigsaw puzzle. The geometry is the same as Gravesen et al. used in [7]. The geometry is obtained through a spline representation with suitable control points.

Figure 5.37 shows the geometry and the control points we are going to use to test the methods. Figure 5.37a shows how the boundary of the geometry is divided into four edges and Figure 5.37b shows the control polygon in black with control points in red. The control points and further specifications of this geometry can be found in Appendix B.

As for the previous geometries, we will also be operating with a difficulty parameter κ for this geometry, but contrary to the other two geometries we have studied, this parameter is not as straight forward to define for the jigsaw geometry. Instead of simply having a κ connected to the amplitude of a sine function, the parameter is connected with the placement of the control points.

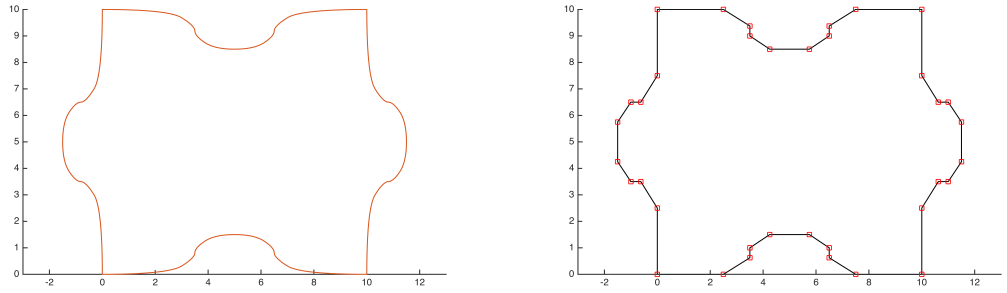
The geometry in Figure 5.37a is defined as having $\kappa = 1$, and any other geometry is defined by the control points found by multiplying the associated κ-value to the difference between the control points shown in Figure 5.37b and points giving a square.

Figure 5.38 shows the geometry and associated control points for $\kappa = 0.5$. This geometry is obviously easier to parameterize compared to the geometry with the higher κ-value.



(a) Jigsaw geometry (b) Control polygon

Figure 5.37: Jigsaw geometry in spline representation and control points with $\kappa = 1$.



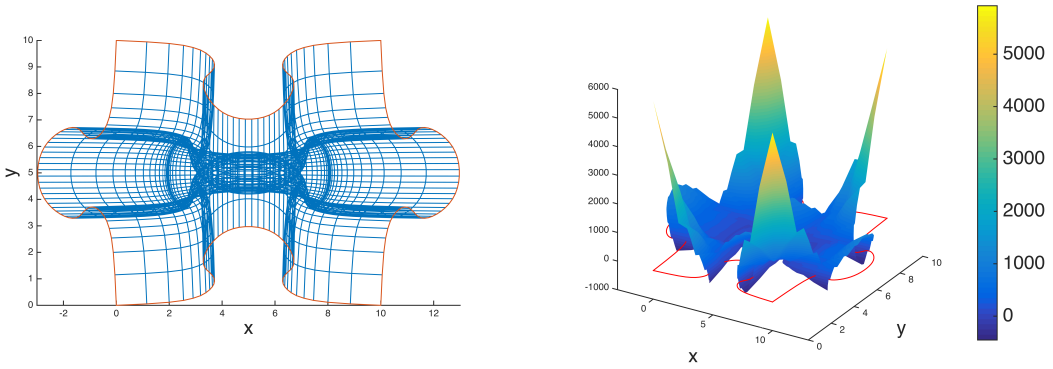
(a) Jigsaw geometry (b) Control polygon

Figure 5.38: Jigsaw geometry in spline representation and control points for $\kappa = 0.5$.

When testing the different methods on the jigsaw geometry we will use the geometry shown in Figure 5.37. Unless otherwise stated, we use the specifications $p = q = 2$, $\Xi = \mathcal{H}$ with $n = 20$, $E = 1200$, and $\nu = 0.2$ for all the methods.

5.3.1 Gordon-Hall

The first method to be tested on the jigsaw geometry is the Gordon-Hall method. The results are presented in Figure 5.39.



(a) Parameterization mesh (b) Jacobian

Figure 5.39: **Gordon-Hall:** Parameterization and corresponding Jacobian obtained with the Gordon-Hall algorithm on the Jigsaw geometry.

Figure 5.39b shows the Jacobian of the parameterization, which contains several negative areas. It is evident that the Gordon-Hall algorithm is unable to find a valid parameterization of this geometry.

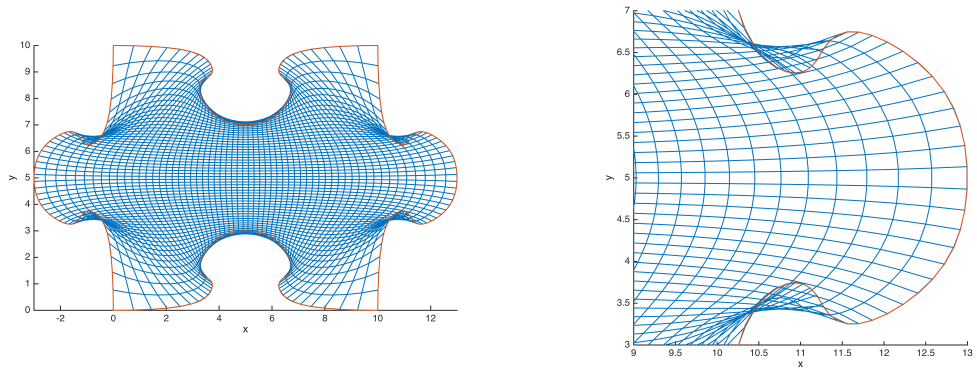
In fact, the method is unable to parameterize the domain in a satisfactory way for any $\kappa \geq 0.4$. The geometry needs to be much easier for the method to be able to produce a valid parameterization. We conclude that the Gordon-Hall algorithm is not suitable to parameterize geometries of this kind.

5.3.2 Uncoupled Poisson

Next, we investigate if the uncoupled Poisson method performs better on the jigsaw geometry. The results are presented in Figure 5.40.

Although the resulting mesh is not as bad as the one obtained with the Gordon-Hall approach, it is still invalid due to the parameterization points outside the domain, occurring around all the jigsaw joints. This is clearly shown in the enlarged portion of the mesh shown in Figure 5.40b, and is confirmed by the Jacobian in Figure 5.41, which becomes negative in several areas.

The natural next step is to try to find a $\gamma(x, y)$-function that can enable a valid parameterization. However, despite persevere attempts, we are not able to find such a function for this specific parameterization problem, and thus the method is unable to obtain a valid mesh.



(a) Parameterization mesh (b) Right jigsaw joint

Figure 5.40: **Uncoupled Poisson:** Parameterization obtained with the uncoupled Poisson method on the jigsaw geometry with constant γ.

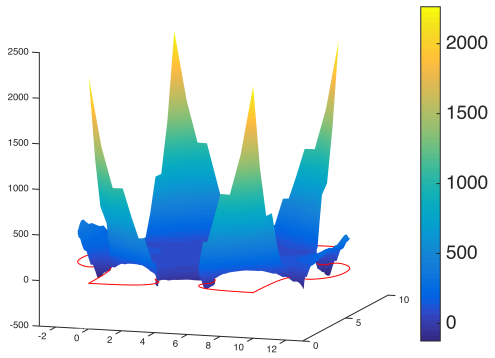
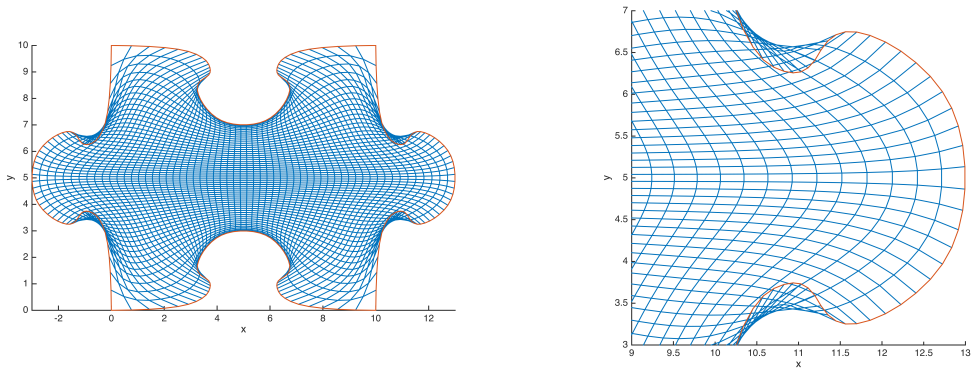


Figure 5.41: **Uncoupled Poisson:** Jacobian of the parameterization obtained with the uncoupled Poisson method on the jigsaw geometry.

5.3.3 Linear Elasticity

The penultimate method for the jigsaw geometry is the linear elasticity method. The resulting parameterization when the elasticity matrix is kept constant is presented in Figure 5.42, and the Jacobian is presented in Figure 5.43.

Once again the parameterization is not valid due to parameterization points laying outside of the boundary in the neighborhood of the four jigsaw joints. Figure 5.43 confirms this by exposing negative values in the Jacobian in these areas.



(a) Parameterization mesh

(b) Right jigsaw joint

Figure 5.42: **Linear elasticity:** Parameterization obtained with the linear elasticity method on the jigsaw geometry with constant elasticity matrix

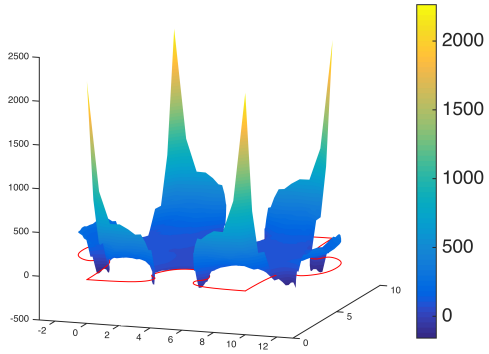
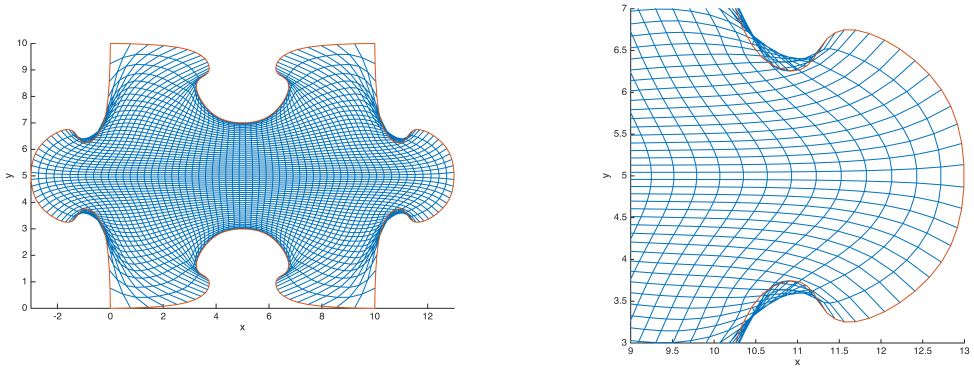


Figure 5.43: **Linear elasticity:** Jacobian of the parameterization obtained with the linear elasticity method on the jigsaw geometry with constant elasticity matrix.

We proceed by attempting to modify the elasticity matrix in order to obtain a valid parameterization. Our usual approach of finding an expression for $\mathbf{D}(x, y)$ dependent on the x- and y-coordinates of the geometry proves to be fruitless.

Instead we try to utilize the Jacobian of the parameterization of the geometry. That is, after attempting to parameterize the geometry by the linear elasticity method with constant elasticity matrix, we use the Jacobian of this parameterization in the elasticity matrix so that $\mathbf{D} = \mathbf{D}(\det(\mathbf{J}))$, and try to solve the problem again. The elasticity matrix is set to correspond to a rigid material where $\det(\mathbf{J})$ has a high value, and to a more elastic material for lower values of $\det(\mathbf{J})$.

The results are shown in Figure 5.44. The parameterization, although more



(a) Parameterization mesh (b) Right jigsaw joint

Figure 5.44: **Linear elasticity:** Parameterization obtained with the linear elasticity method on the jigsaw geometry when $\mathbf{D}$ is no longer constant.

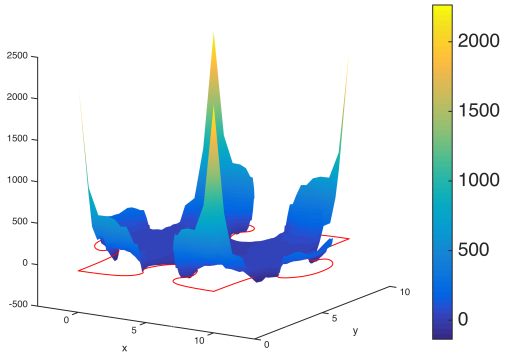


Figure 5.45: **Linear elasticity:** Jacobian of the parameterization obtained with the linear elasticity method on the jigsaw geometry when the elasticity matrix is no longer constant.

restricted to the inside of the domain Ω, is still not valid as the parameterization results in mesh lines outside of the domain and negative Jacobian values as shown in Figure 5.45.

The fact that we are not able to produce valid parameterizations with neither the uncoupled Poisson nor the linear elasticity methods, emphasizes these methods' dependence on finding good expressions for $\gamma(x, y)$ and $\mathbf{D}(x, y)$.

5.3.4 Quasistatic

Finally, we apply the quasistatic approach to the jigsaw geometry. The parameterization result when the quasistatic method is run on the geometry with 15 iterations is presented in Figure 5.46.

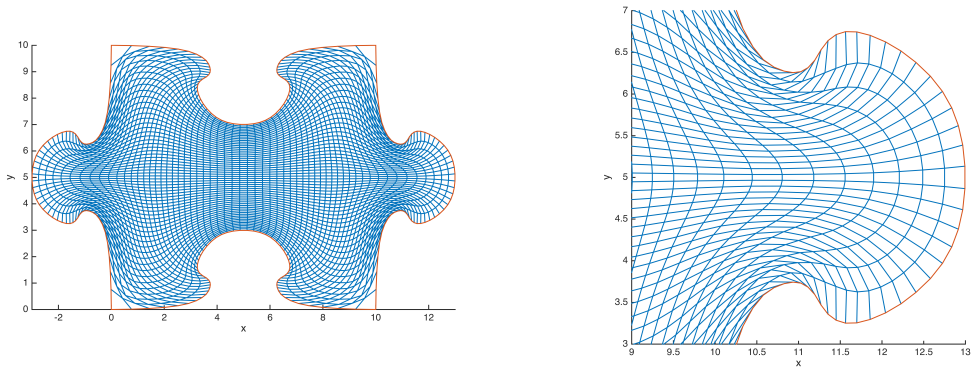
Figure 5.46a shows the entire mesh, while Figure 5.46b shows a magnified version of the right jigsaw joint. This parameterization does not encounter problems with parameterization points outside of the domain. However, the Jacobian is still negative at a few points along the left and right joints. This is shown in Figure 5.47a.

The Jacobian drops just below zero for two elements on each side of the left and right jigsaw joints. Figure 5.47b shows the two elements containing a negative Jacobian on the lower side of the right joint in yellow.

Upon closer inspection it is revealed that all the eight elements containing points resulting in a negative Jacobian are actually arrow shaped and thus quite similar to the elements causing problems in the clover geometry.

