Isogeometric Analysis: Fundamentals and details of implementation.

From first steps to two-dimensional non-linear problems.

Heidi Burger

Masters Dissertation
in the Department of Mechanical Engineering
Faculty of Engineering and the Built Environment
University of Cape Town

November 2018

Declaration

I, Heidi Burger, know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This dissertation has been submitted to the Turnitin module and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

I am now presenting the report for examination for the degree of M.Sc Eng (Mechanical).

Signed by candidate

Heidi Burger

November 21, 2018

APPLICATION FORM

Please Note:
Any person planning to undertake research in the Faculty of Engineering and the Built Environment (EBE) at the University of Cape Town is required to complete this form **before** collecting or analysing data. The objective of submitting this application *prior* to embarking on research is to ensure that the highest ethical standards in research, conducted under the auspices of the EBE Faculty, are met. Please ensure that you have read, and understood the **EBE Ethics in Research Handbook** (available from the UCT EBE, Research Ethics website) prior to completing this application form: http://www.ebe.uct.ac.za/ebe/research/ethics1

APPLICANT'S DETAILS		
Name of principal researcher, student or external applicant		Heidi Burger
Department		Mechanical Engineering
Preferred email address of applicant:		Heidi.burger42@gmail.com
If Student	Your Degree: e.g., MSc, PhD, etc.	MSc
	Credit Value of Research: e.g., 60/120/180/360 etc	120
	Name of Supervisor (if supervised)	Ernesto Bram Ismail
If this is a researchcontract, indicate the source of funding/sponsorship		Funding from supervisor, Daya Reddy and UCT internal funding
Project Title		Isogeometric Analysis for Computational Solid Mechanics

I hereby undertake to carry out my research in such a way that:

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

SIGNED BY	Full name	Signature	Date
Principal Researcher/ Student/External applicant	Heidi Burger	Signed by candidate	**22 Jan 2018**

APPLICATION APPROVED BY	Full name	Signature	Date
Supervisor (where applicable)	Ernesto Ismail	Signed by candidate	**22 Jan 2018**
HOD (or delegated nominee) Final authority for all applicants who have answered NO to all questions in Section1; and for all Undergraduateresearch (Including Honours).	Click here to enter text.	Signed by candidate	26/01/2018 Click here to enter a date.
Chair : Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the above questions.			

Abstract

Isogeometric analysis (IGA) is a computational analysis technique that can serve as an alternative to the traditional finite element method (FEM) in approximating solutions to differential equations. IGA is not necessarily more efficient that traditional FEM, but because of its nature, can naturally handle a greater variety of complex geometries. IGA is based on the use of NURBS (non-uniform rational B-splines), mathematical descriptions of geometry which are the standard of representing geometry in computer aided design (CAD) modeling software. IGA therefore links the CAD world to the world of analysis. Traditional FEM was developed before NURBS, in the 1950s and therefore developed quite separately.

This project focuses on the fundamentals and implementation of IGA for problems, including one-dimensional, two-dimensional scalar, two-dimensional vector-valued and simple non-linear problems. For each new problem, the underlying mathematics is developed and the implementation is discussed in detail. One of the major contributions of this project is considered to be the detail in which the implementation of the Neumann boundary condition is described. There is none of this level of detail in any of the available literature. All problems solved are demonstrative and was written in a modular way that is easy to read and understand. Furthermore, how to extract NURBS data from CAD software is discussed, which would prove useful for future problems with more complex geometry.

While the work done in this project is not considered novel, the thoroughness in which the project was approached is hoped to be useful for future projects. From this project, the work can be expanded to more complex geometries, multi-patch problems with the help of CAD programs or more complex non-linear problems.

Acknowledgements

I could not have done this project alone and would like to thank everyone that helped me finish and enjoy this masters:

First and foremost, thank you to my supervisor, Ernesto, for everything. I throughly enjoyed working with you and you lived up to all expectations and more. There was so much that I learnt from you in addition to computational mechanics and I have the utmost respect for you. My masters journey would have been a lot less interesting and probably a lot longer without you! Thank you.

Thank you to Prof Reddy for co-supervising my masters and supporting my CERECAM team-building pub-quiz evenings. It has been great to work in your research group.

Thank you to Natalie Bent for being the admin queen and going over and above to make sure that the admin side of my masters was completely painless. It has been great to get to know you over the past two years.

Thank you to Emma Griffths and Nkosi Vundla for always being happy to take the time out of your respective PhDs and masters to help me. And a big thank you to you guys and the rest of the CERECAM lunch time crew for ensuring that I will only ever be able to eat lunch at 12.30 sharp from now on!

Thank you to my fellow masters friends, Dane and Aaron for being my moral and procrastinating support for the past few years and for always being keen for an afternoon drink at the pub.

Thank you to my family for annoyingly never doubting my ability to hand in on time.

Finally, thank you to Mike for bringing me picnics on campus when I was too busy to do anything else!

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Isogeometric analysis (IGA) is a computational analysis technique that can serve as an alternative to the traditional finite element method (FEM) in approximating solutions to differential equations. IGA is not necessarily more efficient than traditional FEM, but by nature can naturally handle a larger variety of complex geometries. IGA is based on NURBS (non-uniform rational B-splines); mathematical descriptions of geometry which are the standard for representing geometry in computer aided design (CAD) modeling software [1, p. 7]. IGA therefore links the CAD world to the world of analysis. NURBS are an extension of Bézier curves, which were first introduced by Pierre Bézier, an engineer at the car-manufacturing company, Renault, in the 1960s [2]. NURBS are an amalgamation of rational B-splines and non-uniform B-splines, first used by Boeing in the 1980s [2]. Traditional FEM developed before NURBS, in the 1950s [3] and therefore developed quite separately.

Hughes *et al.*[1] report that one of the most time consuming parts of finite element analysis is the creation of analysis-suitable geometry and mesh generation (about 80% of the total analysis time), which motivated them to find a more efficient way to integrate engineering design and the analysis process. Isogeometric analysis was the result of their research [1].

While there are many academic papers on IGA, most focus on the underpinning mathematics or on results, but skip over the implementation process. [1] and [3] cover the founding ideas of IGA and present the concept well, but brush over many of the implementation steps. [4] on the other hand, explains some of the implementation, but oversimplifies the ideas, omitting many key features. It is therefore beneficial that [5] does a good job at explaining the fundamentals and implementation of NURBS, which provides a solid foundation for introducing IGA.

For a newcomer to the field, a significant period of time may be required to understand the concepts, understand the key features and develop the code. This dissertation aims to be a comprehensive guide to the mathematics of IGA and NURBS and to bridge the gap in literature, detailing the implementation of various IGA problems. Focus is placed on the implementation and the results are briefly discussed. While the work in this dissertation cannot be considered novel, it is hoped that it will be valuable to researchers wanting to learn how to implement IGA. In order to aid the reader to learn IGA, where possible, the relevant page numbers will be included next to references of supporting research.

The objectives of this dissertation are therefore to introduce the fundamentals of NURBS along with how to implement the code to describe NURBS curves and surfaces, introduce the fundamentals of IGA and how to implement it for various problems, encompassing one-dimensional, two-dimensional scalar and vector-valued problems, including simple non-linearity, and finally to discuss more complex geometries and how they can be obtained. This is done by developing an in-house code that approximates various problems.

1.1 Scope and Limitations

The intention is that this dissertation serve as a comprehensive guide on how to implement IGA for the approximations of solutions to various problems. The code is intended to be easy to read and understand by someone with a background in finite element analysis. The code has therefore not been optimized for speed or performance. MATLAB® is used for this project for this reason. Because the problems that are solved are demonstrative and quick to solve, the cost of running and computational efficiency was not important for this project.

Where possible, a modular library of functions was created that can be used in various applications. For example, the function that creates a NURBS shape, can be used to do so in a one-, two- or three-dimensional physical domain. This also allows anyone wanting to take this project further the opportunity to use these functions in their own code or to improve the efficiency of the functions.

It was decided to perform NURBS constructions and IGA analysis only on curves and surfaces and to omit solids. The steps to extend the NURBS and IGA to three-dimensional solids is primarily a "book-keeping" exercise, but were omitted due to the time constraints.

1.2 Plan of Development

This dissertation starts with the introduction of B-splines, from which NURBS descend. These are then extended to a more complete description of NURBS in Chapter 2. Details of NURBS usage, including representation of key elements of geometry and concepts of refinement are introduced so that they are in place when NURBS are used in analysis. This is followed by a section focusing on how to implement and apply NURBS curves and surfaces, including how to represent certain key geometries with NURBS, such as circles.

In Chapters 3-6, the majority of the dissertation, focuses on isogeometric analysis (IGA). The fundamentals of IGA are introduced with a one-dimensional Poisson equation, after which a one-dimensional radial temperature problem is used to introduce specific features. Thereafter, the same radial temperature problem is revisited using two-dimensional NURBS, to introduce two-dimensional scalar problems. Additional features relating to refinement and geometry are further discussed with application to a heat conduction problem. The geometry used represents a plate with a centrally located hole. The same geometry is then used to approximate linear elastic deformation to introduce two-dimensional vector-valued problems. Various features and complexities are highlighted in this problem. Non-linear problems account for a significant proportion of computational analysis problems. This dissertation does not attempt to tackle these in depth, but simple geometric non-linearity is considered to demonstrate how IGA can be extended to this type of problem. The final chapter of this dissertation introduces techniques to extract more complex NURBS data from commercial software, such as Rhinoceros 3D.

Recommendations and conclusions are made in Chapter 8. Recommendations are made as to how the current project can be improved, as well as ideas on how this project can be taken further are outlined. The conclusion summarizes what was achieved with this dissertation. In the appendices, the details of notations are explained, as well as sets of sample code for various sections. There is also a digital portion to this project, which includes all the code that has been developed and a readme file, `IGA_readme` outlines how to access and use each code. A list of all available code is provided in the appendices. The link to the digital portion to this project is:

https://1drv.ms/f/s!AotxHfVTLUe8gtcV2hKpmQ39V6CvXg

Chapter 2

Non-Uniform Rational B-Splines

2.1 Parametric Forms

There are multiple ways to represent geometry such as curves, surfaces and solids. Describing this type of geometry is commonly done in a non-parametric method. For example, a curve or surface on the xy-plane is represented by an equation: $f(x, y) = 0$, where $f(x, y)$ is the function describing the relationship between the variables, x and y. An example of a curve in a two-dimensional domain, is a circle with unit radius about the origin, where the function is described by:

$$f(x, y) = x^2 + y^2 - 1 \quad . \tag{2.1}$$

Geometry can also be represented in a parametric form. Parametric equations are equations that describe points in a domain using an additional variable, called a parameter, t. Each value of the parameter determines a point in the domain, depending on the parametric equations [6]. A curve in the xy-plane can be described by

$$\boldsymbol{C}(t) = (x(t), y(t)) \qquad\qquad a \leq t \leq b \tag{2.2}$$

where a and b are the limits of the parametric domain. The parametric equations

$$\left.\begin{array}{l} x(t) = \cos(t) \\ y(t) = \sin(t) \end{array}\right\} \qquad 0 \leq t \leq \frac{\pi}{2}$$

can be used to describe the same domain as seen in equation (2.1) of a circle with unit radius about the origin.

A single parameter is used to describe a curve, two parameters to describe a surface and three parameters to describe a solid. There are numerous advantages and disadvantages for using either parametric and non-parametric forms. For different tasks, each form would be differently suited. For example, the limits of a non-parametric curve need to be additionally specified, whereas in the parametric form, it is naturally included. Conversely, it is difficult to express an infinite curve parametrically. It is convenient to check whether a particular point lies on a curve or surface by substituting the point into the given non-parametric equation. With a parametric description, although easy to calculate a particular point, it is difficult to check whether a certain point is on the curve. Refer to [5, Sec. 1.1] for more information on how geometry can be expressed.

Purely parametric forms, non-parametric equations and by extension, polynomials (a type of a non-parametric equation) have their particular benefits in certain applications, however, it is generally not intuitive as to how they will look by just looking at the equation or parametric description. Drawing shapes is easy for a human, but apart from the most basic shapes, defining the shape algebraically is not. For a geometry designer, a visual approach is beneficial, as the geometry can be altered more naturally with the coefficients and parameters more intuitively linked. There are several parametric forms that can describe complex geometry. This dissertation mainly uses NURBS (non-uniform rational B-splines) to describe the geometry and can be understood more easily by first considering non-rational B-splines, often referred to simply as B-splines. NURBS are a superset of B-splines and can collapse to B-splines. Two other methods of functions are the power basis and Bézier methods. Details of both the power basis and Bézier method can be found in [5, p. 5].

2.2 B-Splines

While NURBS are required for IGA, the concepts are most simply demonstrated using B-splines. Both B-splines and NURBS represent a shape in the physical domain that has been mapped from a region in the parametric domain using basis functions and control points. The physical domain is the domain in the real world in which shapes are described. In order to define a curve parametrically, linear combinations of overlapping basis functions can be taken as

$$\boldsymbol{C}(\xi) = \sum_{i=1}^{n} N_i^p(\xi)\boldsymbol{P}_i \tag{2.3}$$

where each value $\boldsymbol{P}_i$ determines the contribution of a specific basis function $N_i^p(\xi)$ to the curve $\boldsymbol{C}(\xi)$ [4, p. 10], which can be called the control point. The values of $\boldsymbol{P}_i$ are collected in a vector $\boldsymbol{P}$ and can be visualized in the same domain as $\boldsymbol{C}(\xi)$. When the control points are connected together, they are referred to as a control polygon. The term control polygon is used generically across all three dimensions, but is technically a control net for two-dimensional B-splines and a control lattice in three-dimensional B-splines.

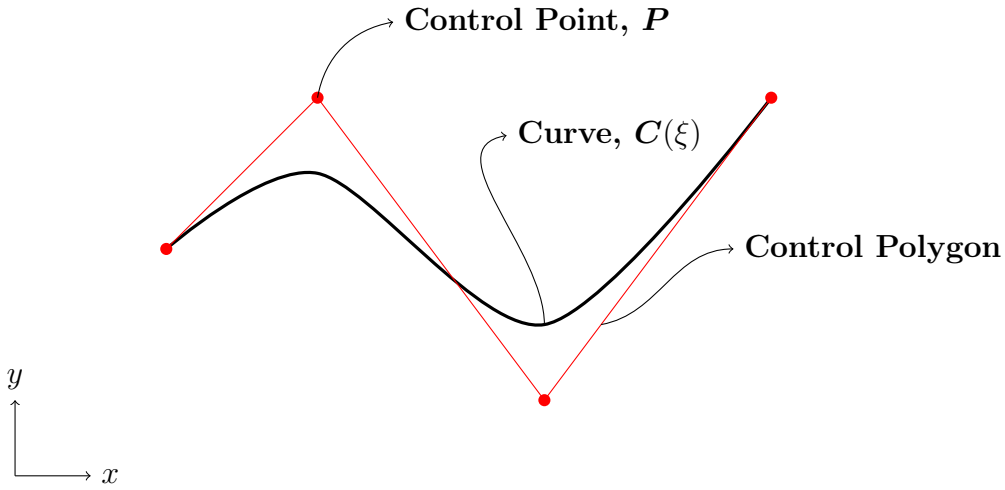


Figure 2.1 Demonstrative B-spline curve and control points in physical space

$\boldsymbol{C}(\xi)$ is the B-spline curve, corresponding to a vector in the parameter domain, ξ. The vector in the parametric domain depends on which points are required to be projected into the physical domain. The vector can be a single point, a uniformly spaced vector of multiple points or non-uniformly spaced. The basis functions, $N_i^p(\xi)$, are a function of the

knot vector (Ξ), the required order (p), the basis function number (i) and the number of basis functions across the domain (n).

The parametric domain is defined by the knot vector, Ξ. A knot vector is an ascending list of values in the parametric domain of the form: $\Xi = \{\xi_1, \xi_2, \xi_3, \ldots, \xi_{n+p+1}\}$ where n and p are the same as above. If unique knot values are evenly spaced, the knot vector is said to be uniform and conversely, non-uniform if unevenly spaced. An example of a uniform knot vector is

$$\Xi = \{0, 0, 0, 1, 2, 3, 4, 5, 5, 5\} \quad .$$

An element is defined as a knot span in the parametric domain; the difference between two adjacent knot values. The number of times that a unique knot value is repeated, is referred to as the multiplicity of a knot value, m_i. For example, the knot vector above has a multiplicity of three at the first and last knots and multiplicities of one at the interior knot values. Furthermore, the vector above is said to be open if the start and end knots have multiplicities of $p + 1$. This implies that the boundaries are discontinuous. If the multiplicity was less than $p + 1$, the knot vector would be closed. Open knot vectors are required for the imposition of boundary conditions in IGA. An open knot vector is therefore always of length $n + p + 1$. This is an important feature to note when performing basis function calculations. It can be useful to have the knot vector domain as $[0, 1]$. To convert the above knot vector to a unit domain, simply divide each component by the largest value in the knot vector. It will have no difference to the final solution. It is also possible to shift the knot vector in the parametric domain with no effect to the final result [1, p. 60].

The basis functions are calculated with the Cox-de Boor recursion formula [4]

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi) \tag{2.4}$$

where when p=0:

$$N_i^{p=0}(\xi) = \begin{cases} 1, & \text{if } \xi_i \geq \xi > \xi_{i+1} \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

ξ is a value in the parametric domain that is mapped to the physical domain and ξ_i (with subscripts) refers to individual entries in the knot vector, Ξ. During the recursive function, if any of the rational fractions go to $\frac{0}{0}$, they are defined as zero. The order of a basis function is denoted by using the superscript. There is no danger of confusion in this notation, because the functions are never exponentiated.

The first derivative of a B-spline basis function is also a recursive function

$$\frac{d}{d\xi}N_i^p(\xi) = \frac{p}{\xi_{i+p} - \xi_i}N_i^{p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}N_{i+1}^{p-1}(\xi) \quad . \tag{2.6}$$

The first order derivatives are used to create the stiffness matrix in IGA. The algorithm for higher order derivatives can be found in [1, p. 28], although higher order derivatives were not used in this dissertation.

B-spline basis functions have the following properties:

1. They form a partition of unity: $\sum_{i=1}^{n} N_i^p(\xi) = 1$
2. Each basis function is non-negative over the domain: $N_i^p(\xi) \geq 0$ for all $i{=}1 : n$.
3. They are linearly independent.
4. Each basis function has exactly one maximum on the interval.
5. If basis functions are of order p there will be $p - m_i$ continuous derivatives across knots, ξ_i, if the multiplicity of the knot is m_i.
6. If a control point $\boldsymbol{P}_i$ is changed, only the section of the curve related to the interval $\xi \in [\xi_i, \xi_{i+p+1})$ is changed [5, p. 120].

If a control point $\boldsymbol{P}_i$ or weight w_i is changed, only the section of the curve related to the interval $\xi \in [\xi_i, \xi_{i+p+1})$ is changed [5, p. 120].

Figure 2.2 shows the basis functions for a particular knot vector. The knot vector is open and the multiplicity of the interior knots are varied to illustrate the effect of increasing the multiplicity. Where the multiplicity is less than $p + 1$, the basis functions are C^{p-m_i} at each knot value and are therefore interpolatory between control points.

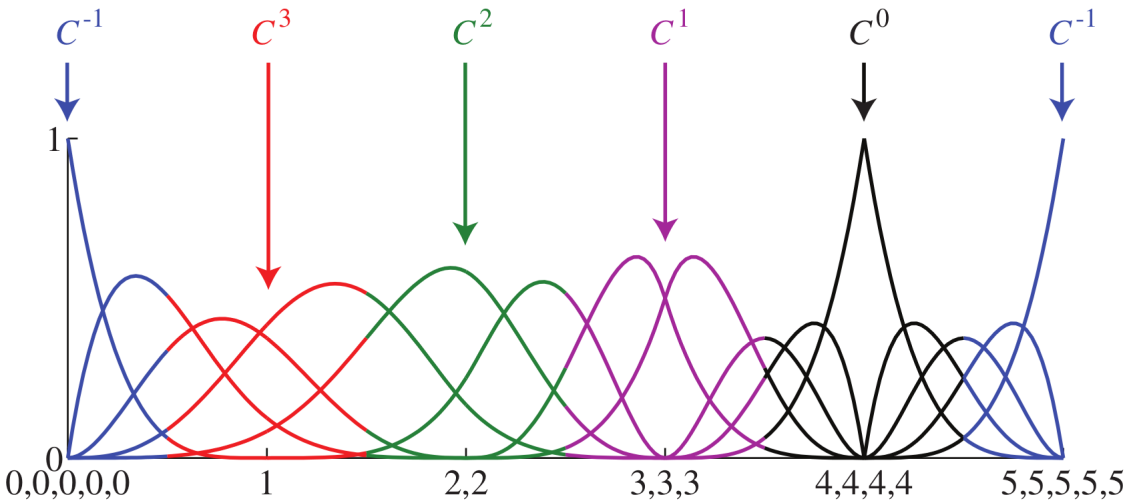


Figure 2.2 Fourth order ($p = 4$) basis functions for an open, non-uniform knot vector $\Xi = \{0, 0, 0, 0, 0, 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5\}$. From [1, p. 23].

The control points $\boldsymbol{P}$ are revisited. $\boldsymbol{P}$ is a set of points in the physical domain that can be used to describe a curve in one-, two- or three-dimensional domain. An example of a three-dimensional curve is a helix. A surface can exist in a two- or three-dimensional domain and a solid exists in a three-dimensional domains. In a one-dimensional physical domain, each point in $\boldsymbol{P}$ will be a single value (x). In a two dimensional physical domain, each point will be a point with two coordinates (x, y) and in a three-dimensional physical domain it will be a point with three coordinates (x, y, z).

An example of a B-spline curve can be seen in Figure 2.3. Control points are indicated by red circles and the interpolated lines between the circles make up the control polygon. The curve is C^1 continuous everywhere, except at the ends where there are repeated knot values.

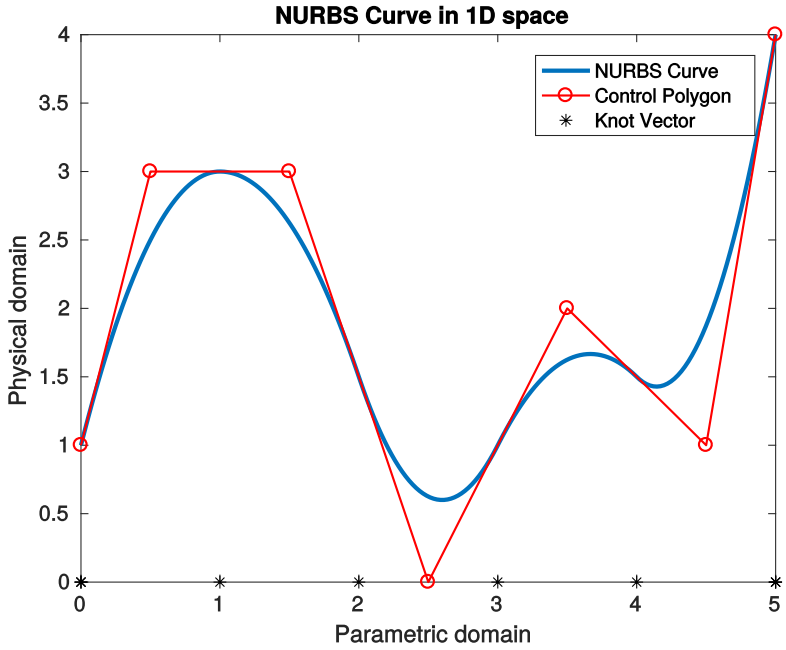


Figure 2.3 B-spline curve with knot vector: $\Xi = 0, 0, 0, 1, 2, 3, 4, 5, 5, 5$ and order $p = 2$.

B-splines can also describe surfaces and solids. For a B-spline surface, there are two parametric directions, ξ and η, with corresponding knot vectors Ξ and $\mathscr{H}$. The order of the basis functions in each direction, p and q, can be different. There are n basis functions in the ξ direction and m basis functions in the η direction. The control points are stored in a $n \times m$ two-dimensional array of points (2D or 3D points) and denoted by $\boldsymbol{P}_{ij}$.

The surface is described by

$$\mathbf{C}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_i^p(\xi) N_j^q(\eta) \boldsymbol{P}_{ij} \quad . \tag{2.7}$$

This can also be written more compactly as

$$\mathbf{C}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,j}^{p,q}(\xi, \eta) \boldsymbol{P}_{ij} \tag{2.8}$$

where

$$N_{i,j}^{p,q}(\xi, \eta) = N_i^p(\xi) N_j^q(\eta) \quad . \tag{2.9}$$

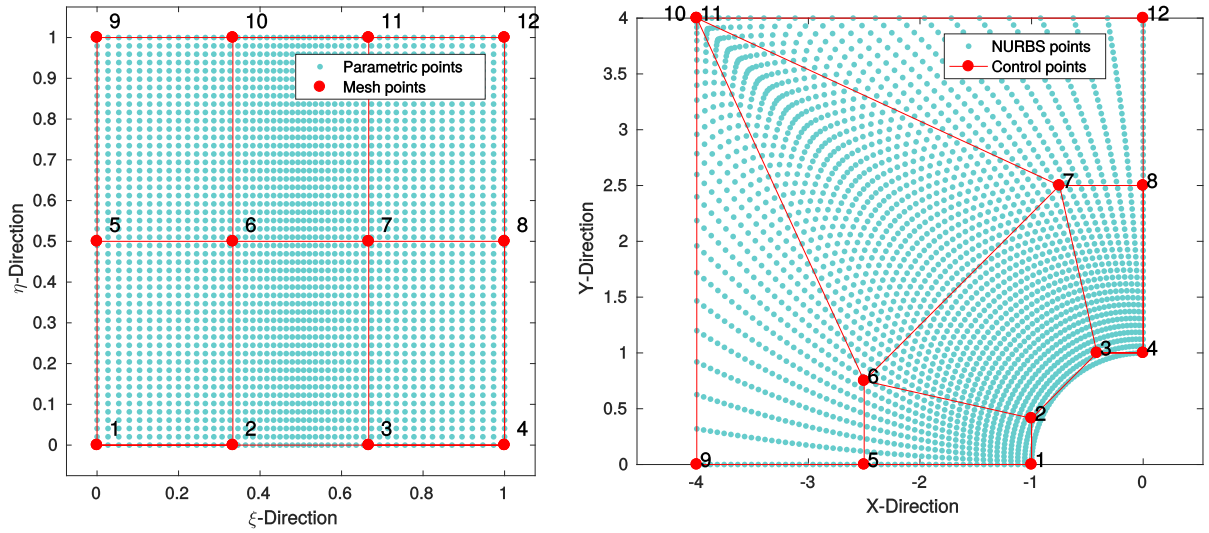
Similarly, a solid requires a three-dimensional parametric domain (ξ, η and ζ) with knot vectors Ξ, $\mathscr{H}$ and $\mathscr{Z}$ and orders p, q and r. The governing equation for a solid is

$$\mathbf{C}(\xi,\eta,\zeta) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} N_i^p(\xi)N_j^q(\eta)N_k^r(\zeta)\boldsymbol{P}_{ijk} \tag{2.10}$$

or compactly as

$$\mathbf{C}(\xi,\eta,\zeta) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} N_{i,j,k}^{p,q,r}(\xi,\eta,\zeta)\boldsymbol{P}_{ijk} \quad . \tag{2.11}$$

In Figures 2.4b and 2.4a, the mapping between the parametric and physical domain can be seen graphically. A series of parametric points are selected in the parametric domain, that are then mapped as points on the NURBS surface in the physical domain, using the basis functions and control polygon. In the parametric domain, the control polygon is represented by a simple rectilinear grid. This is useful in isogeometric analysis.



(a) Control points in parametric domain

(b) NURBS surface with control points in physical domain

Figure 2.4 NURBS surface

2.3 NURBS

Where B-splines fail to create conic shapes, such as circles, ellipsoids, paraboloids and hyperboloids [4, p. 11], NURBS do not have this limitation. Non-uniform rational B-splines (NURBS) are a superset of B-splines, as NURBS collapse to B-splines when the control point weights are all the same.

A NURBS curve

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} R_i^p(\xi)\boldsymbol{P}_i \tag{2.12}$$

is constructed in the same way to a non-rational B-spline curve, except that the basis functions, $R_i^p(\xi)$, are different.

The NURBS basis function

$$R_i^p(\xi) = \frac{N_i^p(\xi)w_i}{W(\xi)} \tag{2.13}$$

where

$$W(\xi) = \sum_{\hat{i}=1}^{n} N_{\hat{i}}^p(\xi)w_{\hat{i}} \tag{2.14}$$

can be constructed using the B-spline basis functions, where w_i is the weight which is collected in a vector $\boldsymbol{w}$. The properties of NURBS basis functions are the same as the B-spline basis function properties listed on page 8.

NURBS are the projective transformation of B-splines [1]. This means that a larger range of geometries can be created from the B-spline formulation by using the projective weights. There is an associated weight to each control point that scales the control points. If $\boldsymbol{P}$ is a vector of control points and $\boldsymbol{w}$ is a vector of the associated weights, the projected control points can be described as

$$\boldsymbol{P}_i^w = [\boldsymbol{P}_i w_i,\ w_i] \tag{2.15}$$

where each component in the first columns are multiplied by the associated weight and the last column contains the weight. More details on projective transformations may be found in Appendix B.

The derivative of the NURBS basis function can be calculated using the quotient rule to give

$$\frac{d}{d\xi}R_i^p(\xi) = \frac{\frac{d}{d\xi}N_i^p(\xi)\ W(\xi) - N_i^p(\xi)\ W'(\xi)}{W(\xi)^2} \tag{2.16}$$

where

$$W'(\xi) = \sum_{i=1}^{n} \frac{d}{d\xi}N_i^p(\xi)w_i \quad . \tag{2.17}$$

NURBS surfaces and solids are constructed similarly to B-spline surfaces and solids. Using the notation introduced in equation (2.9), a surface can be constructed from

$$\mathbf{C}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q}(\xi,\eta)\boldsymbol{P}_{ij} \tag{2.18}$$

where

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_i^p(\xi)N_j^q(\eta)w_{ij}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m} N_{\hat{i}}^p(\xi)N_{\hat{j}}^q(\eta)w_{\hat{i}\hat{j}}} \tag{2.19}$$

and w_{ij} is the weight associated with the control point $\boldsymbol{P}_{ij}$. Note that w_{ij} refers to a single weight, not a pair of weights. The i and j refer to the control point number in each direction. Refer to Figure 2.5 for the matrix labeling convention used in this dissertation. For example, the 11 entry in a matrix sits on the bottom left corner in the physical domain.

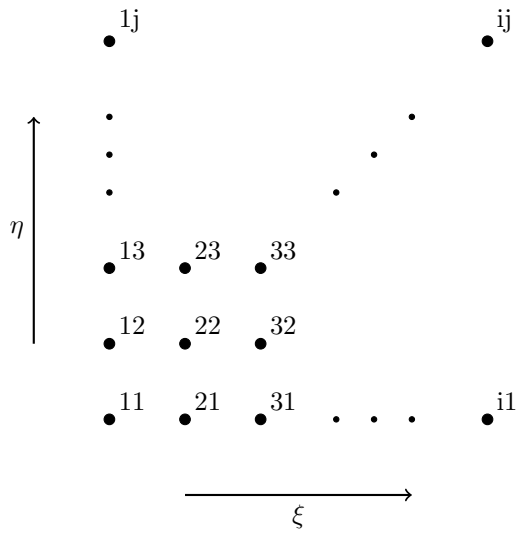


Figure 2.5 Diagram showing control points and labeling, as a rectilinear grid in the parametric domain.

By extension, a solid can be constructed with

$$\boldsymbol{C}(\xi,\eta,\zeta) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta)\boldsymbol{P}_{ijk} \quad . \tag{2.20}$$

Although modern CAD (computer aided design) software appear to use solid modeling, many of these programs actually use surfaces with no interior to represent graphics[1]. Although the interior is not important in CAD, it has a significant influence in FEA where it provides structural strength. This is an important concept to note for future work where the geometry will be obtained directly from CAD programs. For this project however, only surfaces will be considered.

2.4 Refinement

Refinement of the spatial domain can be carried out in three distinct manners in isogeometric analysis (IGA). Two are analogous to traditional FEM refinement and the other is unique to IGA [3]. Refinement is performed in the spatial domain, but becomes useful in IGA to capture more accurate results. So while the geometry requires minimal control points to be described accurately, the field variable often requires a more refined mesh to capture the solution field accurately. If the parametric space is refined (knot vector or order), the control points in the physical domain also change.

A NURBS curve is described by

$$\mathbf{C}^w(\xi) = \sum_{i=1}^{n} R_i^p(\xi)\boldsymbol{P}_i^w \tag{2.21}$$

where $\mathbf{C}^w(\xi)$ is a projected NURBS curve, using the weighted control points that are described in equation (2.15). After refinement, the refined NURBS curve is identical to the original curve and can be described by

$$\mathbf{C}^w(\xi) = \sum_{i=1}^{\bar{n}} R_i^{\bar{p}}\boldsymbol{Q}_i^w \tag{2.22}$$

where the $\bar{n}, \bar{p}$ and $\boldsymbol{Q}$ refer to the new refined parameters. The NURBS curves are identical. To solve for the new refined curves for each of the refinements, this principle will hold. This is different to traditional FEM, where the approximated geometry changes significantly with refinement.