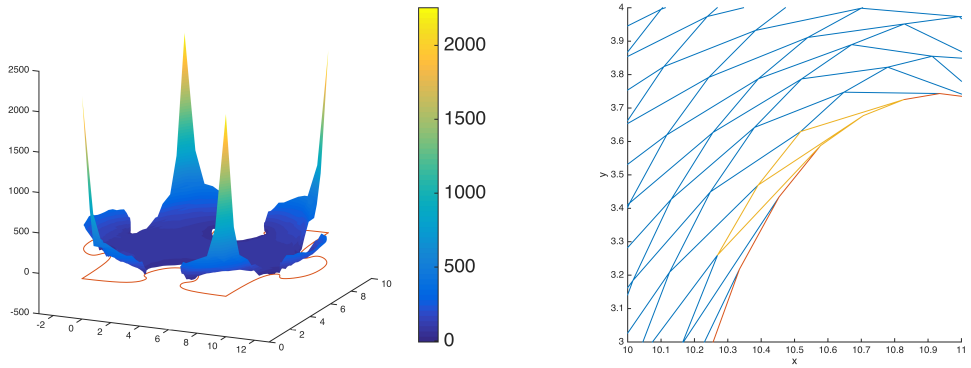
We want to avoid getting these negative elements, and instead get an all positive Jacobian. We try to elevate the polynomial degree by one, so that $p = 3$. Then, the knot vectors get two more elements each, and the number of degrees of freedom is also increased. The results from this order elevation are shown in Figure 5.48, with the full mesh to the left and a magnified version to the right.

This parameterization is valid, since all mesh lines are contained inside the domain, and the Jacobian is strictly positive, as can be seen in Figure 5.49.



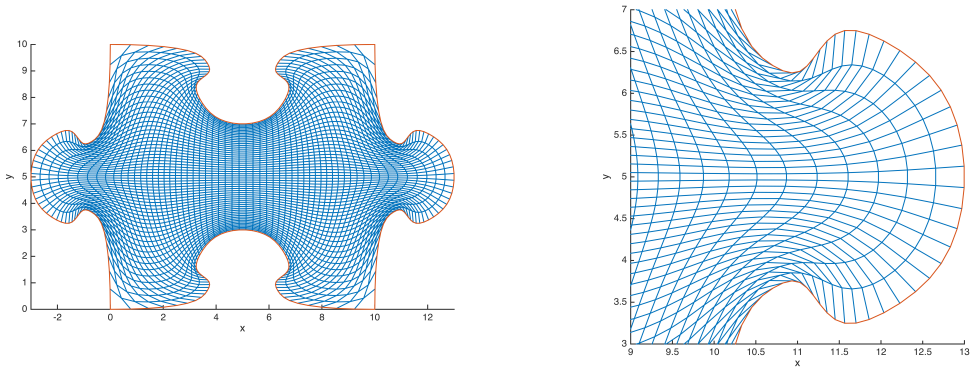
(a) Parameterization mesh (b) Right jigsaw joint

Figure 5.46: **Quasistatic:** Parameterization obtained with the quasistatic approach on the jigsaw geometry.



(a) Jacobian (b) Elements causing negative Jacobian

Figure 5.47: **Quasistatic:** Jacobian of the parameterization obtained with the quasistatic method and mesh zoomed in on the elements resulting in a negative Jacobian. The problem elements are shown in yellow.



(a) Parameterization mesh (b) Magnified right joint.

Figure 5.48: **Order elevated quasistatic:** Parameterization obtained with the quasistatic method on the jigsaw geometry when the polynomial order is set to $p = 3$.

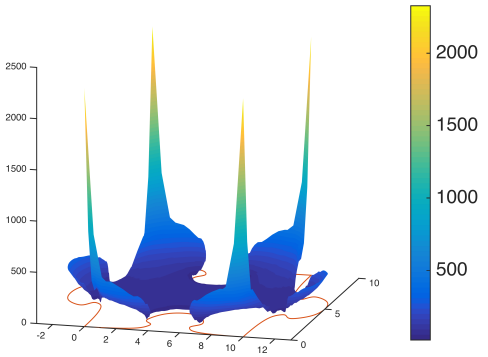
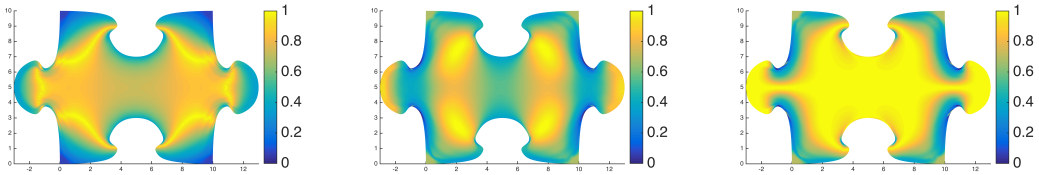


Figure 5.49: **Order elevated quasistatic:** Jacobian of the parameterization obtained with the quasistatic method on the jigsaw geometry when the polynomial order is set to $p = 3$.



(a) Size metric (b) Shape metric (c) Skew metric

Figure 5.50: **Order elevated quasistatic:** Metrics of the parameterization generated by the quasistatic method when $p = 3$.

Table 5.9: **Order elevated quasistatic:** Measure of mesh quality for the parameterization obtained on the jigsaw geometry with the quasistatic method when $p = 3$.

k	$\mathcal{M}_{Size}$	$\mathcal{M}_{Shape}$	$\mathcal{M}_{Skew}$	$\mathcal{M}_{SizeShape}$	$\mathcal{M}_{SizeSkew}$
$MM(\mathcal{M}_k)$	0	0	0	0	0
$RMS(\mathcal{M}_k)$	0.7462	0.6816	0.8815	0.5322	0.6661

Now that we have a valid parameterization, we use the mesh metrics defined in Chapter 2 to obtain the characteristics of the mesh. The metrics of the mesh in Figure 5.48a are presented in Figure 5.50.

Unsurprisingly, the large elements in each corner of the mesh is labeled poor by the size metric, shown in Figure 5.50a, but aside from these elements, it seems that the mesh is deemed rather good in the rest of the geometry. The shape metric in Figure 5.50b shows that generally only two areas in the interior of the geometry and the areas the very end of the outward joints are considered satisfactory from a shape point of view. The rest of the mesh is considered to have a lower quality. The skew metric, shown in Figure 5.50c, shows that apart from the four areas along the boundary connecting the jigsaw joints, the parameterization is quite good with regards to skewness.

Table 5.9 contains the measure of each metric for the parameterization. The min-max measure tells us that even though the parameterization is valid, it contains elements that are degenerate from a mesh metric point of view, specifically they have three collinear nodes.

The root mean square measure on the other hand, reveals that the parameterization is relatively good, especially from a size and skew point of view, and consequently also the combination of these. The shape metric is lower than the other pure metrics, and this also influences the size-shape metric. All things con-

sidered, the mesh holds a high quality.

We are interested in the number of iterations needed to arrive at the best possible parameterization of the geometry. This is done in exactly the same way as for the previous quasistatic method cases, namely by using different number of iterations and compare the measures of each parameterization. The results are presented in Figure 5.51.

We can read from the figure that if the quasistatic method is applied to the jigsaw geometry with less than four iterations, the resulting parameterization becomes invalid. That is, a minimum of four iterations are needed to get a valid parameterization implying that more iterations are required for the jigsaw geometry than the bottom sine and clover geometries.

We see that there is a small improvement in all the measures for increasing number of iterations, but the measure rapidly cease to improve with increasing number of iterations.

The min-max measure became zero for all the metrics regardless of the number of iterations. This means that we can not avoid getting degenerate elements by increasing the number of iterations.

As the improvements are minimal for higher number of iterations, we conclude that the 15 iterations used to obtain the parameterization in Figure 5.48a are sufficient for the quasistatic method on the jigsaw geometry.

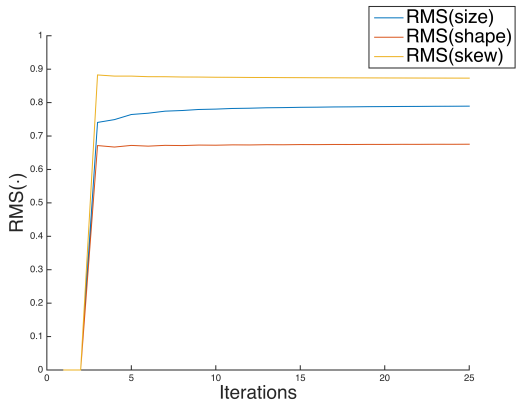


Figure 5.51: **Order elevated quasistatic:** Root mean square measure of the parameterization meshes obtained with the quasistatic method as a function of number of iterations. The size metric is shown in blue, shape metric in red and skew metric in yellow.

5.4 Method Comparison

Now that we have seen how all the methods perform on the three different geometries, it is time to see how they compare with each other overall. The methods can already be compared to some degree on each geometry through the measures presented in the tables under each method. To provide a more comprehensive basis for comparing the methods we have performed additional calculations by varying the difficulty parameter κ and the polynomial degree p to see under which conditions each method is effective.

5.4.1 Performance with respect to κ

We start by examining how the methods perform on the geometries with different κ values. On the bottom sine geometry with amplitude $\kappa = 0.5$ the quasistatic method provided the best mesh in the root mean square measure for all the metrics. That is, the quasistatic mesh had the highest $RMS(\mathcal{M}_k)$, regardless of the metric k. The Gordon-Hall method and uncoupled Poisson method had the worst performance in this measure. At the same time, the Gordon-Hall parameterization provided the highest results in the min-max measure.

In order to get a better impression on how well the methods perform on the bottom sine geometry, they are set to parameterize the domain with different κ-values, ranging from 0 to 1. The results are presented in Figure 5.52.

The figures show that the quasistatic approach has the best general performance as long as this method is able to produce a valid parameterization. For $\kappa > 0.65$, only the Gordon-Hall method is able to provide a valid parameterization, but this is of low quality in the root mean square measure compared with the quasistatic method as long as $\kappa < 0.65$.

From the figure, we see that the min-max measure of the size metric is superior for the Gordon-Hall algorithm, while the min-max measures of the shape and skew metrics are quite similar for all the methods for as long as they are applicable. This is with the exception of the modified uncoupled Poisson and modified linear elasticity methods. These are inferior for all $\kappa \neq 0.5$ in the min-max measures.

This is as expected as the modification functions, $\gamma(x, y)$ and $\mathbf{D}(x, y)$ respectively, have both been optimized for the case $\kappa = 0.5$. They therefore perform poorly on any other geometry specified by a different κ. This is also the reason why these methods produce lower quality meshes on the unit square when $\kappa = 0$, since we no longer have the identity mapping on this geometry.

On the clover geometry with $\kappa = 0.5$ we have already seen that the quasistatic method provide strong results in the root mean square measure, and that the modified uncoupled Poisson method also performs well in the shape and skew metrics.

The clover geometry turned out to be too challenging for the Gordon-Hall method, which was unable to parameterize the domain for this κ-value.

For a more complete comparison of the performance of the methods on this geometry, they are set to parameterize the clover domain for $\kappa \in [0,\ 1]$. The results when we consider the root mean square and min-max measure of the size, shape, and skew metrics of the produced meshes are shown in Figure 5.53.

The figure shows that the Gordon-Hall method has the worst performance, closely followed by the uncoupled Poisson and linear elasticity methods with constant γ and $\mathbf{D}$. These three methods are all unable to produce a valid parameterization of the clover geometry when κ becomes larger than approximately 0.315. This is also the point at which the angle between two neighboring boundary curves becomes too large, causing the arrow shaped elements that destroy the parameterization to occur.

The modified linear elasticity method has a relatively good performance for $\kappa < 0.65$, and actually improves its min-max measure as long as there are no arrow-shaped elements. This is also the case for the modified uncoupled Poisson method, although the improvements in the min-max measure is not as large as for the linear elasticity method.

The modified uncoupled Poisson method is the sole method able to produce valid parameterizations of the geometry with $\kappa = 0.8$, and the root mean square measures of the shape and skew metrics imply high quality meshes from this method. The size metric is however the lowest of all the methods.

Finally, the quasistatic method performs very well on geometries up until $\kappa = 0.75$, after which it is unable to produce valid parameterizations. As long as the method is valid, the meshes it produces performs on a high level in all the metrics, both in the root mean square measure and in the min-max measure.

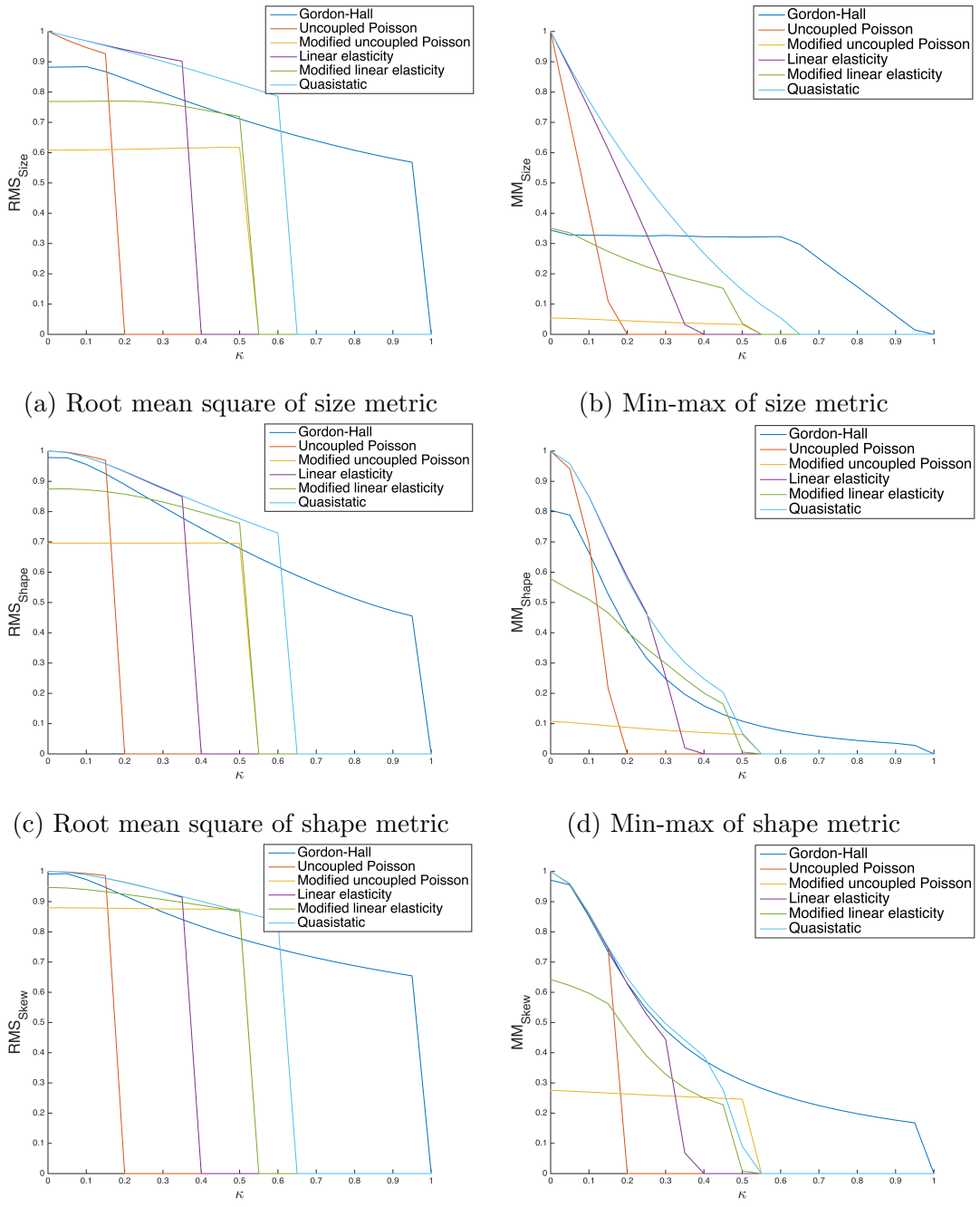
The jigsaw geometry proved to be too much of a challenge for most of the parameterization methods. If we had been able to find suitable functions for $\gamma(x, y)$ and $\mathbf{D}(x, y)$, it is possible that the uncoupled Poisson method and the linear elasticity method would have been able to produce valid parameterizations. The quasistatic approach is the only method capable of parameterizing this domain when $\kappa = 1$.