2.4.1 *h*-refinement

h-refinement refers to increasing the number of elements that describe the domain to improve the accuracy of results. *h*-refinement in IGA works similarly to *h*-refinement in traditional FEM. However, it is important to note that refinement in IGA occurs in the parametric space, where the knot vectors are refined and not the control points directly. Consequently, because knot values are added, the control points will change. The number of control points are a function of the knot vector length and the order of the basis functions ($n = length(\Xi) - p - 1$). Therefore, because the number of control points are directly proportional to the number of basis functions, increasing the number of elements

results in an increase in the number of basis functions which allows for a more accurate representation of the field solution being described.

h-refinement in IGA is essentially knot insertion. h-refinement can be performed locally and globally. Local h-refinement refers to single knot insertions to refine one section of the domain. Global h-refinement refers to multiple knot insertions across the whole domain. To solve for the new control points, $\boldsymbol{Q}^w$, that are associated with the new knot vector, consider the curve $\boldsymbol{C}(\xi)$ defined by the control points $\boldsymbol{P}_i^w$. The knot vector can be modified and the new control points can be computed which yield the same curve.

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{n} R_i^p(\xi)\boldsymbol{P}_i^w = \sum_{i=0}^{n+1} \bar{R}_i^p(\xi)\boldsymbol{Q}_i^w \tag{2.23}$$

This method, however, is computationally expensive and requires one to repeat the same calculations multiple times and then solve a system of simultaneous equations. The only control points being affected are the ones close to the new knot that is being inserted and therefore this process can be made more efficient by only calculating those local control points and not recalculating the entire set of control points. Thereby, the number of calculations can be reduced. The algorithm is shown below, but for more detail on why this method works, refer to [5]. To calculate the new control points, the following formula is used:

$$\boldsymbol{Q}_i^w = \alpha_i \boldsymbol{P}_i^w + (1 - \alpha_i)\boldsymbol{P}_{i-1}^w \tag{2.24}$$

where

$$\alpha_i = \begin{cases} 1, & \text{if } i \leq k - p \\ \dfrac{\hat{\xi} - \xi_i}{\xi_{i+p} - \xi_i}, & \text{if } k - p + 1 \leq i \leq k \\ 0, & \text{if } i \geq k + 1 \end{cases} \tag{2.25}$$

$\hat{\xi}$ is the new knot value to be inserted and k is the index after which the knot should be inserted. In other words, $\hat{\xi}$ will be inserted into Ξ between $\{\xi_k, \xi_{k+1}\}$. The new control points, $\boldsymbol{Q}_i$ will be the same if the new knot values lies out of the element span and interpolatory if the knot is in the element span. This process must be repeated for each knot that is inserted. Algorithms that allow for multiple concurrent knot insertions exist, but for the sake of understanding the essence of knot refinement, single knot insertion was implemented. See [5, p.162] for more information on simultaneous knot insertions. Knot removal was not implemented, but additional information can be found in [5, p. 170].

In this dissertation, global h-refinement is implemented by halving the elements at each subsequent refinement level. This is done by inserting a knot between two element boundaries until all the elements are halved. This can be done for multiple levels. h-refinement in this dissertation refers to global h-refinement unless stated otherwise.

Often, the level of refinement required to accurately describe the geometry and the level of refinement required to describe the field parameter are different; however, because the stiffness matrix and force vectors are generated from the basis functions of the geometry, the levels of refinement must be the same. In other words, the mesh used to describe the geometry will be refined to obtain accurate finite element results. After calculating the solution field control points, the NURBS geometry can be described with either the refined or unrefined control points.

In order to visualize the effect of global knot insertion on the control points and knot vector, a parabola is represented using NURBS. Consider the function

$$f(x) = -2x^2 + 2x \quad x \in [0, 1]. \tag{2.26}$$

This curve is described exactly with a single element NURBS curve of order $p = 2$ in Figure 2.6. Figure 2.7 shows one level of refinement of the original mesh. This curve uses two elements and four control points. The NURBS curve in Figure 2.8 is made up of four elements and six control points, where the original mesh is refined twice. In terms of describing a second order polynomial, this level of mesh refinement is not required, but may be required to obtain the desired level of accuracy of the numerical approximations of a given field variable in the isogeometric analysis.

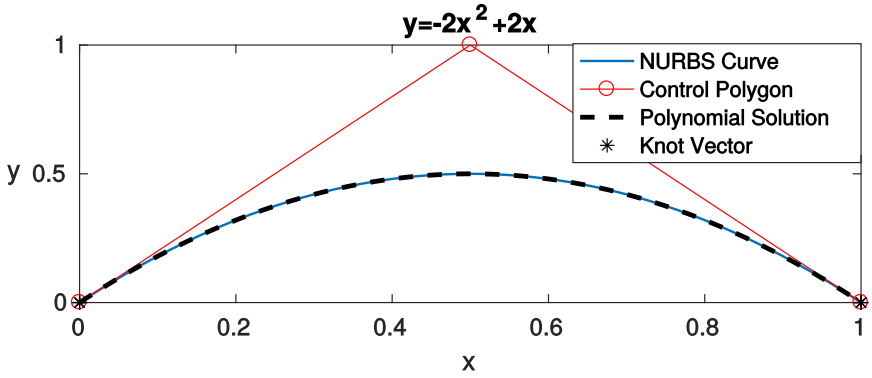


Figure 2.6 1D NURBS plot for $y = -2x^2 + 2x$ for no h-refinement

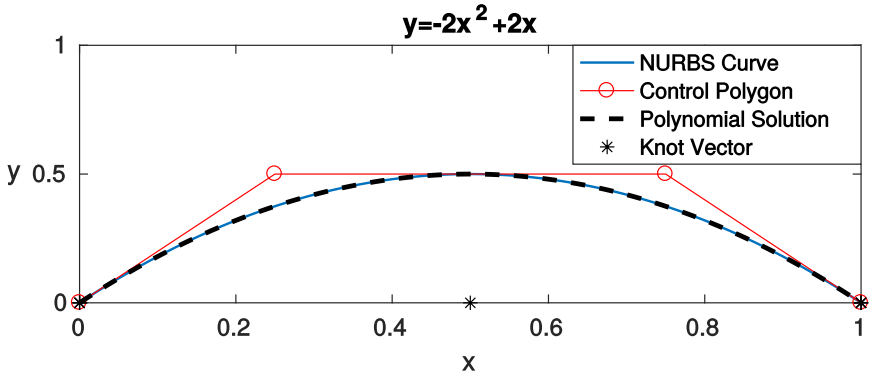


Figure 2.7 1D NURBS plot for $y = -2x^2 + 2x$ for one level of h-refinement

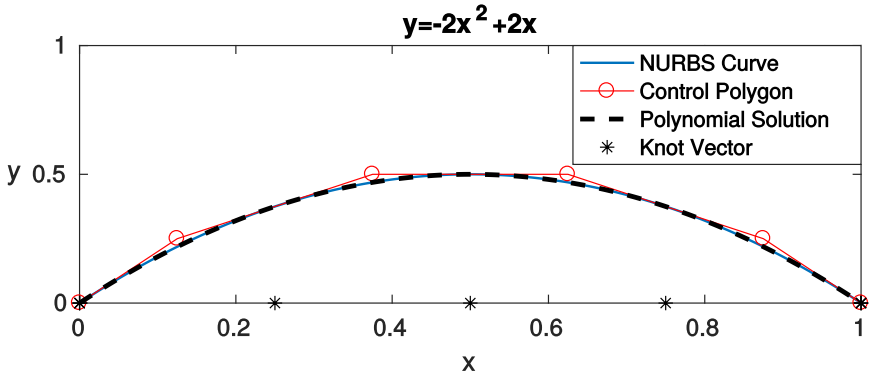


Figure 2.8 1D NURBS plot for $y = -2x^2 + 2x$ for two levels of h-refinement

In Figures 2.9-2.11, the basis functions corresponding to the graphs in Figures 2.6 to 2.8 are shown. Note the partition of unity of the basis functions across the domain. At each element boundary at least one basis function reduces to zero and another one starts at zero for the next element. A key feature of IGA is that the number of basis functions across

the domain are the same as the number of control points as seen in the respective graphs in Figures 2.6 to 2.8. Each control point is associated with a basis function.

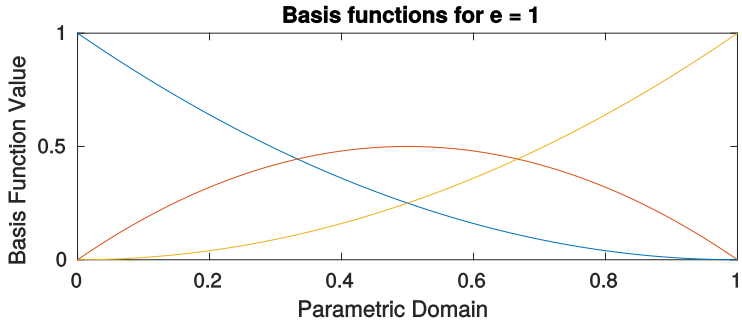


Figure 2.9 1D NURBS basis functions with one element, $p = 2$, $\Xi = [0, 0, 0, 1, 1, 1]$.

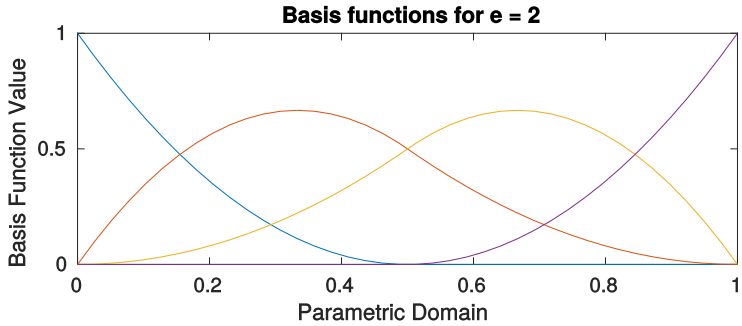


Figure 2.10 1D NURBS basis functions with two elements, $p = 2$, $\Xi = [0, 0, 0, 0.5, 0.5, 1, 1, 1]$.

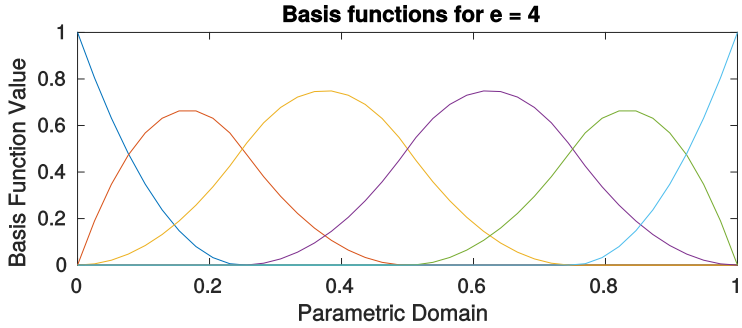


Figure 2.11 1D NURBS basis functions with four elements, $p = 2$, $\Xi = [0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1]$.

A key feature is that non-zero basis function are C^{p-1} continuous across element boundaries. This is in contrast to traditional FEM basis functions that are C^0 continuous across element boundaries. The change of continuity across the boundaries is explained further in the p-refinement section.

2.4.2 p-refinement

p-refinement in IGA, also called order elevation [1] or degree elevation [5] is similar to p-refinement in traditional finite element analysis, where the polynomial order of the basis functions [1] is increased. In IGA, this is done by increasing each of the knot multiplicities by one, although it is important to note that no new knot values are added, but rather existing knot values are repeated. This ensures that the number of elements remains constant. This leads to the new curve being identical to the original curve in a geometric and parametric sense [5]. Once again consider a curve

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{n} R_i^p(\xi)\boldsymbol{P}_i^w = \sum_{i=0}^{\hat{n}} R_i^{p+1}(\xi)\boldsymbol{Q}_i^w \tag{2.27}$$

from which the new control points, $\boldsymbol{Q}^w$, can be calculated by solving a system of linear equations made up of a set of collocation points (arbitrary points across the parametric domain). This is not the most efficient way to solve for the new control points, but is the simplest. The number of collocation points is equal to the number of new control points ($\hat{n}$). There are multiple other p-refinement algorithms that have been developed and are listed in [5, p. 200]. This section also describes another mathematically simple algorithm that can be found in [5, p. 201].

Figures 2.12 to 2.14 show the progression of p-refinement of a two element domain. It can be seen that the curve remains identical, but the control points change, converging closer and closer to the actual curve.

Figures 2.15 to 2.18 show the basis functions corresponding to the refined knot vectors and elevated order p in Figures 2.12 to 2.14, including what a $p = 1$ basis function domain with two elements looks like. In p-refinement, the total number of basis functions across the domain is $(r + 1)n - rp$ where r is the number of order elevations.

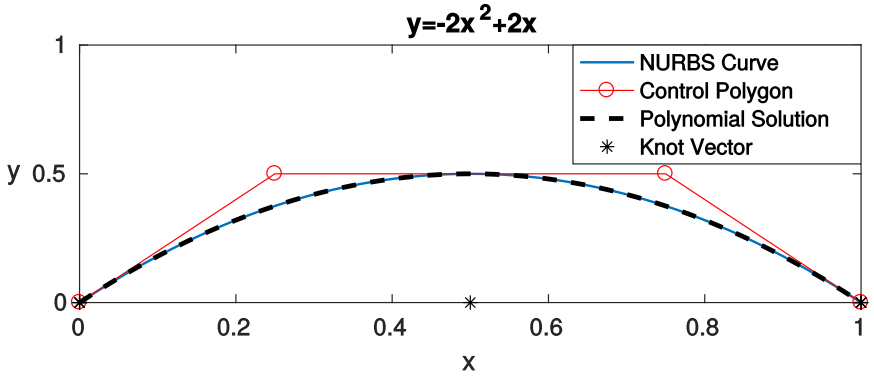


Figure 2.12 1D NURBS plot for $y = -2x^2 + 2x$ with two elements, $p = 2$.

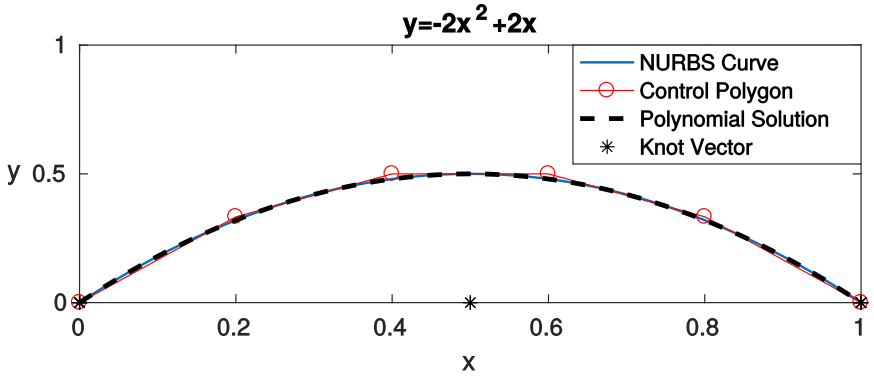


Figure 2.13 1D NURBS plot for $y = -2x^2 + 2x$, under p-refinement $p = 3$.

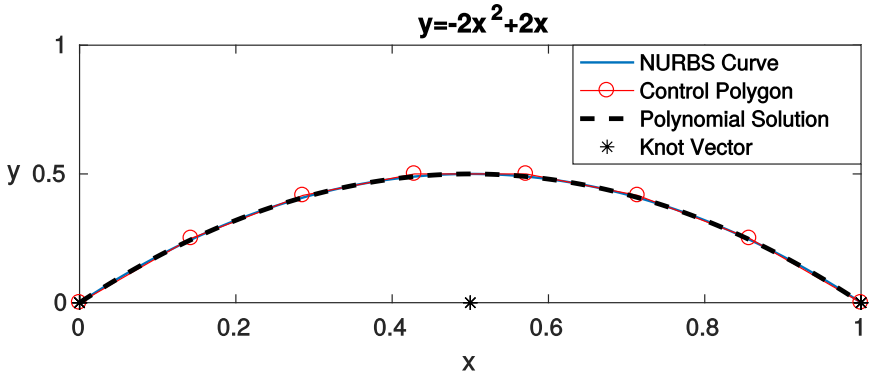


Figure 2.14 1D NURBS plot for $y = -2x^2 + 2x$, under p-refinement $p = 4$.

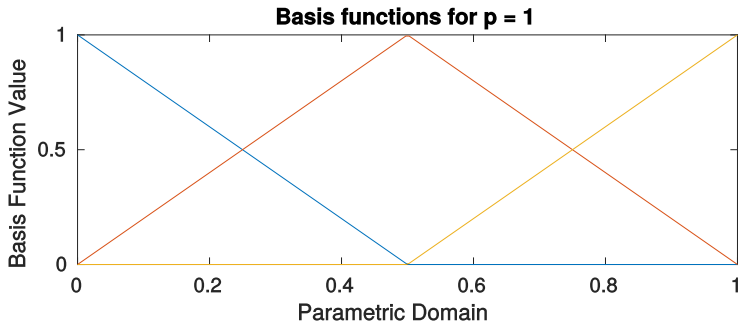


Figure 2.15 1D NURBS basis functions with two elements, $p = 1$, $\Xi = [0, 0, 0.5, 1, 1]$.

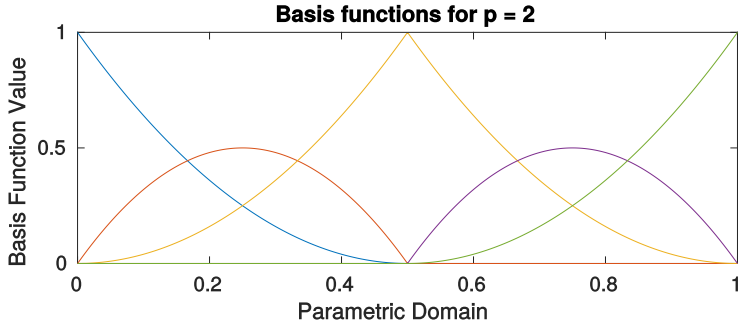


Figure 2.16 1D NURBS basis functions with two elements, $p = 2$, $\Xi = [0, 0, 0, 0.5, 0.5, 1, 1, 1]$.

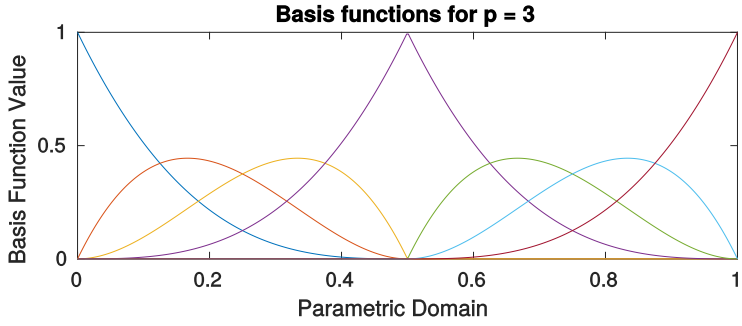


Figure 2.17 1D NURBS basis functions with two elements, $p = 3$, $\Xi = [0, 0, 0, 0, 0.5, 0.5, 0.5, 1, 1, 1, 1]$.

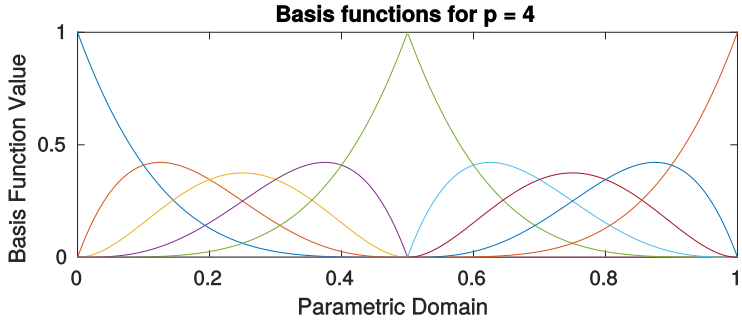


Figure 2.18 1D NURBS basis functions with two elements, $p = 4$, $\Xi = [0, 0, 0, 0, 0, 0.5, 0.5, 0.5, 0.5, 1, 1, 1, 1, 1]$.

Figure 2.16 shows the basis functions for a domain with two elements. Each basis function is defined across the entire domain, but due to the Cox De-Boor formulation go to zero for the elements where they are not relevant. The number of non-zero basis functions in any given element is therefore always equal to $p+1$. This means that each calculation for a point on a NURBS curve can call all functions, because those not needed, calculate to be zero. Efficiency could be gained by only evaluating those needed; however, this is not done in this dissertation. Note that at least one basis function goes to zero at the inner element boundary. At the ends of the domain, all go to zero, except for one. A reduction of the basis function order is detailed in [5, p.212], but was not implemented.

2.4.3 k-refinement

k-refinement is an order-elevation refinement and is often described as a mixture of p- and h-refinement [4]. p-refinement involves increasing the order from p to $\bar{p}$ to more accurately capture the solution space, while simultaneously increasing the multiplicity of the knots. This maintains the same level of continuity across the element boundaries. The number of continuous derivatives across the element borders stays the same. h-refinement involves increasing the number of elements in a domain to increase the resolution and thereby increasing the accuracy of the solution space. This also maintains the same level of continuity across the element boundary.

For 'pure' k-refinement, a one element domain is required. The element is first p-refined and then h-refined. The domain now has C^{p-1} continuity across the elements, as opposed to C^0 continuity with p-refinement. The number of continuous derivatives across the element borders increase to $\bar{p}-1$, from $p-1$ before refinement. This results in more continuous basis functions and fewer basis functions with a higher order. This leads to increased efficiency, because there are fewer basis functions to compute for each domain [1]. In p-refinement, the total number of basis functions across the domain is $(r+1)n-rp$ where r is the number of order elevations. In k-refinement, the total number of basis functions across the domain is only $n+r$. This is significantly less than for p-refinement. Additionally, this number is squared or cubed for 2D or 3D NURBS respectively.

k-refinement can also be performed on a domain that consists of more than one element, but a different refinement algorithm will have to be implemented. As opposed to 'pure' k-refinement where p- and then h-refinement are performed, the new knot and order will be defined, after which the new control points will be determined by creating a system of linear equations with a set of collocation points. The governing equation is the same as equation (2.27).

Figures 2.19 to 2.21 shows k-refinement from $p = 2$ to $p = 4$. Note that there are fewer control points than in Figures 2.12 to 2.14 for the same basis function order.

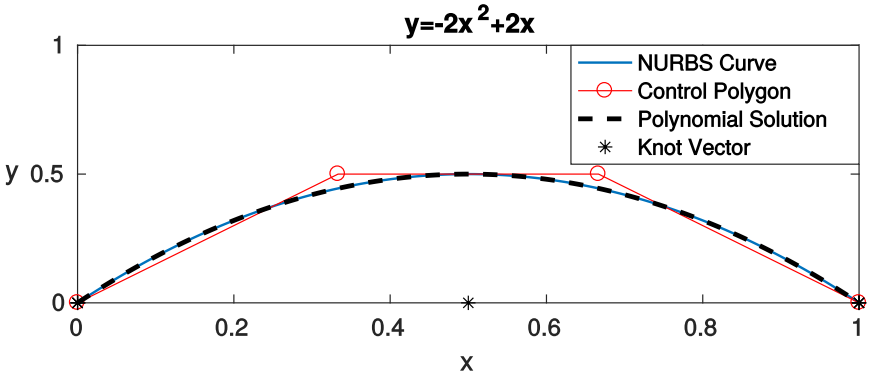


Figure 2.19 1D NURBS plot for $y = -2x^2 + 2x$ with two elements, $p = 2$.

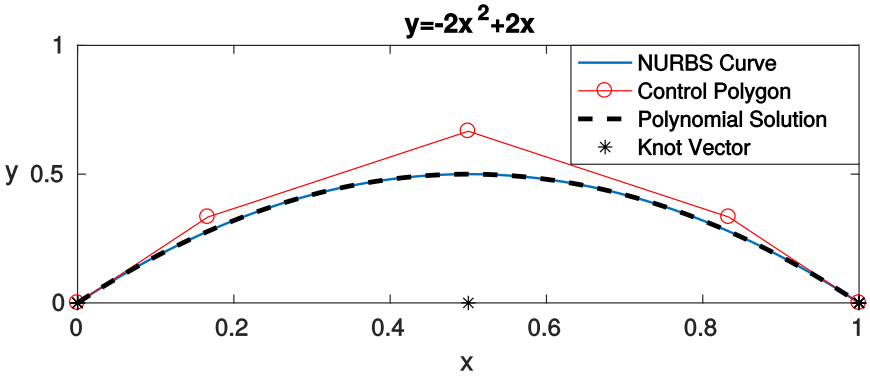


Figure 2.20 1D NURBS plot for $y = -2x^2 + 2x$, under k-refinement $p = 3$.

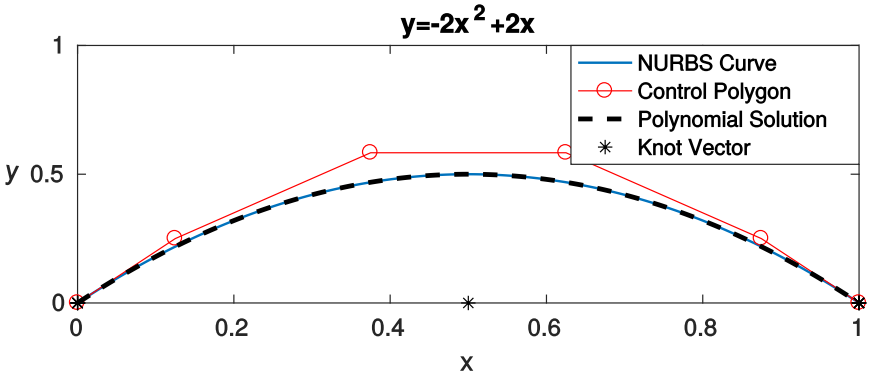


Figure 2.21 1D NURBS plot for $y = -2x^2 + 2x$, under k-refinement $p = 4$.

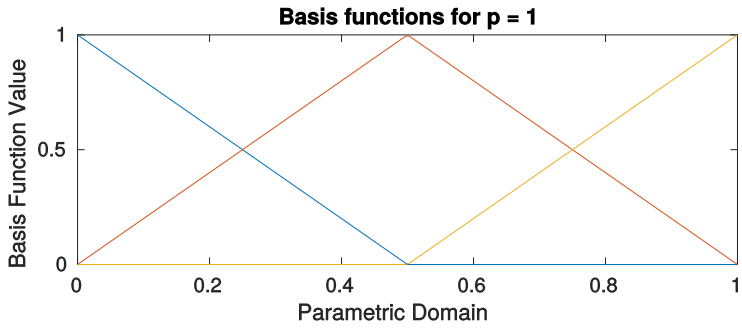


Figure 2.22 1D NURBS basis functions with two elements, $p = 1$, $\Xi = [0, 0, 0.5, 1, 1]$.

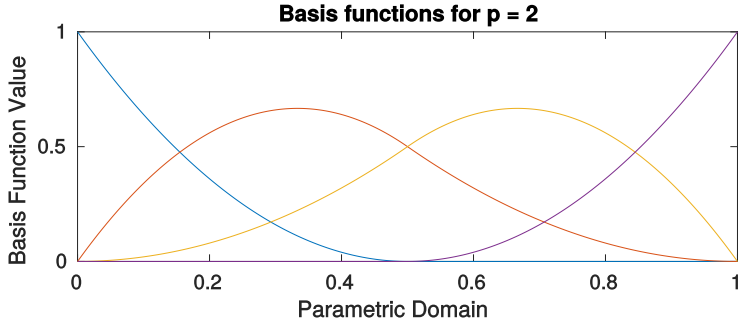


Figure 2.23 1D NURBS basis functions with two elements, $p = 2$, $\Xi = [0, 0, 0, 0.5, 1, 1, 1]$.

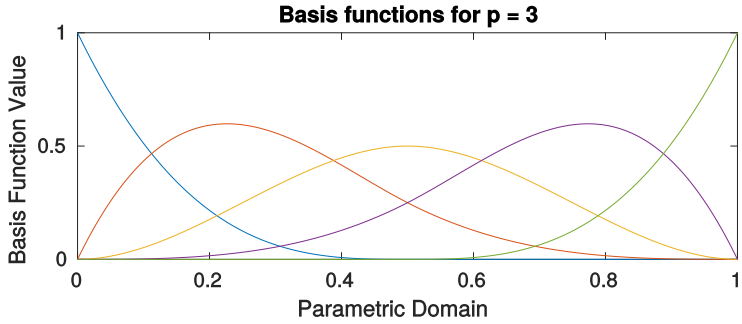


Figure 2.24 1D NURBS basis functions with two elements, $p = 3$, $\Xi = [0, 0, 0, 0, 0.5, 1, 1, 1, 1]$.

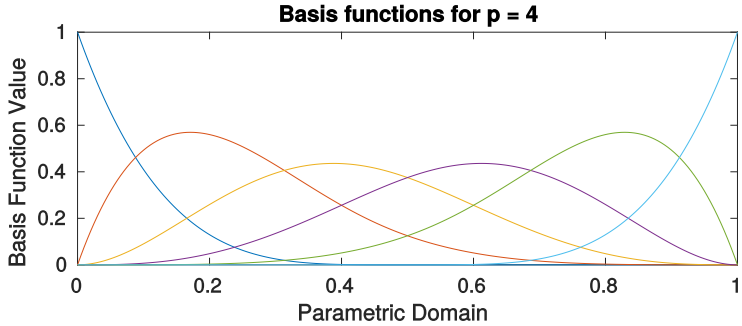


Figure 2.25 1D NURBS basis functions with two elements, $p = 4$, $\Xi = [0, 0, 0, 0, 0, 0.5, 1, 1, 1, 1, 1]$.

Figures 2.22 to 2.25 show k-refinement of a two element domain with various order elevations. The k-refined domain of Figures 2.22 to 2.25 has fewer basis functions than the equivalently p-refined domain of Figures 2.15 to 2.18.

In order to compare p- and k-refinement, consider a single element discretization with basis functions of order one, as in Figure 2.26. The domain is locally h-refined twice to give the element domain in Figure 2.27.

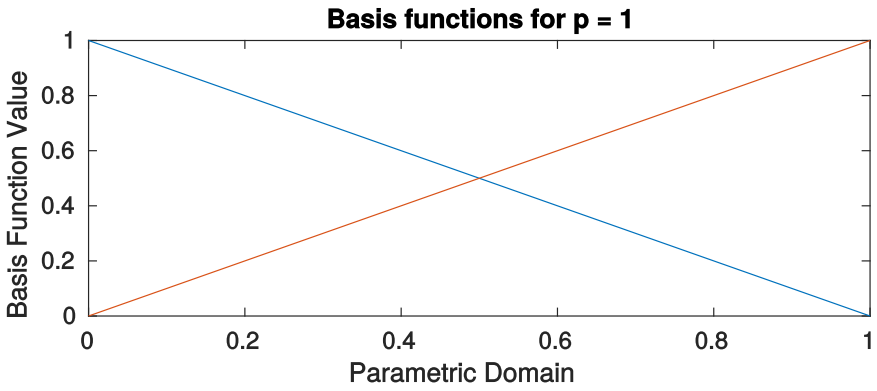


Figure 2.26 Base case, $\Xi = [0, 0, 1, 1]$, $p = 1$

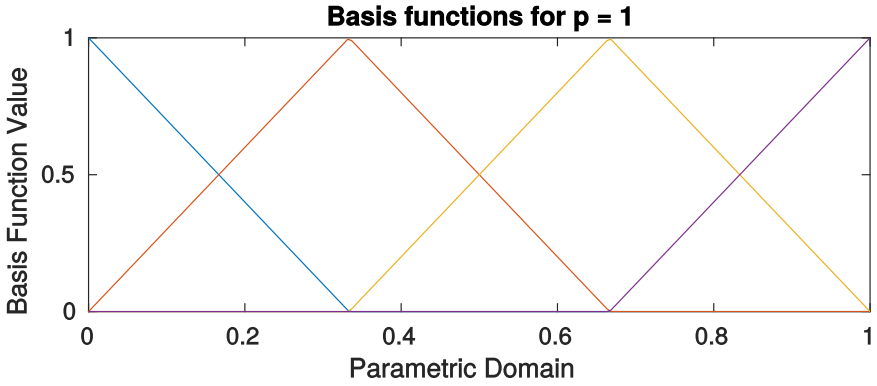


Figure 2.27 Knot Insertion, $\Xi = [0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1]$, $p = 1$

This discretization is then p-refined and k-refined, with the results of each subsequent level of refinement shown in Figure 2.28. The progression of p- and k-refinement on Figure 2.27 is shown in the left and right columns respectively. The multiplicity of the internal knot values, as well as the end knot values increase in multiplicity for p-refinement, while only the multiplicity of the end knot values increase for k-refinement. Note that p-refinement creates elements that have C^0 continuity while k-refinement creates elements that are C^{p-1} continuous.