The performance of the methods for different values of κ on the jigsaw geometry is shown in Figure 5.54. The plots are produced using the same specifications as for the results presented in Section 5.3, i.e. $p = 2$, and $n = 20$ on uniform knot vectors, and material constants corresponding to a quite rigid material.

The figure shows that the Gordon-Hall algorithm is the least persevering and is the first method to become unable to produce a valid mesh of the geometry. With the exception of the skew metric, its performance is also the worst, both in the root mean square measure and the min-max measure.

The uncoupled Poisson and linear elasticity methods have a quite similar performance in the root mean square measure, but the uncoupled Poisson shows better results in the min-max measure.

None of the other methods can compete with the endurance of the quasistatic method. It is capable of parameterizing the domain for far larger values of κ than any of the other methods, and the meshes it produces is of a high quality. The min-max measure reveals that for large κ-values, the shape and skew metrics pick up on some degenerate elements, but the parameterization is still valid until κ exceeds approximately 0.85, at which point we have to elevate the polynomial degree for the method to be able to produce valid parameterization.



(a) Root mean square of size metric

(b) Min-max of size metric

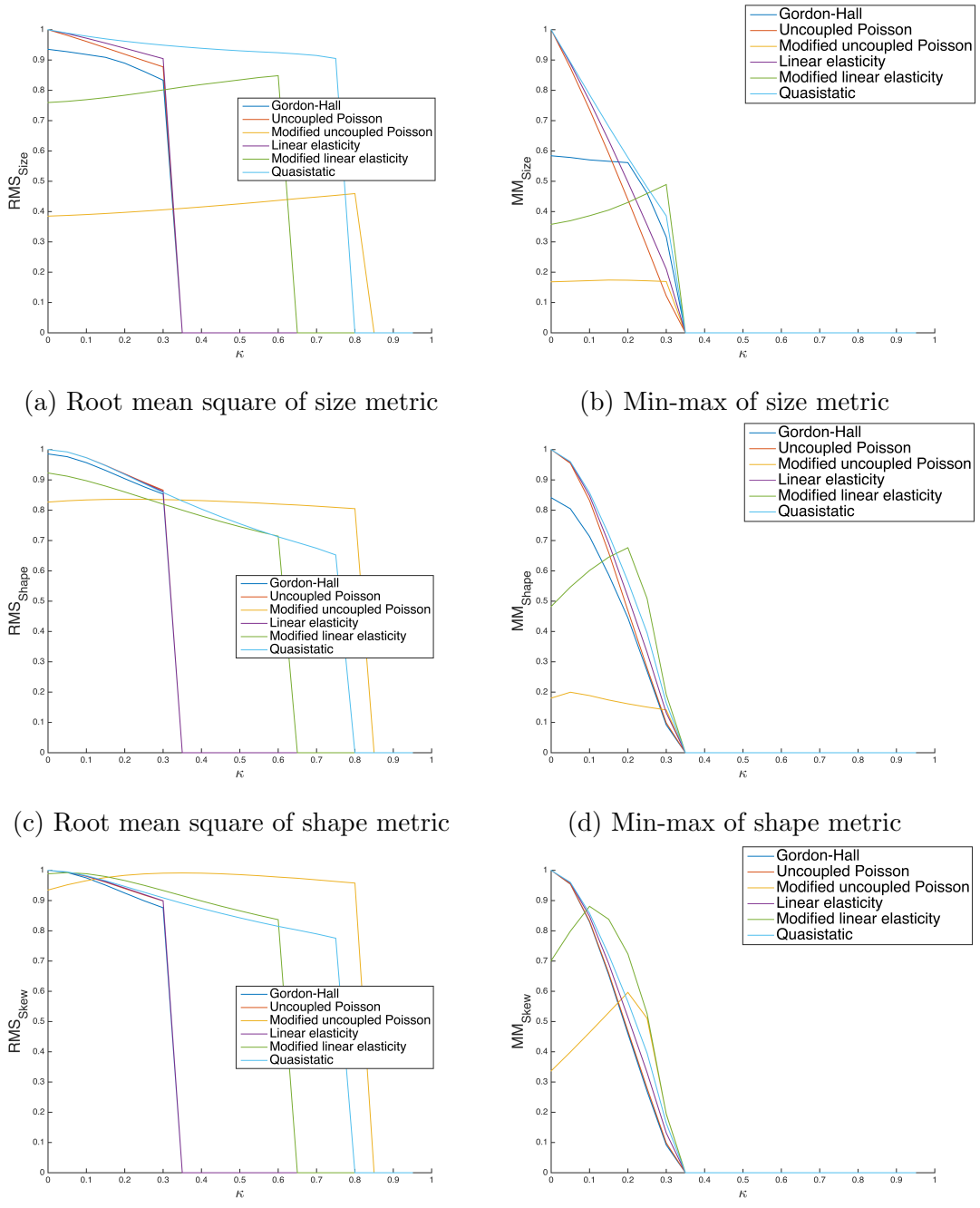
(c) Root mean square of shape metric

(d) Min-max of shape metric

(e) Root mean square of skew metric

(f) Min-max of skew metric

Figure 5.52: **Method comparison on bottom sine geometry:** Root mean square measure (left) and min-max measure (right) with respect to κ for the different methods on the bottom sine geometry in the size, shape, and skew metric. The blue lines belong to the Gordon-Hall method, the red and yellow belong to the uncoupled Poisson without and with a γ-function respectively, the purple and green belong to the linear elasticity method without and with a modified elasticity matrix, and the turquoise belong to the quasistatic method.



(a) Root mean square of size metric

(b) Min-max of size metric

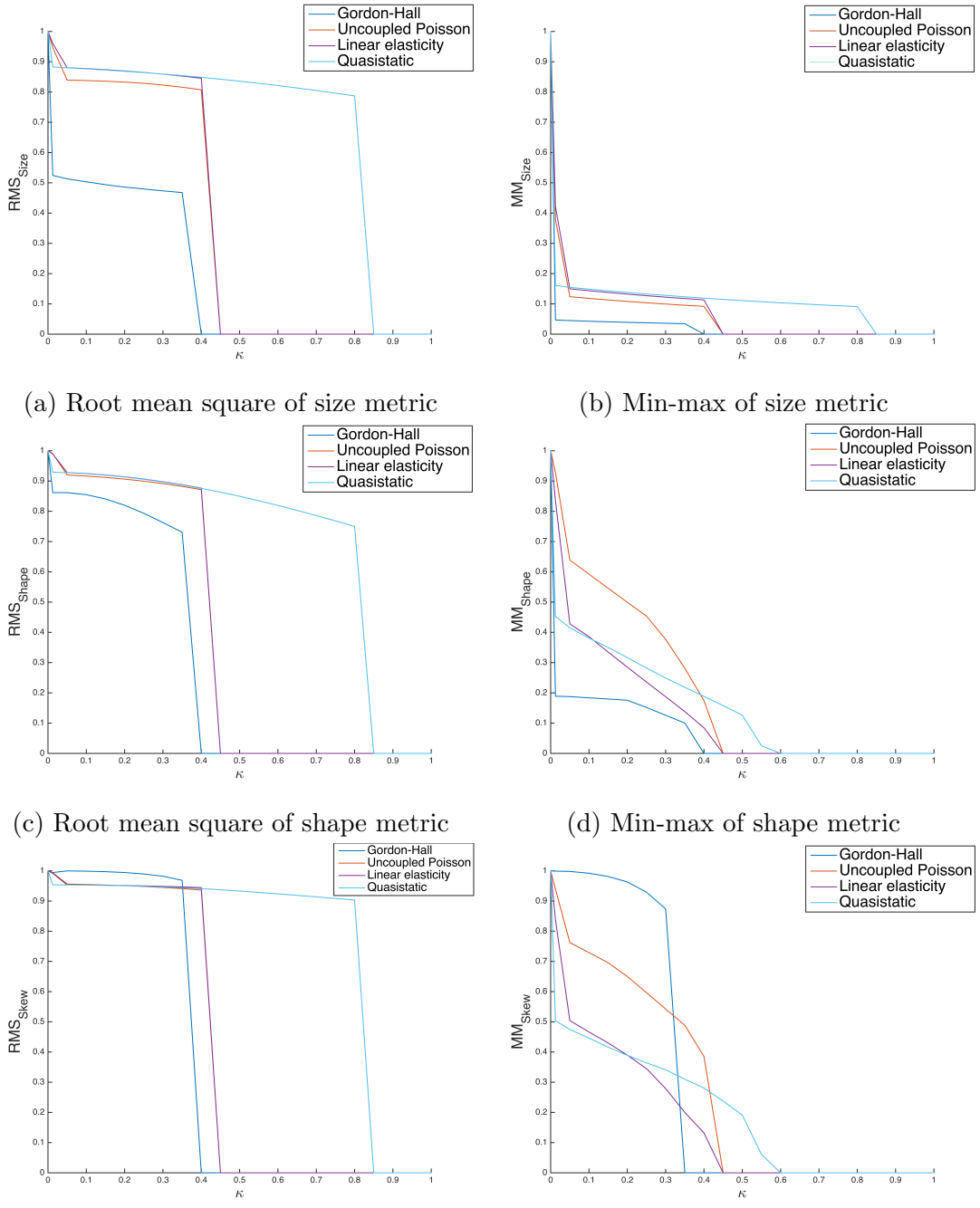
(c) Root mean square of shape metric

(d) Min-max of shape metric

(e) Root mean square of skew metric

(f) Min-max of skew metric

Figure 5.53: **Method comparison on clover geometry:** Root mean square measure (left) and min-max measure (right) with respect to κ for the different methods on the clover geometry in the size, shape, and skew metric. The blue lines belong to the Gordon-Hall method, the red and yellow belong to the uncoupled Poisson without and with a $\gamma(x, y)$-function respectively, the purple and green belong to the linear elasticity method without and with a modified elasticity matrix, and the turquoise belong to the quasistatic method.



(a) Root mean square of size metric

(b) Min-max of size metric

(c) Root mean square of shape metric

(d) Min-max of shape metric

(e) Root mean square of skew metric

(f) Min-max of skew metric

Figure 5.54: **Method comparison on the jigsaw geometry:** Root mean square measure (left) and min-max measure (right) with respect to κ for the different methods on the clover geometry in the size, shape, and skew metric. The blue lines belong to the Gordon-Hall method, the red to the uncoupled Poisson with a constant $\gamma(x, y)$-function, the purple to the linear elasticity method with a constant elasticity matrix, and the turquoise belong to the quasistatic method.

5.4.2 Performance with respect to polynomial degree

Now that we have seen how the methods perform on the three geometries with different values of the difficulty parameter κ, it is time to see how the polynomial degree, p, influence the parameterization ability of the methods. As we saw in Section 5.3.4, the polynomial degree influenced the quasistatic method's ability to generate a valid parameterization.

The dependence on choice of polynomial degree is explored further by running the methods several times with different choices of p. To get a complete picture of the performance of the methods, κ is also varied, and we consider the results of the root mean square measure and the min-max measure of each of the metrics relative size, shape, and skew.

The results presented here are obtained on the jigsaw geometry. Similar results were obtained on the bottom sine geometry and the clover geometry.

Figure 5.55 shows the dependence of the Gordon-Hall algorithm on the polynomial order for $p = 1, 2, 3, 4$ and 5 when parameterizing the jigsaw geometry. The figure shows that the polynomial order has some effect on the min-max measure of the relative size and shape metric, and the root mean square measure of the size metric. The parameterizations obtained with $p = 1$ and 2 contain degenerate elements in the skew metric for lower κ's than for parameterizations for higher polynomial degree, as can be seen in Figure 5.55f. Aside from the early introduction of the degenerate elements, the skew metric seems less affected by the polynomial degree.

The figures also shows that elevating the polynomial degree p does not affect the durability of the Gordon-Hall algorithm. The method is still unable to produce valid parameterizations on the jigsaw geometry for any $\kappa > 0.4$.

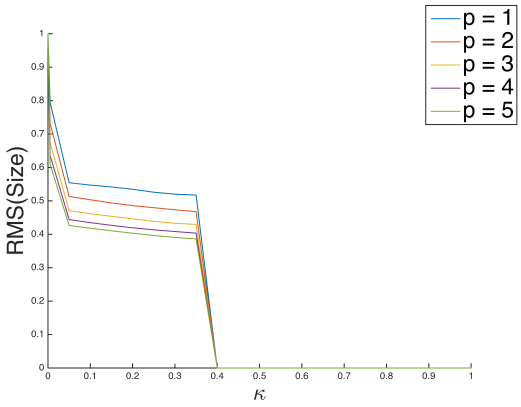
The dependence on p for the uncoupled Poisson method on the jigsaw geometry is illustrated in Figure 5.56. The figure shows that $p = 1$ and $p = 2$ yield the exact same values in the root mean square measure, regardless of which metric is being considered. As is the case for $p = 3, 4$ and 5, but with higher polynomial degree, the method is able to produce valid parameterization for slightly higher values of κ. In the min-max measures the polynomial degrees yield slightly different values, but they follow the same trends.

The results for the linear elasticity method is illustrated in Figure 5.57, and it shows more distinction between the lower polynomial degrees. For $p = 1$ the method is able to produce a valid parameterization for very low κ-values only, but once the degree is increased to $p = 2$ it is able to find valid parameterizations up until $\kappa = 0.4$. Further elevation of the polynomial degree makes the method able to tackle slightly higher κ's, but it is not able to produce a valid mesh for $\kappa > 0.5$ regardless of the polynomial degree. The figure shows that the root mean square measure values for $p = 3, 4$ and 5 are identical, and only small value differences

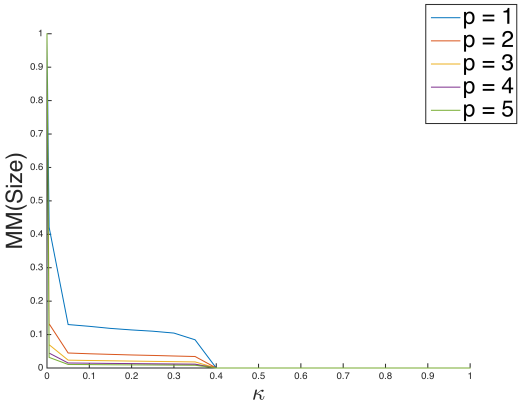
are visible in the min-max measures.

Finally, the results for the quasistatic method are presented in Figure 5.58. Similarly to the linear elasticity method, the quasistatic method performs poorly for $p = 1$, however, just by raising the polynomial order by one, to $p = 2$, it is able to produce valid parameterizations for higher κ's than any of the other methods. However, as we saw in Section 5.3.4, it is not able to handle κ's all the way up to 1 with $p = 2$. By raising the degree to $p = 3$, the method is able to produce a valid mesh for this κ-value, but as seen in Figures 5.58d and 5.58f, the mesh contains degenerate elements in the shape and skew metric respectively. This can be avoided by raising the degree further to $p = 4$. Then, the quasistatic method is able to produce a valid parameterization on the jigsaw geometry for $\kappa = 1$, without any degenerate elements.

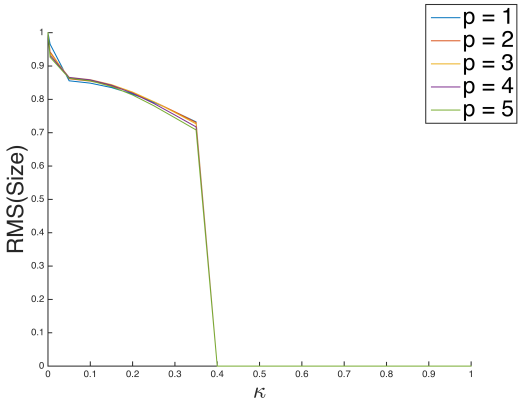
We observe that all the methods require more time to perform the computations and produce the parameterization with increasing polynomial degree. For the least complex method, this is hardly noticeable on the geometries we have been examining, but for the more complex methods the slow-down is significant.



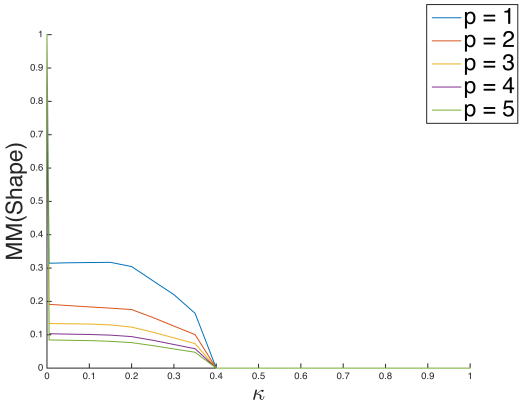
(a) Root mean square of size metric



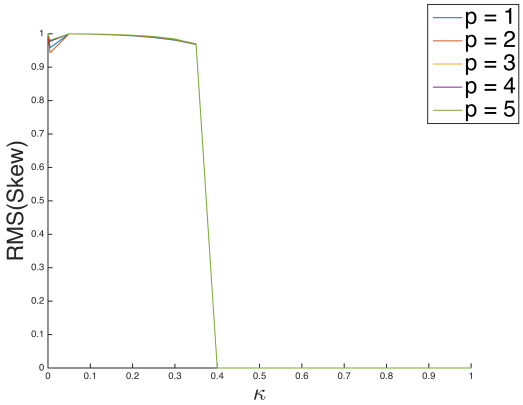
(b) Min-max of size metric



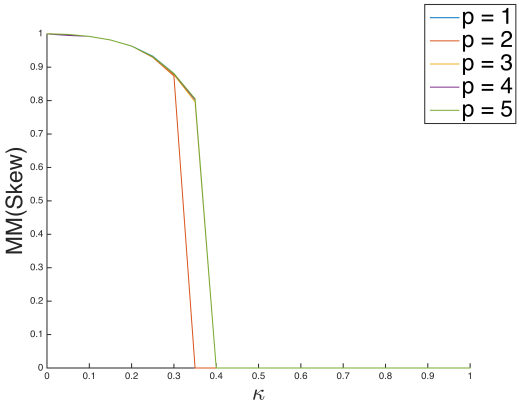
(c) Root mean square of shape metric



(d) Min-max of shape metric

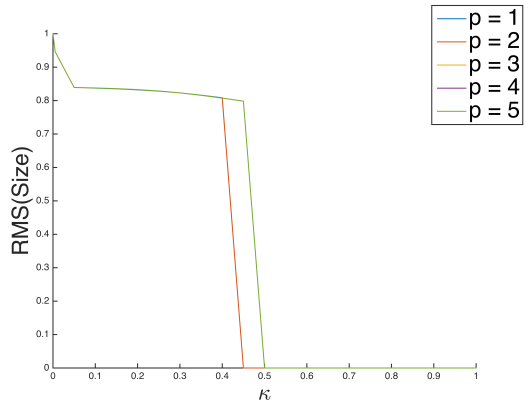


(e) Root mean square of skew metric

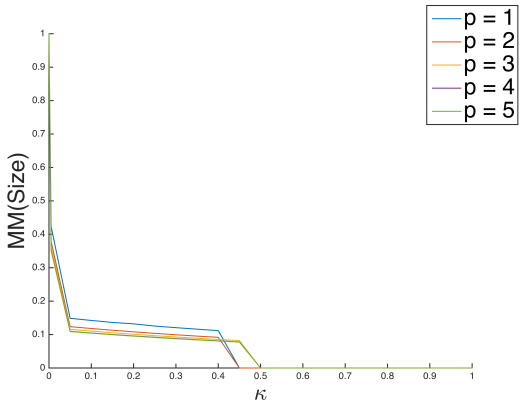


(f) Min-max of skew metric

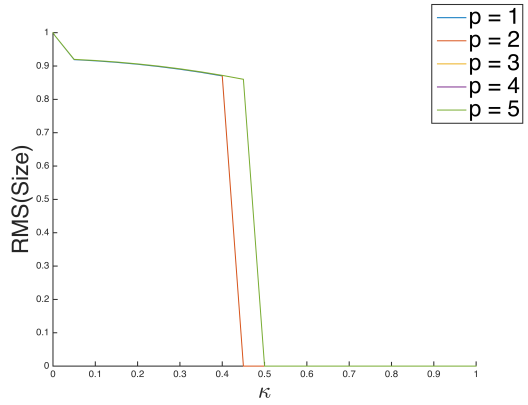
Figure 5.55: **Polynomial order dependency, Gordon-Hall:** Dependence on polynomial order p for the Gordon-Hall algorithm when parameterizing the jigsaw geometry with different values of κ.



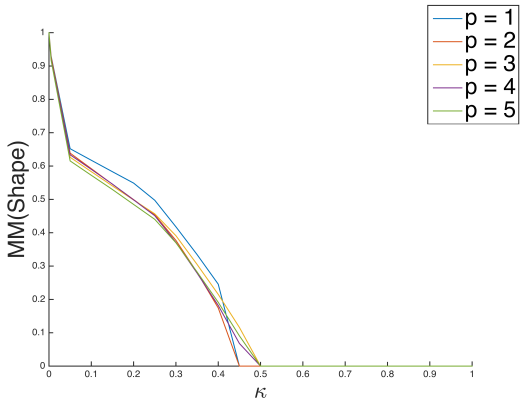
(a) Root mean square of size metric



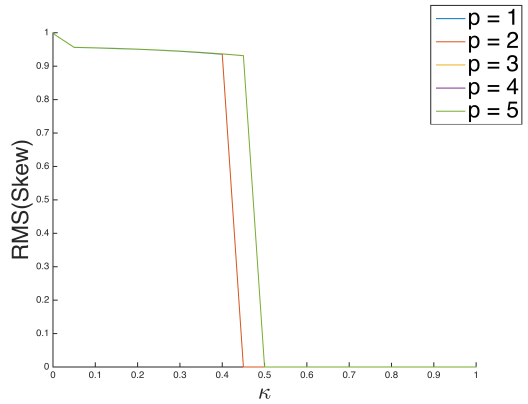
(b) Min-max of size metric



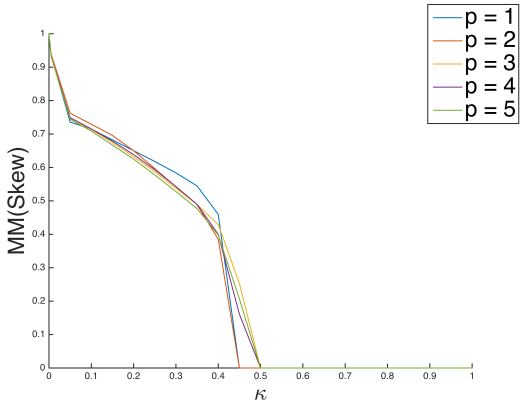
(c) Root mean square of shape metric



(d) Min-max of shape metric

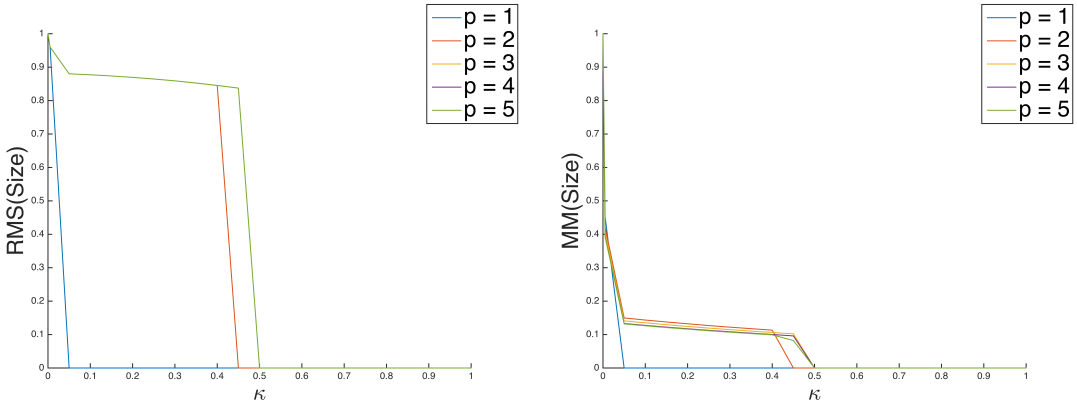


(e) Root mean square of skew metric



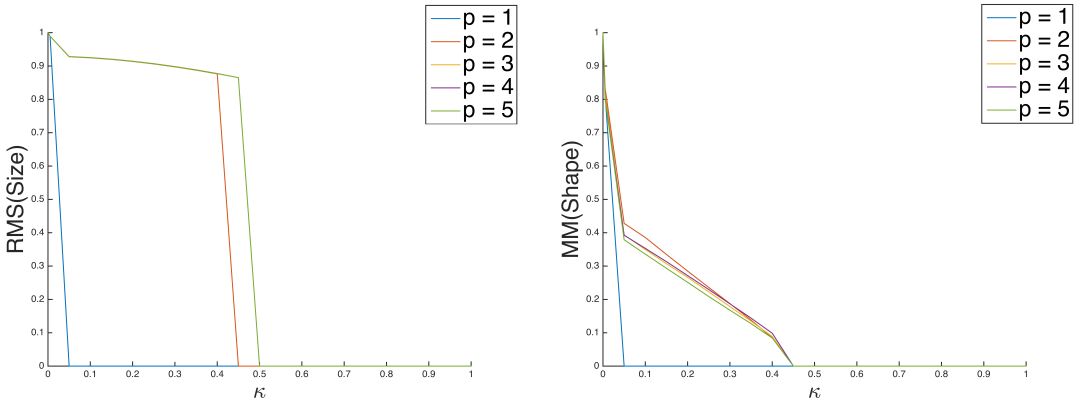
(f) Min-max of skew metric

Figure 5.56: **Polynomial order dependency, uncoupled Poisson:** Dependence on polynomial order p for the uncoupled Poisson method when parameterizing the jigsaw geometry with different values of κ.



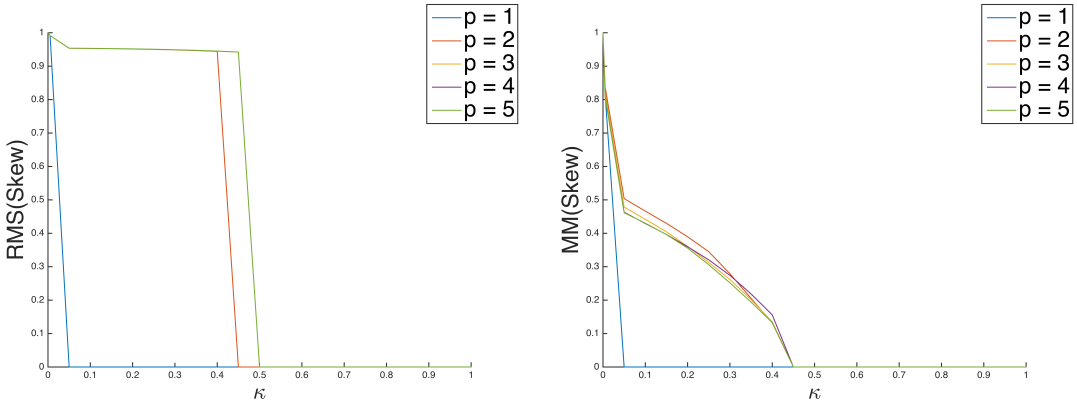
(a) Root mean square of size metric

(b) Min-max of size metric



(c) Root mean square of shape metric

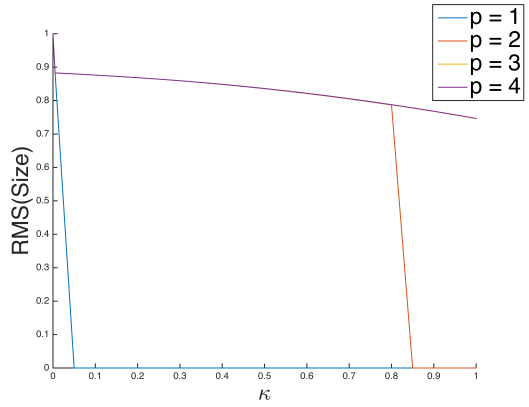
(d) Min-max of shape metric



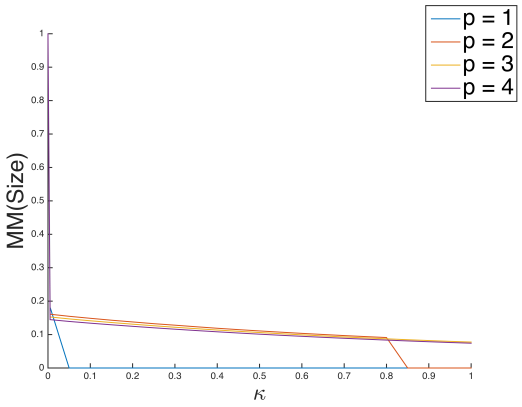
(e) Root mean square of skew metric

(f) Min-max of skew metric

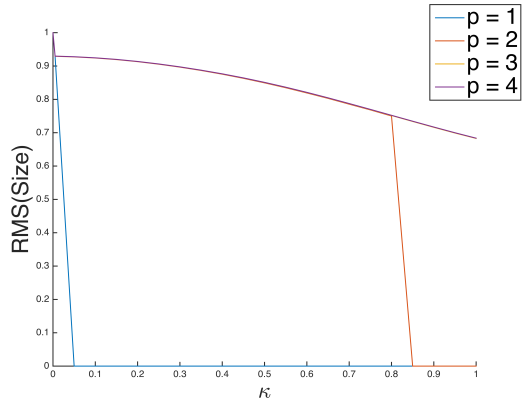
Figure 5.57: **Polynomial order dependency, linear elasticity:** Dependence on polynomial order p for the linear elasticity method when parameterizing the jigsaw geometry with different values of κ.



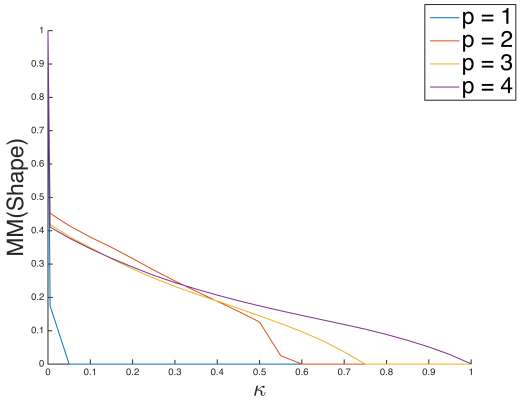
(a) Root mean square of size metric



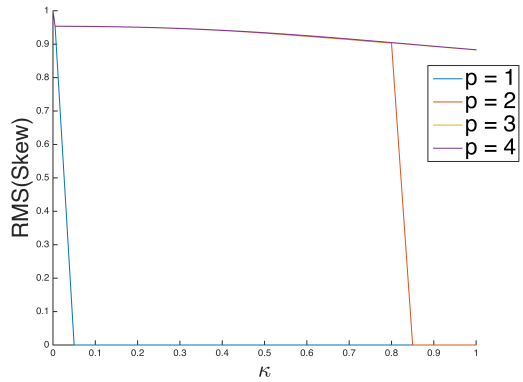
(b) Min-max of size metric



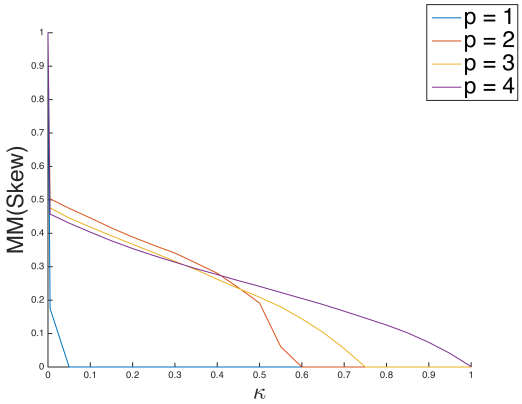
(c) Root mean square of shape metric



(d) Min-max of shape metric



(e) Root mean square of skew metric



(f) Min-max of skew metric

Figure 5.58: **Polynomial order dependency, quasistatic:** Dependence on polynomial order p for the quasistatic method when parameterizing the jigsaw geometry with different values of κ.