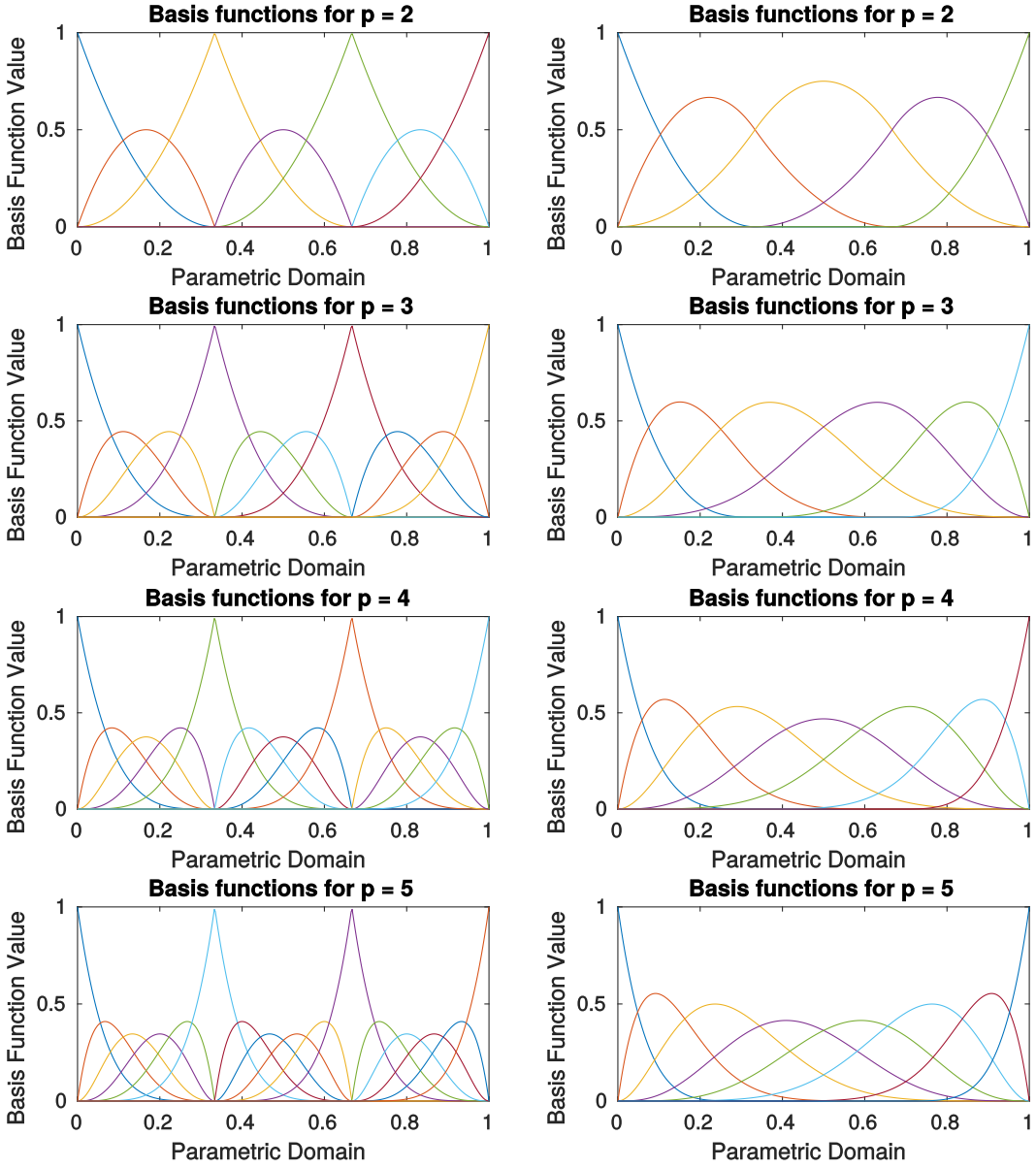


Figure 2.28 p-refinement on the left and k-refinement on the right for a three element domain

2.4.4 Refinement in Two Dimensions

Refinement in a two-dimensional parametric space uses the same algorithms as in the one-dimensional space. With h-refinement, refinement in each direction happens consecutively, while refinement in each direction for p- and k-refinement can occur simultaneously. There is therefore a h-refinement code for each direction and one of each type of order refinement.

Starting from the unrefined mesh in Figure 2.29, the mesh is then refined first in the η-direction (or ξ-direction) and then refined in the other direction. Figure 2.30 shows refinement in the η-direction. Figure 2.31 then shows the mesh in Figure 2.30 having refinement applied in the ξ-direction as well. In this work, global h-refinement is performed in both directions. For p- and k-refinement, it was chosen so that different orders of refinement may be applied in different directions.

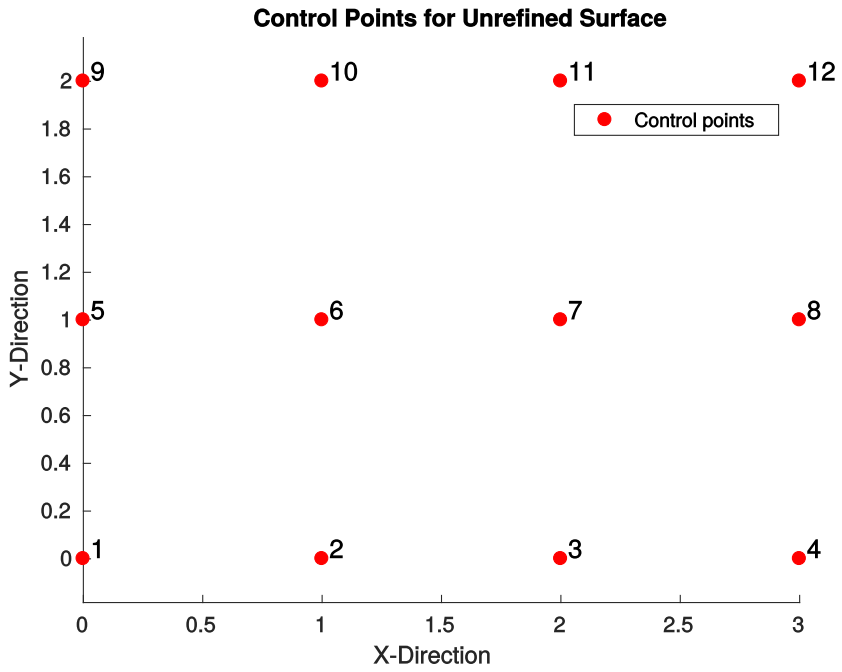


Figure 2.29 Control points of an unrefined two-dimensional NURBS Surface.

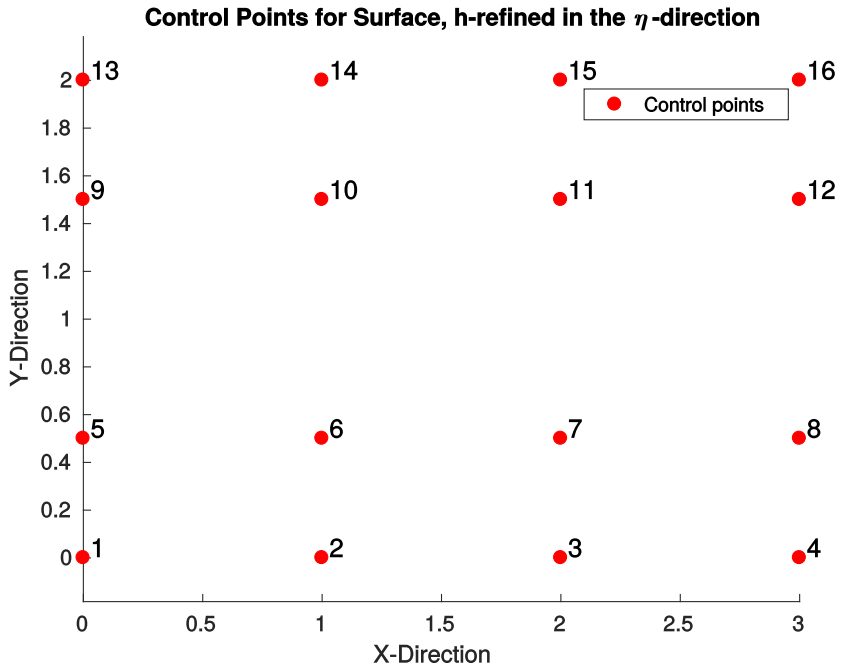


Figure 2.30 Control points of a two-dimensional surface, h-refined in the η-direction only.

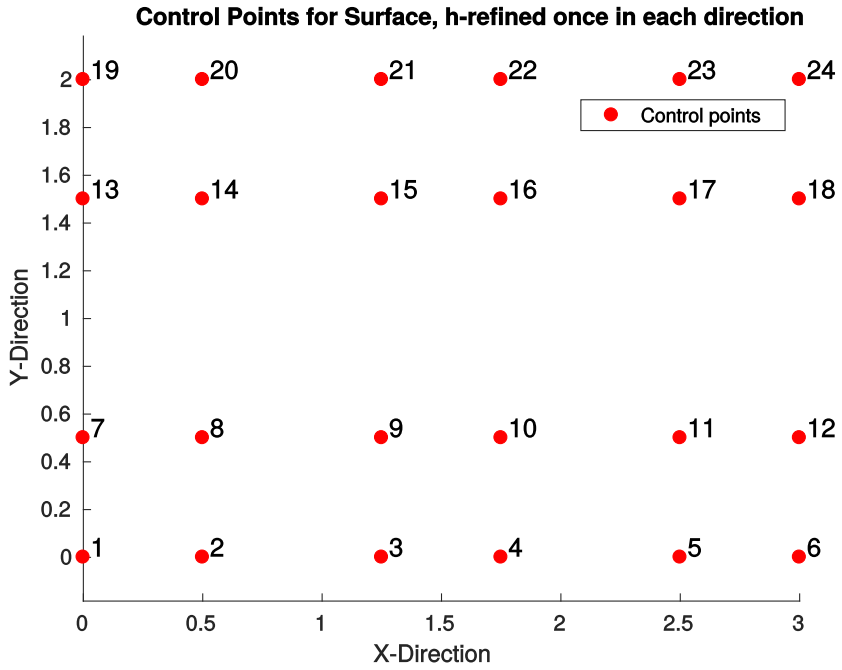


Figure 2.31 Control points of a two-dimensional surface, h-refined once in each direction.

2.5 NURBS Implementation and Examples

2.5.1 One Dimensional NURBS

A one-dimensional NURBS curve can be described by equation (2.12); that is,

$$\boldsymbol{C}(\xi) = \sum_{i=1}^{n} N_i^p(\xi)\boldsymbol{P}_i$$

and can exist in a one, two or three-dimensional domain. It uses univariate shape functions as described in equation (2.13) and NURBS control points, $\boldsymbol{P}$ and weights, $\boldsymbol{w}$. A NURBS curve in two-dimensional physical domain is used as an example to demonstrate the one-dimensional NURBS implementation.

2.5.1.1 One Dimensional Implementation

Arbitrary points and associated weights in a two-dimensional domain were chosen to illustrate the construction of a NURBS curve. The order of the basis functions was taken to be $p = 2$ and a knot vector of $\Xi = [0, 0, 0, 0.25, 0.5, 0.7, 1, 1, 1]$ was chosen. This means that the number of basis functions across the domain is equal to $n = length(\Xi) - p - 1 = 6$. Therefore, six control points in the two-dimensional domain are required. The control points and associated weights were chosen to be

$$\boldsymbol{P} = [(0,0),\ (1,3),\ (2,3),\ (3,1),\ (3.5,3),\ (5,4)]^T \tag{2.28}$$

$$\boldsymbol{w} = [1,\ 0.3,\ 0.7,\ 0.5,\ 0.5,\ 1]^T \quad . \tag{2.29}$$

Finally, the evenly spaced parametric vector which will be the points mapped from the parametric to the physical domain is created. These points are what make up the points on the NURBS curve and is defined to be ξ_{vec} which is of length $ppoints$. The NURBS curve assembly procedure is shown in Algorithm 1 below.

Algorithm 1 Procedure for constructing a NURBS curve

Initialize NURBS vector `c_vec` made of 1D, 2D or 3D points to length *ppoints*
Initialize *count*
for $i_{points} = 1 : ppoints$ **do**
    Initialize c and W
    for i=1:n **do**
        Assemble $W = W + N_i^p(\xi(i_{points}))w_i$
    end for
    for i=1:n **do**
        Assemble $c = c + N_i^p(\xi(i_{points}))w_i\boldsymbol{P}_i$
    end for
    Increment *count*
    Assemble $c_{vec}(count, :) = c/W$
end for

Note that $\boldsymbol{P}_i$ can be a two- or three-dimensional point or a single entry.

See Figure 2.32 for the generated NURBS curve.

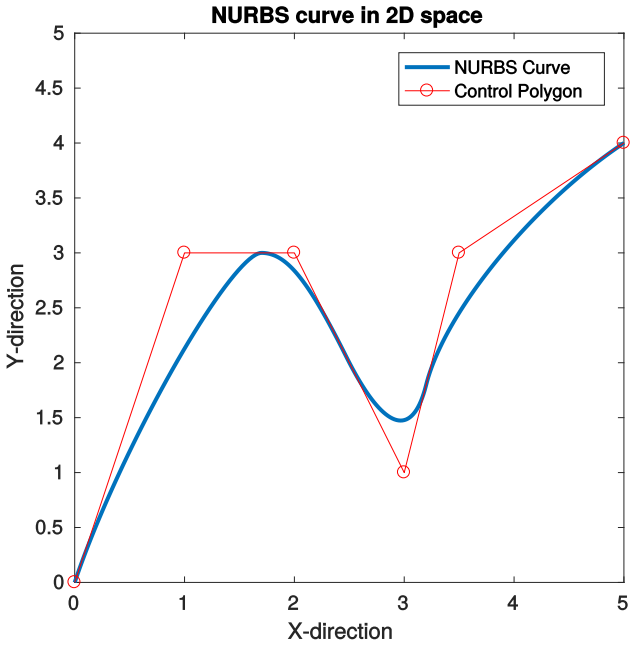


Figure 2.32 Arbitrary two-dimensional NURBS curve.

2.5.1.2 Circle

One of the biggest advantages of using NURBS is that it can draw conic sections, such as circles, exactly and therefore it is appropriate to discuss how to construct the control points of a circle. There is more than one way to construct a circle. For example, there is a way to construct a circle using negative weights [1, p. 57], but the following method is simple and easy to follow and has been adapted from [1, p. 57] by using arcs.

Arcs of angle less than 180°can be constructed with only one element and an order of $p \geq 2$. The knot vector if $p = 2$ will be $\Xi = [0, 0, 0, 1, 1, 1]$ which make the knot ends interpolatory and therefore control points $\boldsymbol{P}_1$ and $\boldsymbol{P}_3$ are the arc start and end points with each of the associated weights, w_1 and w_3 equal to one. The second control point, $\boldsymbol{P}_2$ is calculated by taking the intersection point of the two lines that are tangent to the other two points. The weight of that point can be calculated as the cosine of half the arc angle, $w_2 = cos(\theta/2)$. See Figure 2.33 below.

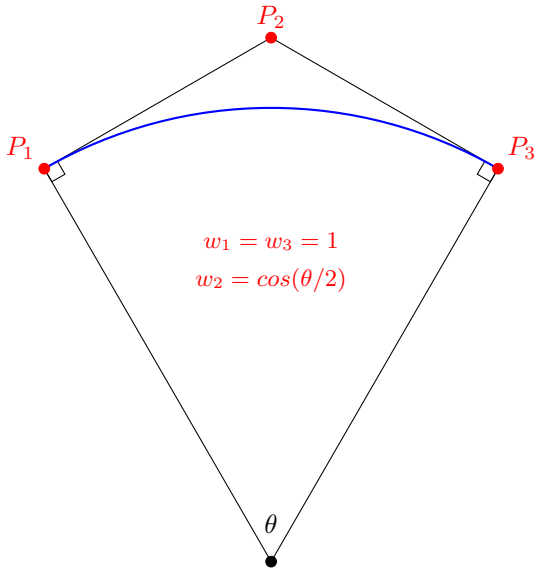


Figure 2.33 Diagram illustration how to build a circular arc, adapted from Figure 2.34 in [1].

A quarter circle with unit radius can be created with control points $\boldsymbol{P} = [(0, 1),\ (1, 1),\ (1, 0)]^T$ with the associated weights $\boldsymbol{w} = [1,\ \frac{1}{\sqrt{2}},\ 1]$. Arcs with an inner angle of more than 180°can be constructed using multiple arcs. To link the arcs together, the knot values associated with each element boundary needs a multiplicity of p to allow the control point to represent the curve exactly at that point. A circle can be constructed out of multiple arcs. To

construct a full circle, four quarter circle arcs are used and are joined by making the first and last control points coincident. The control points and associated weights for a circle would be

$$\boldsymbol{P} = [(1,0),\ (1,1),\ (0,1),\ (-1,1),\ (-1,0),\ (-1,-1),\ (0,-1),\ (1,-1),\ (1,0)]^T \tag{2.30}$$

$$\boldsymbol{w} = \left[1,\ \frac{1}{\sqrt{2}},\ 1,\ \frac{1}{\sqrt{2}},\ 1,\ \frac{1}{\sqrt{2}},\ 1,\ \frac{1}{\sqrt{2}},\ 1\right]^T \quad . \tag{2.31}$$

The knot vector for $p = 2$ order basis functions would be $\Xi = [0,0,0,1,1,2,2,3,3,4,4,4]$ with double knot values on each element boundary. Figure 2.34 shows the NURBS circle and associated control points.

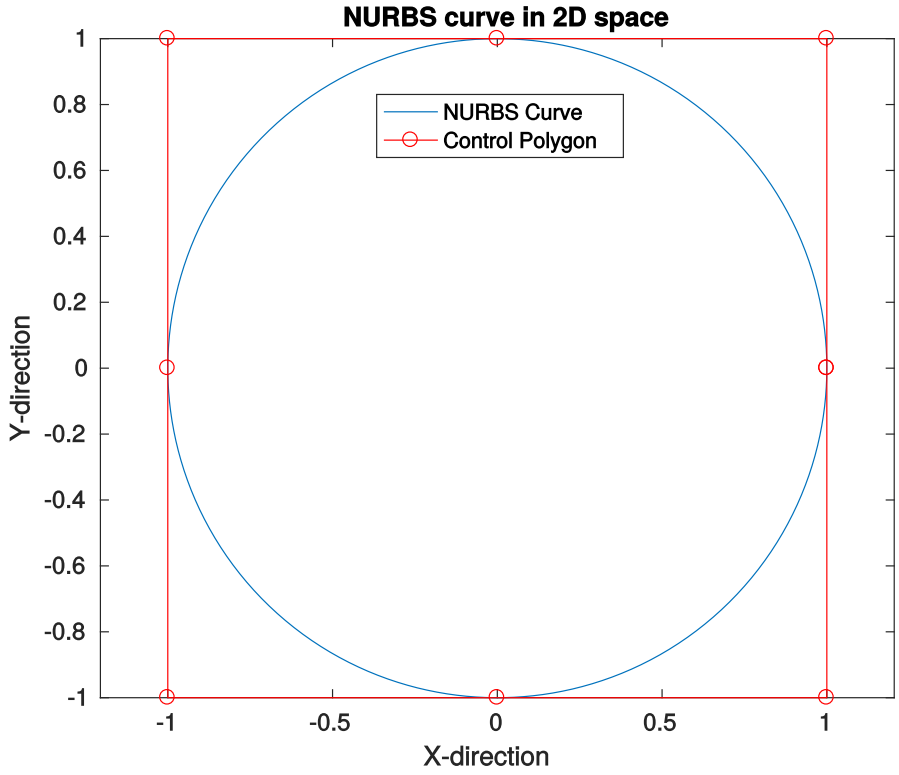


Figure 2.34 Two-dimensional NURBS circle

2.5.1.3 Helix

A helix is a curve in a three-dimensional domain. The x and y control point coordinates can be constructed in a similar way as for a circle space, adding additional rotations as necessary. The z control point coordinates can be varied, as the helix increases in height. For a circle by a constant offset, running from a height of zero to one, the control points are

$$\begin{aligned}\boldsymbol{P} = [&(1,0,0),\ (1,1,0.125),\ (0,1,0.25),\ (-1,1,0.375),\ (-1,0,0.5),\\ &(-1,-1,0.625),\ (0,-1,0.75),\ (1,-1,0.875),\ (1,0,1)]^T\end{aligned} \tag{2.32}$$

$$\boldsymbol{w} = [1,\ \frac{1}{\sqrt{2}},\ 1,\ \frac{1}{\sqrt{2}},\ 1,\ \frac{1}{\sqrt{2}},\ 1,\ \frac{1}{\sqrt{2}},\ 1]^T \quad . \tag{2.33}$$

The helix with control points shown above, can be described as seen in Figure 2.35.

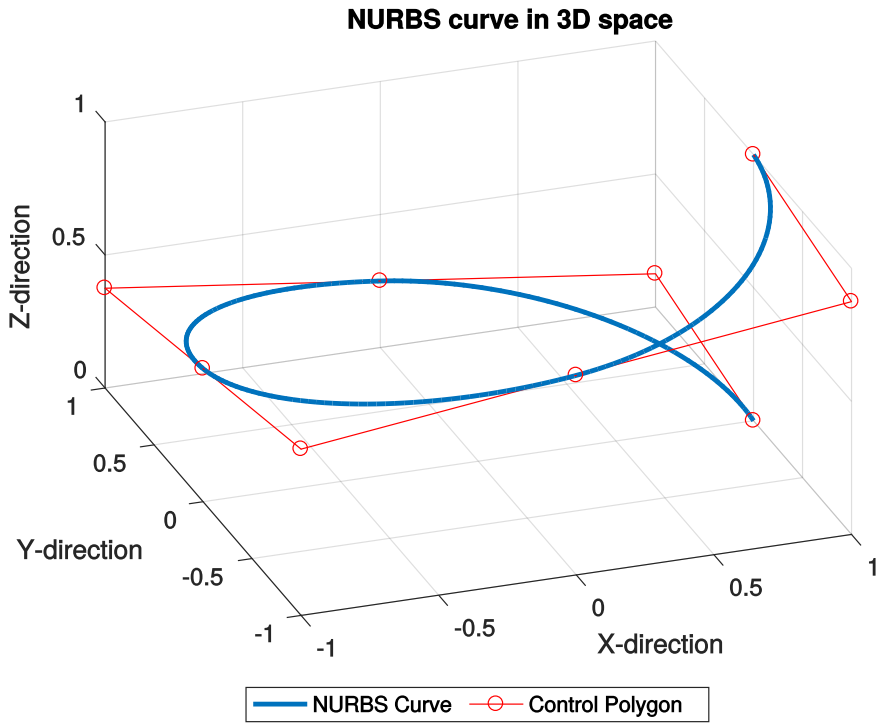


Figure 2.35 Three-dimensional NURBS helix

2.5.1.4 Polynomial

An important outcome of this work is creating a B-spline curve from a given polynomial. This technique makes it possible to construct more curves exactly and is used to implement inhomogeneous boundary conditions. The relevant control points are found using known solution points from the polynomial and solving a set of simultaneous equations. Using the mapping equation (2.3) as a function of random collocation points, ξ', to calculate the basis function matrix, $\boldsymbol{N}(\xi')$ and the known points on the polynomial $\boldsymbol{C}(\xi')$, the control points, $\boldsymbol{P}$ can be computed by

$$\boldsymbol{N}(\xi')_{IJ}\boldsymbol{P}_J = \boldsymbol{C}(\xi')_I \tag{2.34}$$

where

$$\boldsymbol{N}(\xi')_{IJ} = N_J^p(\xi'_I) = \begin{bmatrix} N_1^p(\xi'_1) & N_2^p(\xi'_1) & \dots & N_n^p(\xi'_1) \\ N_1^p(\xi'_2) & N_2^p(\xi'_2) & \dots & N_n^p(\xi'_2) \\ \cdot & \cdot & & \cdot \\ \cdot & & \cdot & \cdot \\ \cdot & & & \cdot \\ N_1^p(\xi'_n) & N_2^p(\xi'_n) & \dots & N_n^p(\xi'_n) \end{bmatrix} \quad I, J = 1 \text{ to } n. \tag{2.35}$$

$\boldsymbol{C}(\xi')$ are the curve values at each point in ξ', while ξ'_i are the components of ξ'. In this dissertation, ξ' was chosen to be

$$\xi' = \frac{\Xi_{p+1\,:\,end-p+1} + \Xi_{p\,:\,end-p}}{2} \quad , \tag{2.36}$$

so that the number of sample points equal the number of control points required $(length(\Xi) - p - 1)$. This method only works for B-splines and therefore if a NURBS code is used, the weights should all be the same value, for example,

$$w_J = 1 \qquad \text{for } J = 1, 2, \dots, n. \tag{2.37}$$

2.5.2 Two Dimensional NURBS surfaces

A NURBS surface is created with equation (2.18) using two parameters. The process is very similar to a NURBS curve (one-dimensional NURBS), with the difference being that the shape function at each associated control point is a bivariate shape function and is the product of the two univariate shape functions in each direction. A NURBS surface can be described in the two- or three-dimensional physical domain. The creation of NURBS surfaces is demonstrated with two constructions. The first considers the construction of

one quarter of an annulus while the other considers the construction of a quarter of a plate with a centrally located hole. This example contains important features of NURBS surfaces for future sections.

2.5.2.1 Implementation

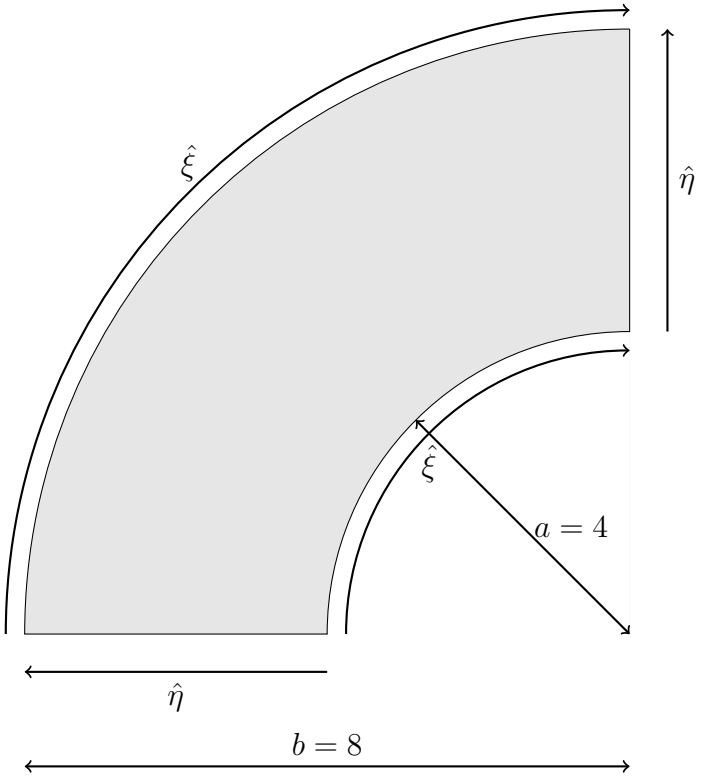


Figure 2.36 Annular plate

A quarter of an annulus with an outer radius of 8 and an inner radius of 4. The ξ-direction of the parametric domain is chosen to be along the circumference, while the η-direction is radially outward. The control points associated with the circumferential direction is constructed as per Section 2.5.1.2. Because the geometry in the circumferential direction is circular, the minimum order of the basis functions required is two, $p = 2$. Therefore three control points are required to draw the arcs as only one element is required in the ξ-direction. Control points of the inner edge are $(-4, 0)$, $(-4, 4)$ and $(0, 4)$ with associated weights of 1, $\frac{1}{\sqrt{2}}$ and 1 respectively. The outer radius can be defined using control points $(-8, 0)$, $(-8, 8)$ and $(0, 8)$ with associated weights of 1, $\frac{1}{\sqrt{2}}$ and 1 respectively. The geometry in the radial direction has very few features and runs straight outwards. This means that the control points in the radial direction can be evenly spaced from the inner to the outer

boundary. Because there are no features in the radial direction, the minimum order that is required in the radial direction is only one, $p = 1$ and because again, only one element is required to describe the geometry accurately, only two control points are required which are already included by defining the circumferential control points. The control point and associated NURBS weights arrays are therefore defined as

$$\boldsymbol{P} = \begin{bmatrix} (-4,0) & (-8,0) \\ (-4,4) & (-8,8) \\ (0,4) & (0,8) \end{bmatrix} \tag{2.38}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 & 1 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & 1 \end{bmatrix} \tag{2.39}$$

The number of control points in the ξ-direction is $n = 3$ and in the η-direction is $m = 2$. Because only one element in each direction is required to represent this geometry exactly, the knot vectors in each respective direction are $\Xi = [0, 0, 0, 1, 1, 1]$ and $\mathscr{H} = [0, 0, 1, 1]$. Finally, two evenly spaced parametric vectors in each direction are used to describe the NURBS surface namely ξ_{vec} and η_{vec} which are each of length $ppoints_N$ and $ppoints_M$ respectively. The following algorithm is used to describe the NURBS surface.

The NURBS surface and respective control points can be seen in Figure 2.37.

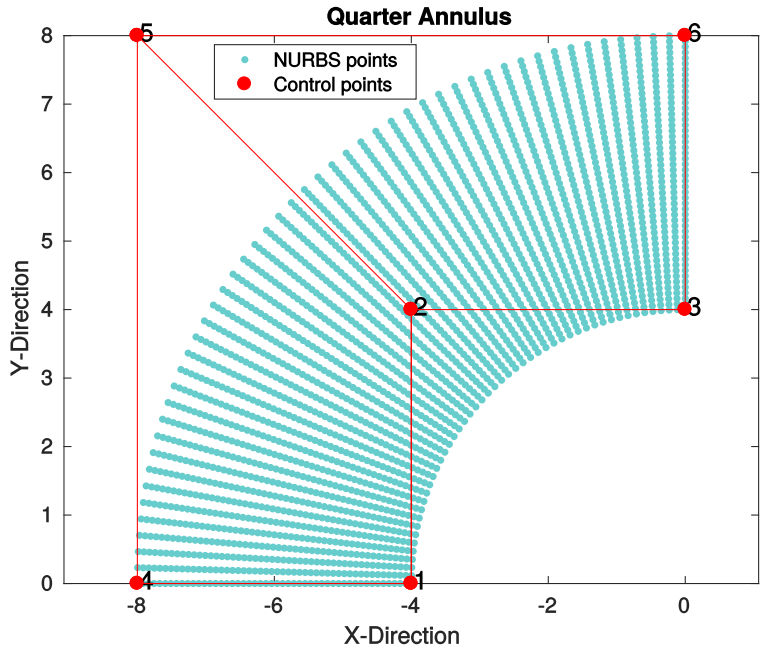


Figure 2.37 Quarter Annulus

Algorithm 2 Procedure for constructing a NURBS surface

Initialize NURBS vector `c_vec` of 2D points to length $ppoints_N \times ppoints_M$
Initialize *count*
for $j_{points} = 1 : ppoints_M$ **do**
    for $i_{points} = 1 : ppoints_N$ **do**
        Initialize c and W
        for j=1:m **do**
            for i=1:n **do**
                Assemble $W = W + N_i^p(\xi(i_{points}))N_j^q(\eta(j_{points}))w_{ij}$
            end for
        end for
        for j=1:m **do**
            for i=1:n **do**
                Assemble $c = c + N_i^p(\xi(i_{points}))N_j^q(\eta(j_{points}))w_{ij}\boldsymbol{P}_{ij}$
            end for
        end for
        Increment *count*
        Assemble $c_{vec}(count, :) = c/W$
    end for
end for

Note that $\boldsymbol{P}_{ij}$ can be a two- or three-dimensional point and is not just a single entry. Hint: it can be easier to convert $\boldsymbol{P}$ into a vector of points to make it easier to handle.

2.5.2.2 Quarter plate with a centrally located hole

The quarter plate with a centrally located hole is a unexpectedly complex geometry to construct as the two directions in the parametric domain $(\hat{\xi}, \hat{\eta})$ do not correspond directly to the directions of the physical domain $(\vec{x}, \vec{y})$. Each edge of the mesh in the parametric domain does not necessarily correspond to a single edge in the physical domain. Figure 2.38 shows the NURBS shape in the physical domain, with dimensions $a = 1$ and $b = 4$ as well as the corresponding parametric directions as projected into the physical domain.

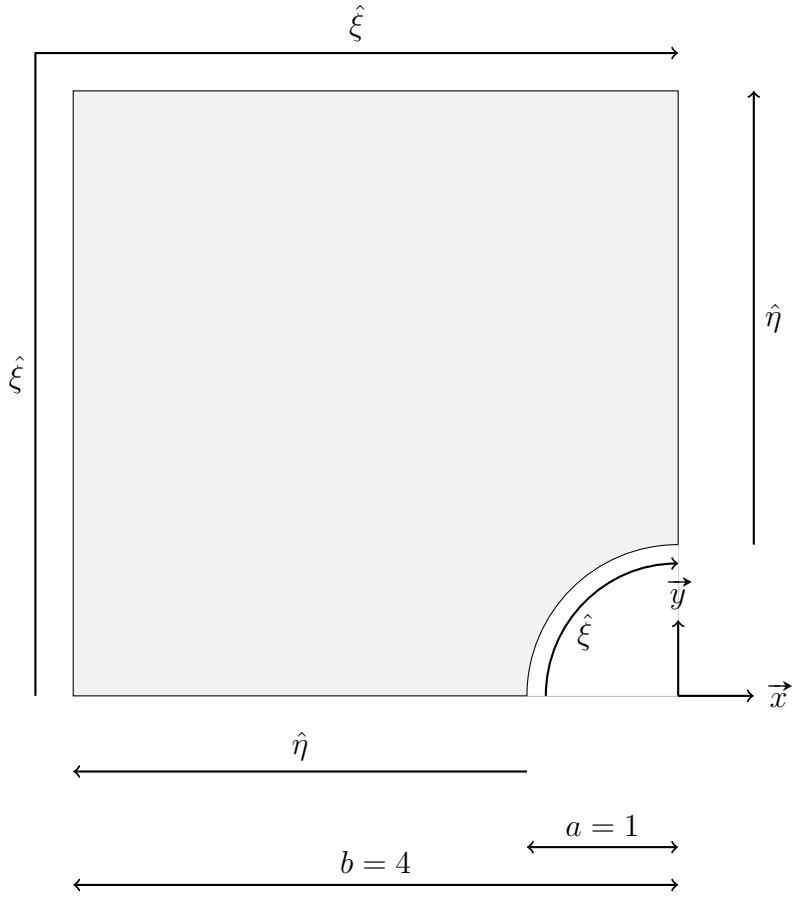


Figure 2.38 Quarter plate with hole NURBS surface

The control points domain can be described as

$$\boldsymbol{P} = \begin{bmatrix} (-1,0) & (-2.5,0) & (-4,0) \\ (-1,\sqrt{2}-1) & (-2.5,0.75) & (-4,4) \\ (1-\sqrt{2},1) & (-0.75,2.5) & (-4,4) \\ (0,1) & (0,2.5) & (0,4) \end{bmatrix} \tag{2.40}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 & 1 & 1 \\ (1+1/\sqrt{2})/2 & 1 & 1 \\ (1+1/\sqrt{2})/2 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.41}$$

where the first column represents the arc of the hole, the final column the left and top edges with a corner in the middle and the middle column the area between the arc and the corner. Alternatively, the first row represents the bottom edge in the physical domain

and the last row represents the right edge of the domain. Control points $\boldsymbol{P}_{23}$ and $\boldsymbol{P}_{33}$ are coincident. These control points define the top-left corner of the domain. This is to capture the right-angled corner of the domain, but this double control point feature cannot be captured accurately in k-refinement. Both h- and p-refinement are able to retain this feature. Points on the NURBS surface are created using Algorithm 2 and is plotted below in Figure 2.39 along with its control points.

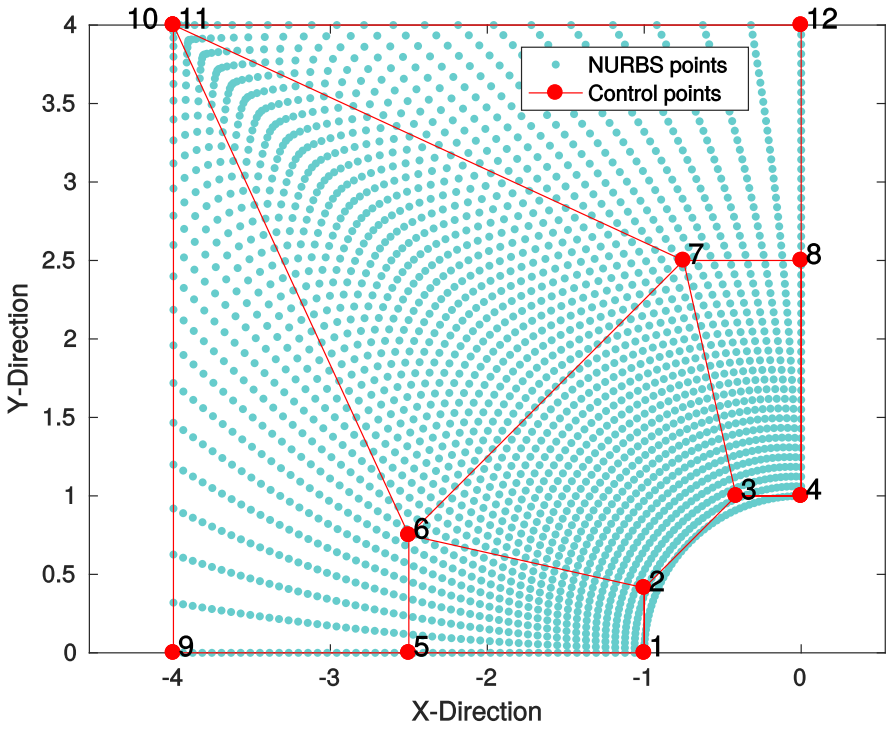


Figure 2.39 Plate with hole NURBS surface with control polygon

Figure 2.40 shows the result of k-refinement in order to see the effect of order refinement on a domain with a repeated control point. The mesh below is order refined to have $p = 3$ and $q = 3$. This leads to the conclusion that k-refinement cannot be used in all cases and does not work for the case of coincident points.

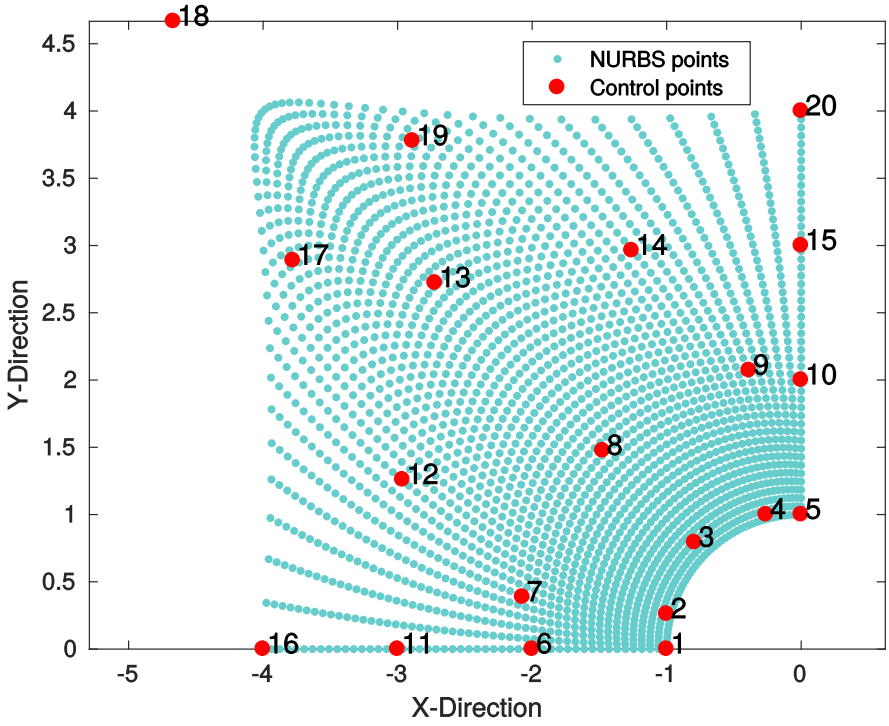


Figure 2.40 Plate with centrally located hole, after k-refinement to $p, q = 3$.

2.5.2.3 Surface in a Three-dimensional domain

The final example of NURBS in this project is a surface in a three-dimensional physical domain. Figure 2.41 shows the mapping of equispaced parameter values onto the physical domain. The control points are also shown and labeled. The knot vectors and control points for a toroidal surface, adapted from [3, p. 4187] are

$$\Xi = [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4] \tag{2.42}$$

$$\mathscr{H} = [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4] \tag{2.43}$$

with $p = 2$ and $q = 2$ and

$$\boldsymbol{P} =$$
$$\begin{bmatrix}
(5,0,-1) & (6,0,-1) & (6,0,0) & (6,0,1) & (5,0,1) & (4,0,1) & (4,0,0) & (4,0,-1) & (5,0,-1) \\
(5,5,-1) & (6,6,-1) & (6,6,0) & (6,6,1) & (5,5,1) & (4,4,1) & (4,4,0) & (4,4,-1) & (5,5,-1) \\
(0,5,-1) & (0,6,-1) & (0,6,0) & (0,6,1) & (0,5,1) & (0,4,1) & (0,4,0) & (0,4,-1) & (0,5,-1) \\
(-5,5,-1) & (-6,6,-1) & (-6,6,0) & (-6,6,1) & (-5,5,1) & (-4,4,1) & (-4,4,0) & (-4,4,-1) & (-5,5,-1) \\
(-5,0,-1) & (-6,0,-1) & (-6,0,0) & (-6,0,1) & (-5,0,1) & (-4,0,1) & (-4,0,0) & (-4,0,-1) & (-5,0,-1) \\
(-5,-5,-1) & (-6,-6,-1) & (-6,-6,0) & (-6,-6,1) & (-5,-5,1) & (-4,-4,1) & (-4,-4,0) & (-4,-4,-1) & (-5,-5,-1) \\
(0,-5,-1) & (0,-6,-1) & (0,-6,0) & (0,-6,1) & (0,-5,1) & (0,-4,1) & (0,-4,0) & (0,-4,-1) & (0,-5,-1) \\
(5,-5,-1) & (6,-6,-1) & (6,-6,0) & (6,-6,1) & (5,-5,1) & (4,-4,1) & (4,-4,0) & (4,-4,-1) & (5,-5,-1) \\
(5,0,-1) & (6,0,-1) & (6,0,0) & (6,0,1) & (5,0,1) & (4,0,1) & (4,0,0) & (4,0,-1) & (5,0,-1)
\end{bmatrix} \tag{2.44}$$

$$\boldsymbol{w} =$$
$$\begin{bmatrix}
1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 \\
1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} \\
1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 \\
1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} \\
1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 \\
1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} \\
1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 \\
1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} & 1/2 & 1/\sqrt{2} \\
1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1 & 1/\sqrt{2} & 1
\end{bmatrix} \tag{2.45}$$

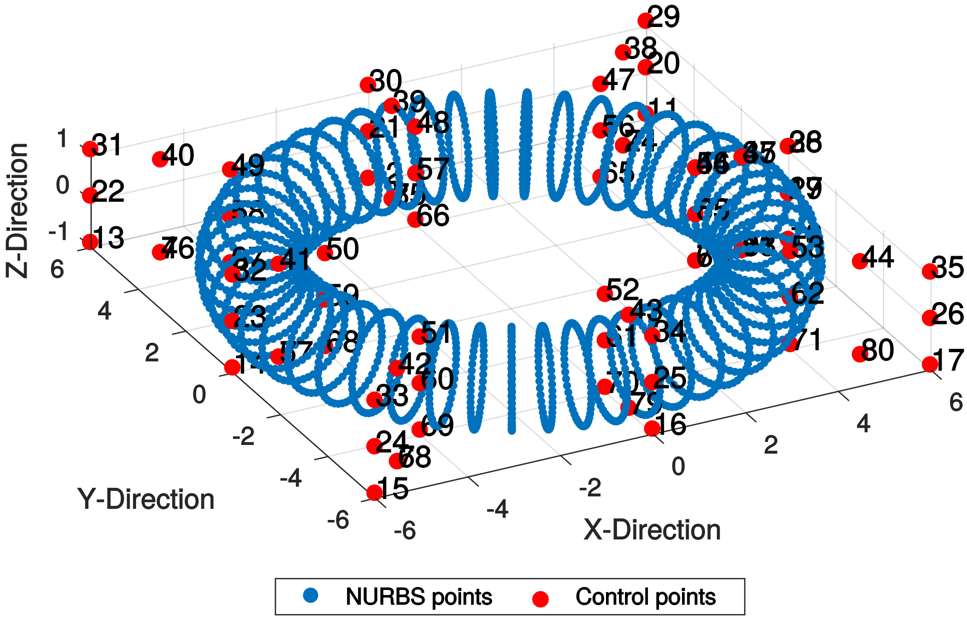


Figure 2.41 Toroidal surface

Chapter 3

One-dimensional Isogeometric Analysis

Isogeometric analysis (IGA) is a finite element analysis technique that uses the same basis functions for the unknown variables as those used to describe the geometry. The idea of using the same basis functions for both the geometry and the analysis, called the isoparametric concept is not a new concept in finite element analysis, but the key of IGA is that it can represent geometries exactly, as well as approximate the solution field [1, 4]. Traditional finite element methods (FEM) also commonly use the isoparametric concept for both, however, the meshing procedure renders complex geometries in a faceted manner and not always exactly. The NURBS geometry is the mesh and therefore there is no need to do any additional meshing of the domain. The benefits of IGA can be seen in modeling fillet and hole radii. With IGA, these features are always modeled exactly and therefore the exact stress concentration will be reached with convergence. With traditional FEM, it is difficult to capture sharp fillets and hole radii and it results in either a mesh that needs to be so highly refined that it is difficult to work with or the fillet or hole radii are removed which results in inaccurate stress distributions. IGA allows for these features to be captured [1, p. 53]. A one-dimensional IGA problem will be approximated in order to demonstrate the IGA procedure in a step by step manner.

As in Section 2.3, equation (2.12) maps the parametric domain to the physical domain,

$$\boldsymbol{x}(\xi) = \sum_{i=1}^{n} R_i^p(\xi)\boldsymbol{P}_i \quad .$$

The field distribution is then approximated using a NURBS curve, $u(x)$, that shares the knot vector and basis functions with the geometry NURBS curve:

$$u(\boldsymbol{x}) = \sum_{i=1}^{n} R_i^p(\xi)\boldsymbol{d}_i \quad , \tag{3.1}$$

where $u(\boldsymbol{x})$ is the solution field as a function of the physical position, $R_i^p(\xi)$ are the shape functions, $\boldsymbol{d}$ is the set of control points for the solution field, compared to $\boldsymbol{P}$ being the set of control points for the geometry field. $\boldsymbol{x}$, $\boldsymbol{P}$ and $\boldsymbol{d}$ are vectors of individual values and are written in bold. Although the curve is a one-dimensional NURBS curve, the curve could be in one-, two- or three-dimensional physical space and one point can consist of more than one coordinate. A single point of $\boldsymbol{x}_i$, $\boldsymbol{P}_i$ or $\boldsymbol{d}_i$, is therefore always written in bold.

The approach to obtaining the field solution control points, $\boldsymbol{d}$, from the differential equation, using the IGA technique, is detailed in Sections 3.1.1 to 3.1.4, after which the implementation of the procedure will be described in Section 3.1.5.

3.1 One-dimensional IGA Solution Procedure

A one-dimensional Poisson's problem will be approximated in order to demonstrate the IGA procedure. This is the same problem treated in [4, p. 18]. The problem is defined using a strong form equation with appropriate boundary conditions. The governing equations are then converted into the weak form, after which the functions are approximated with shape functions. The combination of the weak form and the approximation functions allows the problem to be represented as a set of discrete linear equations, that can then be solved numerically [7].

3.1.1 Strong Form

The one-dimensional Poisson equation,

$$\frac{d^2u(x)}{dx^2} + b(x) = 0 \ , \tag{3.2}$$

which in this case will be defined over the domain $x \in [0, 1]$, where for convenience the source term is chosen as

$$b(x) = x \quad . \tag{3.3}$$

The boundary conditions for this problem are both homogeneous Dirichlet boundary (Γ_D) conditions; that is,

$$u(0) = 0 \tag{3.4}$$
$$u(1) = 0 \tag{3.5}$$

and therefore there are no Neumann boundary conditions.

The exact solution

$$u(x) = -\frac{1}{6}x^3 + \frac{1}{6}x \quad . \tag{3.6}$$

3.1.2 Weak From

The strong form is now converted into the weak form. Each component of the differential equation is multiplied by a weighting function and integrated across the relevant domain. The weighting function, $\nu(x)$, is an arbitrary function with the property that $\nu = 0$ for all Dirichlet boundary conditions. The weighting function is arbitrary to ensure that the strong form is equivalent to the weak form [7, p. 48]. $u(x)$ is referred to as the trial solution, which is the set of admissible functions that describe the solution. In this case, the physical domain is one-dimensional and therefore x is written in standard weight script. The same holds for the solution field control points. $\boldsymbol{d}_i$ will be written as d_i, while $\boldsymbol{d}$ still represents the full vector of control points in the domain. The weak form of the equation (3.2) is

$$\int_0^1 \nu(x)\left[\frac{d^2u(x)}{dx^2} + b(x)\right]dx = 0 \qquad \forall\nu. \tag{3.7}$$

For convenience, we write the weak form as

$$\int_0^1 \nu(x)\frac{d}{dx}\left(\frac{du(x)}{dx}\right)dx + \int_0^1 \nu(x)b(x)dx = 0 \qquad \forall\nu. \tag{3.8}$$

and using integration by parts, the following is obtained:

$$-\int_0^1 \frac{d\nu(x)}{dx}\frac{du(x)}{dx}dx + \nu(x)\frac{du}{dx}\bigg|_{x=0}^{x=1} = -\int_0^1 \nu(x)b(x)dx \qquad \forall\nu. \tag{3.9}$$

In order for the integrals in equation (3.9) to make sense it suffices to require that the functions $u(x)$ and $\nu(x)$, as well as their first derivatives, are square integrable. Furthermore, the solution and the arbitrary test functions are assumed to satisfy the homogeneous boundary conditions. The space of functions satisfying these conditions is denoted by V_0: that is,

$$V_0 = \{\nu \mid \int_0^1 [\nu(x)]^2 \, dx < \infty, \qquad \int_0^1 [\nu'(x)]^2 \, dx < \infty, \nu(\Gamma_D) = 0\} \, . \tag{3.10}$$

Although the boundary condition in (3.9) can be removed for this example, it will be retained for illustrative purposes. We are therefore left with the weak problem of finding $u(x) \in V_0$ that satisfies

$$\int_0^1 \frac{d\nu(x)}{dx}\frac{du(x)}{dx}dx - \nu(x)\frac{du}{dx}\Big|_{x=0}^{x=1} = + \int_0^1 \nu(x)b(x)dx \qquad \forall \nu \in V_0 \quad . \tag{3.11}$$

3.1.3 Galerkin Method

There are multiple different numerical methods that can be used in isogeometric analysis. The Galerkin method will be used in this work, which is a finite-dimensional approximation method where the trial and the weighting spaces are the same. Other numerical methods include the Collocation, Least-Squares FEM and Meshless methods which are described in detail in [1, p. 72]. The weak form can be represented with approximation functions from the definition equations (2.12) and (3.1). By extension, $\nu(x)$ and the derivatives of the functions, $\frac{du(x)}{dx}$ and $\frac{d\nu(x)}{dx}$ are defined as

$$\nu(x) = \sum_{i=1}^{n} R_i^p(\xi)\mathsf{v}_i \tag{3.12}$$

$$\frac{du(x)}{dx} = \sum_{i=1}^{n} R_{i,x}^p(\xi)d_i \tag{3.13}$$

$$\frac{d\nu(x)}{dx} = \sum_{i=1}^{n} R_{i,x}^p(\xi)\mathsf{v}_i \tag{3.14}$$

where $\mathbf{v}$ is the vector of weighting points that describe the weighting function. These equations are substituted into the weak form to give

$$\int_0^1 \left(\sum_{i=1}^{n} R_{i,x}^p(\xi)\mathsf{v}_i\right)\left(\sum_{i=1}^{n} R_{i,x}^p(\xi)d_i\right)dx = \left(\sum_{i=1}^{n} R_i^p(\xi)\mathsf{v}_i\right)\bar{q}\Big|_{x=0}^{x=1} + \int_0^1 \left(\sum_{i=1}^{n} R_i^p(\xi)\mathsf{v}_i\right)b(x)dx \qquad (3.15)$$
$$\forall\nu \text{ with } \nu(0)=0, \nu(1)=0.$$

In order to represent the approximation in equation (3.16) as a system of linear equations, shape function arrays and control points are introduced so that the approximation may be solved numerically. The shape functions and shape function derivatives can be written in arrays as

$$\mathbf{R} = [R_i^p(\xi), R_{i+1}^p(\xi), R_{i+2}^p(\xi)] \qquad (3.16)$$
$$\mathbf{B} = [R_{i,x}^p(\xi), R_{i+1,x}^p(\xi), R_{i+2,x}^p(\xi)] \qquad (3.17)$$

where both $\mathbf{R}$ and $\mathbf{B}$ are row vectors of length $p+1$ for $i \in [1:n]$ which contain the relevant shape functions of that element. In this case, i is the first shape function number associated with the relevant element or domain. More detail on shape function selection will follow in Section 3.1.5. By defining these arrays, the approximations (3.12-3.14) are written as

$$u(x) = \mathbf{R}\boldsymbol{d} \qquad (3.18)$$
$$\nu(x) = \mathbf{R}\mathsf{v} = \mathsf{v}^T\mathbf{R}^T \qquad (3.19)$$
$$\frac{du(x)}{dx} = \mathbf{B}\boldsymbol{d} \qquad (3.20)$$
$$\frac{d\nu(x)}{dx} = \mathbf{B}\mathsf{v} = \mathsf{v}^T\mathbf{B}^T \quad . \qquad (3.21)$$

Equation (3.16) can then be written as

$$\underbrace{\int_0^1 \mathsf{v}^T\mathbf{B}^T\mathbf{B}dx}_{\boldsymbol{K}}\,\boldsymbol{d} = \underbrace{\mathsf{v}^T\mathbf{R}^T\bar{q}\Big|_{x=0}^{x=1} - \int_0^1 \mathsf{v}^T\mathbf{R}^T b(x)dx}_{\boldsymbol{f}}\,. \qquad (3.22)$$

v^T is arbitrary and thus

$$\underbrace{\int_0^1 \mathbf{B}^T\mathbf{B}dx}_{\boldsymbol{K}}\,\boldsymbol{d} = \underbrace{\mathbf{R}^T\bar{q}\Big|_{x=0}^{x=1} - \int_0^1 \mathbf{R}^T b(x)dx}_{\boldsymbol{f}}\,. \tag{3.23}$$

Equation (3.23) is written in the Bubnov-Galerkin form, where $\boldsymbol{K}$ is usually referred to as the stiffness matrix and $\boldsymbol{f}$ as the force vector in accordance to many other FEM conventions. While the terms stiffness matrix and force vector only strictly apply to certain categories of problems, they will be used throughout this work for simplicity and clarity. For example, with a heat problem, the $\boldsymbol{K}$ should be referred to as the conductance matrix. Finally, $\boldsymbol{d}$ is the vector of control points that define the solution field, that is found from the constructed system of linear equations $\boldsymbol{K}\boldsymbol{d} = \boldsymbol{f}$.

Both IGA and traditional FEM techniques for linear problems result in a system of linear equations in the form $\boldsymbol{K}\boldsymbol{d} = \boldsymbol{f}$. The difference is while $\boldsymbol{d}$ in traditional FEM are the actual nodal degrees of freedom in the solution field, $\boldsymbol{d}$ in IGA are the control points of a NURBS curve that approximates the solution field.

Because this problem does not contain any Neumann boundary conditions and $\nu = 0$ at both ends, the Neumann term is now excluded. Implementation of Neumann boundary conditions will be explained in Section 3.2.

$$\underbrace{\int_0^1 \mathbf{B}^T\mathbf{B}dx}_{\boldsymbol{K}}\,\boldsymbol{d} = \underbrace{-\int_0^1 \mathbf{R}^T b(x)dx}_{\boldsymbol{f}}\,. \tag{3.24}$$

In order to solve the integrals computationally, Gauss-Legendre quadrature (or simply Gauss quadrature) is used [4]. An element of order p requires $p+1$ point Gauss quadrature if exact integration is required [4, p. 17]. There are multiple other integration approximation techniques, but because IGA has rectilinear elements in the parametric domain by default, it lends itself well to Gauss quadrature and will be used in this dissertation. There is more than one domain transformation to get from the physical domain to the isoparametric domain where the Gaussian quadrature occurs and therefore all transformations need to be discussed in depth before equation (3.24) can be implemented.

3.1.4 Domain Transformation

The transformations between the different domains are best described with a two-dimensional space, where ξ is the parameter in the first direction, η is the parameter that describes the

second direction in the parametric domain. Similarly, the isoparametric domain is defined by $\bar{\xi}$ and $\bar{\eta}$. The diagram below illustrates the domains that are used in IGA. Integration via Gaussian quadrature occurs in the isoparametric domain. The Gauss points are mapped into the parametric domain in which analysis occurs and elements are defined and finally the control points are used to map the solution to the physical domain.

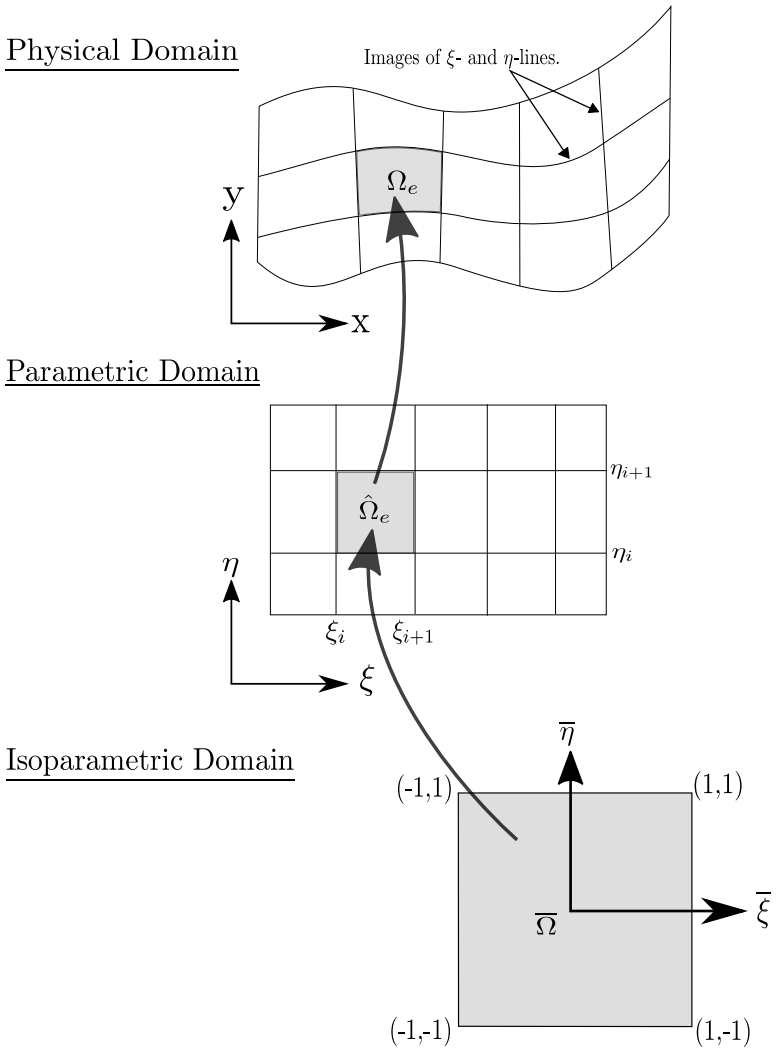


Figure 3.1 Illustration of domains used for integration in analysis. Gauss quadrature is performed in the isoparametric domain which is mapped to each of the elements defined in the parametric domain and finally to the physical domain.

The parametric domain is discretized into elements defined by the knot vectors, Ξ and $\mathscr{H}$. This allows equation (3.24) to be assembled element wise, where Gaussian quadrature occurs for each element. The system is comprised of a series of elemental stiffness matrix

and force vector contributions that are assembled into global stiffness matrix and force vector respectively. The transformation of a general function, $f(x, y)$ from the physical domain to the parametric and finally the isoparametric domain is described as

$$\begin{aligned}\int_{\Omega} f(x,y)d\Omega = \sum_{e=1}^{el_{total}} \int_{\Omega_e} f(x,y)d\Omega_e &= \sum_{e=1}^{el_{total}} \int_{\hat{\Omega}_e} f(x(\xi),y(\eta))|\boldsymbol{J_\xi}|d\hat{\Omega}_e \\ &= \sum_{e=1}^{el_{total}} \int_{\bar{\Omega}_e} f(x(\bar{\xi}),y(\bar{\eta}))|\boldsymbol{J_\xi}||\boldsymbol{J_{\bar{\xi}}}|d\bar{\Omega}_e \quad . \end{aligned} \tag{3.25}$$

The scaling and transformation between the domains is done with a series of Jacobians: the isoparametric Jacobian, $J_{\bar{\xi}}$, and the parametric Jacobian, J_ξ.

The transformation of points in the isoparametric domain $(\bar{\xi}, \bar{\eta})$ to the parametric domain (ξ, η) is done as follows:

$$\begin{aligned}\xi &= \frac{1}{2}[(\xi_{i+1} - \xi_i)\bar{\xi} + (\xi_{i+1} + \xi_i)] \\ \eta &= \frac{1}{2}[(\eta_{j+1} - \eta_j)\bar{\eta} + (\eta_{j+1} + \eta_j)] \quad . \end{aligned} \tag{3.26}$$

Here $\bar{\xi}$ and $\bar{\eta}$ are points (typically quadrature points) in the isoparametric domain and $[\xi_i, \xi_{i+1}] \times [\eta_j, \eta_{j+1}]$ is the element domain. These equations are typically used during numerical integration for equation (3.24). The determinant of the isoparametric Jacobian for the isoparametric to parametric domain transformation is

$$|J_{\bar{\boldsymbol{\xi}}}| = \frac{1}{4}(\xi_{i+1} - \xi_i)(\eta_{j+1} - \eta_j) \tag{3.27}$$

where the subscript $\bar{\boldsymbol{\xi}}$ represents $(\bar{\xi})$ in the one-dimensional NURBS case, $(\bar{\xi}, \bar{\eta})$ in the two-dimensional NURBS case and so forth.

The parametric Jacobian of the transformation between the parametric domain and the physical domain is described as

$$J_{\boldsymbol{\xi}} = \begin{bmatrix} x_{,\xi} & x_{,\eta} \\ y_{,\xi} & y_{,\eta} \end{bmatrix} \quad . \tag{3.28}$$

Each component in the parametric Jacobian is calculated from the parametric domain, using the control points. By expressing $\boldsymbol{x}$ as a function of ξ, using equation (2.12), it is

possible to determine the entries of J_ξ from the elemental control points and non-zero basis functions for that element. Each component of the matrix above can thus be described by

$$(J_{\boldsymbol{\xi}})_{IJ} = \frac{dx_I}{d\xi_J} = \frac{dR^{p,q}_{i,j}(\xi,\eta)}{d\xi_J} P_{iI} \tag{3.29}$$

where I, J are the I^{th} and J^{th} components on the physical, $\boldsymbol{x}$, and parametric, $\boldsymbol{\xi}$ domains respectively. For a two-dimensional NURBS surface in a two-dimensional physical domain, both I and J run from 1:2. Again, bold $\boldsymbol{x}$ represents (x) in a one-dimensional domain and (x, y) in a two-dimensional domain, *etc.* Similarly, $\boldsymbol{\xi}$ represents (ξ) in a one-dimensional parametric domain and (ξ, η) in a two-dimensional parametric domain. $\boldsymbol{P}_{iI}$ refers to the I^{th} component of the i^{th} control point.

The next step is to implement the isogeometric process in a MATLAB® code.

3.1.5 One-dimensional IGA Implementation

This section describes the code for the Poisson problem outlined in Section 3.1.1 to guide the user through a MATLAB® implementation. The IGA work flow process is very similar to the assembly and solving methods used in solving traditional FEM problems. Figure 3.2 shows an overview of the assembly and solving process for a one-dimensional IGA problem. The corresponding MATLAB® code is available as part of the digital submission of this dissertation. Variables from the digital submission of this code that are included in this dissertation, are written in `teletype font`.