Chapter 6

Summary and Concluding Remarks

6.1 Summary and Conclusion

In this thesis we have looked at four distinct methods for parameterization of the interior of a physical domain, given its boundary. All the methods use B-splines as basis functions, but they differ in computational complexity and sophistication.

We have seen that the least complex method, the Gordon-Hall algorithm, performs well on simple geometries, but quickly unravels and becomes unable to produce valid parameterizations for more challenging geometries, regardless of the polynomial degree p.

The uncoupled Poisson method persists longer for increasingly challenging geometries, compared with the Gordon-Hall method, especially if we are able to find a suitable gamma function, $\gamma(x, y)$, for the geometry in question. We have seen that if we are able to find such a function, the method is able to produce valid meshes of geometries it otherwise would be unable to parameterize.

The quality of the mesh is strongly connected with $\gamma(x, y)$, as an optimal γ-function may enable the method to produce superior meshes, while a suboptimal function may not improve the method at all. Furthermore, we wish to find a general, automatic method for finding optimal parameterizations. It is then undesirable to be dependent on finding a suitable $\gamma(x, y)$ for the geometry in question.

The polynomial degree has minor impact on the results of the uncoupled Poisson method.

The linear elasticity approach has proved to be more tenacious than the Gordon-Hall and uncoupled Poisson methods, but even this approach has to surrender when used on the most challenging geometries. For more complex geometries, also the linear elasticity method becomes dependent on finding an appropriate expres-

sion for the elasticity matrix, $\mathbf{D}(x, y)$. This means that the performance of the method, as with the uncoupled Poisson method, is governed by our ability to find an appropriate expression for this matrix. We have seen that it can be challenging to find a suitable modification of the elasticity matrix, and for some geometries we are unable to find an expression of $\mathbf{D}(x, y)$ for the method to produce a valid parameterization.

When applying the linear elasticity method, we have to choose values for the material properties of Young's modulus, E, and Poisson's ratio, ν. This, combined with the need to customize the elasticity matrix, $\mathbf{D}(x, y)$, makes automation more difficult, and requires the method to be specifically adapted to each geometry before it can be applied.

We have seen that the linear elasticity method can be enhanced, to some extent, by increasing the polynomial degree, and thereby making the method capable of parameterizing slightly more complex geometries.

The most complex method we have examined, the quasistatic method, has proven to give the best overall performance, and to be the most versatile. We have seen that the method performs on a high level, producing valid parameterizations even on the most challenging geometries, where the other methods have failed.

Except for the specification of Young's modulus and Poisson's ratio, this method also avoids dependency on finding geometry specific adaptations like the $\gamma(x, y)$ or $\mathbf{D}(x, y)$ for the uncoupled Poisson and linear elasticity methods.

The quasistatic method responded very well when the polynomial degree was elevated, enabling the method to achieve high quality meshes on even more challenging geometries.

However, the results from a complex procedure such as the quasistatic method comes with a trade off. Several iterations need to be performed to obtain the parameterization meshes. As each iteration essentially solves an elasticity problem, the computing multiplies proportional with the number of iterations as compared with the second most complex method which was the linear elasticity method. This may not be of large importance for small problems, but it may become a limiting factor for larger problems.

Ultimately, the parameterization method to be used needs to be decided based on the problem at hand and the requirements associated with the given geometry.

On a simple geometry, it may be sufficient to employ one of the less computationally intensive and time consuming methods, without standing the risk of sacrificing the parameterization quality. The quasistatic method will in general produce the mesh with the best overall performance, but this method may be unnecessary complex for simple geometries. On simple geometries the other methods may also produce parameterizations of sufficient quality with less computing and in less time.

6.2 Further Work

A natural continuation of the work done in this thesis would be to incorporate the handling of the arrow shaped elements in the clover geometry for κ's larger than approximately 0.315, found in Section 5.2. Then, we would not have to disregard these elements in the same way, and we would get a more complete impression of the different parameterizations.

The scope of this thesis has been limited to consider parameterizations of planar geometries on non-convex domains. It would have been interesting to consider more complex domains, but the methods we have presented are, for example, unable to parameterize domains with holes. By, for instance, introducing different mappings between the parametric and physical domain, the methods can be extended to be able to handle such geometries as well.

Incorporating multipatch is also a possible extension to the work carried out in this thesis. A possible application of multipatch is to make it possible to use different parameterization methods on different parts of a geometry. That is, if one method is best suited on one part of the geometry, and an other method on another part of the geometry, these areas may be parameterized almost independently, provided that their boundaries coincide.

Bibliography

[1] A. Quarteroni. *Numerical Models for Differential Problems*. Springer, 2009.

[2] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. *Isogeometric Analysis Toward Integration of CAD and FEA*. John Wiley & Sons, 2009.

[3] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Engrg.*, 194(39-41):4135–4195, 2005.

[4] T. Dokken, T. Lyche, and K. F. Pettersen. Polynomial splines over locally refined box-partitions. *Comput. Aided Geom. Design*, 30(3):331–356, 2013. ISSN 0167-8396.

[5] K. A. Johannessen, T. Kvamsdal, and T. Dokken. Isogeometric analysis using LR B-splines. *Comput. Methods Appl. Mech. Engrg.*, 269:471–514, 2014. ISSN 0045-7825.

[6] K. A. Johannessen, F. Remonato, and T. Kvamsdal. On the similarities and differences between classical hierarchical, truncated hierarchical and LR B-splines. *Comput. Methods Appl. Mech. Engrg.*, 291:64–101, 2015.

[7] J. Gravesen, A. Evgrafov, N. D. Manh, and P. Nørtoft. Planar parametrization in isogeometric analysis. In *Mathematical methods for curves and surfaces*, volume 8177 of *Lecture Notes in Comput. Sci.*, pages 189–212. Springer, Heidelberg, 2014.

[8] N. D. Manh, A. Evgrafova, A. R. Gersborgb, and J. Gravesen. Isogeometric shape optimization of vibrating membranes. *Comput. Methods Appl. Mech. Engrg.*, 200:1343 – 1353, 2011.

[9] Knut Nordanger, Runar Holdahl, Trond Kvamsdal, Arne Morten Kvarving, and Adil Rasheed. Simulation of airflow past a 2d {NACA0015} airfoil using an isogeometric incompressible navier-stokes solver with the spalarrt-allmaras turbulence model. *Comput. Methods Appl. Mech. Engrg.*, 290:183 – 208, 2015.

[10] K. Nordanger, R. Holdahl, A. M. Kvarving, A. Rasheed, and T.Kvamsdal. Implementation and comparison of three isogeometric navier-stokes solvers applied to simulation of flow past a fixed 2d {NACA0012} airfoil at high reynolds number. *Comput. Methods Appl. Mech. Engrg.*, 284:664 – 688, 2015. Isogeometric Analysis Special Issue.

[11] D.A. Field. Qualitative measures for initial meshes. *Int. J. Numer. Meth. Engng.*, 47:887–906, 2000.

[12] P. M. Knupp. Algebraic mesh quality metrics for unstructured initial meshes. *Finite Elem. Anal. Des.*, 39:217–241, 2003.

[13] P. M. Knupp. Algebraic mesh quality metrics. *SIAM J. Sci. Comput*, 23: 193–218, 2001.

[14] L. Piegl and W. Tiller. *The NURBS Book*. Springer, 1995.

[15] T. Lyche and K. Mørken. Spline methods. lecture notes in MAT-INF4170, spring 2014. *Lecture Notes*, 2011.

[16] V. P. Nguyen, R. N. Simpson, S. P. A. Bordas, and T. Rabczuk. An introduction to isogeometric analysis with matlab implementation: FEM and XFEM formulations. *Lecture Notes*, 2012.

[17] E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, 2006.

[18] J. Gravesen and P. Nørtoft. Isogeometric analysis: A practical primer. *Lecture Notes*, 2014.

[19] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, 2009.

[20] S. S. Sastry. *Introductory Methods of Numerical Analysis*. PHI Learning, 2012.

[21] E. Süli and D. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, Cambridge, 2006.

[22] Gaussian quadrature weights and abscissae, september 2014. `http://pomax.github.io/bezierinfo/legendre-gauss.html`.

[23] P. Holoborodko. Numerical integration, 2014. www.holoborodko.com/pavel/numerical-methods/numerical-integration.

[24] V. P. Nguyen, T. Rabczuk, S. Bordas, and M. Duflot. Meshless methods: A review and computer implementation aspects. *Math. Comput. Simulation*, 79 (3):763–813, 2008.

[25] D. Wang and J. Xuan. An improved NURBS-based isogeometric analysis with enhanced treatment of essential boundary conditions. *Comput. Methods Appl. Mech. Engrg.*, 199(37-40):2425–2436, 2010.

[26] W. J. Gordon and C. A. Hall. Construction of curvilinear co-ordinate systems and applications to mesh generation. *Internat. J. Numer. Methods Engrg.*, 7: 461–477, 1973.

[27] E. Rønquist. Deformed geometries - part 2. *Lecture Notes*, 2012.

[28] J. Fish and T. Belytschko. *A First Course in Finite Elements*. John Wiley & Sons, 2007.

[29] G. T. Mase and G. E. Mase. *Continuum Mechanics for Engineers*. CRC Press, 1999.

[30] O. C. Zienkiewicz and J. Z. Zhu. A simple error estimator and adaptive procedure for practical engineering analysis. *Internat. J. Numer. Methods Engrg.*, 24:337–357, 1987.

[31] S. Lipton, J.A. Evans, Y. Bazilevs, T. Elguedj, and T.J.R. Hughes. Robustness of isogeometric structural discretizations under severe mesh distortion. *Comput. Methods Appl. Mech. Engrg.*, 199:357 – 373, 2010.

Appendices

Appendix A

Source Code

The most important source codes for the four parameterization methods presented in this thesis is included here. The code is presented in the same order as the methods in Chapter 4. Functions and scripts used by all the parameterization procedures are presented in the last section of this appendix, Appendix A.5.

A.1 Gordon Hall Solver

The Gordon-Hall algorithm is executed by initializing the values for p, n, and κ and running `GordonHallAlgorithm.m`. The geometry also needs to be specified in `DomainCurves.m`. The `GordonHallAlgorithm.m` script makes a call to the function `GetNewBasis.m`, which returns the control points after performing linear interpolation on the domain curves. The control points are found through order elevation in `OrderElevation.m`, and knot insertion in `KnotInsertion.m`. The procedure also involves a function that evaluates spline functions in their associated Greville points, `EvaluateGreville.m`, and some functions that are common for all the methods which is included in Appendix A.5.

`GordonHalAlgorithm.m`

```
geometry = 'sine';
kappa = 0.5;
p = 2;
n = 22;

[KnotVec, BottomTop, LeftRight, CornerPoints] = GetNewBasis(p,n,kappa);

F = BottomTop + LeftRight - CornerPoints;
UU = F;

KnotVecX = KnotVec;
KnotVecY = KnotVec;
```

```
q = p;
NoDofs = length(UU);

Sx = reshape(F(:,1),[n,n])';
Sy = reshape(F(:,2),[n,n])';

GetGeometry
GenerateMesh

[N, dN] = PlotBasis(KnotVec, p, nviz);

ux = N*Sx*N';
uy = N*Sy*N';

figure;
hold on
plot(ux, uy, 'Color',[0 0.447 0.741]);
plot(ux',uy', 'Color',[0 0.447 0.741]);
title('Mesh')
axis equal

JacobianTest

if JacobianMinimum < 0
    disp('Negative Jacobian')
else
    Metrics
end %if
```

DomainCurves.m

```
function point = DomainCurves(s, x,kappa)
    % Bottom sine geometry:
    n = length(x);
    if strcmp(s, 'b')
        point = [x; kappa*sin(2*pi*x)];
    elseif strcmp(s, 't')
        point = [x; ones(1,n)];
    elseif strcmp(s, 'l')
        point = [zeros(1,n);x];
    elseif strcmp(s, 'r')
        point = [ones(1,n);x];
    end
end
```

GetNewBasis.m

```
function [t, BottomTop3, LeftRight3, CornerPoints3] = GetNewBasis(p,n,kappa)
addpath ../SharedFunctions/
% Full basis
t = linspace(0,1,n+1-p);
o = ones(1,p);
z = zeros(1,p);
t = [z t o];
x = linspace(0,1,n);
```

```
c = [x; 0.5*sin(2*pi*x)]';

% Linear basis
tau1 = [0 0 1 1];
p_l  = 1;
elevation = p-p_l;

% bottom and top boundaries:
x = linspace(0,1,n);
c1 = DomainCurves('b',x,kappa)';%[x; 0.5*sin(2*pi*x)]';
c2 = DomainCurves('t',x,kappa)';%[x; o]';
j = 1;
BottomTop1 = [];
for i = 1:n
    BottomTop1(j,:)     = c1(i,:);
    BottomTop1(j+1,:)   = c2(i,:);
    j = j+2;
end %i
clear c1 c2

% left and right boundaries:
x = linspace(0,1,n);
c1 = DomainCurves('l',x,kappa)';
c2 = DomainCurves('r',x,kappa)';
j = 1;
LeftRight1 = [];
for i = 1:n
    LeftRight1(j,:)     = c1(i,:);
    LeftRight1(j+1,:)   = c2(i,:);
    j = j+2;
end %i
clear c1 c2

% Corner points:
x = linspace(0,1,2);
c1 = DomainCurves('l',x,kappa)';
c2 = DomainCurves('r',x,kappa)';
CornerPoints1 = [c1; c2];
clear c1 c2

%Order Elevation botton-top and left-righ:
[tau2, BottomTop2] = OrderElevation(tau1, p_l, BottomTop1, elevation,n);
[tau2, LeftRight2] = OrderElevation(tau1, p_l, LeftRight1, elevation,n);

%Knot insertion botton-top and left-righ:
[tau3, BottomTop3] = KnotInsertion(t,tau2,p, BottomTop2,n);
[tau3, LeftRight3] = KnotInsertion(t,tau2,p, LeftRight2,n);

BT3 = [];
m = [1:n];
k = [1:n:n^2-n+1];
for i = 1:n
    BT3(m,:) = BottomTop3(k,:);
    m = m+n;
    k = k+1;
end %i

BottomTop3 = BT3;

%Order elevation and knot insertion corner points:
[tau2, CP2] = OrderElevationXiEta(tau1, p_l, CornerPoints1, elevation);
[tau3, CP3] = KnotInsertionXiEta(t,tau2,p,CP2,n);
```

```
 cpts3 = [];
 counter = 1;
for i = 1:n
    cpts3(counter,:) = CP3(i,:);
    cpts3(counter+1,:) = CP3(i+n,:);
    counter = counter + 2;
end %i
[tau2, CornerPoints2] = OrderElevation(tau1,p_l,cpts3,elevation,n);
[tau3, CornerPoints3] = KnotInsertion(t,tau2,p,CornerPoints2,n);

end
```

OrderElevation.m

```
function [KnotVec, controlPts] = OrderElevation(KnotVecOriginal, p,...
                                        controlPtsOriginal, elevation,n)

    % Order elevation of knot vector
    uniqueKnots = unique(KnotVecOriginal);
    KnotVec = KnotVecOriginal;
    for i = 1:elevation
        KnotVec = [KnotVec uniqueKnots];
    end
    KnotVec = sort(KnotVec);

    % Finding approriate control points
    [N_p, N_pe] = evaluateGreville(KnotVecOriginal,KnotVec,p,p+elevation);

    h = length(controlPtsOriginal);
    teller = [1:p+elevation+1];
    controlPts = [];
    for i = 1:2:2*n
        controlPts(teller,:) = N_pe\N_p*controlPtsOriginal(i:i+p,:);
        teller = teller + p + elevation + 1;
    end
end
```

KnotInsertion.m

```
function [KnotVec, controlPts] = KnotInsertion(t,tau,p, controlPts_Original,n)

controlPts = [];
m = length(t) - (p+1);

counter = [1:p+1];
for j = 1:n
    b = [];
    c = controlPts_Original(counter,:);
    for i = 1:m
        mu = FindMu(tau, t(i));
        cp = c(mu-p:mu,:);
        if p == 0
            b(i,:) = cp(mu);
        else
            b(i,:) = getB(t(i+1:i+p), mu, tau, p) * cp;
```

```
        end
    end
    controlPts = [controlPts; b];
    counter = counter + p + 1;
end

    KnotVec = t;
end
```

A.2 Uncoupled Poisson Solver

The procedure for the uncoupled Poisson method starts by running `UncoupledPoisson.m`. Here, the physical domain boundaries are parameterized, and the function `Main.m` is run twice, once for the x-coordinate and once for the y-coordinate of the final parameterization $u = [x, y]$. The function `Main.m` makes calls to several other functions in order to set up and solve the Poisson problem.