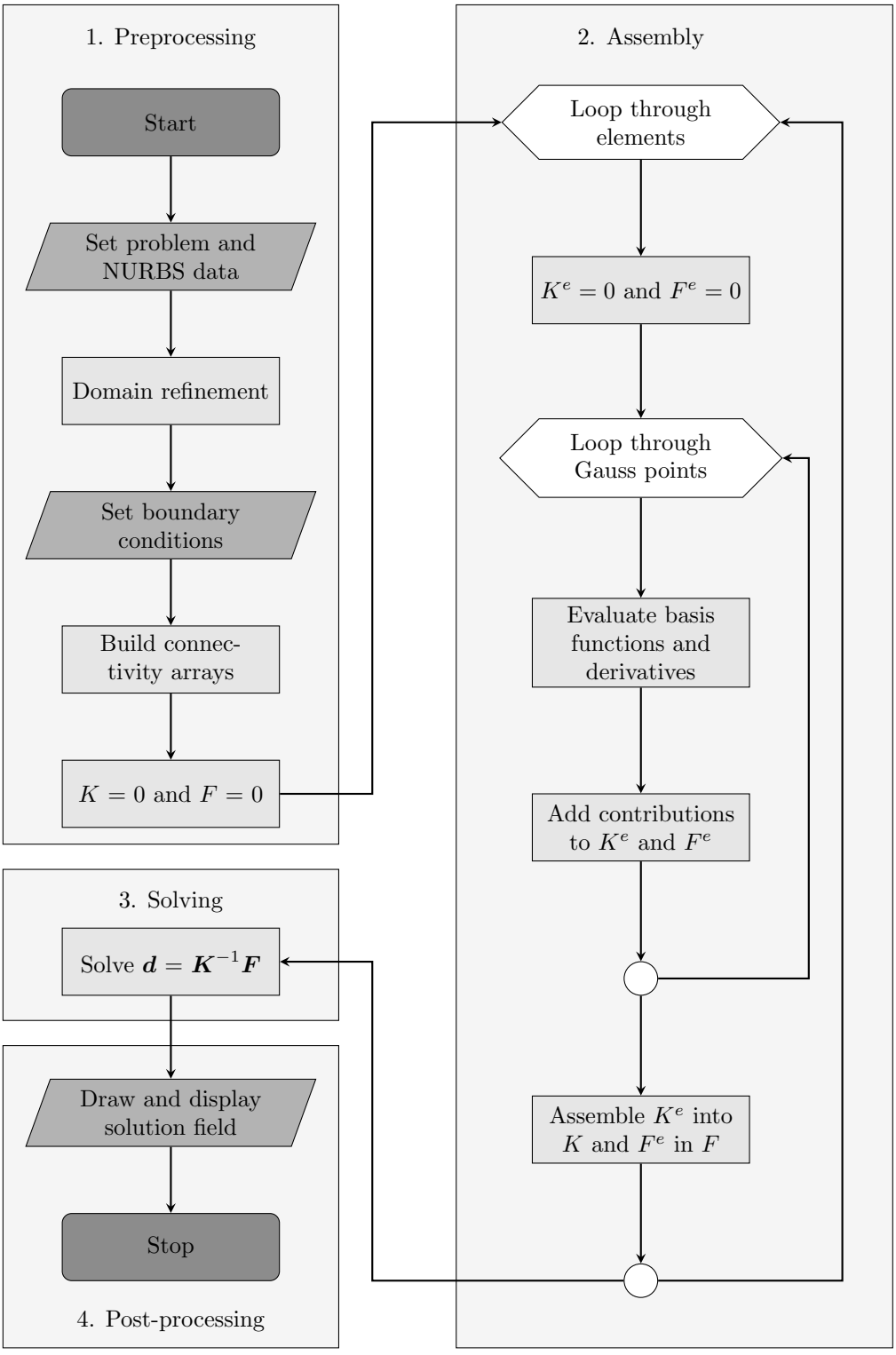


Figure 3.2 Flow chart of a Galerkin one-dimensional isogeometric analysis process

The solution of a one-dimensional IGA problem can be broken up into four main stages [7]:

1. Preprocessing, where problem and NURBS variables are defined.
2. Assembling the discrete system of equations $\boldsymbol{K}\boldsymbol{d} = \boldsymbol{f}$.
3. Solving the system of equations.
4. Postprocessing, where the solution is displayed, additional information is calculated and the NURBSs are described.

The initial setup of the code includes defining what type of refinement is required, setting the problem specific specifications such as the source term and exact solution, defining the NURBS geometry of the problem and how many points are required to plot the final results. The domain is then refined to allow for improved accuracy in the final result. The boundary conditions are defined after refinement. Algorithm 3 lists the steps required in the preprocessing procedure.

Algorithm 3 Preprocessing procedure for one-dimensional IGA

1: Set `refinementlevel` of h-refinement
2: Set `p_target` to select a target order if order refining
3: Set `refinementType` to select an order refinement
4: Set Problem boundaries
5: Set NURBS variables: p, Ξ, $\boldsymbol{P}$ and $\boldsymbol{w}$
6: Set number of plotting points, `ppoints`
7: Create parametric vector of points, `xi_vec`
8: Select Gauss quadrature order
9: Perform refinement
10: Update number of elements, `el_num`
11: Set Dirichlet node numbers, `n_E`
12: Set Dirichlet node values, `d_E`
13: Set Free node numbers, `n_F`

h-refinement is defined by how many times each element is halved (`refinementlevel`). p- and k-refinement are set by setting an order target (`p_target`) and which type of order refinement is preferred (`refinementType`).

The boundary conditions are defined by grouping the Dirichlet essential and remaining free degrees of freedom (DOFs) into vectors `n_E` and `n_F` respectively. The degree of freedom values, used to impose the Dirichlet boundary conditions, associated with the DOF numbers in `n_E` are stored in vector `d_E`. When homogeneous Dirichlet boundary conditions are used, the actual values are used for the control points. If the Dirichlet boundary conditions are inhomogeneous, the control point values need to be calculated with a method such as that detailed in Section 2.5.1.4. There are no Neumann boundary conditions in this problem. The DOF numbering convention in the one-dimensional scalar case is equal to the associated control point numbers. The connectivity array (`asm`) defines which elements interact with each other and which nodes contribute to which element. In the case of a one-dimensional problem, it is simply the adjacent elements, which is equal to the element number plus the order, p. The connectivity array also determines which shape function numbers are used in an element. Although elements are defined by the knot vector, it can be useful to consider the span of the basis functions associated with each element. Each element is associated with $p+1$ control points. For example, as can be seen in Figure 3.3 below, if a domain consists of six elements and the order, $p = 2$, then there will be eight control points. The first three control points will be associated to the first elements. The second to fourth control points are associated with the second element and so forth. Two-dimensional connectivity arrays will be dealt with in Section 4.1.

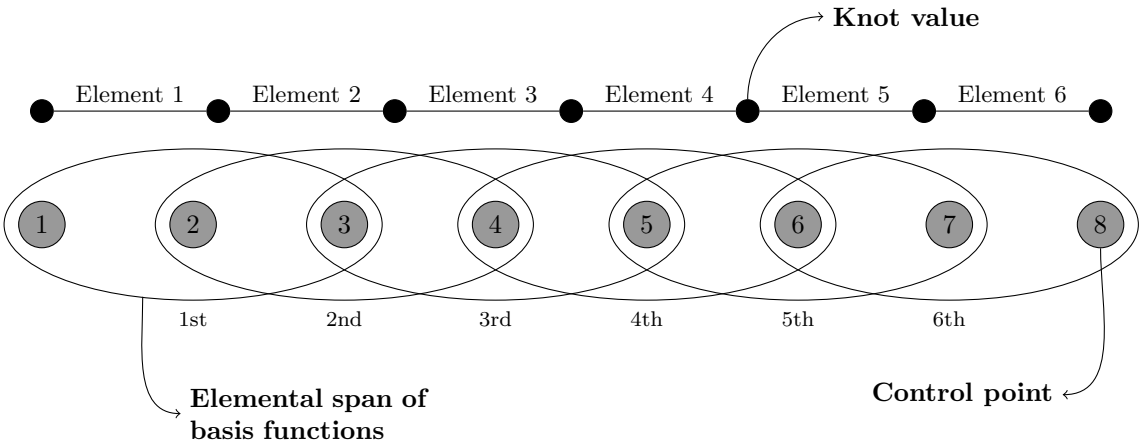


Figure 3.3 Diagram showing the relationship between control points and elements.

Assembly of the global stiffness matrix and force vectors occurs in the main section of the code. This procedure is outlined in Algorithm 4. It starts by initializing the stiffness

and force arrays. The elemental loop is then started, running through all the elements in the domain. Each time the elemental loop is traversed, the elemental stiffness and force contributions are added to the global stiffness matrix and force vector. Note that an element is defined as the non-zero knot span between two knot values. However, in calculating the elemental contributions to the stiffness matrix and force vector, the zero knot spans are also included and therefore the number of elemental loops required are `el_total=n-p`. The connectivity array (`asm`) selects the control points associated with the relevant element (`x`) as well as determines the size of the shape function array $(p+1)$.

The elemental stiffness matrix and force vector contributions are then initialized as well as setting i, which is the counter that determines which knot span is being worked with. i corresponds to the i seen in equations (2.13) and (3.26).

Next, the Gauss loop is started, running for the number of Gauss points chosen. In each Gauss loop, the contributions are added to the elemental stiffness matrix and force vector. This is done by mapping the Gauss points from the isoparametric domain to the parametric domain, using equation (3.26).

The shape function array, $\mathbf{R}$ and the derivative of the shape function array, $\mathbf{R}_{,\boldsymbol{\xi}}$, which are associated with particular Gauss points in the parametric domain, are then computed, using the shape function are associated with particular elements as seen in Figure 3.3. In each element, there are Gauss points that are mapped from the isoparametric domain using equations (3.26) into the parametric domain and are used to calculate the required shape functions associated with that element. For example, using Figure 3.3, if the second element is being considered, the Gauss points would be mapped into the parametric domain in the range of the element and the associated shape function numbers would be 2, 3 and 4 for a $p = 2$ element. The same numbers are used for the derivative of the shape functions. The shape function array of any one-dimensional element of order p would be

$$\mathbf{R} = \begin{bmatrix} R^p_{el}(\xi), & R^p_{el+1}(\xi), & \dots & R^p_{el+p}(\xi) \end{bmatrix} \tag{3.30}$$

where el is the element number which is currently being chosen.

The parametric Jacobian, J_ξ, scales the parametric domain to the physical domain using equation (3.28). The isoparametric Jacobian, $J_{\bar{\xi}}$, that scales the isoparametric domain to the parametric domain is calculated from equation (3.27). The parametric Jacobian, together with the derivative of the shape function array is used to calculate the derivative shape function array, $\mathbf{B}$, which is of the form

$$\mathbf{B} = \begin{bmatrix} R^p_{el,x}, & R^p_{el+1,x}, & \dots & R^p_{el+p,x} \end{bmatrix} \quad . \tag{3.31}$$

Finally, before the elemental stiffness matrix and force vector can be calculated, the Gauss point needs to be mapped to the physical domain, because the source term, $b(x)$ is a function of the physical domain, x. Equation (2.12) is used to calculate the Gauss point on the NURBS curve in the physical space. The elemental stiffness matrix and force vector are assembled into their respective global arrays.

Algorithm 4 shows the global assembly procedure of the stiffness matrix and force vector for a one-dimensional problem. The algorithm is based off $p = 2$ order basis functions. Note that the shape function numbers in line 9 and 10 are illustrative and show the first element shape functions only. The shape function numbers associated with each element would be `el` to `el+p`.

Algorithm 4 Procedure for evaluating stiffness matrix and force vector

1: Initialize $\boldsymbol{K}$ and $\boldsymbol{f}$
2: **for** `el=1:el_total` **do**
3: Initialize $\boldsymbol{K}^e$ and $\boldsymbol{f}^e$
4: Define elemental connectivity array, `asm=el:el+p`
5: Select elemental control points, `x=P(asm)`
6: Set i to select element domains, `i = el+p`
7: **for** $gp = 1 : \#GaussPoints$ **do**
8: Calculate parametric coordinate, using equation (3.26)
9: Compute shape function array at point ξ, $\mathbf{R} = [R^p_1(\xi), R^p_2(\xi), R^p_3(\xi)]$
10: Compute derivative of shape function array, $\mathbf{R}_\xi = [R^p_{1,\xi}(\xi), R^p_{2,\xi}(\xi), R^p_{3,\xi}(\xi)]$
11: ▷ *Note in lines 9 and 10,* $\mathbf{R}$ *and* $\mathbf{R}_\xi$ *would have p+1 entries.*
12: Compute parametric Jacobian, $J_\xi = ||\mathbf{R}_\xi \boldsymbol{x}||$
13: Compute isoparametric Jacobian, $J_{\bar{\xi}} = \frac{1}{2}(\xi_{i+1} - \xi_i)$
14: Compute derivative shape function array, $\mathbf{B} = \mathbf{R}_{\boldsymbol{x}} = J_\xi^{-1} R^T_\xi$
15: Compute $\bar{\xi}$ in physical domain, x_{source}
16: Compute $\boldsymbol{K}^e$, $\boldsymbol{K}^e = \boldsymbol{K}^e + J_\xi J_{\bar{\xi}} w_{gp} \mathbf{B}^T \times \mathbf{B}$
17: Compute $\boldsymbol{f}^e$, $\boldsymbol{f}^e = \boldsymbol{f}^e + J_\xi J_{\bar{\xi}} w_{gp} b(x_{source}) \times \mathbf{R}^T$
18: **end for**
19: Assemble $\boldsymbol{K}_{\texttt{asm asm}} = \boldsymbol{K}_{\texttt{asm asm}} + \boldsymbol{K}^e$
20: Assemble $\boldsymbol{f}_{\texttt{asm}} = \boldsymbol{f}_{\texttt{asm}} + \boldsymbol{f}^e$
21: **end for**

Note: w_{gp} is the Gauss point weight, not the NURBS weight.

Now that $\boldsymbol{K}$ and $\boldsymbol{f}$ have been obtained, the Dirichlet boundary conditions can be applied and $\boldsymbol{d}$ can be solved for. Because the multiplicities of the knot vector start and end vector is $m_i = p + 1$, the knot vector is considered to be open. This means that the domain boundaries are discontinuous. It also means that there is only one basis function that is of unity and therefore boundary conditions can be applied exactly through the corresponding Dirichlet control points. This is an important feature in isogeometric analysis. There are numerous ways in which boundary conditions can be imposed, but in this dissertation the partition method is used. This is done by creating a system of linear equations and partitioning the equations by selecting the components associated with the essential (Dirichlet) DOFs and the free DOFs, using vectors `n_E` and `n_F` respectively. The system of linear equations is then reordered such that the DOFs are grouped as $\boldsymbol{d}_E$ and $\boldsymbol{d}_F$, as well as partitioning the associated stiffness matrix and force vector components as follows:

$$\begin{bmatrix} \boldsymbol{K}_E & \boldsymbol{K}_{EF} \\ \boldsymbol{K}_{FE} & \boldsymbol{K}_F \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_E \\ \boldsymbol{d}_F \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_E \\ \boldsymbol{f}_F \end{bmatrix} \quad . \tag{3.32}$$

By rearranging the above the unknown control points, $\boldsymbol{d}_F$ can be solved for as

$$\boldsymbol{d}_F = \boldsymbol{K}_F^{-1}(\boldsymbol{f}_F - \boldsymbol{K}_{EF}^T \boldsymbol{d}_E) \quad . \tag{3.33}$$

$\boldsymbol{d}_E$ and $\boldsymbol{d}_F$ are then assembled into $\boldsymbol{d}$ to give the control points for the field solution.

In the post-processing stage, the field solution and geometry NURBS curves are constructed, using the control points and definitions (2.12) and (3.1). They can then be plotted on a graph to compare against the exact solution. Care should be taken to plot the field solution against the NURBS curve in the physical domain and not the parametric vector. Although both vectors might have the same domain range, the internal part of the domain can be warped and plotting the field solution against the parametric domain might portray inaccurate results.

3.1.6 Results

Figure 3.4 shows an unrefined two-element mesh with the shape function order p=2. The IGA solution, represented by the thin blue line is slightly off the exact solution (dashed thicker black line). The mesh is then globally h-refined once so that there are four elements in the domain and the result is shown in Figure 3.5. It can be seen that the IGA solution approximates the exact solution much more closely. Next, order refinement is illustrated.

k-refinement of the original two element domain is performed to increase the shape function order to three. Again, the IGA solution matches the exact solution closely with minimal refinement, where the domain is still only comprised of two elements.

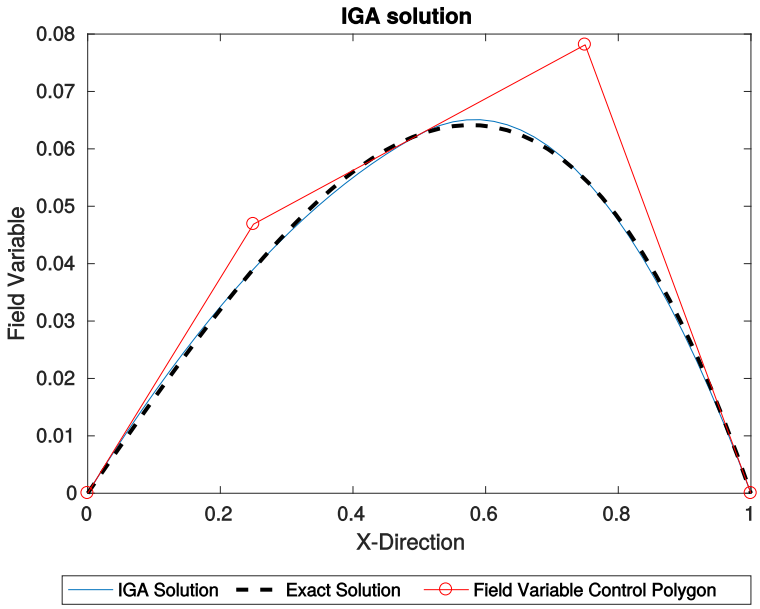


Figure 3.4 Solution to equation (3.2) with two elements.

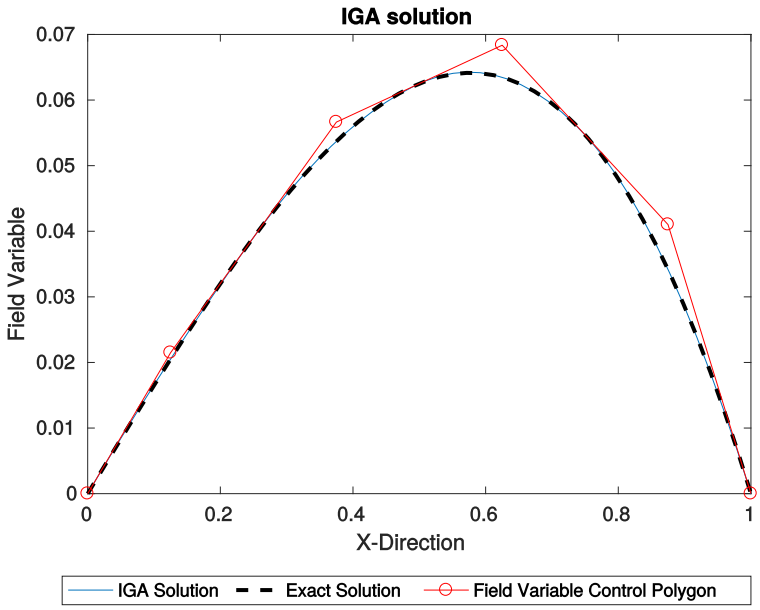


Figure 3.5 Solution to equation (3.2), h-refined to four elements.

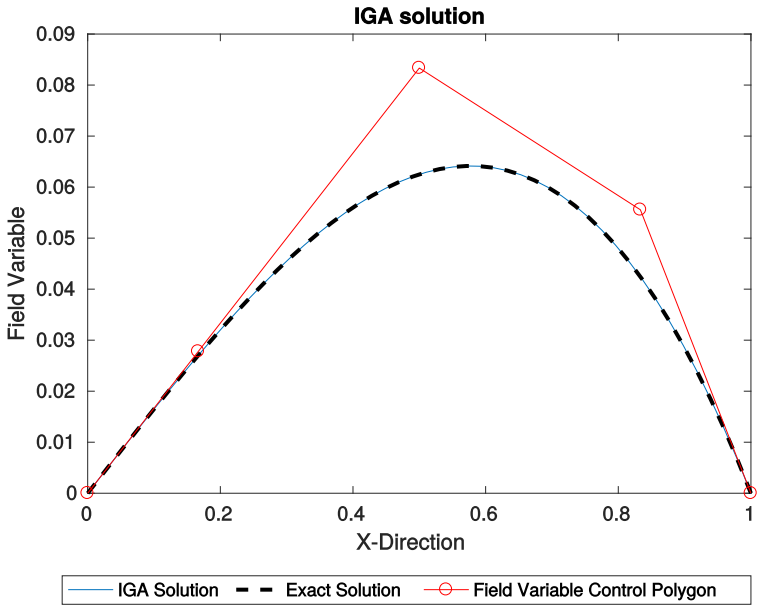


Figure 3.6 Solution to equation (3.2), k-refined to have an order of three and two elements.

Figure 3.7 below shows the L2 error graphs for each refinement type. The L2 error was calculated by comparing the IGA solution to the exact solution with the formula [7, p. 114]

$$||e_{L2}|| = \left(\int_{\Omega} \left(u_{ex}(x) - u_h(x) \right)^2 d\Omega \right)^{\frac{1}{2}} \quad , \tag{3.34}$$

where $u_{ex}(x)$ is the exact solution and $u_h(x)$ is the IGA solution.

The first graph in Figure 3.7 shows the L2 error convergence rate for global h-refinement on a log scale. From the graph it can be seen that the gradient is linear, which indicates that it is still busy converging. For both p- and k-refinement, the L2 error converges after order refinement equal to p=3.

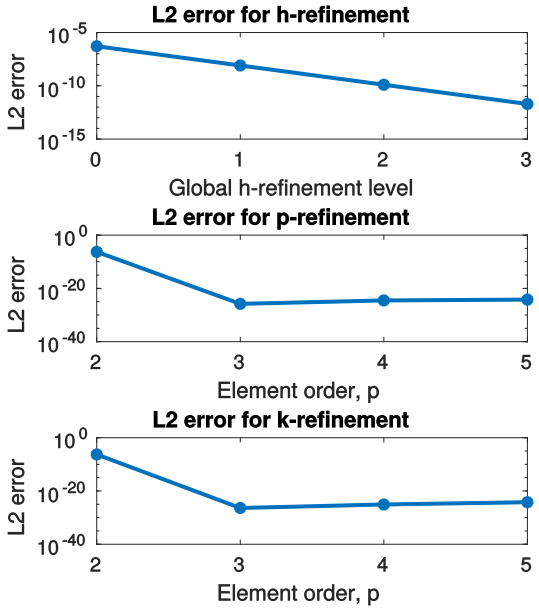


Figure 3.7 L2 error of h, p and k-refinement.

3.2 Radial Temperature

The next problem that will be approximated is a heat conduction problem across a two-dimensional domain for which the domain and source terms have radial symmetry and can therefore be expressed as a one-dimensional problem. It will be shown how a Neumann boundary conditions is implemented for a one-dimensional problem. The problem is described by

$$\boldsymbol{\nabla}^{\boldsymbol{T}}\boldsymbol{q} - s(x,y) = \mathbf{0} \tag{3.35}$$

where

$$\boldsymbol{q} = -\boldsymbol{D}\boldsymbol{\nabla}T \tag{3.36}$$

$$\boldsymbol{D} = k\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \quad . \tag{3.37}$$

$\boldsymbol{D}$ is the conductivity matrix and it is assumed that the material of the domain is isotropic. k is the thermal conductivity with the units $\frac{W}{m\cdot K}$, $T(x,y)$ is the temperature across the domain as a function of the spatial position and s is the source term, also a function of the spatial position.

A circular domain with boundary conditions that result in a temperature field that varies one-dimensionally, radially, is considered. The domain is shown in Figure 3.8 where there are both Dirichlet boundary conditions, Γ_D and Neumann boundary conditions Γ_N which are defined as

$$\begin{aligned}\bar{T} &= T_{r=r_o} \quad \text{on } \Gamma_D\\ \bar{q} &= \frac{T_{r=r_i}}{dr} \quad \text{on } \Gamma_N\end{aligned} \tag{3.38}$$

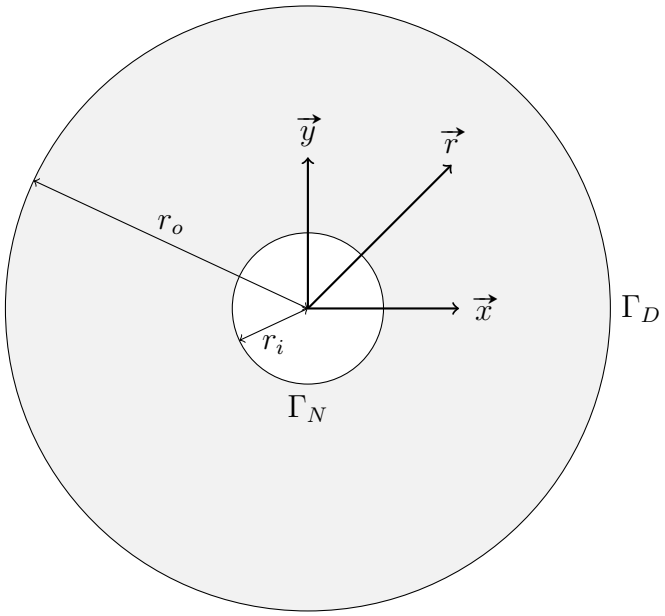


Figure 3.8 Diagram of one-dimensional radial temperature problem.

Because this problem is radially symmetric, it can be reduced to a one-dimensional IGA problem. The same problem will also be approximated as a two-dimensional problem in Section 4.1.5. The one-dimensional strong form can be written as

$$k\frac{d}{dr}\left(r\frac{dT(r)}{dr}\right) + rs(r) = 0 \quad . \tag{3.39}$$

The difference between this equation and the equation 3.2 is that the source term is multiplied by radial position. The problem specifics are

$$\begin{aligned}
k &= 10\frac{W}{m\cdot K} \\
s(r) &= -10\frac{W}{m^3} \\
\bar{q}(r) &= 0\frac{K}{m} \\
r_i &= 1m \\
r_o &= 4m \\
\bar{T} &= 0K \quad \text{on } \Gamma_D \\
\bar{q} &= 0\frac{K}{m} \quad \text{on } \Gamma_N \\
T_{exact}(r) &= \frac{r_i^2}{2}\frac{s(r)}{k}ln(r) + \frac{r_o^2}{4}\frac{s(r)}{k} - \frac{r_i^2}{2}\frac{s(r)}{k}ln(r_o) - \frac{r^2}{4}\frac{s(r)}{k} \quad .
\end{aligned} \tag{3.40}$$

The space of admissible functions, similar to that described previously, are

$$V_0 = \{\nu \mid \int_{r_i}^{r_o} [\nu(r)]^2 \, dr < \infty, \quad \int_{r_i}^{r_o} [\nu'(r)]^2 \, dr < \infty, \nu(\Gamma_D) = 0\} \, . \tag{3.41}$$

The weak form can find $T(r) \in V_0$ that satisfy $T(r_o) = 0$ such that:

$$\int_{r_i}^{r_o} \frac{d\nu}{dr} kr \frac{T(r)}{dr} dr = \nu kr \frac{dT}{dr}\bigg|_{r_o} + \int_{r_i}^{r_o} \nu r s(r) dr \quad \forall \nu \in V_0 \quad . \tag{3.42}$$

Similar to the process performed in Section 3.1.3, approximations are substituted into the weak form in equation (3.42) to give

$$\int_{r_i}^{r_o} \mathbf{v}^T \mathbf{B}^T(kr) \mathbf{B} \boldsymbol{d} \, dr = \mathbf{v}^T \mathbf{R}^T(kr) \bar{q}(r)\Big|_{r_o} + \int_{r_i}^{r_o} \mathbf{v}^T \mathbf{R}^T r s(r) dr \quad \forall \nu \tag{3.43}$$

and can be written in the form

$$\underbrace{\int_{r_i}^{r_o} \mathbf{B}^T(kr) \mathbf{B} \, dr}_{\boldsymbol{K}} \, \boldsymbol{d} = \underbrace{\mathbf{R}^T(kr) \bar{q}(r)\Big|_{r_o} + \int_{r_i}^{r_o} \mathbf{R}^T r s(r) dr}_{\boldsymbol{f}} \quad \forall \nu \quad . \tag{3.44}$$

3.2.1 Implementation

The same implementation procedure is performed as described in Section 3.1.5, except that there is now a Neumann term to consider and implement, as well as the slight change calculating the elemental stiffness matrices and force vectors. Algorithm 5 details the assembly procedure, highlighting how the Neumann contribution is added to the force vector. The procedure for the assembly of the stiffness matrix and source contribution of the force vector is the same as in Algorithm 4 with the exception of the lines 6 and 7 in Algorithm 5, where the final stiffness and force contribution formula include the value of the Gauss quadrature point in the physical domain, r_{source}. All differences of this problem to the Poisson one-dimensional problem will be illustrated in Algorithm 5. If nothing is explicitly stated, it can be assumed that it is the same as the Poisson problem.

Algorithm 5 Procedure for evaluating Neumann component of force vector

Initialize $\boldsymbol{K}$ and $\boldsymbol{f}$
for `el=1:el_total` **do**
  Initialize $\boldsymbol{K}^e$ and $\boldsymbol{f}^e$
  Initialize $\boldsymbol{f}_N^e$
  for $gp = 1 : \#GaussPoints$ **do**
    Compute $\boldsymbol{K}^e$, $\boldsymbol{K}^e = \boldsymbol{K}^e + J_\xi J_{\bar{\xi}} w_{gp} \mathbf{B}^T (k\, r_{source}) \mathbf{B}$
    Compute $\boldsymbol{f}^e$, $\boldsymbol{f}^e = \boldsymbol{f}^e + J_\xi J_{\bar{\xi}} w_{gp} (r_{source}\, s(r_{source})) \mathbf{R}^T$
  end for
  Assemble $\boldsymbol{K}_{\texttt{asm asm}} = \boldsymbol{K}_{\texttt{asm asm}} + \boldsymbol{K}^e$
  Assemble $\boldsymbol{f}_{\texttt{asm}} = \boldsymbol{f}_{\texttt{asm}} + \boldsymbol{f}^e$
  if `el=el_Neum` **then** ▷ *Only when the current element has a flux contribution.*
    Define elemental connectivity array, `asm=el:el+p`
    Select elemental control points, `x=P(asm)`
    Set i to select element domains, `i = el+p`
    for $gp = 1 : \#GaussPoints$ **do**
      Calculate parametric coordinate, using equation (3.26)
      Compute shape function array at point ξ, $\mathbf{R} = [R_1^p(\xi), R_2^p(\xi), R_3^p(\xi)]$
      Compute derivative of shape function array, $\mathbf{R}_\xi = [R_{1,\xi}^p(\xi), R_{2,\xi}^p(\xi), R_{3,\xi}^p(\xi)]$
      Compute parametric Jacobian, $J_\xi = ||\mathbf{R}_\xi \boldsymbol{x}||$
      Compute isoparametric Jacobian, $J_{\bar{\xi}} = \frac{1}{2}(\xi_{i+1} - \xi_i)$
      Compute $\bar{\xi}$ in physical domain, r_{source}
      Compute $\boldsymbol{f}_N^e$, $\boldsymbol{f}_N^e = \boldsymbol{f}_N^e + J_\xi J_{\bar{\xi}} w_{gp} (k\, r_{source}\, \bar{q}(r_{source})) \mathbf{R}^T$
    end for
  end if
  ▷ *Where* `n_flux` *is the elemental node number on which the flux occurs.*
  Assemble $\boldsymbol{f}^e_{\texttt{asm}_{\texttt{n\_flux}}} = \boldsymbol{f}^e_{\texttt{asm}_{\texttt{n\_flux}}} + \boldsymbol{f}_N^e$
end for

Note: w_{gp} is the Gauss point weight, not the NURBS weight.

Solving for $\boldsymbol{d}$ gives the control points for the NURBS curve approximating the temperature distribution across the domain. Boundary conditions are implemented using the partition method in equation (3.33). The free degrees of freedom can then be calculated using equation (3.33) and assembled into the global vector $\boldsymbol{d}$. As post-processing, the temperature and spatial NURBS curves are created to compare the approximated solution to the exact solution.

3.2.2 Results

The results of the IGA approximations for various refinements are plotted against the exact solution, along with the respective control polygons. The first figure shows the approximation with no refinement. While the IGA solution is close to the exact solution, it is still slightly inaccurate. Similar to the previous problem, with either one level of h-refinement or order refinement for $p = 3$, the accuracy can be improved significantly.

The key feature to note from this problem is how the Neumann boundary condition is implemented into the code. This problem will also be solved as a two-dimensional problem in Section 4.1.5. The corresponding MATLAB® code is included in Appendix D.1 and is available as part of the digital submission of the dissertation.