Figure A.1 schematically shows the calls the main function makes each time it is run. First, knot vectors, control points, boundary conditions and external loads are specified in `GetGeometry.m`. Then, the elements are defined and the connectivity between the basis functions are found in`GenerateMesh.m`. These two functions make up the pre-processing part of the solver.

Next comes the processing part. It consist of the functions `AssembleSystem.m` and `BoundaryConditionApplication.m`, as well as the actual solving of the system $\mathbf{AU} = \mathbf{f}$. The system is solved by using MATLAB's built-in operator.

The solution $\mathbf{U}$ is not the "real" solution to the problem, but rather it contains the value of the control variables. In order to get the actual solution we need to post-process. This is done in `postProcessing.m`, where the spline basis functions are used in combination with the control variables to represent the solution in the same way as in equation (3.8) and (3.11). Once the post-processing is done, the main sequence returns the solution to `UncoupledPoisson.m` where it is interpreted as either the x- or y-coordinate.

Some of the functions used by the Main-sequence is used by the uncoupled Poisson solver, as well as the linear elasticity and quasistatic solver. These functions can be found collectively in Appendix A.5.

`UncoupledPoisson.m`

```
alpha = @(x,y) ((y^(7/4))*(1-0.25*x));
kappa = 0.5;

% Solve for x-coordinate:
BottomBoundary  = @(x,y) x;
TopBoundary     = @(x,y) x;
LeftBoundary    = @(x,y) 0 + (y-y);
```

```
RightBoundary   = @(x,y) 1 + (y-y);

Main
ux = u;
Ux = U;

% Solve for y-coordinate:
BottomBoundary  = @(x,y) kappa * sin(2*pi*x);
TopBoundary     = @(x,y) 1 + (x-x);
LeftBoundary    = @(x,y) y;
RightBoundary   = @(x,y) y;

Main
uy = u;
Uy = U;

% Test if |J| > 0
JacobianTest

if min(min(Jacobian)) < 0
    disp('Negative Jacobian')
else
    Metrics
end
```

Main.m

```
% Initialization
GetGeometry

GenerateMesh

% Assemble the system Au = f
AssembleSystem

% Applying boundary conditions to A and f
BoundaryConditionApplication

% Solve system
U = A\f;

postProcessing

plotting
```

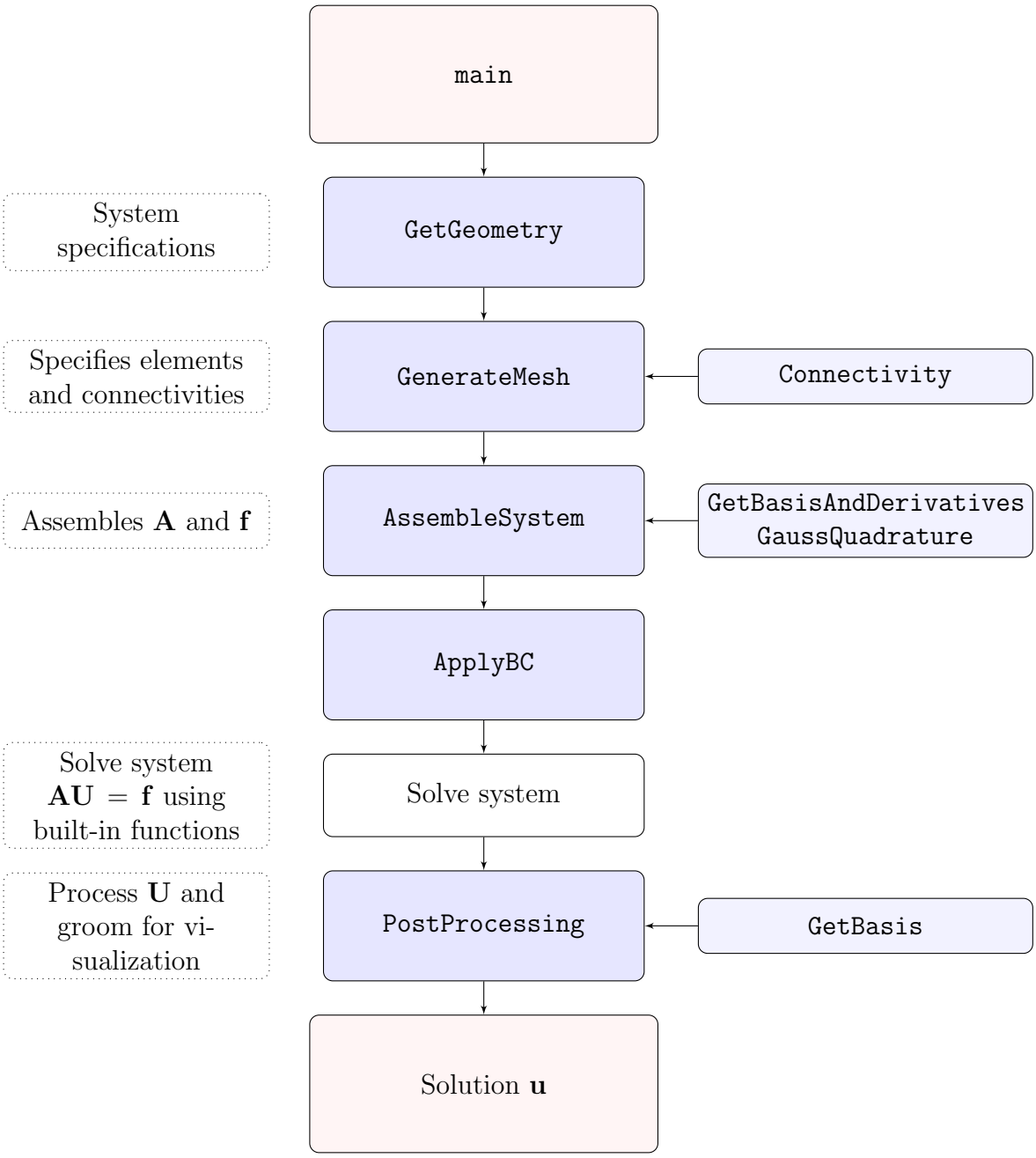


Figure A.1: Overview of the code implementation. The main function makes a call to all the functions in the central column. Each of these utilizes the functions in the right column, and some additional minor functions.

GetGeometry.m

```
func    = @(x,y) 0;
startX = 0; endX = 1;
startY = 0; endY = 1;

% Spline specification
n = 20;
KnotVecX = linspace(0,1,n+1);
KnotVecX = [0 0 KnotVecX 1 1];
p = 2;
KnotVecY = KnotVecX;
q = p;

NoPtsX       = FindMu(KnotVecX, KnotVecX(end));
NoPtsY       = FindMu(KnotVecY, KnotVecY(end));
NoGaussPts   = max(p,q) + 1;
NoCtrlPts    = NoPtsX * NoPtsY;

GetControlPts

% Visualization specification
NoPx = p+1;
NoPy = q+1;
uniqueX      = unique(KnotVecX);
uniqueY      = unique(KnotVecY);
NoElemX      = length(uniqueX) - 1;
NoElemY      = length(uniqueY) - 1;
nviz = NoPx*NoElemX;
mviz = NoPy*NoElemY;
```

AssembleSystem.m

```
% Assemble stiffness matrix A and right hand side vector f

% Initialize A and f
A   = zeros(NoCtrlPts, NoCtrlPts);
f   = zeros(NoCtrlPts, 1) ;

% Gauss quadrature points and weights
[P, W]      = GaussQuadrature(NoGaussPts, 2);

for e = 1:NoElem
    % Element interval (xi_i, xi_(i+1))
    Xi_e    = ElemRange(e,1:2);
    Eta_e   = ElemRange(e,3:4);

    % Connectivities on element e
    Conn    = ElemConn(e,:);

    for g = 1:size(W,1)
        xi_hat  = P(g,1);
        eta_hat = P(g,2);
        w       = W(g);

        % Coordinates in parametric space:
        Xi  = 0.5 * ((Xi_e(2) - Xi_e(1))*xi_hat ...
                   + (Xi_e(2)  + Xi_e(1)));
```

```
        Eta = 0.5 * ((Eta_e(2) - Eta_e(1))*eta_hat ...
                     +(Eta_e(2) + Eta_e(1)));

        % Jacobian unit square - parametric:
        J2  = 0.25  * ( Xi_e(2) - Xi_e(1) )*( Eta_e(2) - Eta_e(1) );

        % Basis functions and derivatives
        [N, dNdxi, dNdeta] = GetBasisAndDerivatives(Xi, Eta, p, q, ...
                                                    KnotVecX, KnotVecY);

        % Jacobian parametric - physical:
        jacob       = controlPts(Conn,:)'*[dNdxi' dNdeta'];
        J1          = det(jacob);
        invJacob    = inv(jacob);

        % Use the derivative wrt x and y instead of xi and eta
        dNdx        = [dNdxi' dNdeta'] * invJacob;

        Alpha = alpha(Xi,Eta);
        A(Conn, Conn) = A(Conn, Conn) + dNdx * Alpha * dNdx' * J1 * J2 * w;
        x = N*controlPts(Conn,1); y = N*controlPts(Conn,2);
        f(Conn) = f(Conn) + (func(x,y) * N' * J1 * J2 * w);

    end %g
end %e
```

BoundaryConditionApplication.m

```
% Solves least squares problem on boundary

% Find basis functions at boundary
rightNodes  = find(controlPts(:,1) == endX)';
topNodes    = find(controlPts(:,2) == endY)';
leftNodes   = find(controlPts(:,1) == startX)';
bottomNodes = find(controlPts(:,2) == startY)';

NoP = 10;

NobottomNodes   = length(bottomNodes);
NoleftNodes     = length(leftNodes);
NorightNodes    = length(rightNodes);
NotopNodes      = length(topNodes);

% Used for local element indexing
bottomElement   = zeros(NoElemX, p+1);
leftElement     = zeros(NoElemY, q+1);
rightElement    = zeros(NoElemY, q+1);
topElement      = zeros(NoElemX, p+1);

% Used to keep track on the active basis functions and
% control points on each element boundary
bottomEdgeMesh  = zeros(NoElemX, p+1);
leftEdgeMesh    = zeros(NoElemY, q+1);
rightEdgeMesh   = zeros(NoElemY, q+1);
topEdgeMesh     = zeros(NoElemX, p+1);

for i = 1:NoElemX
    bottomEdgeMesh(i,:) = bottomNodes(i:i+p);
    topEdgeMesh(i,:)    = topNodes(i:i+p);
```

```
    bottomElement(i,:)   = (i:i+p);
    topElement(i,:)      = (i:i+p);
end %i

for i = 1:NoElemY
    leftEdgeMesh(i,:)    = leftNodes(i:i+q);
    rightEdgeMesh(i,:)   = rightNodes(i:i+q);
    leftElement(i,:)     = (i:i+q);
    rightElement(i,:)    = (i:i+q);
end %i

B = zeros(NoCtrlPts,1);
C = zeros(NoCtrlPts, NoCtrlPts);

bottomC = zeros(NobottomNodes, NobottomNodes);
leftC   = zeros(NoleftNodes, NoleftNodes);
rightC  = zeros(NorightNodes, NorightNodes);
topC    = zeros(NotopNodes, NotopNodes);

bottomB = zeros(NobottomNodes,1);
leftB   = zeros(NoleftNodes,1);
rightB  = zeros(NorightNodes,1);
topB    = zeros(NotopNodes,1);

% Bottom boundary
for e = 1:NoElemX
    sctr = bottomEdgeMesh(e,:);
    sctrC = bottomElement(e,:);
    Xi_e = ElemRangeX(e,:);
    xiC = linspace(Xi_e(1), Xi_e(2), NoP);

    N = GetBasisBoundary(KnotVecX,p,xiC);
    bottomC(sctrC,sctrC) = bottomC(sctrC,sctrC) + N'*N;
    x = N*controlPts(sctr,1); y = N*controlPts(sctr,2);
    ff = BottomBoundary(x,y);
    bottomB(sctrC) = bottomB(sctrC) + N'*ff;

end %e

C(bottomNodes,bottomNodes) =  C(bottomNodes,bottomNodes) + bottomC;
B(bottomNodes) = B(bottomNodes) +  bottomB;
fBottom = bottomC\bottomB;

% Left boundary
for e = 1:NoElemY
    sctr = leftEdgeMesh(e,:);
    sctrC = leftElement(e,:);
    Eta_e = ElemRangeY(e,:);
    etaC = linspace(Eta_e(1), Eta_e(2), NoP);

    M = GetBasisBoundary(KnotVecY,q,etaC);
    leftC(sctrC,sctrC) = leftC(sctrC,sctrC) + M'*M;
    x = M*controlPts(sctr,1); y = M*controlPts(sctr,2);
    ff = LeftBoundary(x,y);
    leftB(sctrC) = leftB(sctrC) + M'*ff;
end %e

C(leftNodes,leftNodes) =  C(leftNodes,leftNodes) + leftC;
B(leftNodes) = B(leftNodes) +  leftB;
fLeft = leftC\leftB;

% Right boundary
```

```
for e = 1:NoElemY
    sctr = rightEdgeMesh(e,:);
    sctrC = rightElement(e,:);
    Eta_e = ElemRangeY(e,:);
    etaC = linspace(Eta_e(1), Eta_e(2), NoP);

    M = GetBasisBoundary(KnotVecY,q,etaC);
    rightC(sctrC,sctrC) = rightC(sctrC,sctrC) + M'*M;
    x = M*controlPts(sctr,1); y = M*controlPts(sctr,2);
    ff = RightBoundary(x,y);
    rightB(sctrC) = rightB(sctrC) + M'*ff;
end %e

C(rightNodes,rightNodes) = C(rightNodes,rightNodes) + rightC;
B(rightNodes) = B(rightNodes) + rightB;
fRight = rightC\rightB;

% Top boundary
for e = 1:NoElemX
    sctr = topEdgeMesh(e,:);
    sctrC = topElement(e,:);
    Xi_e = ElemRangeX(e,:);
    xiC = linspace(Xi_e(1), Xi_e(2), NoP);

    N = GetBasisBoundary(KnotVecX,p,xiC);
    topC(sctrC,sctrC) = topC(sctrC,sctrC) + N'*N;
    x = N*controlPts(sctr,1); y = N*controlPts(sctr,2);
    ff = TopBoundary(x,y);
    topB(sctrC) = topB(sctrC) + N'*ff;
end %e

C(topNodes,topNodes) = C(topNodes,topNodes) + topC;
B(topNodes) = B(topNodes) + topB;
fTop = topC\topB;

% Solving least-mean-square
Fix = [bottomNodes leftNodes rightNodes topNodes];
FixedNodes = unique(Fix);
for i = size(C,1):-1:1;
    if not(ismember(i,FixedNodes))
        C(i,:) = [];
        C(:,i) = [];
        B(i) = [];
    end %if
end %if

% Apply to boundary
Fixed = (C\B)';
f              = f - A(:,FixedNodes)*Fixed';
f(FixedNodes)  = Fixed;
A(FixedNodes,:) = 0;
A(:,FixedNodes) = 0;

for i = 1:length(FixedNodes)
    A(FixedNodes(i), FixedNodes(i)) = 1;
end %i
```