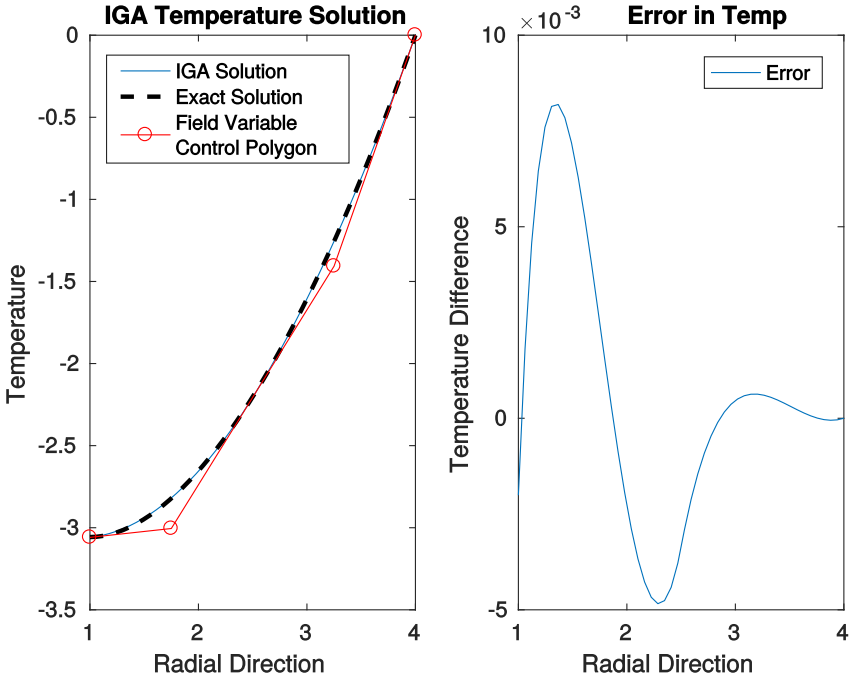


Figure 3.9 Temperature distribution for a radial heat problem with no refinement and the associated error.

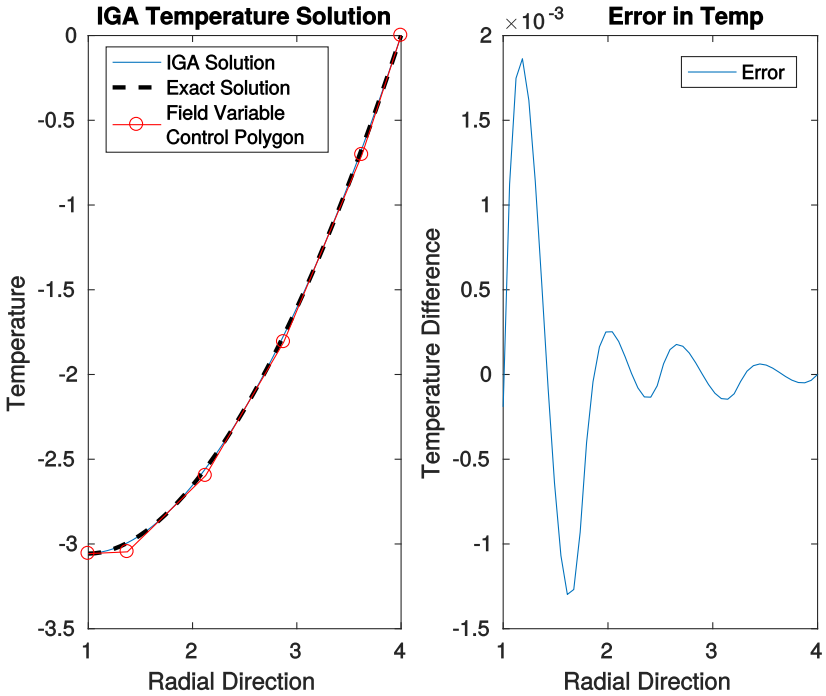


Figure 3.10 Temperature distribution for a radial heat problem with one level of h-refinement to four elements and the associated error.

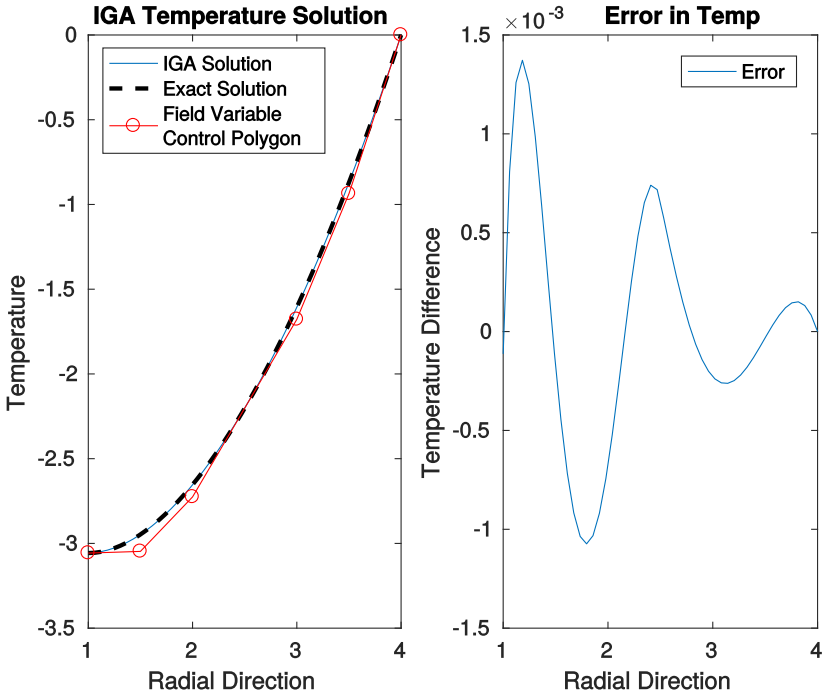


Figure 3.11 Temperature distribution for a radial heat problem with p-refinement to order $p = 3$ and the associated error.

Figure 3.12 below shows the L2 error graphs for each refinement type. The L2 error was calculated by comparing the IGA solution to the exact solution with the formula [7]

$$||e_{L2}|| = \left(\int_{\Omega} \left(u_{ex}(x) - u_h(x) \right)^2 d\Omega \right)^{\frac{1}{2}} \quad , \tag{3.45}$$

where $u_{ex}(x)$ is the exact solution and $u_h(x)$ is the IGA solution.

The first two graphs in Figure 3.12 both have linear convergence gradients of the error on a log scale. This indicates that while the error is busy converging, it is not expected to converge soon. Therefore, after a certain amount of refinement when the error is deemed sufficiently small for the application, the results should be taken. The third graph starts to converge after the problem is k-refined to p=4.

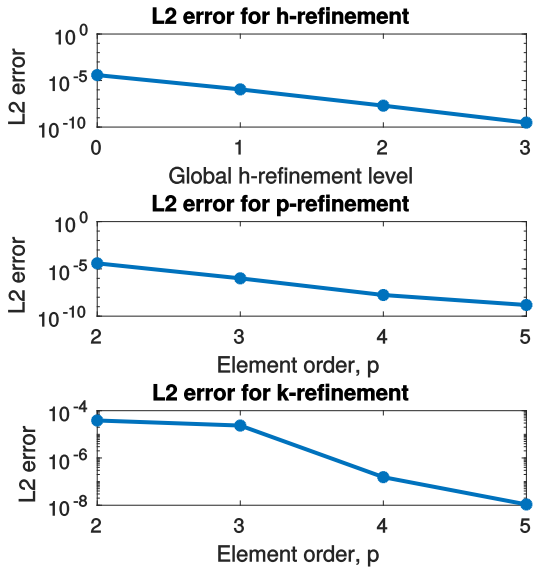


Figure 3.12 L2 error of h, p and k-refinement.

Chapter 4

Two-dimensional Scalar IGA

4.1 Two-dimensional IGA Solution Process

The implementation for a two-dimensional scalar IGA problem is very similar to one-dimensional IGA. Two-dimensional scalar problems have one field variable (degree of freedom) at each node in a two-dimensional parametric domain. It follows the same format as the one-dimensional case with the key difference that there are elements in two directions, which makes the basis function matrices and element connectivity arrays interact differently.

First, the strong and weak forms will be developed and from the weak form, the discrete Galerkin formulation will be developed. With the Galerkin formulation in place, the problem can be implemented into the software and used to approximate the solution. To demonstrate how to approximate a solution to a two-dimensional scalar problem, the same radial temperature problem that was introduced in Section 3.2, is revisited.

4.1.1 Strong Form

The strong form of the heat conduction equation is seen in equations (3.35) to (3.37) as well as the boundary conditions seen in equations (3.38). The problem specifics are the same, except that source and flux terms will be expressed as a function of x and y instead of r as

$$s(x, y) = -10 \tag{4.1}$$

$$\bar{\boldsymbol{q}}(x, y) = 0 \quad . \tag{4.2}$$

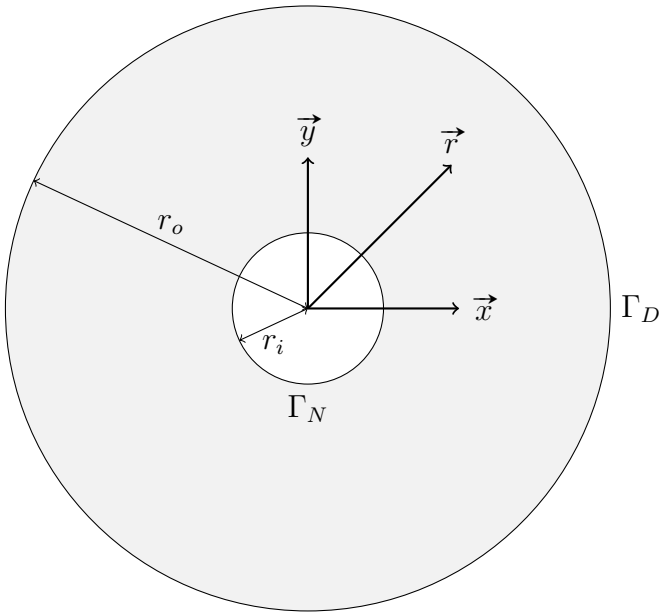


Figure 4.1 Diagram of two-dimensional radial temperature problem.

The problem domain is show in Figure 4.1.

4.1.2 Weak Form

The weak form of the equation (3.35) is the integral of the equation multiplied by the weighting function, $\nu(x, y)$. ν is an arbitrary function with $\nu = 0$ on Γ_D.

$$\int_{\Omega} \nu \boldsymbol{\nabla}^T \boldsymbol{q} - \nu s(x, y) d\Omega = 0 \quad \forall \nu \quad . \tag{4.3}$$

By applying Green's formula

$$\int_{\Omega} (\boldsymbol{\nabla}\nu)^T \boldsymbol{q} d\Omega = \int_{\Gamma} \nu \boldsymbol{q}\boldsymbol{n} d\Gamma - \int_{\Omega} \nu \boldsymbol{s}(x, y) d\Omega \quad \forall \nu \tag{4.4}$$

is obtained.

Functions $T(x, y)$ and $\nu(x, y)$, as well as their first derivatives are square integrable. Furthermore, the solution and the arbitrary test functions are assumed to satisfy the Dirichlet boundary conditions. The space of functions satisfying these conditions is denoted by V: that is,

$$V_0 = \{\nu \mid \int_{r_i}^{r_o} [\nu(x,y)]^2 \, d\Omega < \infty, \quad \int_{r_i}^{r_o} [\nu'(x,y)]^2 \, d\Omega < \infty, \nu(\Gamma_D) = 0\} \, . \tag{4.5}$$

By substituting equations (3.36) and (3.38) into equation (4.4), find $T(x,y) \in V_0$ so that $T = \bar{T}$ on Γ_D that satisfies the equation below where the weighting function, $\nu \in V_0$.

$$\int_\Omega (\boldsymbol{\nabla}\nu)^T \boldsymbol{D} \boldsymbol{\nabla} T(x,y) d\Omega = \int_\Omega \nu \boldsymbol{s}(x,y) d\Omega - \int_\Gamma \nu \bar{\boldsymbol{q}} d\Gamma \quad \forall \nu \in V_0 \tag{4.6}$$

4.1.3 Galerkin Form

The functions that form part of the weak form in equation (4.6) are now approximated with the NURBS definitions as

$$\nu = \sum_{i=1}^{n} \sum_{j=1}^{m} R_i^p(\xi) R_j^q(\eta) \mathbf{v}_{ij} \tag{4.7}$$

$$(\boldsymbol{\nabla}\nu)^T = \sum_{i=1}^{n} \sum_{j=1}^{m} R_{i,\boldsymbol{x}}^p(\xi) R_{j,\boldsymbol{x}}^q(\eta) \mathbf{v}_{ij} \tag{4.8}$$

$$T(x,y) = \sum_{i=1}^{n} \sum_{j=1}^{m} R_i^p(\xi) R_j^q(\eta) \boldsymbol{d}_{ij} \tag{4.9}$$

$$(\boldsymbol{\nabla} T)^T = \sum_{i=1}^{n} \sum_{j=1}^{m} R_{i,\boldsymbol{x}}^p(\xi) R_{j,\boldsymbol{x}}^q(\eta) \boldsymbol{d}_{ij} \tag{4.10}$$

Each of these approximations can be represented in a manner similar to equations (3.18-3.21) by using shape function and shape function derivative arrays and their corresponding control points as

$$\nu = \mathbf{R}\mathbf{v} = \mathbf{v}^T \mathbf{R}^T \tag{4.11}$$

$$(\boldsymbol{\nabla}\nu)^T = \mathbf{B}\mathbf{v} = \mathbf{v}^T \mathbf{B}^T \tag{4.12}$$

$$T(x,y) = \mathbf{R}\boldsymbol{d} \tag{4.13}$$

$$(\boldsymbol{\nabla} T)^T = \mathbf{B}\boldsymbol{d} \tag{4.14}$$

where $\mathbf{R}$ and $\mathbf{B}$ are the shape function and shape function derivative arrays respectively. Entries of the shape function array can be defined as

$$\mathsf{R}_{i+(p+1)(j-1)} = R_{i,j}^{p,q}(\xi,\eta) \quad . \tag{4.15}$$

$\mathbf{R}$ is a one-dimensional array of bivariate basis functions that are the product of two basis functions from each direction at a particular node. In expanded form, it reads

$$\begin{aligned}\mathbf{R} = [&R^{p,q}_{1,1}(\xi,\eta), R^{p,q}_{2,1}(\xi,\eta), \ldots, R^{p,q}_{p+1,1}(\xi,\eta), R^{p,q}_{1,2}(\xi,\eta), \ldots, \\ &R^{p,q}_{p+1,2}(\xi,\eta), \ldots, R^{p,q}_{1,q+1}(\xi,\eta), \ldots, R^{p,q}_{p+1,q+1}(\xi,\eta)] \quad .\end{aligned} \tag{4.16}$$

The shape function derivative array, $\mathbf{B}$ is

$$\mathbf{B} = \begin{bmatrix} \mathsf{R}^{sca}_{1,x}, & \mathsf{R}^{sca}_{2,x}, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),x} \\ \mathsf{R}^{sca}_{1,y}, & \mathsf{R}^{sca}_{2,y}, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),y} \end{bmatrix} \quad . \tag{4.17}$$

Equations (4.16-4.17) define the scalar two-dimensional shape function and shape function derivative arrays. For one-dimensional problems and two-dimensional vector problems, the arrays will be different.

The Galerkin formulation is

$$\int_\Omega \mathbf{v}^T \mathbf{B}^T \boldsymbol{D}\mathbf{B}\boldsymbol{d}d\Omega = \int_\Omega \mathbf{v}^T \mathbf{R}^T \boldsymbol{s}(x,y)d\Omega - \int_\Gamma \mathbf{v}^T \mathbf{R}^T \bar{\boldsymbol{q}}d\Gamma \quad \forall\nu \quad . \tag{4.18}$$

Because $\mathbf{v}^T$ is arbitrary, equation (4.18) can be manipulated to yield

$$\underbrace{\int_\Omega \mathbf{B}^T \boldsymbol{D}\mathbf{B}d\Omega}_{\boldsymbol{K}}\,\boldsymbol{d} = \underbrace{\int_\Omega \mathbf{R}^T \boldsymbol{s}(x,y)d\Omega - \int_\Gamma \mathbf{R}^T \bar{\boldsymbol{q}}d\Gamma}_{\boldsymbol{f}} \quad \forall\nu \tag{4.19}$$

which is implemented in MATLAB®.

4.1.4 Implementation

Two-dimensional scalar problems are implemented in a very similar manner to one-dimensional problems. It is set out in the same main stages as listed on page 53, namely preprocessing, assembling, solving and post-processing.

The preprocessing stage is similar to that set out in Algorithm 3 with the additional two-dimensional aspect. It involves defining the order refinement type (p-, k-refinement or neither) and the global h-refinement level. Then the problem specifics are applied,

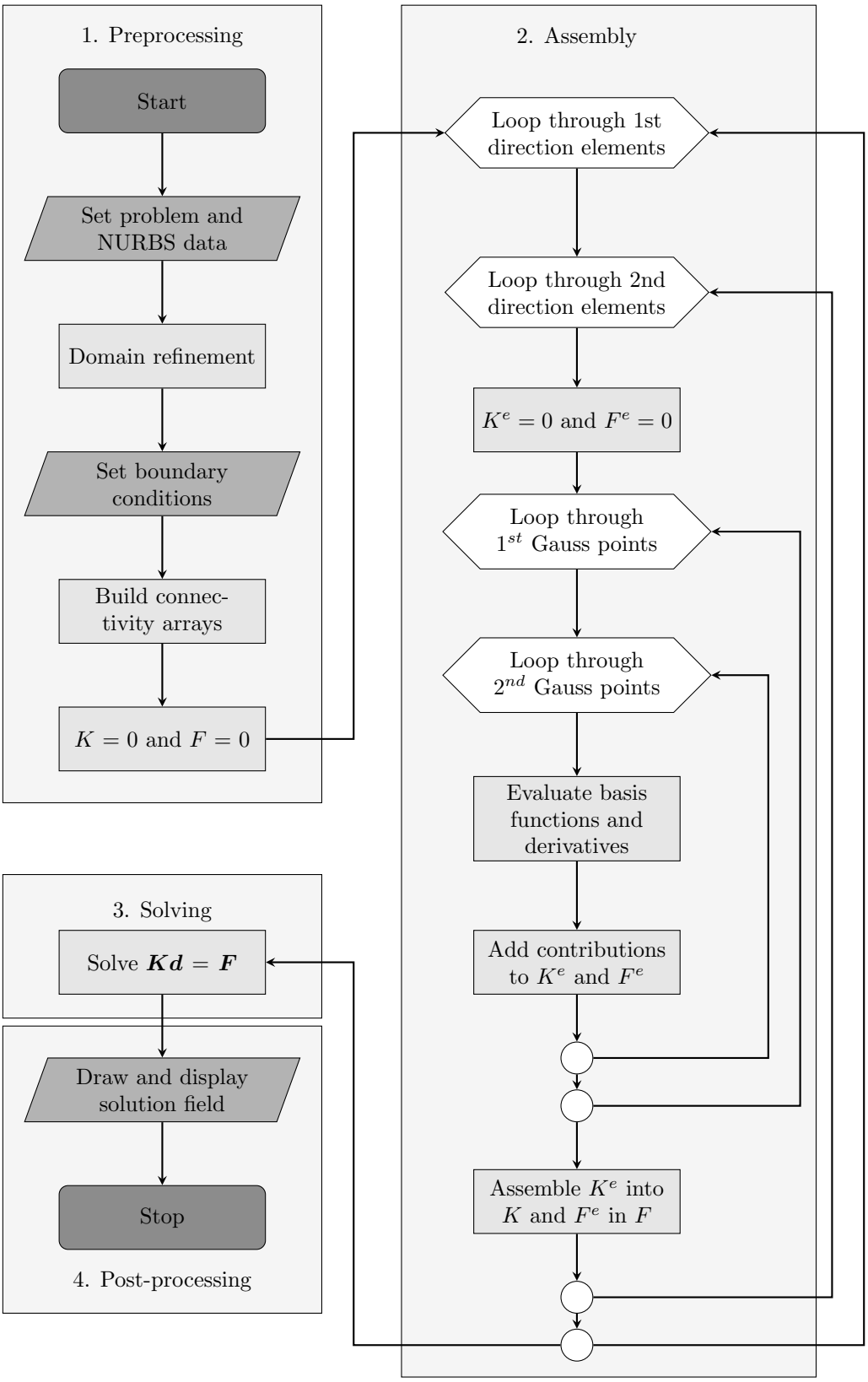


Figure 4.2 Flow chart of a Galerkin two-dimensional IGA process

including the conductance of the domain material, the source term, the domain range and analytical solution to the problem.

Next, the spatial NURBS variables are set in each direction, including the control points and weights, knot vectors, basis function orders and consequently, the number of control points and basis functions. The control points and weights are defined in a matrix of points (which can be two- or three-dimensional points) corresponding to where they lie in the parametric rectilinear grid. This matrix can be converted into a vector of points, `Pvec`, which make the points easier to work with in certain cases.

For this particular problem, the temperature changes more in the radial direction than in the circumferential direction and therefore the initial domain is set up such that there are more elements in the ξ-direction, than the η-direction. Therefore, for an initial domain the knot vectors are $\Xi = [0, 0, 0, 1, 1, 1]$ and $\mathscr{H} = [0, 0, 0, 0.5, 1, 1, 1]$ while the orders of the basis functions are $p = 2$ and $q = 2$.

The next step is to define the connectivity array. This is more complicated than the one-dimensional case where the elements were all next to each other. For the labeling convention in two dimensions, the elements and control points (nodes) are numbered from left to right, bottom to top as seen in Figure 4.3 below.

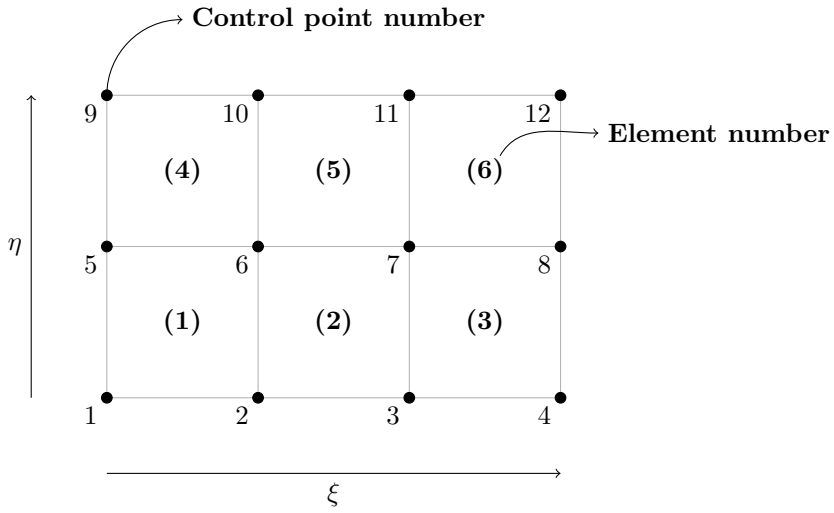


Figure 4.3 Diagram illustrating control point and element labeling conventions for a $p = 1$ mesh with six elements.

Each line in the connectivity array (`NODE`) represents the control point/node numbers associated with an element, labeled from left to right, bottom to top. For example, the first

line associated with the mesh shown in Figure 4.3, if $p = 1$ would be $\texttt{NODE}(1,:) = [1, 2, 5, 6]$. Similarly to the one-dimensional case, the number of control points associated with each element is equal to $(p+1) \times (q+1)$. If $p = 2$, the mesh would be as shown in Figure 4.4.

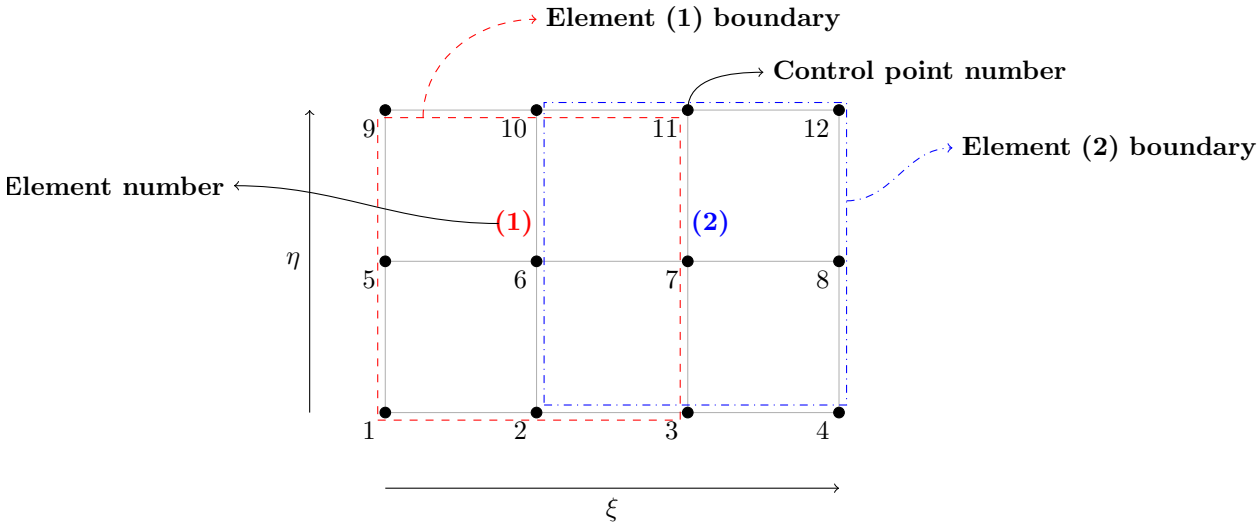


Figure 4.4 Diagram illustrating control point and element labeling conventions for a $p = 2$ mesh with two elements. The control point contributions to the relevant element are enclosed in the relevant boxes.

The first entry for the connectivity array for the mesh in Figure 4.4 would be $\texttt{NODE}(1,:) = [1, 2, 3, 5, 6, 7, 9, 10, 11]$. In each elemental loop of the stiffness matrix and force vector construction, the corresponding line from the connectivity array is selected.

After the connectivity array is established, the boundary conditions are set in accordance to the labeling conventions and the variables introduced on page 54. As in the one-dimensional case, the control point numbers that impose the Dirichlet boundary condition are placed in a vector `n_E` with the control points values that would impose a homogeneous Dirichlet boundary condition in a different vector `d_E`, corresponding to the positions of `n_E`. The remaining free DOFs are all stored in vector `n_F`. The Neumann nodes will not be set because this problem has a homogeneous Neumann boundary condition. The contributions to the force vector will be zero and therefore will not be calculated in this problem, but will be dealt with in the next two-dimensional scalar code in Section 4.2.

The main section of the code consists of constructing the stiffness matrix and force vector. This is achieved by running two nested loops, one for each direction of elements and building the stiffness matrix and force vector elementally. At each element in these loops, Gauss

quadrature is performed to calculate the elemental stiffness and force contributions from equation (4.19) which is assembled into the global stiffness and force matrices. Algorithm 6 describes this construction process.

Algorithm 6 Procedure for evaluating stiffness matrix and force vector for two-dimensional scalar problems

Initialize $\boldsymbol{K}$ and $\boldsymbol{f}$
`el=0`
for `elM=1:el_totalM` **do**
  for `elN=1:el_totalN` **do**
    `el=el+1`
    Initialize $\boldsymbol{K}^e$ and $\boldsymbol{f}^e$
    Define elemental connectivity array, `asm`
    Select elemental control points in each direction,
    `x=Pvec(asm,1)`, `y=Pvec(asm,2)`
    Set i, j to select element domains, `i = elN+p`, `j = elM+q`
    for $gp2 = 1 : \#GaussPoints$ **do**
      for $gp1 = 1 : \#GaussPoints$ **do**
        Calculate parametric coordinates, using equation (3.26)
        Compute shape function arrays at point (ξ, η),
        $\mathbf{R} = [R^{p,q}_{1,1}(\xi,\eta), R^{p,q}_{2,1}(\xi,\eta), \ldots, R^{p}_{p+1}(\xi)q+1]$
        Compute derivative of shape function arrays in each direction,
        $\mathbf{R}_\xi = [R^{p}_{1,\xi}(\xi)R^{q}_{1}(\eta), R^{p}_{2,\xi}(\xi)R^{q}_{1}(\eta), \ldots, R^{p}_{p+1,\xi}(\xi)R^{q}_{q+1}(\eta)]$
        $\mathbf{R}_\eta = [R^{p}_{1}(\xi)R^{q}_{1,\eta}(\eta), R^{p}_{2}(\xi)R^{q}_{1,\eta}(\eta), \ldots, R^{p}_{p+1}(\xi)R^{q}_{q+1,\eta}(\eta)]$
        Compute parametric Jacobian, $J_\xi = ||\mathbf{R}_{\boldsymbol{\xi}}\boldsymbol{x}||$
        Compute isoparametric Jacobian, $J_{\bar{\xi}} = \frac{1}{2}(\xi_{i+1} - \xi_i)(\eta_{j+1} - \eta_j)$
        Compute shape function derivative array, $\mathbf{B} = \mathbf{R}_{\boldsymbol{x}} = R^T_{\boldsymbol{\xi}} J^{-1}_{\xi}$
        Compute $(\bar{\xi}, \bar{\eta})$ in physical domain, (x_{source}, y_{source})
        Compute $\boldsymbol{K}^e$, $\boldsymbol{K}^e = \boldsymbol{K}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} \mathbf{B}^T \times \boldsymbol{D} \times \mathbf{B}$
        Compute $\boldsymbol{f}^e$, $\boldsymbol{f}^e = \boldsymbol{f}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} s(x_{source}, y_{source}) \times \mathbf{R}^T$
      end for
    end for
    Assemble $\boldsymbol{K}_{\texttt{asm asm}} = \boldsymbol{K}_{\texttt{asm asm}} + \boldsymbol{K}^e$
    Assemble $\boldsymbol{f}_{\texttt{asm}} = \boldsymbol{f}_{\texttt{asm}} + \boldsymbol{f}^e$
  end for
end for

Note: w_{gp1}, w_{gp2} are the Gauss point weights, not the NURBS weights.

The first key feature to note of Algorithm 6, is that the ξ-direction loop is within the η-direction loop and not visa versa. This is because of the element numbering conventions as seen in Figure 4.3. The same applies to the Gauss quadrature loops.

The shape function and the shape function derivative arrays are notably different to the one-dimensional case and will be elaborated on. The shape function array is a vector of bivariate shape functions which comprise of a product of the two univariate shape functions associated with each direction. Each univariate shape function number depends on which column in the ξ-direction and which row, in the η-direction the relevant control point stands. The NURBS weights that are to be used in calculating each of the univariate shape functions depend on which row the associated control point is in the ξ-direction and which column the associated control point is in the η-direction. The univariate shape function numbering will be illustrated in Figure 4.5 below, with a 4×3 control point mesh that has p=q=2 and is therefore made of a 2×1 elements.

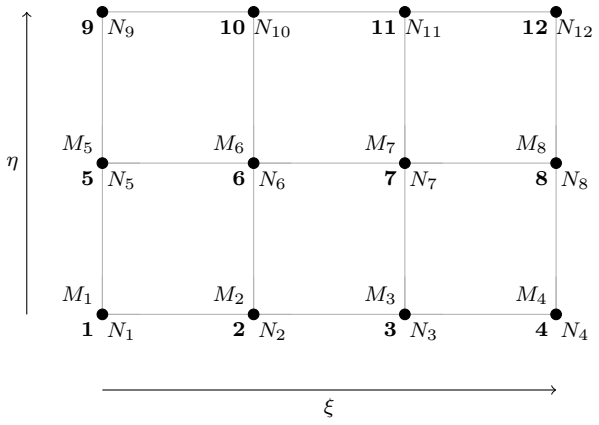


Figure 4.5 Diagram showing a 4×3 mesh of control points and the univariate shape functions for each point.

The control points of a $p = 2$, 2×1 element mesh can be written as

$$\boldsymbol{P} = \begin{bmatrix} P_1 & P_4 & P_7 & P_{10} \\ P_2 & P_5 & P_8 & P_{11} \\ P_3 & P_6 & P_9 & P_{12} \end{bmatrix} \tag{4.20}$$

with the weights matrix defined as

$$\boldsymbol{w} = \begin{bmatrix} w_1 & w_4 & w_7 & w_{10} \\ w_2 & w_5 & w_8 & w_{11} \\ w_3 & w_6 & w_9 & w_{12} \end{bmatrix} \quad . \tag{4.21}$$

The univariate shape function vectors for each direction, N_I, M_I, with I running from one to $(p+1)\times(q+1)$, have $(p+1)\times(q+1)$ entries, the same number of control points in an element. Each entry is made of the univariate shape function associated with that control point and the shape function number of that particular entry in that row/column. For example, the univariate shape function array for the ξ-direction of the second element of the mesh above is

$$\boldsymbol{N} = [R_2^p(\xi), R_3^p(\xi), R_4^p(\xi), R_2^p(\xi), R_3^p(\xi), R_4^p(\xi), R_2^p(\xi), R_3^p(\xi), R_4^p(\xi)] \tag{4.22}$$

where each of the shape functions above corresponds to the NURBS weights $\boldsymbol{w}$ in the same row as the entry. Therefore, N_1, N_2 and N_3 use the weights corresponding to the first row from the bottom, namely w_1, w_2, w_3 and w_4. Entries N_4, N_5 and N_6 use the weights corresponding to the second row from the bottom, namely w_5, w_6, w_7 and w_8. Lastly, the entries N_7, N_8 and N_9 use the weights corresponding to the third row from the bottom, namely w_9, w_{10}, w_{11} and w_{12}.