`PostProcessing.m`

```
k = 1;
S = zeros(NoPtsY, NoPtsX);

for i = 1:NoPtsY
    S(i,:) = U(k:i*NoPtsX);
    k = k + NoPtsX;
end %i

[N, dN] = PlotBasis(KnotVecX, p, nviz);
[M, dM] = PlotBasis(KnotVecY, q, mviz);

% Solution:
u = N*S*M';
u_x = dN*S*M';
u_y = N*S*dM';

x = zeros(NoPtsY,NoPtsX);
y = zeros(NoPtsY,NoPtsX);

d = 1;
for i = 1:NoPtsY
    for j = 1:NoPtsX
        x(i,j) = controlPts(d,1);
        y(i,j) = controlPts(d,2);
        d = d + 1;
    end %j
end %i

X = N*x*M';
Y = N*y*M';
```

A.3 Linear Elasticity Solver

The frame of the linear elasticity solver is relatively similar to the uncoupled Poisson solver, with some minor differences. In the linear elasticity case the solver is run directly from the `Main.m` script and the geometry is defined in `GetGeometry.m` in stead of being defined before running the `Main.m`. Aside from this, the basic structure of the solvers are the same. Some differences occur in the way the system is assembled and how the boundary conditions are enforced. In addition, the elasticity matrix is included in the assembly of the stiffness matrix. Here, we have only included the functions and scripts that have changed. That is, if a script is the same as in the uncoupled Poisson solver it is not included here and the reader is referred to the previous section.

LinearElasticity.m

```
GetGeometry

GenerateMesh

LinearElasticity_AssembleSystem

LinearElasticity_BoundaryConditionApplication

U = A\f;

Ux = U(1:NoCtrlPts);
Uy = U(NoCtrlPts+1:end);

postProcessing

plotting

JacobianTest

if min(min(Jacobian)) < 0
    disp('Negative Jacobian')
else
    Metrics
end %if
```

LinearElasticity_AssembleSystem.m

```
% Assemble stiffness matrix A and right hand side vector f
% Initialize A and f
A   = zeros(NoDofs, NoDofs);
f   = zeros(NoDofs, 1) ;

% Gauss quadrature points and weights
[P, W]      = GaussQuadrature(NoGaussPts, 2);

for e = 1:NoElem
    % Element interval (xi_i, xi_(i+1))
    Xi_e    = ElemRange(e,1:2);
    Eta_e   = ElemRange(e,3:4);

    % Connectivities on element e and matrix B
    Conn    = ElemConn(e,:);
    ConnB   = [Conn Conn+NoCtrlPts];

    nn  = length(Conn);
    B   = zeros(3, 2*nn);
    pts = controlPts(Conn,:);
    R   = zeros(2,2*nn);

    for g = 1:size(W,1)
        xi_hat  = P(g,1);
        eta_hat = P(g,2);
        w       = W(g);

        % Coordinates in parametric space:
        Xi  = 0.5 * ((Xi_e(2) - Xi_e(1))*xi_hat ...
```

```
                    + (Xi_e(2)  + Xi_e(1)));
        Eta = 0.5 * ((Eta_e(2) - Eta_e(1))*eta_hat ...
                    +(Eta_e(2) + Eta_e(1)));

        % Jacobian unit square - parametric:
        J2  = 0.25  * ( Xi_e(2) - Xi_e(1) )*( Eta_e(2) - Eta_e(1) );

        % Basis functions and derivatives
        [N, dNdxi, dNdeta] = GetBasisAndDerivatives(Xi, Eta, p, q, ...
                                                    KnotVecX, KnotVecY);

        % Jacobian parametric - physical:
        jacob       = controlPts(Conn,:)'*[dNdxi' dNdeta'];
        J1          = det(jacob);
        invJacob    = inv(jacob);

        % Use the derivative wrt x and y instead of xi and eta:
        dNdx        = [dNdxi' dNdeta'] * invJacob;

        % B-matrix:
        B(1,1:nn)       = dNdx(:,1)';
        B(2,nn+1:2*nn)  = dNdx(:,2)';
        B(3,1:nn)       = dNdx(:,2)';
        B(3,nn+1:2*nn)  = dNdx(:,1)';

        R(1,1:length(Conn)) = N';
        R(2,length(Conn)+1:end) = N';

        Dmat = ElasticityMatrix(Xi, Eta, Youngs, poissons);

        A(ConnB, ConnB) = A(ConnB, ConnB) + B'*Dmat*B*J1*J2*w;
        x = N*controlPts(Conn,1); y = N*controlPts(Conn,2);
        f(ConnB) = f(ConnB) + (func(x,y) * R * J1 * J2 * w)';
    end %g
end %e
```

ElasticityMatrix.m

```
function D = ElasticityMatrix(x, y, Y, v)
    E = Y*(y^(5/8)+0.3)*(1-0.25*x);
    d = [1 v 0 ;
         v 1 0 ;
         0 0 (1-v)/2];

    D = E/(1-v^2) * d;
end
```

LinearElasticity_BoundaryConditionApplication.m

```
% Solves least squares problem on boundary
% Find basis functions at boundary
rightNodes  = find(controlPts(:,1) == endX)';
topNodes    = find(controlPts(:,2) == endY)';
leftNodes   = find(controlPts(:,1) == startX)';
bottomNodes = find(controlPts(:,2) == startY)';
```

```
7
NoP = 10;
NobottomNodes   = length(bottomNodes);
NoleftNodes     = length(leftNodes);
NorightNodes    = length(rightNodes);
12 NotopNodes      = length(topNodes);

% Used for local element indexing
bottomElement   = zeros(NoElemX, p+1);
leftElement     = zeros(NoElemY, q+1);
17 rightElement    = zeros(NoElemY, q+1);
topElement      = zeros(NoElemX, p+1);

% Used to keep track on the active basis functions and
% control points on each element boundary
22 bottomEdgeMesh  = zeros(NoElemX, p+1);
leftEdgeMesh    = zeros(NoElemY, q+1);
rightEdgeMesh   = zeros(NoElemY, q+1);
topEdgeMesh     = zeros(NoElemX, p+1);

27 for i = 1:NoElemX
    bottomEdgeMesh(i,:) = bottomNodes(i:i+p);
    topEdgeMesh(i,:)    = topNodes(i:i+p);
    bottomElement(i,:)  = (i:i+p);
    topElement(i,:)     = (i:i+p);
32 end %i

for i = 1:NoElemY
    leftEdgeMesh(i,:)   = leftNodes(i:i+q);
    rightEdgeMesh(i,:)  = rightNodes(i:i+q);
37  leftElement(i,:)    = (i:i+q);
    rightElement(i,:)   = (i:i+q);
end %i

Bx = zeros(NoCtrlPts,1);
42 By = zeros(NoCtrlPts,1);
C  = zeros(NoCtrlPts, NoCtrlPts);

bottomC = zeros(NobottomNodes, NobottomNodes);
leftC   = zeros(NoleftNodes, NoleftNodes);
47 rightC  = zeros(NorightNodes, NorightNodes);
topC    = zeros(NotopNodes, NotopNodes);

bottomBx= zeros(NobottomNodes,1);
leftBx  = zeros(NoleftNodes,1);
52 rightBx = zeros(NorightNodes,1);
topBx   = zeros(NotopNodes,1);

bottomBy= zeros(NobottomNodes,1);
leftBy  = zeros(NoleftNodes,1);
57 rightBy = zeros(NorightNodes,1);
topBy   = zeros(NotopNodes,1);

% Bottom boundary
for e = 1:NoElemX
62  sctr    = bottomEdgeMesh(e,:);
    sctrC   = bottomElement(e,:);
    Xi_e    = ElemRangeX(e,:);
    xiC     = linspace(Xi_e(1), Xi_e(2), NoP);

67  N = GetBasisBoundary(KnotVecX,p,xiC);
```

```
    bottomC(sctrC, sctrC) = bottomC(sctrC, sctrC) + N'*N;

    x = N*controlPts(sctr,1); y = N*controlPts(sctr,2);
    ff = BottomBoundary(x,y);
    bottomBx(sctrC) = bottomBx(sctrC) + N'*ff(:,1);
    bottomBy(sctrC) = bottomBy(sctrC) + N'*ff(:,2);
end %e
C(bottomNodes,bottomNodes) =  C(bottomNodes,bottomNodes) + bottomC;
Bx(bottomNodes) = Bx(bottomNodes) +  bottomBx;
By(bottomNodes) = By(bottomNodes) +  bottomBy;

% Left boundary
for e = 1:NoElemY
    sctr    = leftEdgeMesh(e,:);
    sctrC   = leftElement(e,:);
    Eta_e   = ElemRangeY(e,:);
    etaC    = linspace(Eta_e(1), Eta_e(2), NoP);

    M = GetBasisBoundary(KnotVecY, q, etaC);

    leftC(sctrC, sctrC) = leftC(sctrC, sctrC) + M'*M;

    x = M*controlPts(sctr,1); y = M*controlPts(sctr,2);
    ff = LeftBoundary(x,y);
    leftBx(sctrC) = leftBx(sctrC) + M'*ff(:,1);
    leftBy(sctrC) = leftBy(sctrC) + M'*ff(:,2);
end %e
C(leftNodes,leftNodes) =  C(leftNodes,leftNodes) + leftC;
Bx(leftNodes) = Bx(leftNodes) +  leftBx;
By(leftNodes) = By(leftNodes) +  leftBy;

% Right boundary
for e = 1:NoElemY
    sctr = rightEdgeMesh(e,:);
    sctrC = rightElement(e,:);
    Eta_e = ElemRangeY(e,:);
    etaC = linspace(Eta_e(1), Eta_e(2), NoP);

    M = GetBasisBoundary(KnotVecY, q, etaC);

    rightC(sctrC, sctrC) = rightC(sctrC, sctrC) + M'*M;

    x = M*controlPts(sctr,1); y = M*controlPts(sctr,2);
    ff = RightBoundary(x,y);
    rightBx(sctrC) = rightBx(sctrC) + M'*ff(:,1);
    rightBy(sctrC) = rightBy(sctrC) + M'*ff(:,2);
end %e
C(rightNodes,rightNodes) =  C(rightNodes,rightNodes) + rightC;
Bx(rightNodes) = Bx(rightNodes) +  rightBx;
By(rightNodes) = By(rightNodes) +  rightBy;

% Top boundary
for e = 1:NoElemX
    sctr = topEdgeMesh(e,:);
    sctrC = topElement(e,:);
    Xi_e = ElemRangeX(e,:);
    xiC = linspace(Xi_e(1), Xi_e(2), NoP);

    N = GetBasisBoundary(KnotVecX,p,xiC);

    topC(sctrC, sctrC) = topC(sctrC, sctrC) + N'*N;
```

```
    x = N*controlPts(sctr,1); y = N*controlPts(sctr,2);
    ff = TopBoundary(x,y);
    topBx(sctrC) = topBx(sctrC) + N'*ff(:,1);
    topBy(sctrC) = topBy(sctrC) + N'*ff(:,2);
end %e
C(topNodes,topNodes) =  C(topNodes,topNodes) + topC;
Bx(topNodes) = Bx(topNodes) +  topBx;
By(topNodes) = By(topNodes) +  topBy;

% Solving least-mean-square
Fix = [bottomNodes leftNodes rightNodes topNodes];
FixedNodes = unique(Fix);
for i = length(Bx):-1:1;
    if not(ismember(i,FixedNodes))
        C(i,:) = [];
        C(:,i) = [];
        Bx(i)  = [];
        By(i)  = [];
    end %if
end %if

% Apply to boundary
Fixedx = (C\Bx);
Fixedy = (C\By);
FixedNodesX = FixedNodes;
FixedNodesY = FixedNodes + NoCtrlPts;
f = f - A(:,FixedNodesX)*Fixedx;
f = f - A(:,FixedNodesY)*Fixedy;
f(FixedNodesX) = Fixedx';
f(FixedNodesY) = Fixedy';
A(FixedNodesX,:) = 0;
A(FixedNodesY,:) = 0;
A(:,FixedNodesX) = 0;
A(:,FixedNodesY) = 0;

for i = 1:length(FixedNodesX)
    A(FixedNodesX(i), FixedNodesX(i)) = 1;
    A(FixedNodesX(i)+NoCtrlPts, FixedNodesX(i)+NoCtrlPts) = 1;
end %i
```

A.4 Quasistatic Solver

As explained in Section 4.4 of the report, the quasistatic method is basically solving the linear elasticity problem iteratively.

Quasistatic.m

```
geometry = 'sine';
BottomBoundary  = @(x,y) [x, (y-y)];
TopBoundary     = @(x,y) [x, 1+(y-y)];
LeftBoundary    = @(x,y) [(x-x), y];
RightBoundary   = @(x,y) [1+(x-x), y];
Main;

r = 0.0;
```

```
kappa = 0.5;
iteretions = 10;

while r <= kappa
    controlPts = [Ux'; Uy']';
    BottomBoundary  = @(x,y) [x, r*sin(2*pi*x)];
    TopBoundary     = @(x,y) [x, 1+(y-y)];
    LeftBoundary    = @(x,y) [(x-x), y];
    RightBoundary   = @(x,y) [1+(x-x), y];

    LinearElasticity_Main
    r = r + kappa/iterations;
end %while

if min(min(Jacobian)) < 0
    disp('Negative Jacobian')
else
    Metrics
end %if
```

A.5 Shared Functions

The following functions and scripts are used in two or more of the solvers presented.

GenerateMesh.m

```
% Unique knots in xi and eta direction
uniqueX     = unique(KnotVecX);
uniqueY     = unique(KnotVecY);

% Number of elements in xi and eta direction, and in total
NoElemX     = length(uniqueX) - 1;
NoElemY     = length(uniqueY) - 1;
NoElem      = NoElemX * NoElemY;

% Element connectivities and range
ElemConn    = zeros(NoElem, (p+1)*(q+1) );
ElemRange   = zeros(NoElem, 4);

chan        = zeros(NoPtsY, NoPtsX);
counter     = 1;

for i = 1:NoPtsY
    for j = 1:NoPtsX
        chan(i,j) = counter;
        counter = counter + 1;
    end %j
end %i

[ElemRangeX, ElemConnX] = Connectivity(p, KnotVecX, NoElemX);
[ElemRangeY, ElemConnY] = Connectivity(q, KnotVecY, NoElemY);

e = 1;
for y = 1:NoElemY
    yConn = ElemConnY(y,:);
    for x = 1:NoElemX
```

```
        c = 1;
        xConn = ElemConnX(x,:);
        for i = 1:length(yConn)
            for j = 1:length(xConn)
                ElemConn(e,c) = chan(yConn(i), xConn(j));
                c = c + 1;
            end %j
        end %i
        e = e + 1;
    end %x
end %y

e = 1;
for i = 1:size(ElemRangeY,1)
    for j = 1:size(ElemRangeX,1)
        ElemRange(e,1:2)      = ElemRangeX(j,:);
        ElemRange(e,3:4)      = ElemRangeY(i,:);
        e                     = e + 1;
    end %j
end %i
```

Connectivity.m

```
function [ElemRange, ElemConn] = Connectivity(p, KnotVec, NoElem)
    % The function finds the range and connectivities
    % for each element in 1D

    ElemRange   = zeros(NoElem, 2);
    ElemConn    = zeros(NoElem, p+1);
    ElemKnotInd = zeros(NoElem, 2);

    element     = 1;
    previous    = KnotVec(1);

    for i = 1:length(KnotVec)
       current = KnotVec(i);
       if KnotVec(i) ~= previous
           ElemRange(element,:)    = [previous, current];
           ElemKnotInd(element,:)  = [i-1, i];
           element = element + 1;
       end %if
       previous = current;
    end %i

    NoRepeated = 0;
    for i = 1:NoElem
        ind         = ( ElemKnotInd(i,1) - p + 1 ):ElemKnotInd(i,1);
        previous    = KnotVec(ind);
        current     = ones(1,p) * KnotVec(ElemKnotInd(i,1));

        if isequal(previous, current) && (length(nonzeros(previous)) > 1);
            NoRepeated =  NoRepeated + 1;
        end %if

        ElemConn(i,:) = (ElemKnotInd(i,1) - p ):ElemKnotInd(i,1);
    end %i
end
```

FindMu.m

```
function mu = FindMu(KnotVec, x)
    % mu = the value for which xi_mu <= x < xi_(mu+1)

    if abs(x - KnotVec(end)) < eps
        x = KnotVec(end) - eps;
    end %if

    M = length(KnotVec);
    for i = 1:M-1
        if KnotVec(i) <= x && KnotVec(i+1) > x
            mu = i;
        end %if
    end %i

end
```

GetBasisAndDerivatives.m

```
function [basis, dNdxi, dNdeta] = GetBasisAndDerivatives(xi, eta, p, q, KnotVecX,
    KnotVecY)
    % The function returns the basis and first derivatives
    % with respect to the points xi and eta.

    if abs(xi - KnotVecX(end)) < eps
        xi = KnotVecX(end) - eps;
    end %if
    if abs(eta - KnotVecY(end)) < eps
        eta = KnotVecY(end) - eps;
    end %if

    % Get Basis:
    [N, dN] = GetBasisAndDerivatives1D(KnotVecX, p, xi);
    [M, dM] = GetBasisAndDerivatives1D(KnotVecY, q, eta);

    k = 1;
    for j = 1:(q+1)
        for i = 1:(p+1)
            basis(k) = N(i)*M(j);
            dNdxi(k) = dN(i)*M(j);
            dNdeta(k) = dM(j)*N(i);
            k = k + 1;
        end %i
    end %j

end
```

GetBasisAndDerivatives1D.m

```
function [N, dN] = GetBasisAndDerivatives1D(KnotVec, p, x)

    mu = FindMu(KnotVec, x);

    % Basis:
```

```
    N = 1;
    for k = 1:p
        R = 0;
        for i = 1:k
            R(i,i)   = (KnotVec(mu+i) - x)/(KnotVec(mu+i)-KnotVec(mu+i-k));
            R(i,i+1) = (x-KnotVec(mu+i-k))/(KnotVec(mu+i)-KnotVec(mu+i-k));
        end %i
        N = N * R;
    end %k

    % Derivative:
    dN = 1;
    for k = 1:(p-1)
        R = 0;
        for i = 1:k
            R(i,i) = (KnotVec(mu+i) - x)   /(KnotVec(mu+i)-KnotVec(mu+i-k));
            R(i,i+1) =(x-KnotVec(mu+i-k)) /(KnotVec(mu+i)-KnotVec(mu+i-k));
        end %i
        dN = dN*R;
    end %k

    for k = (p-1+1):p
        dR = 0;
        for i = 1:k
            dR(i,i) = (-1)  / (KnotVec(mu+i) - KnotVec(mu+i-k));
            dR(i,i+1) = (1) / (KnotVec(mu+i) - KnotVec(mu+i-k));
        end %i
        dN = dN*dR;
    end %k

    dN = (factorial(p)/factorial(p-1))^(1)*dN;

end
```

GetControlPts.m

```
xpts = ControlPoints(p, startX, endX, KnotVecX);
ypts = ControlPoints(q, startY, endY, KnotVecY);

k = 1;
controlPts = zeros(NoCtrlPts, 2);
for i = 1:NoPtsY
    for j = 1:NoPtsX
        controlPts(k,1) = xpts(j);
        controlPts(k,2) = ypts(i);
        k = k + 1;
    end %j
end %i
```

JacobianTest.m

```
%%% Get the Jacobian of the parameterization
Jacobian = zeros(NoElemY*NoPy, NoElemX*NoPx);
UU  = [Ux Uy];
xxx = 1;
```

```
yyy = 1;

for e = 1:NoElem
    % Element interval (xi_i, xi_(i+1))
    Xi_e    = ElemRange(e,1:2);
    Eta_e   = ElemRange(e,3:4);

    xi  = linspace(Xi_e(1), Xi_e(2)-eps, NoPx);
    eta = linspace(Eta_e(1), Eta_e(2)-eps, NoPy);

    Conn    = ElemConn(e,:);
    D       = zeros(NoPy, NoPx);

    for i = 1:NoPx
        Xi = xi(i);
        for j = 1:NoPy
            Eta = eta(j);

            [N, dNdxi, dNdeta] = GetBasisAndDerivatives(Xi, Eta, p, q, ...
                                                        KnotVecX, KnotVecY);

            jacob   = UU(Conn,:)'*[dNdxi' dNdeta'];
            detJ    = det(jacob);
            D(j,i)  = detJ;

        end %j
    end %i

    if mod(e, NoElemY) == 1 && e~=1
        yyy = yyy + NoPy;
    end %if

    Jacobian(yyy:yyy+NoPy-1,xxx:xxx+NoPx-1) = D;
    xxx = NoPx*mod(e,NoElemX) + 1;
end
```

Metrics.m

```
x = reshape(ux', [1,3600]);
y = reshape(uy', [1,3600]);
NoElemX = length(xx)-1;
NoElem = NoElemX^2;
geometry = 'sine';

[sizeMetric,shapeMetric,skewMetric]=GetMetrics(x,y,NoElem,NoElemX,geometry)

SizeShape   = sizeMetric.*shapeMetric;
SizeSkew    = sizeMetric.*skewMetric;

% Root-Mean-Square:
sizeRMS         = sqrt( (1/NoElem) * (sizeMetric'   *sizeMetric  ) );
shapeRMS        = sqrt( (1/NoElem) * (shapeMetric'  *shapeMetric ) );
skewRMS         = sqrt( (1/NoElem) * (skewMetric'   *skewMetric  ) );
SizeShapeRMS    = sqrt( (1/NoElem) * (SizeShape'    *SizeShape   ) );
SizeSkewRMS     = sqrt( (1/NoElem) * (SizeSkew'     *SizeSkew    ) );

% MinMax:
sizeMM      = min(sizeMetric)   / max(sizeMetric);
shapeMM     = min(shapeMetric)  / max(shapeMetric);
```

```
skewMM       = min(skewMetric)    / max(skewMetric);
SizeShapeMM  = min(SizeShape)     / max(SizeShape);
SizeSkewMM   = min(SizeSkew)      / max(SizeSkew);

RMS = [sizeRMS   shapeRMS     skewRMS  SizeShapeRMS     SizeSkewRMS];
MM  = [sizeMM    shapeMM      skewMM   SizeShapeMM      SizeSkewMM];
```

GetMetrics.m

```
function [sizeMetric, shapeMetric, skewMetric] = GetMetrics(x,y, NoElem, NoElemX,
    geometry)
    %(x,y) are the parameterization points

    totalArea   = zeros(NoElem,1);
    shapeMetric = zeros(NoElem,1);
    skewMetric  = zeros(NoElem,1);
    index       = zeros(NoElem,4);

    index(1,1:2) = [1  2];
    index(1,3:4) = [NoElemX+2 NoElemX+3];

    teller = 1;
    for i = 2:NoElem
        if mod(teller,NoElemX) == 0
            index(i,:) = index(i-1,:) + 2;
        else
            index(i,:) = index(i-1,:) + 1;
        end %if
        teller = teller + 1;
    end %i

    for i = 1:NoElem
        ind1 = index(i,1);
        ind2 = index(i,2);
        ind3 = index(i,3);
        ind4 = index(i,4);

        A1 = [x(ind2)-x(ind1) x(ind3)-x(ind1);
              y(ind2)-y(ind1) y(ind3)-y(ind1);];
        AA1 = A1'*A1;

        A2 = [x(ind4)-x(ind2) x(ind1)-x(ind2);
              y(ind4)-y(ind2) y(ind1)-y(ind2)];
        AA2 = A2'*A2;

        A3 = [x(ind3)-x(ind4) x(ind2)-x(ind4);
              y(ind3)-y(ind4) y(ind2)-y(ind4);];
        AA3 = A3'*A3;

        A4 = [x(ind1)-x(ind3) x(ind4)-x(ind3);
              y(ind1)-y(ind3) y(ind4)-y(ind3)];
        AA4 = A4'*A4;

        totalArea(i) = (det(A1) + det(A3))/2;

        if totalArea(i) < 0 || ...
            norm((det(A1)+det(A3))-(det(A2)+det(A4)))>2*eps
            totalArea(i) = 0;
        end %if
```

```
        shapeMetric(i) = 8/((AA1(1,1) + AA1(2,2))/det(A1) + ...
            (AA2(1,1) + AA2(2,2))/det(A2)    + ...
            (AA3(1,1) + AA3(2,2))/det(A3)    + ...
            (AA4(1,1) + AA4(2,2))/det(A4));

        if shapeMetric(i) < 0 || shapeMetric(i) > 1
            shapeMetric(i) = 0;
        end %if

        skewMetric(i) = 4 / (sqrt(AA1(1,1)*AA1(2,2))/det(A1) + ...
            sqrt(AA2(1,1)*AA2(2,2))/det(A2) +  ...
            sqrt(AA3(1,1)*AA3(2,2))/det(A3) + ...
            sqrt(AA4(1,1)*AA4(2,2))/det(A4));

        if skewMetric(i) < 0 || skewMetric(i) > 1
            skewMetric(i) = 0;
        end %if

    end %i

    NoElemY = NoElem / NoElemX;

    referenceSize = (1/NoElemX)*(1/NoElemY);
    tau = totalArea./referenceSize;
    sizeMetric = min(tau, 1./tau);

end
```

Appendix B

Jigsaw Geometry Specifications

The jigsaw geometry presented in Section 5.3 is defined by four curves, $\partial\Omega_1$, $\partial\Omega_2$, $\partial\Omega_3$ and $\partial\Omega_4$, with a spline representation. As stated in Section 3.1.5, a spline representation of a curve is given as

$$\mathbf{C} = \sum_i N_{i,p}(\xi)\mathbf{B}_i.$$

When defining the curves in the jigsaw geometry, the N_i's are the splines defined by the knot vector $\Xi = [\frac{0}{8}, \frac{0}{8}, \frac{0}{8}, \frac{1}{8}, \frac{2}{8}, \frac{3}{8}, \frac{4}{8}, \frac{5}{8}, \frac{6}{8}, \frac{7}{8}, \frac{8}{8}, \frac{8}{8}, \frac{8}{8}]$ and polynomial order $p = 2$. The control points $\mathbf{B}_i = [\partial\Omega_j(x_i), \partial\Omega_j(y_i)]$ for each of the four boundary curves $\partial\Omega_j$, $j = 1, 2, 3, 4$ are given in Table B.1.

Table B.1: Control points for the jigsaw geometry shown in Figure 5.37a.

$\partial\Omega_1(x)$	$\partial\Omega_1(y)$	$\partial\Omega_2(x)$	$\partial\Omega_2(y)$	$\partial\Omega_3(x)$	$\partial\Omega_3(y)$	$\partial\Omega_4(x)$	$\partial\Omega_4(y)$
0	0	10	0	0	10	0	0
3	0	10	3	3	10	0	3
4	1	11	4	4	9	-1	4
3	1.5	11.5	3	3	8.5	-1.5	3
4	3	13	4	4	7	-3	4
6	3	13	6	6	7	-3	6
7	1.5	11.5	7	7	8.5	-1.5	7
6	1	11	6	6	9	-1	6
7	0	10	7	7	10	0	7
10	0	10	10	10	10	0	10

Appendix C

List of symbols

A_i	Jacobian matrix of element, wrt node i.
α_i	Determinant of Jacobian matrix A_i
a	Area of reference element
A_{tot}	Total area of a domain
$\mathbf{A}$	Stiffness matrix
$a_{i,j}$	Element (i, j) of stiffness matrix $\mathbf{A}$
$\mathbf{B}_i$, $\mathbf{B}_{ij}$	Control points
$\mathbf{b}$, $\mathbf{c}^*$	Vector for imposing boundary conditions
$\mathbf{C}$	Spline curve
$\widetilde{C}$, C	Mappings between spaces
$\mathbf{D}$	Elasticity matrix
$\mathbf{e}$	Error, $e = u - u_h$
E	Young's modulus
$\boldsymbol{\varepsilon}$, ε_{xx}, ε_{yy}, ε_{xy}	Strain
$\mathbf{f}$	Force vector
f_j	Element of force vector
$\mathcal{F}$	Mapping
$\mathbf{G}$	Matrix for imposing boundary conditions
Γ_D	Dirichlet boundary
Γ_N	Neumann boundary
Γ_R	Robin boundary
$\gamma(x, y)$	Function used in the uncoupled Poisson method
J_ξ, $J_{\widetilde{\xi}}$	Jacobian of mappings
κ	Parameter of problem difficulty

λ_{kl}^{i}	Element kl of the matrix product $A_i^T A_i$
$\mathcal{M}_{Size}$	Size metric
$\mathcal{M}_{Shape}$	Shape metric
$\mathcal{M}_{Skew}$	Skew metric
$\mathcal{M}_{SizeShape}$	Combination metric size-shape
$\mathcal{M}_{SizeSkew}$	Combination metric size-skew
m_i^k	The k-metric of element i
$MM(\cdot)$	Min-Max measure
m	Multiplicity of a knot
$M_{j,q}(\eta)$	Spline j induced by $\mathcal{H}$ with polynomial degree q
n, m	Number of splines induced by Ξ, $\mathcal{H}$
N	Number of mesh elements
$N_{i,p}(\xi)$	Spline i induced by Ξ with polynomial degree p
$\mathbf{n}$	Normal vector
ν	Poisson's ratio
Ω	Physical domain
$\hat{\Omega}$	Parametric domain
$\widetilde{\Omega}$	Unit square $(x, y) \in [-1, 1]^2$
$\partial\Omega$	Physical domain boundary
p, q	Polynomial degree
$\varphi_i(\cdot)$	Polynomials
$RMS(\cdot)$	Root-Mean-Square measure
$\mathbb{R}$	Set of all real numbers
$\mathbf{R}$	B-spline matrix
σ_i	$(\alpha_i + \alpha_{i+2})/2a$
$\mathcal{S}_{p,\Xi}$	Space spanned by $N_{i,p}$ defined by Ξ
$\boldsymbol{\sigma}$, σ_{xx},σ_{yy},σ_{xy}	Stress
$\mathbf{u}$	Solution vector
u_h	Numerical solution
$w_{alpha,\beta}$	Weight of Gauss point
Ξ, $\mathcal{H}$	Knot vectors
ξ_i, η_i	Knot i in Ξ and $\mathcal{H}$
$\mathbb{Z}$	Set of all integers
$\|\| \cdot \|\|_{L_2(\Omega)}$	The L_2-norm on Ω
$\|\| \cdot \|\|_{E(\Omega)}$	The energy norm on Ω