Similarly, the univariate shape function array for the η-direction for the second element in the mesh above is

$$\boldsymbol{M} = [R_1^q(\eta), R_2^q(\eta), R_3^q(\eta), R_1^q(\eta), R_2^q(\eta), R_3^q(\eta), R_1^q(\eta), R_2^q(\eta), R_3^q(\eta)] \tag{4.23}$$

where again, different NURBS weights correspond to different shape function entries. This time, the associated weights are the weights in the same column as the given entry. Therefore, for the second element, the weights associated with entries M_1, M_4 and M_7 correspond to the weights in the second column of the mesh, namely w_2, w_6 and w_{10}. The weights associated with entries M_2, M_5 and M_8 correspond to the weights in the third column of the mesh, namely w_3, w_7 and w_{11}. The weights associated with entries M_3, M_6 and M_9 correspond to the weights in the last column of the mesh, namely w_4, w_8 and w_{12}.

If all the weights were equal for all control points or even for every row or column, it would not be required to calculate each new univariate shape function for each direction, for each new row or column. In other words, $\boldsymbol{N}$ would only have three distinct entries corresponding to each column. $\boldsymbol{M}$ would also only have three distinct entries corresponding to each new row. But for meshes with varying weights at each control points, each univariate shape function needs to be calculated to combine into the bivariate shape function array. The bivariate shape function array can now be constructed as

$$\mathsf{R}_I = R_{i,j}^{p,q}(\xi,\eta) = N_I M_I \quad . \tag{4.24}$$

For example, the second element of a $p = q = 2$; 4×3 mesh as shown in Figure 4.5 above, will have a shape function array

$$\mathbf{R} = \left[N_2M_2, N_3M_3, N_4M_4, N_6M_6, N_7M_7, N_8M_8, N_{10}M_{10}, N_{11}M_{11}, N_{12}M_{12}\right] \tag{4.25}$$

which could similarly be expressed as

$$\begin{aligned}\mathbf{R} = \big[&R^{p,q}_{2,1}(\xi,\eta), R^{p,q}_{3,1}(\xi,\eta), R^{p,q}_{4,1}(\xi,\eta), R^{p,q}_{2,2}(\xi,\eta), R^{p,q}_{3,2}(\xi,\eta), R^{p,q}_{4,2}(\xi,\eta),\\ &R^{p,q}_{2,3}(\xi,\eta), R^{p,q}_{3,3}(\xi,\eta), R^{p,q}_{4,3}(\xi,\eta)\big]\end{aligned} \tag{4.26}$$

with the corresponding NURBS weights being used for each shape function.

The derivatives of the shape function array in each direction are

$$\mathsf{R}_{I,\xi} = N_{I,\xi}M_J \tag{4.27}$$

$$\mathsf{R}_{I,\eta} = N_I M_{J,\eta} \quad . \tag{4.28}$$

The parametric Jacobian that scales the parametric domain to the physical domain is defined by equation (3.28) and can be calculated using equation (3.29) as

$$J_{\boldsymbol{\xi}} = \begin{bmatrix} \mathbf{R}_{,\xi}x & \mathbf{R}_{,\eta}x \\ \mathbf{R}_{,\xi}y & \mathbf{R}_{,\eta}y \end{bmatrix} \tag{4.29}$$

where x and y are the coordinates of the control points corresponding to the relevant element as defined in line 7 of Algorithm 6. The isoparametric Jacobian mapping the isoparametric domain to the parametric domain is defined by equation (3.28).

The shape function derivative array, $\mathbf{B}$, can now be computed using $J_{\boldsymbol{\xi}}$ and the derivatives of the shape functions $\mathbf{R}_{,\xi}$ and $\mathbf{R}_{,\eta}$ and is defined as seen in equation (4.17) for a two-dimensional scalar problem. It can be computed by

$$\mathbf{B} = \begin{bmatrix} \mathbf{R}_{,\xi} \\ \mathbf{R}_{,\xi} \end{bmatrix}^T J_{\boldsymbol{\xi}}^{-1} \quad . \tag{4.30}$$

Finally, before the elemental stiffness matrices and force vectors can be calculated, the Gauss points need to mapped into the physical domain using equation (2.12) to calculate the source term which is a function of the physical space, $s(x, y)$. Equation (4.19) can now be assembled using the assembly procedure in Algorithm 6.

Now that the assembly is complete and the stiffness matrix, $\boldsymbol{K}$, and force vector, $\boldsymbol{f}$, exist, the boundary conditions can be applied using the partition method in equation (3.33), after which the unknown temperature control points can be solved for. $\boldsymbol{d}_E$ and $\boldsymbol{d}_F$ are then assembled into $\boldsymbol{d}$ as the control points for the temperature distribution across the domain.

In the post-processing stage, the spatial and temperature NURBS vectors are calculated using equation (2.18) and the results are plotted on graphs presented in the next section. The flux is also calculated across the domain using the definition

$$\boldsymbol{q}(x, y) = \mathbf{B}\boldsymbol{d} \tag{4.31}$$

where, in this case $\mathbf{B}$ is not calculated elementally, but rather across the whole domain. In other words, $\mathbf{B}$ is a $2 \times (n * m)$ array, with the first row corresponding to x, the second row corresponding to y and each column corresponding to a different control point. The flux could be calculated elementally, where the element in which each point occurs is determined and then the elemental $\mathbf{B}$ matrix is calculated. In this way, because the basis functions that are not associated to the element calculate to zero, the number of calculations can be minimized. However, in order to create a function that is modular and code that is easy to read and understand, all basis functions are calculated in this function.

4.1.5 Results

The two-dimensional temperature problem has the temperature decreasing radially inwards from 0°. It is therefore expected that the problem should be radially symmetric. Figure 4.6 shows the temperature distribution across the domain with contour lines linking points of the same temperature.

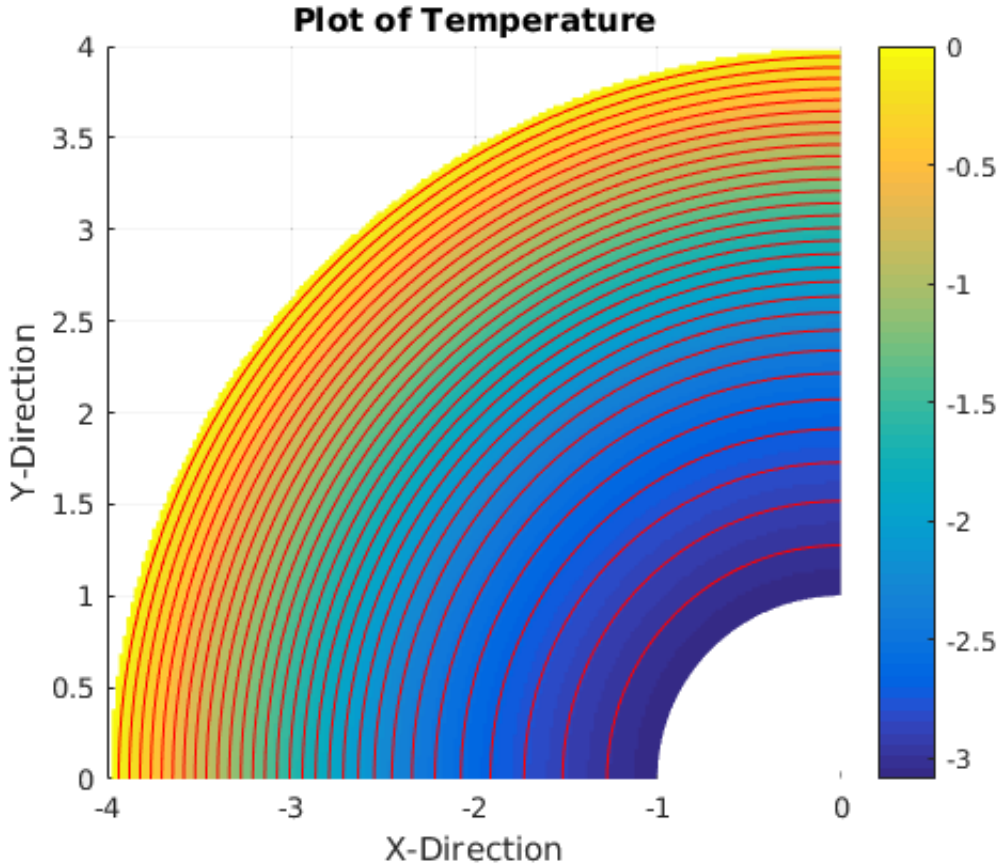


Figure 4.6 Contour plot of radial heat distribution in two dimensions.

Figure 4.7 compares the radial profile along line $x = 0$ to the exact solution defined in equation (3.40). The current IGA solution field is generated with a 2×1 element mesh with second order shape functions in both directions. The maximum error is under 2%. A 2×1 traditional finite element mesh with second order shape functions will immediately not be able to achieve the same order of accuracy, because the base geometry cannot be captured.

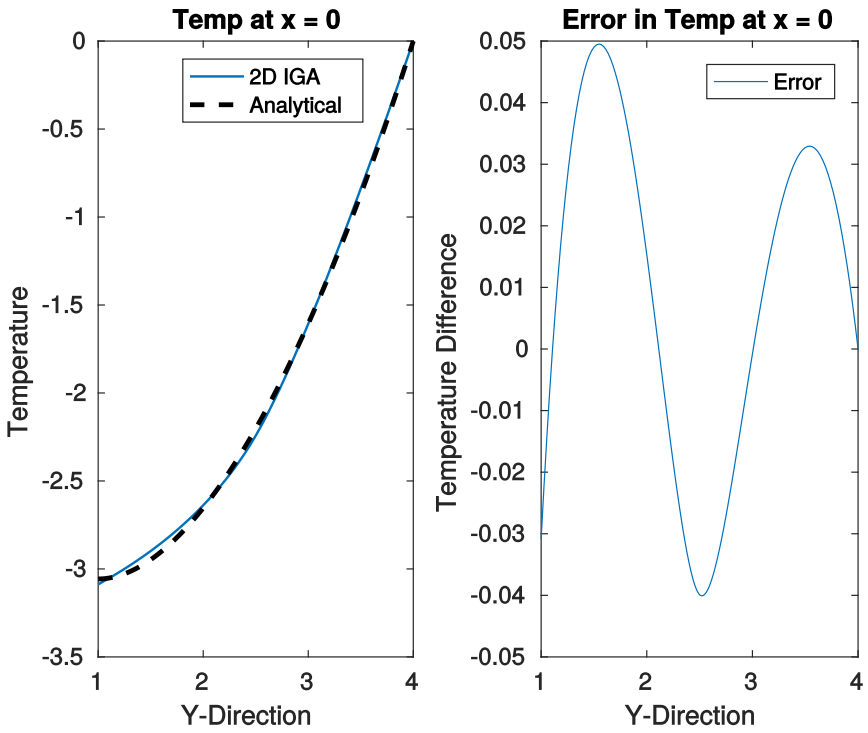


Figure 4.7 Comparison between exact solution and the associated error.

The mesh was globally h-refined once, with twice as many elements in each direction. The resulting error is an order of magnitude less that for the unrefined mesh, as is shown in Figure 4.8.

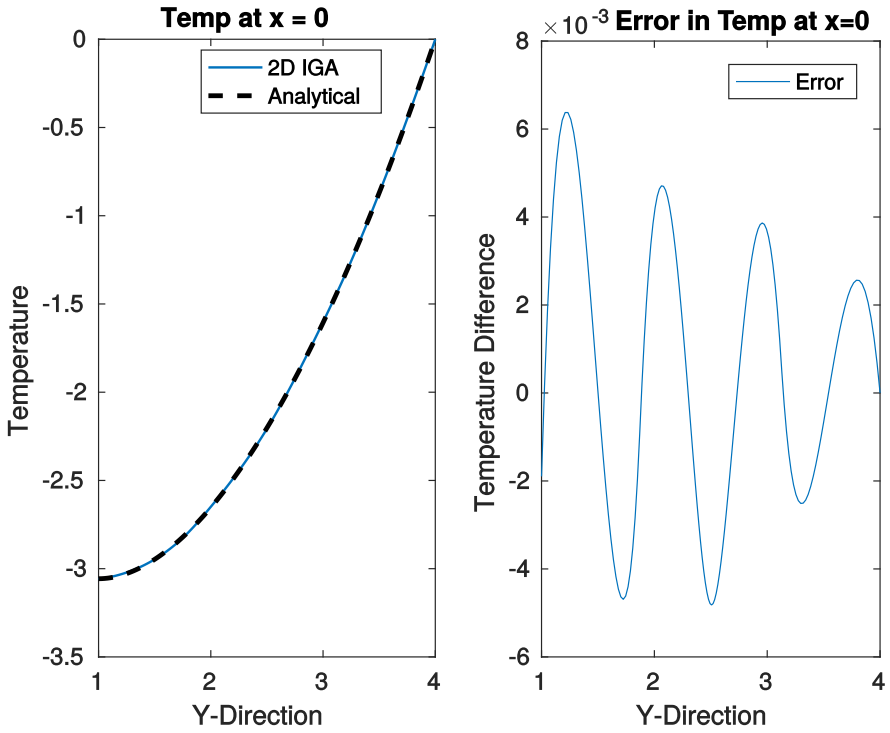


Figure 4.8 Comparison between exact solution and the associated error for one level of h-refinement.

The unrefined mesh is now p-refined so that both p=q=3. Again, the error is an order of magnitude smaller than that for the unrefined mesh. Figure 4.9 shows the error associated with p-refinement to obtain basis functions of order three.

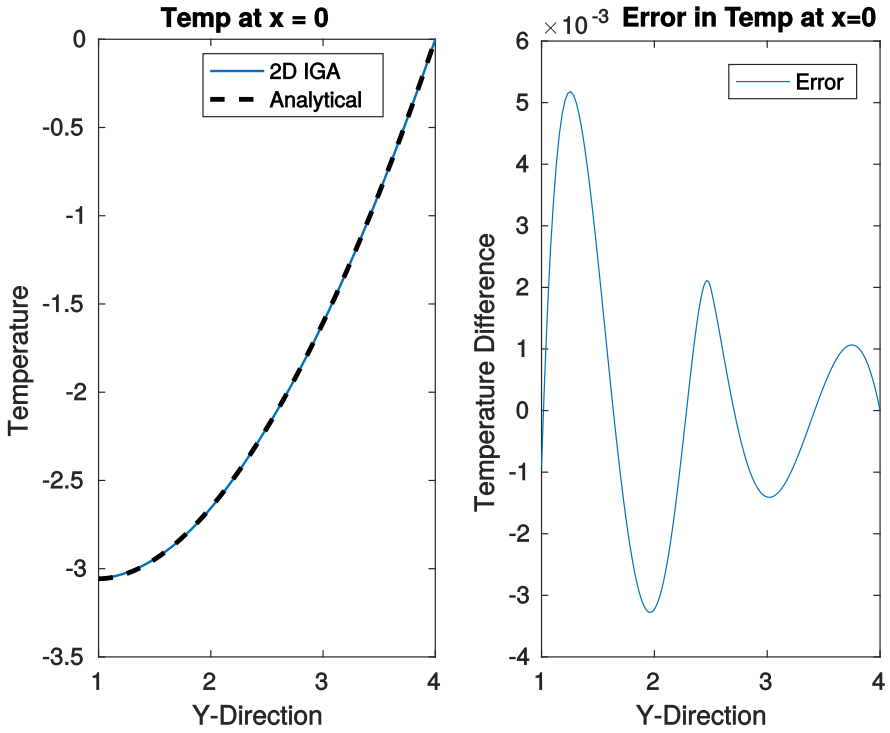


Figure 4.9 Comparison between exact solution and the associated error for p-refinement to $p, q = 3$.

k-refinement to obtain basis functions of order three in both directions of the unrefined mesh also gives smaller error than the original mesh, but has more of an error than the previous two refinements. In this case, the error is roughly 30% of the original error. This is still a significant improvement over the unrefined mesh and is less computationally expensive than both p-refinement and the first level of h-refinement, however, benefits of additional accuracy need to weighed up against computational expense.

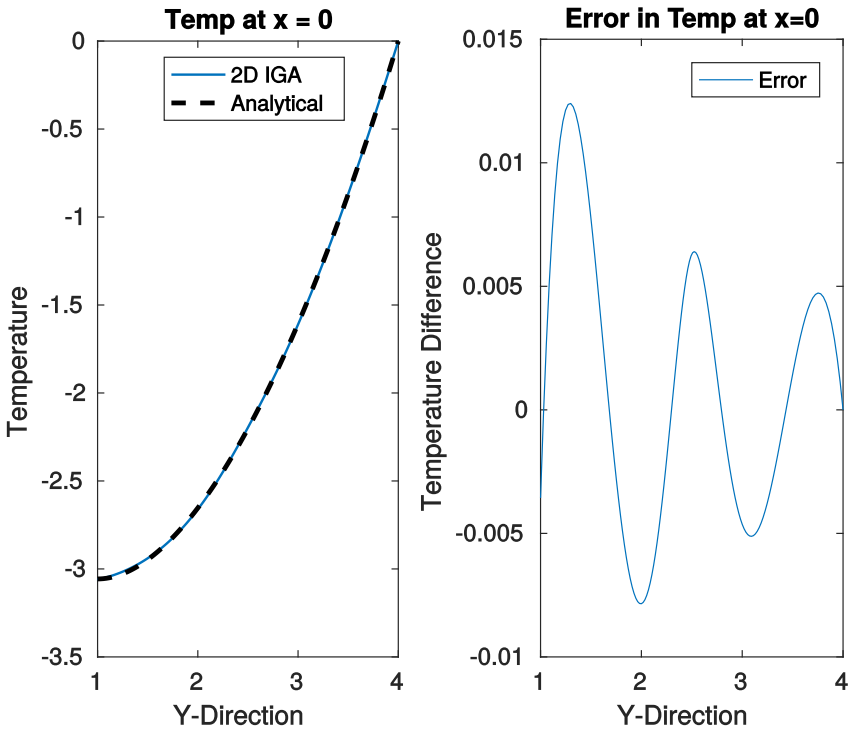


Figure 4.10 Comparison between exact solution and the associated error for k-refinement to $p = q = 3$.

The flux plot is the derivative of the temperature distribution and can be calculated by using equation (4.31). The plot can also serve as an additional source of confirmation that the solution is being captured correctly. See Figure 4.11 below.

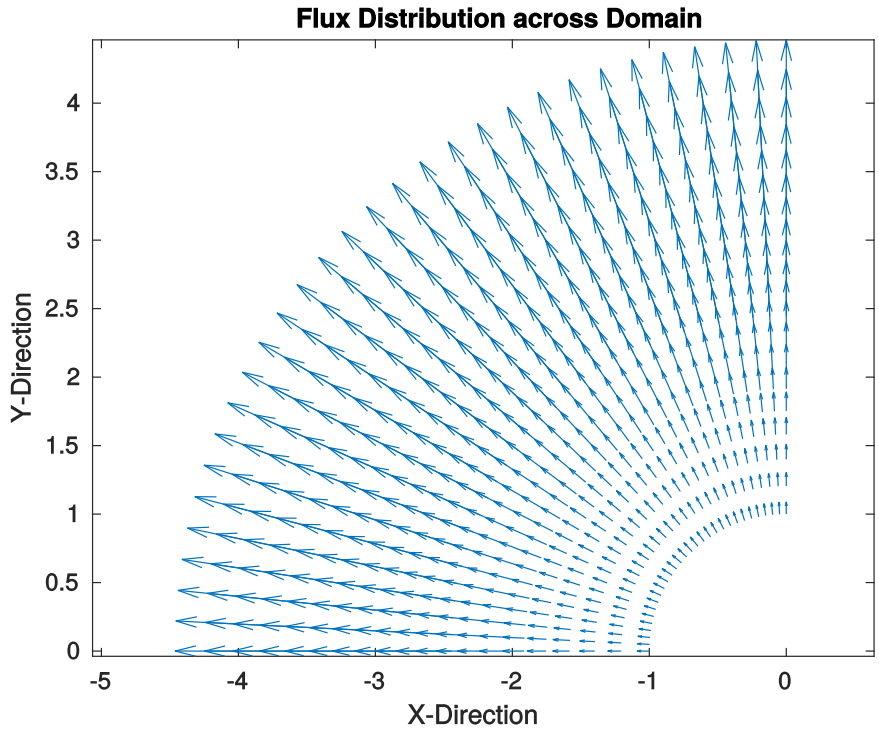


Figure 4.11 The flux across the domain at sampled NURBS points.

Figure 4.12 below shows the L2 error graphs for each refinement type on a log scale. The L2 error was calculated by comparing the IGA solution to the exact solution with the equation [7]

$$||e_{L2}|| = \left(\int_{\Omega} \left(u_{ex}(x,y) - u_h(x,y) \right)^2 d\Omega \right)^{\frac{1}{2}} \quad , \tag{4.32}$$

where $u_{ex}(x,y)$ is the exact solution and $u_h(x,y)$ is the IGA solution.

It can be seen on the first graph that the error in a log scale is decreasing linearly. This indicates that it is still converging. However, after a certain amount of refinement, the error could be deemed sufficiently small for the use of the results.

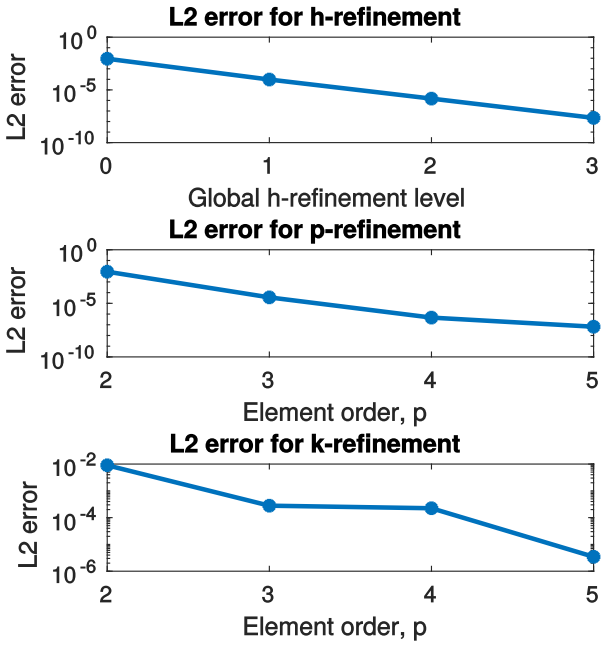


Figure 4.12 L2 error of h, p and k-refinement.

4.2 Temperature: Hole in Plate

The next two-dimensional scalar problem considered is also a heat conduction problem, as in Section 4.1. With this problem, the domain is more complex and different boundary conditions will be applied. In this example, a Neumann boundary condition implementation is shown. The strong form, weak form and Bubnov-Galerkin form are the same as those used in equations (3.35), (4.6) and (4.19).

The strong form of the problem is once again (see equation (3.35))

$$\boldsymbol{\nabla^T q} - s(x,y) = \mathbf{0}$$

where

$$\begin{aligned} \boldsymbol{q} &= -\boldsymbol{D}\boldsymbol{\nabla T} \\ \boldsymbol{D} &= k\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad . \end{aligned} \tag{4.33}$$

Functions $T(x,y)$ and $\nu(x,y)$, as well as their first derivatives are square integrable. Furthermore, the solution and the arbitrary test functions are assumed to satisfy the Dirichlet boundary conditions. The space of functions satisfying these conditions is denoted by V_0 and V, where r_i=1: that is,

$$V_0 = \{\nu \mid \int_\Omega \nu(x,y)]^2 \, d\Omega < \infty, \quad \int_\Omega [\nu'(x,y)]^2 \, d\Omega < \infty, \nu(\Gamma_D) = 0\} \tag{4.34}$$

$$V = \{u \mid \int_\Omega u(x,y)]^2 \, d\Omega < \infty, \quad \int_\Omega [u'(x,y)]^2 \, d\Omega < \infty, u(\Gamma_D) = \bar{u}\}\,, \tag{4.35}$$

where Γ_D is the Dirichlet boundary and $\bar{u}$ is the specified boundary conditions.

The weak form of the problem, equation (4.6), where $\nu(x,y) \in V_0$, $T(x,y) \in V$ and $\nu = 0$ on Γ_D, is

$$\int_\Omega (\boldsymbol{\nabla}\nu)^T \boldsymbol{D}\boldsymbol{\nabla} T(x,y) d\Omega = \int_\Omega \nu \boldsymbol{s}(x,y) d\Omega - \int_\Gamma \nu \bar{\boldsymbol{q}} d\Gamma \quad \forall \nu \in V_0$$

and the Bubnov-Galerkin form of the equation as seen in equation (4.6) is

$$\underbrace{\int_\Omega \mathbf{B}^T \boldsymbol{D}\mathbf{B} d\Omega}_{\boldsymbol{K}} \boldsymbol{d} = \underbrace{\int_\Omega \mathbf{R}^T \boldsymbol{s}(x,y) d\Omega - \int_\Gamma \mathbf{R}^T \bar{\boldsymbol{q}} d\Gamma}_{\boldsymbol{f}} \quad \forall \nu \quad .$$

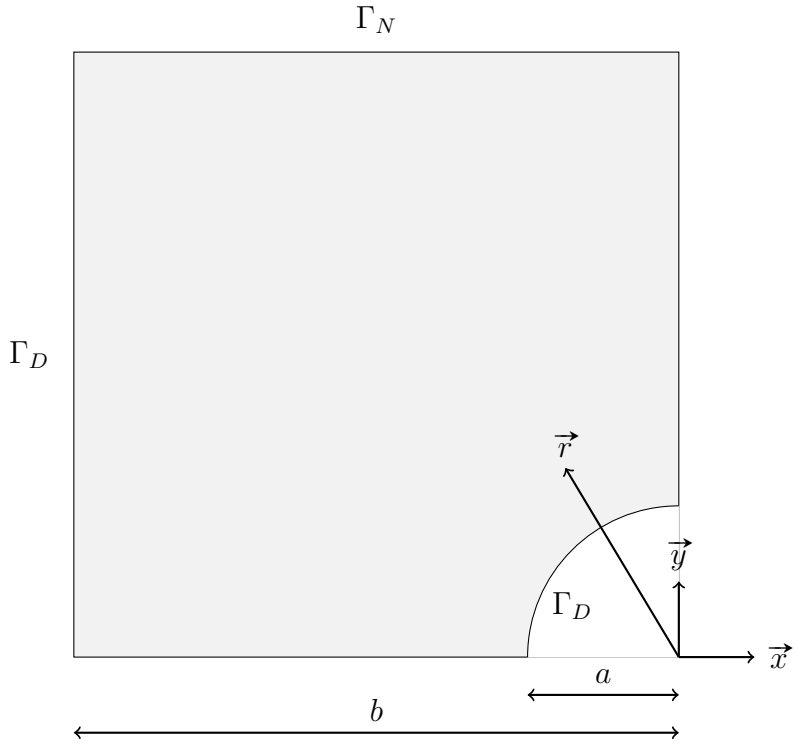


Figure 4.13 Diagram of two-dimensional temperature problem.

The problem is taken from [7, p. 205] and quarter symmetry is used. The problem is created from the manufactured solution

$$T_{exact} = (r-a)^2 = x^2 + y^2 - 2a\sqrt{x^2+y^2} + a^2 \tag{4.36}$$

defined over the domain shown above with

$$a = 1 \tag{4.37}$$

$$b = 4 \quad . \tag{4.38}$$

The plate is isotropically conductive with a heat coefficient of $k = 1W/m^2/K$. The source term varies radially outward from the center of the plate as

$$s(x,y) = \nabla^2 T = 2a\frac{1}{\sqrt{x^2+y^2}} - 4 \quad . \tag{4.39}$$

The problem has both Dirichlet and Neumann boundary conditions. The Dirichlet boundary conditions, Γ_D are

$$T(r = a) = 0 \tag{4.40}$$

$$T(x = -b, y) = a^2 + b^2 + y^2 - 2a\sqrt{y^2 + b^2} \tag{4.41}$$

while the Neumann boundary conditions, Γ_N are

$$\bar{\boldsymbol{q}}(y = b) = -k\boldsymbol{n}^T \nabla T = -k\frac{dT}{dy} = 2kb\left(\frac{2a}{\sqrt{x^2 + y^2}} - 1\right) \tag{4.42}$$

where $\boldsymbol{n}$ is the outward facing normal of the surface on which the flux is prescribed.

The NURBS properties of this domain are described in Section 2.5.2.2. The control point mesh is shown in Figure 4.14.

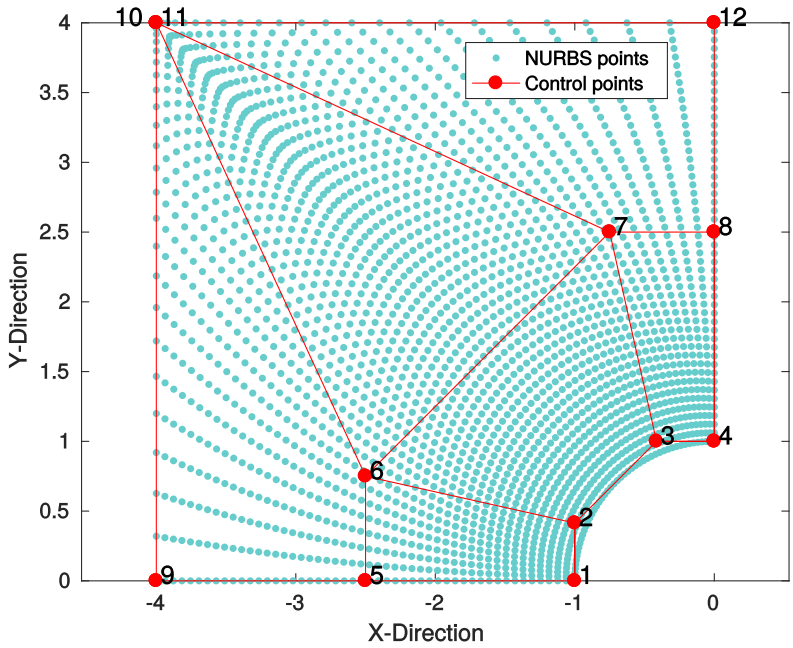


Figure 4.14 NURBS surface with control points in physical domain

4.2.1 Implementation

The heat conduction problem over the domain of a plate with a centrally located hole is implemented in a similar way to the two-dimensional scalar problem in Section 4.1.4, however, there are several added complexities in this problem. First, there is a Neumann (flux) boundary condition that is applied on the top edge of the plate and needs to be implemented in the code. Secondly, the Dirichlet boundary condition on the left edge of the plate is inhomogeneous. The description of geometry requires two control points that coincide at the top-left corner, which adds complexity in refinement as mentioned in Section 2.5.2.2 and in applying both the Neumann and Dirichlet boundary conditions.

Inhomogeneous Dirichlet boundary conditions are boundary conditions that have a prescribed temperature, that is not merely equal to zero. Normally, having a inhomogeneous Dirichlet boundary condition would be estimated by calculating the relevant control point values with the polynomial approximation, as demonstrated in Section 2.5.1.4. However, because the left and top edges of the domain both make up one edge in the parametric domain and the relevant basis functions are continuous about this corner, the control points along the top edge contribute to the Dirichlet distribution along the left edge. This gives an inaccurate boundary solution.

The Dirichlet boundary conditions will therefore have to be imposed by simply calculating the temperature value for each control point using the Dirichlet boundary condition function in equation (4.41). This does not give as accurate results, but with a few levels of refinement, the error is reduced. See [4, p. 30] and Section 6.1.5 on page 120 which will describes how to apply inhomogeneous Dirichlet boundary conditions for a different domain.

The Neumann boundary condition is implemented within the elemental loops, after the elemental stiffness matrix and source contribution to the force vector has been constructed. The Neumann conditions of a temperature problem are in the form of a known flux and the contributions are added to the force vector for each of the element boundaries at which a flux condition occurs. However, depending on which physical and parametric edge the Neumann boundary condition occurs, the code implementation will change. The procedure of calculating the flux contributions follows the one-dimensional assembly along an edge in Section 3.1.5 after defining on which the edge the flux occurs. The remainder of this section will focus on the application of Neumann boundary conditions to different parametric edges. The current heat conduction problem for a plate with a centrally located hole, will be used as an example to illustrate how to apply these boundary conditions. One level of global

h-refinement is applied. The boundary conditions of the problem are illustrated on the physical domain below and labeled accordingly.

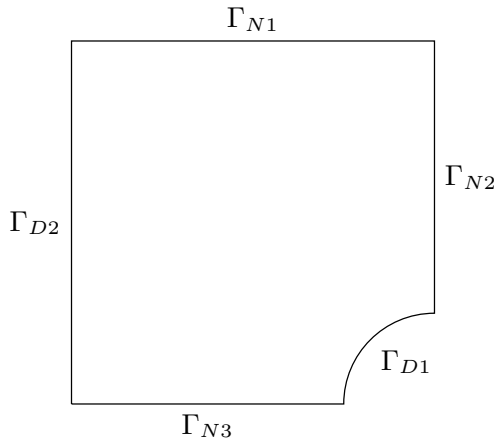


Figure 4.15 Physical domain including boundary conditions

The first step to calculate the flux contributions of the problem is to determine which Neumann boundary conditions are relevant. This domain represents a quarter of the actual problem due to symmetry and therefore the flux on the bottom and right edges (Γ_{N2} and Γ_{N3}) are zero. Hence, they do not need to be added as contributions to the force vector. That leaves boundary condition Γ_{N1} to be considered. It is useful to represent the mesh in the parametric domain and partition the mesh by the knot vector elements as in Figure 4.16. It is also useful to draw the control point mesh as a rectilinear grid to visualize the contributions of control points to the relevant edges as in Figure 4.17. Both diagrams prove useful in visualizing how the Neumann condition is applied.

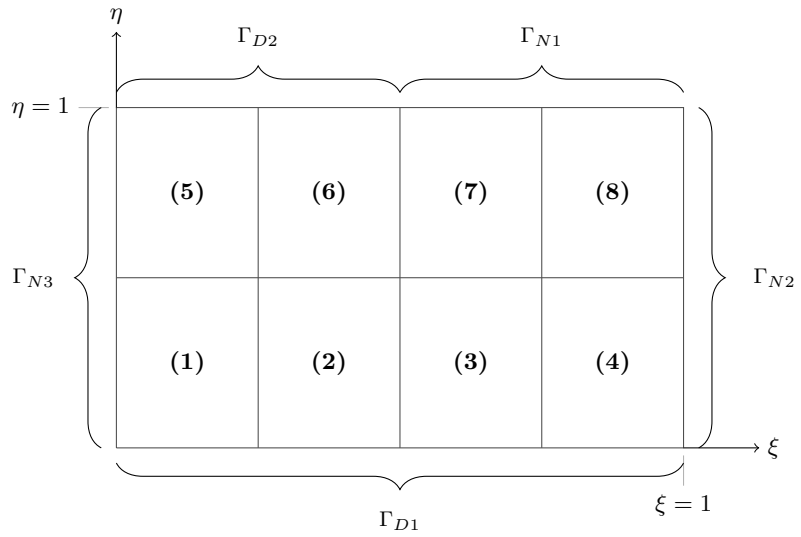


Figure 4.16 Mesh in parametric domain with boundary conditions

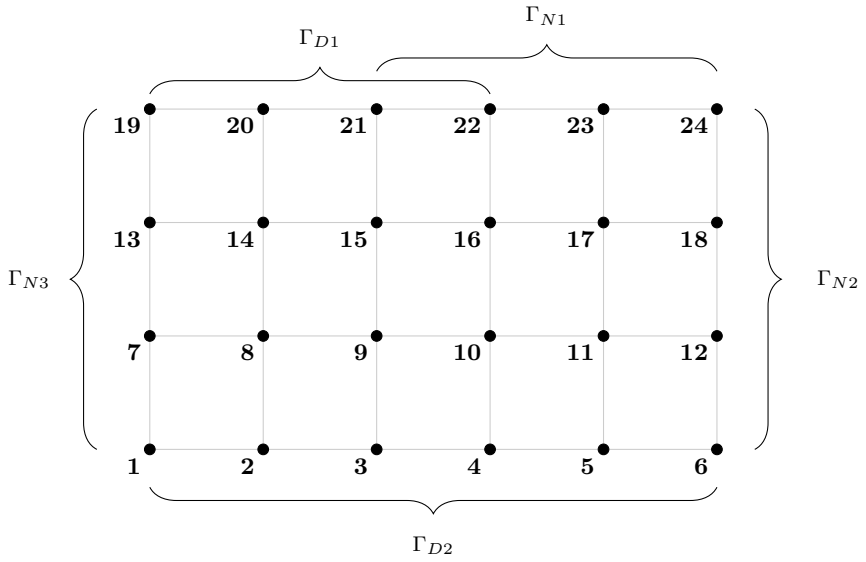


Figure 4.17 Mesh of control points associated with boundary conditions

Now that the edge that contains the relevant flux condition has been identified, it needs to be represented in vectors in the boundary condition section of the code. There are three key variables to describe how the flux is implemented in the code. `el_flux` indicates in which elements the edge with flux exist. For this case, it can be seen that Γ_{N1} affects elements seven and eight and therefore `el_flux`$= [7, 8]$. `n_flux` identifies the control points associated with the Neumann edge and is used to assemble the flux contributions into the force vector. Figure 4.17 shows that the relevant nodes for this case will be

stored as `n_flux`= $[21, 22, 23, 24]$. The last variable that is used to define how the flux is implemented, is `n_flux_side`. This variable determines the elemental nodes associated with the elemental side on which the prescribed flux is defined. For example, if the flux corresponds to the top edge of the parametric domain, the elemental nodes associated with the top of the element will be placed in `n_flux_side`. For the case of a p=q=2 element, the associated elemental nodes are labeled as seen in Figure 4.18 as `n_flux_side`= $[7, 8, 9]$.

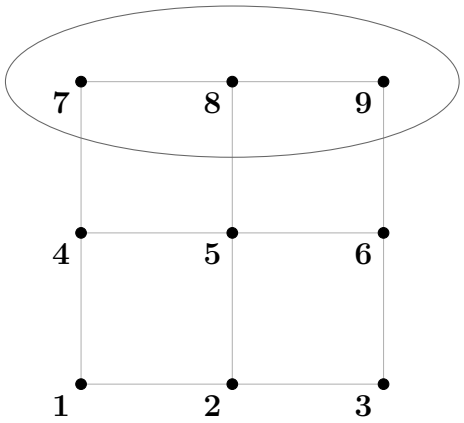


Figure 4.18 Elemental mesh of control points

If the flux was on the right hand side of the element, the variable would be `n_flux_side`= $[3, 6, 9]$ and so forth for the other two edges. The size of this vector also determines the size (`sz`) of the shape function vector that is used create the flux contribution of the elemental force vector. If the flux occurs on the bottom or top edges, the shape function vector will be $p + 1$ long. If the flux occurs on the left or right edges, the shape function vector will be $q + 1$ long.

The variables above are defined in the preprocessing part of the code. The next step in the flux assembly procedure is to build the correct shape function and shape function derivative arrays in the elemental loop that correspond to the correct parametric edge. If the flux occurs on the bottom or top of the parametric domain, all the shape functions and shape function derivatives will be associated to the ξ-direction. If the flux were to be applied on the left or right edges of the parametric domain, the η-direction shape functions and shape function derivatives would be used. This includes changing the shape function selector, `I`, which would start from either `elN` or `elM` respectively. The weights that are used to construct the shape functions need to correspond to the parametric edge on which the flux occurs. Using the mesh in Figure 4.17, the weights associated with flux along the top edge would be w_{19} to w_{24}. The parametric points at which the shape function and

shape function derivatives get evaluated need to be chosen with care. Because the integral to calculate the flux contributions will occur along a edge, one of the parametric variables will stay constant and the other variable will vary, depending on the transformation of the Gauss quadrature points. In this example, because the flux is along the top edge of the parametric domain, the parametric points can be calculated as

$$\begin{aligned}\xi &= \frac{1}{2}[(\xi_{i+1} - \xi_i)\bar{\xi} + (\xi_{i+1} + \xi_i)] \\ \eta &= 1 \quad .\end{aligned}$$

If the flux were to occur on the edges $\xi = 0$, $\xi = 1$ or $\eta = 0$, they would be set accordingly, with the opposite variable varying corresponding to the Gauss quadrature points. These parametric points are transformed into the physical domain to be used in calculating the value of the applied flux at each point, $\boldsymbol{q}(x, y)$.

Once the shape function and shape function derivative arrays have been constructed, the parametric and isoparametric Jacobians need to be calculated. Recall from line 11 in Algorithm 4 that the parametric Jacobian is constructed using the shape function derivative array and the elemental control points along the respective edge. In creating the Jacobian, the correct x or y components of the respective control points need to be selected, which depends on the Neumann edge in the physical domain. In this case, the Neumann boundary condition is along the edge that is constant in the y-direction, and therefore the x-component of the control points need to be used in creating the parametric Jacobian.

The isoparametric Jacobian is calculated based on equation (3.29), while also depending along which parametric edge the flux occurs. If the flux occurs along the ξ-direction, equation (3.29) is used. If the flux occurs in the η-direction, the equation

$$J_{\bar{\eta}} = \frac{1}{2}(\eta_{j+1} - \eta_j) \tag{4.43}$$

is used. Using the implementation details discussed above, Algorithm 7 describes the assembly procedure to include the flux contributions of the current problem into the force vector. The elemental stiffness and force contributions are the same as in Algorithm 6 and details are omitted for the sake of brevity.

Once the stiffness matrix and force vector are constructed, the boundary conditions can be applied, using the partition method in equation (3.32). Where the flux force contributions are applied at Dirichlet control points, the Neumann conditions are overwritten by the

Algorithm 7 Assembly of Neumann flux boundary conditions into the force vector for two-dimensional scalar problems

Initialize $\boldsymbol{K}$ and $\boldsymbol{f}$
Initialize $\boldsymbol{f}^N$
`el=0`
for `elM=1:el_totalM` **do**
  for `elN=1:el_totalN` **do**
    `el=el+1`
    Initialize $\boldsymbol{K}^e$ and $\boldsymbol{f}^e$
    Initialize $\boldsymbol{f}^e_N$
    Define elemental connectivity array, `asm`
    Set i, j to select element domains, `i = elN+p`, `j = elM+q`
    for $gp2 = 1 : \#GaussPoints$ **do**
      for $gp1 = 1 : \#GaussPoints$ **do**
        Compute $\boldsymbol{K}^e$, $\boldsymbol{K}^e = \boldsymbol{K}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} \mathbf{B}^T \boldsymbol{D} \mathbf{B}$
        Compute $\boldsymbol{f}^e$, $\boldsymbol{f}^e = \boldsymbol{f}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} s(x_{source}, y_{source}) \mathbf{R}^T$
      end for
    end for
    Assemble $\boldsymbol{K}_{\texttt{asm asm}} = \boldsymbol{K}_{\texttt{asm asm}} + \boldsymbol{K}^e$
    Assemble $\boldsymbol{f}_{\texttt{asm}} = \boldsymbol{f}_{\texttt{asm}} + \boldsymbol{f}^e$
    if `el=el_flux` **then** ▷ *Only when the current element has a flux contribution.*
      Select relevant elemental control points, `xx=Pvec(asm(n_flux_side),1)`
      for $gp = 1 : \#GaussPoints$ **do**
        Calculate parametric coordinate, ξ using equation (3.26) and $\eta = 1$.
        $I = el_N$
        Compute shape function array at point ξ, $\mathbf{R} = [R^p_I(\xi), R^p_{I+1}(\xi), R^p_{I+2}(\xi)]$
        Compute derivative of $\mathbf{R}$, $\mathbf{R}_\xi = [R^p_{1,\xi}(\xi), R^p_{2,\xi}(\xi), R^p_{3,\xi}(\xi)]$
        Compute parametric Jacobian, $J_\xi = ||\mathbf{R}_\xi \boldsymbol{xx}||$
        Compute isoparametric Jacobian, $J_{\bar{\xi}} = \frac{1}{2}(\xi_{i+1} - \xi_i)$
        Compute $(\bar{\xi}, \bar{\eta})$ in physical domain, (x_{source}, y_{source})
        Compute $\boldsymbol{f}^e_N = \boldsymbol{f}^e_N + J_\xi J_{\bar{\xi}} w_{gp} \bar{q}(x_{source}, y_{source}) \mathbf{R}^T$
      end for
      Assemble $\boldsymbol{f}^N_{\texttt{asm}_{\texttt{n\_flux\_side}}} = \boldsymbol{f}^N_{\texttt{asm}_{\texttt{n\_flux\_side}}} - \boldsymbol{f}^e_N$
    end if
  end for
end for
Assemble $\boldsymbol{f}_{\texttt{n\_flux}} = \boldsymbol{f}_{\texttt{n\_flux}} + F^N_{\texttt{n\_flux}}$

Note: w_{gp1}, w_{gp2} are the Gauss point weights, not the NURBS weights.

control points in `d_E` that ensure that the prescribed Dirichlet condition is enforced. The system of linear equations can now be used to solve for the unknown control points, $\boldsymbol{d}_F$. Along with the known control points $\boldsymbol{d}_E$, they can be assembled into the field variable control point vector $\boldsymbol{d}$. The NURBS surface can now be constructed using the control points, $\boldsymbol{P}$, as well as the temperature values on the corresponding NURBS surface, using the control points, $\boldsymbol{d}$.

4.2.2 Results

The heat conduction problem over a plate with a centrally located hole is a good example of a problem with sufficient control points to represent a complex geometry, but not enough control points to approximate the solution field accurately. Figure 4.19 below shows the contour plot of the temperature solution field. The contour lines make it obvious that the temperature solution field does not increase radially outward. By plotting the difference between the IGA solution and the exact solution as stated in equation (4.36) this error is shown in the "Error Plot" graph in Figure 4.20. The error is greatest on the left and top edges and is less in the middle. The error is calculated as

$$error = T(x, y) - T_{exact}(x, y) \quad . \tag{4.44}$$

Comparing the "IGA Temperature Plot" graph to the "Analytical Temperature Plot" graph, the left and top edges are straight, where they should be curved and increasing exponentially towards the top-left corner. The "Error Plot (%)" graph shows the percentage error relative to the analytical temperature.

$$error\% = \frac{T(x, y) - T_{exact}(x, y)}{T_{exact}(x, y)} \tag{4.45}$$

Because the boundary condition at the edge of the circular hole is zero, the error percentage is higher, because the error comes close to being divided by zero.

After two levels of h-refinement, the IGA results have significantly improved. Where the maximum error difference was 5°C with no refinement, the maximum error is only 0.5°C after two levels of h-refinement which corresponds to a 8×4 element mesh. See Figures 4.21 and 4.22 for the effect of h-refinement on the results across the domain.

Accuracy can also be improved by p-refining the domain so that $p = 4$ and $q = 4$. The elements are kept as a 2×1 mesh, while the shape function orders are raised. Figures

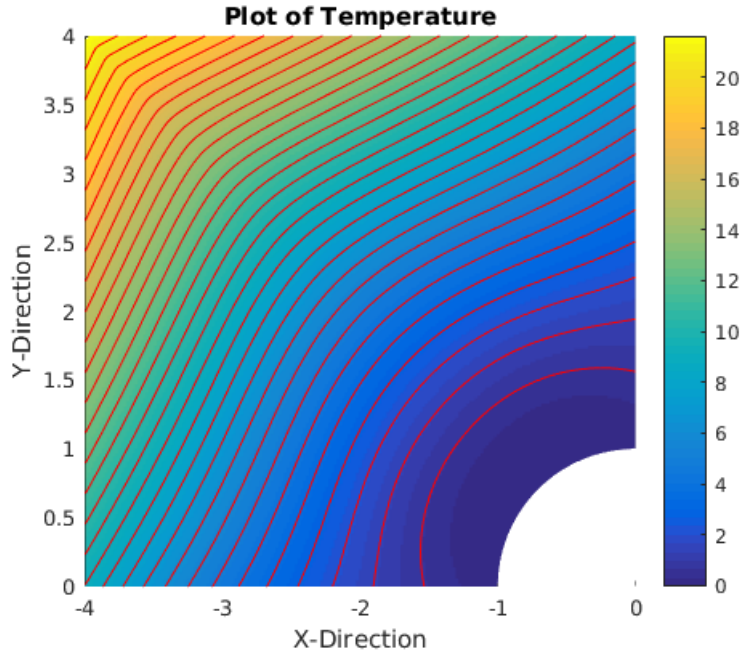


Figure 4.19 Contour plot of temperature distribution.

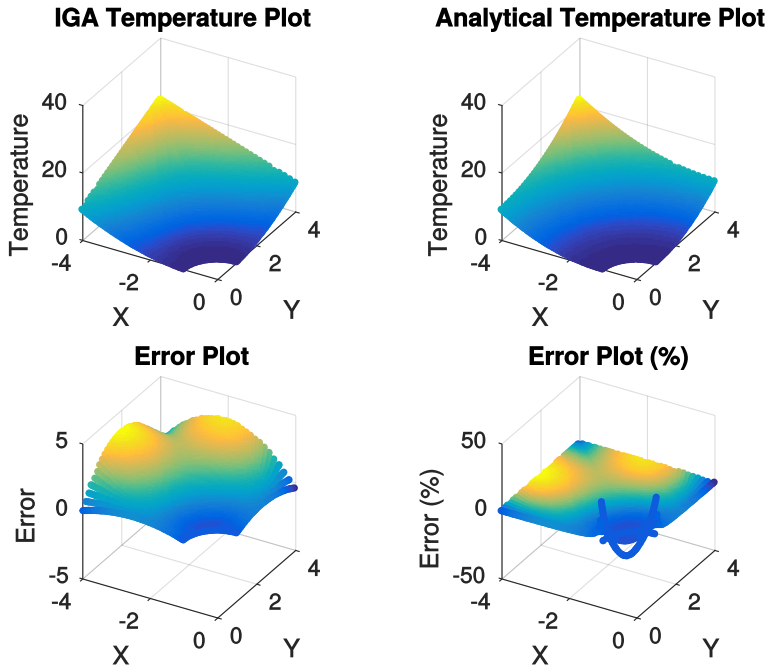


Figure 4.20 Comparison of an unrefined IGA solution to the exact solution.

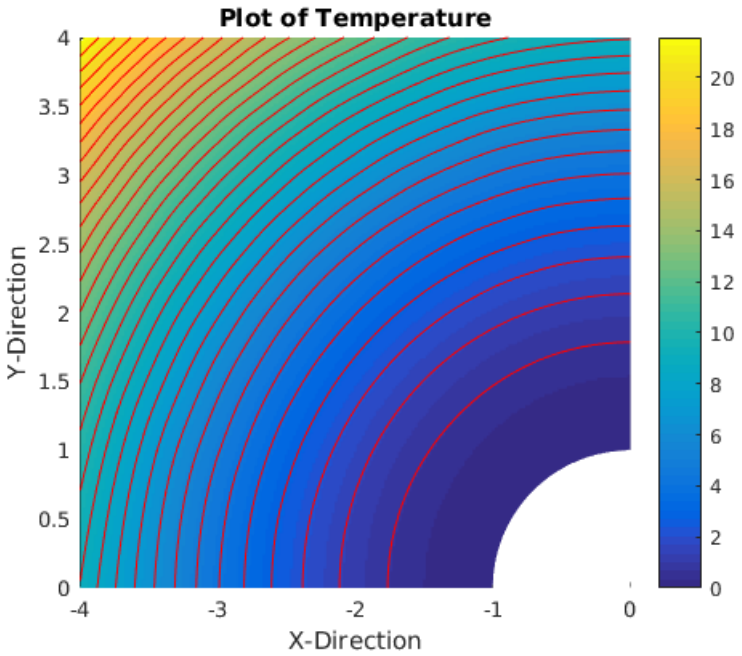


Figure 4.21 Contour plot of temperature distribution, after two levels of h-refinement.

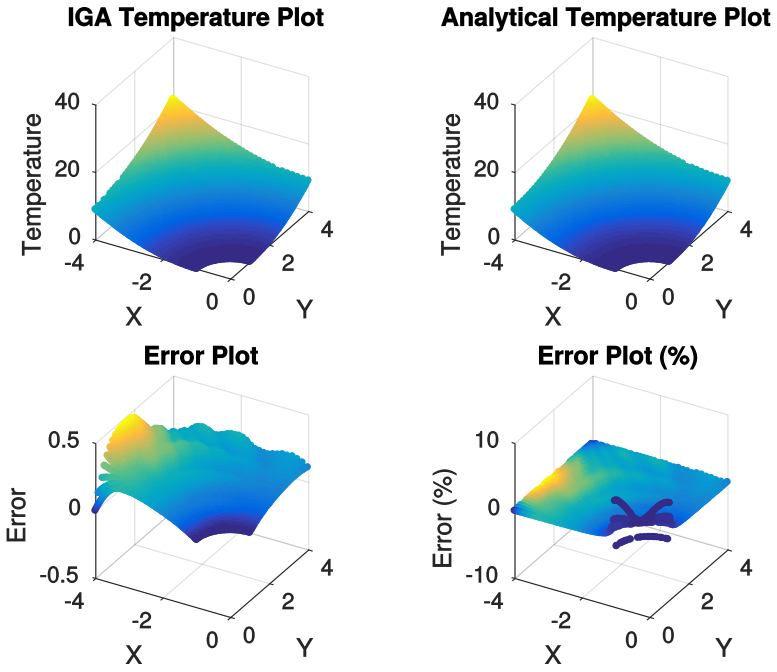


Figure 4.22 Comparison of a IGA solution field to the exact solution, after two levels of h-refinement.

4.23 and 4.24 below show the temperature solution field and error plots associated with p-refinement. The maximum error, again along the Dirichlet boundary is approximately 1.5°C.

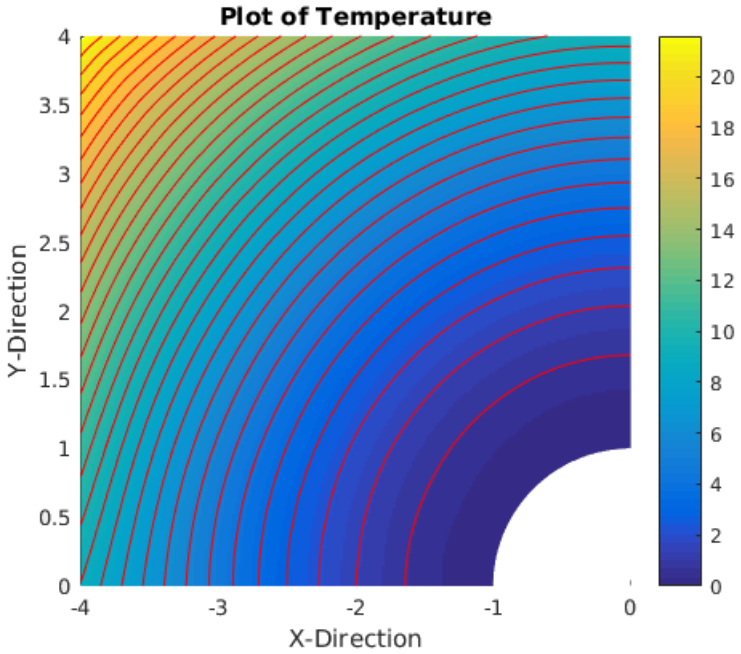


Figure 4.23 Contour plot of temperature distribution, after p-refinement to $p = q = 4$.

Because k-refinement cannot represent the geometry accurately, it is not applied to this problem in IGA, as discussed in Section 2.5.2.2.

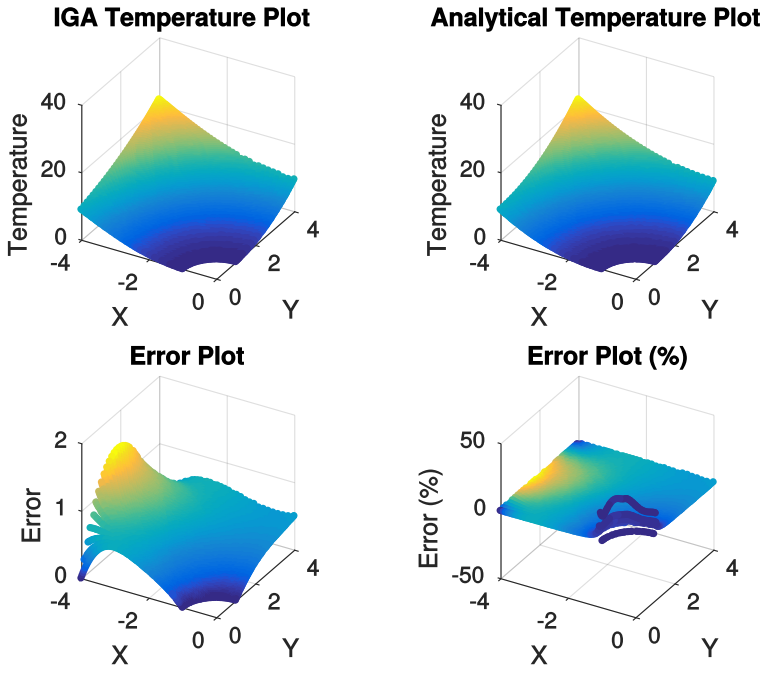


Figure 4.24 Comparison of a IGA solution field to the exact solution, after p-refinement to $p, q = 4$.

Chapter 5

Two Dimensional Vector IGA

5.1 Two Dimensional Vector Solution Process

A two-dimensional vector problem refers to a problem in a two-dimensional physical domain where the field of interest is vector-valued and has two degrees of freedom (DOFs) at each node. Elasticity is a common application of a two-dimensional vector-valued problem where displacement in the x- and y-direction is considered. To demonstrate the procedure for approximating the two-dimensional elastic deformation of a solid, a plate with a centrally located hole domain, as in the previous section, is modeled with quarter symmetry. As with the two-dimensional scalar problems, the strong, weak and Bubnov-Galerkin form will be developed and then approximated numerically.

5.1.1 Strong Form

The strong form of the elasticity equation is

$$\boldsymbol{\nabla}_{\boldsymbol{s}}^{T}\boldsymbol{\sigma} + \boldsymbol{b}(x,y) = \mathbf{0} \tag{5.1}$$

where $\boldsymbol{b}(x,y)$ is the body force across the domain which will be varied to investigate different effects in Section 5.1.5. $\boldsymbol{\nabla}_{\boldsymbol{s}}$ is the symmetric gradient operator defined as

$$\boldsymbol{\nabla}_{\boldsymbol{s}} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{bmatrix} \quad . \tag{5.2}$$

$\boldsymbol{\sigma}$ is the Cauchy stress across the domain

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} \tag{5.3}$$

and can be represented as a function of the elasticity matrix, $\boldsymbol{D}$ and the strain, $\boldsymbol{\varepsilon}$ and can be written as

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\varepsilon} \tag{5.4}$$

and in turn, the strain is

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{bmatrix} \tag{5.5}$$

and can be expressed as the derivative of the infinitesimal displacement, $\boldsymbol{u}$ as

$$\boldsymbol{\varepsilon} = \boldsymbol{\nabla_s u} \quad . \tag{5.6}$$

For this problem, the domain is the same as that in Figure 4.13 with the following properties

$$a = 1m, \quad b = 4m, \quad E_y = 10^7 \ Pa, \quad \tau = 0.3, \quad t = 0.1m$$

where E_Y is the Young's Modulus, τ is the Poisson's ratio and t is the thickness of the plate. For this problem, plane stress is assumed. Therefore

$$\boldsymbol{D} = \frac{E_y}{1-\tau^2} \begin{bmatrix} 1 & \tau & 0 \\ \tau & 1 & 0 \\ 0 & 0 & \frac{1-\tau}{2} \end{bmatrix} \quad . \tag{5.7}$$

There are both traction and Dirichlet boundary conditions for this problem. Because the plate is modeled using quarter symmetry, the Dirichlet boundary conditions of the problem are

$$\boldsymbol{u_x}(x = 0, y) = \boldsymbol{0} \tag{5.8}$$

$$\boldsymbol{u_y}(x, y = 0) = \boldsymbol{0} \quad . \tag{5.9}$$

The traction is applied on the left edge in the direction of negative x. Therefore

$$\bar{\boldsymbol{t}} = \begin{bmatrix} -10000 \\ 0 \end{bmatrix} N \quad . \tag{5.10}$$

Figure 5.1 shows the domain and applied boundary conditions.

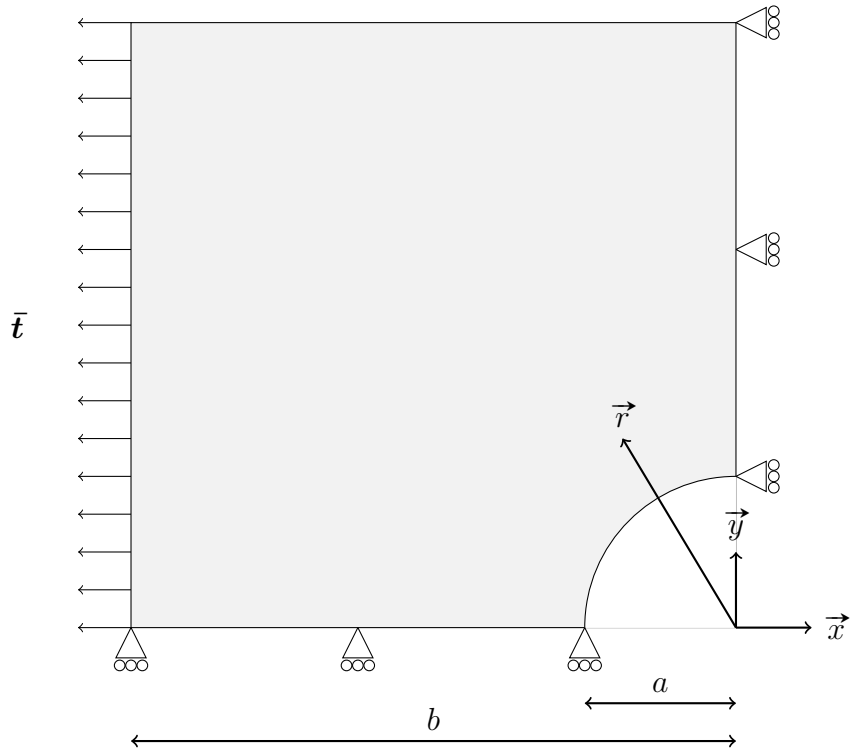


Figure 5.1 Diagram of two-dimensional displacement problem.

5.1.2 Weak Form

The weak form is obtained by multiplying each term by a weighting function, $\boldsymbol{\nu}(x, y)$ and taking the integral across the domain.

$$\int_{\Omega} \boldsymbol{\nu} \boldsymbol{\nabla}_s^T \boldsymbol{\sigma} d\Omega + \int_{\Omega} \boldsymbol{\nu}^T \boldsymbol{b}(x, y) d\Omega = \mathbf{0} \qquad \forall \boldsymbol{\nu} \tag{5.11}$$

Green's theorem is applied to the first term to yield

$$\int_{\Omega} (\boldsymbol{\nabla}_s \boldsymbol{\nu})^T \boldsymbol{\sigma} d\Omega - \int_{\Gamma} \boldsymbol{\nu}^T \boldsymbol{t} d\Gamma = \int_{\Omega} \boldsymbol{\nu}^T \boldsymbol{b}(x, y) d\Omega \qquad \forall \boldsymbol{\nu} \tag{5.12}$$

where $\boldsymbol{t}$ is the traction and is equal is $\boldsymbol{t} = \bar{\boldsymbol{t}}$ on Γ_N. The equation above can be rearranged to form

$$\int_{\Omega} (\boldsymbol{\nabla}_s \boldsymbol{\nu})^T \boldsymbol{\sigma} d\Omega - \int_{\Gamma} \boldsymbol{\nu}^T \bar{\boldsymbol{t}} d\Gamma = \int_{\Omega} \boldsymbol{\nu}^T \boldsymbol{b}(x, y) d\Omega \qquad \forall \boldsymbol{\nu} \quad . \tag{5.13}$$

Functions $\boldsymbol{u}(x, y)$ and $\boldsymbol{\nu}(x, y)$, as well as their first derivatives are square integrable. Furthermore, the solution and the arbitrary test functions are assumed to satisfy the Dirichlet boundary conditions. The space of functions satisfying these conditions is denoted by V_0: that is,

$$V_0 = \{\nu \mid \int_\Omega \nu(x,y)]^2 \, d\Omega < \infty, \quad \int_\Omega [\nu'(x,y)]^2 \, d\Omega < \infty, \nu(\Gamma_D) = 0\} \, . \tag{5.14}$$

By substituting equations (5.4) and (5.6) into the equation above, where $\boldsymbol{u} = \bar{\boldsymbol{u}}$ on Γ_D, $\boldsymbol{\nu}(x, y), \boldsymbol{u}(x, y) \in V_0$ and where $\boldsymbol{\nu}(x, y)$ is arbitrary over the domain, the weak form reads

$$\int_\Omega (\boldsymbol{\nabla}_s \boldsymbol{\nu})^T \boldsymbol{D} \boldsymbol{\nabla}_s \boldsymbol{u} d\Omega - \int_\Gamma \boldsymbol{\nu}^T \bar{\boldsymbol{t}} d\Gamma = \int_\Omega \boldsymbol{\nu}^T \boldsymbol{b}(x, y) d\Omega \qquad \forall \boldsymbol{\nu} \in V_0 \quad . \tag{5.15}$$

5.1.3 Bubnov-Galerkin Form

The components in the weak form are approximated by the following NURBS definitions

$$\boldsymbol{\nu}(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{m} R_i^p(\xi) R_j^q(\eta) \mathbf{v}_{ij} \tag{5.16}$$

$$(\boldsymbol{\nabla}_s \boldsymbol{\nu})^T = \sum_{i=1}^{n} \sum_{j=1}^{m} R_{i,\boldsymbol{x}}^p(\xi) R_{j,\boldsymbol{x}}^q(\eta) \mathbf{v}_{ij} \tag{5.17}$$

$$\boldsymbol{u}(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{m} R_i^p(\xi) R_j^q(\eta) \boldsymbol{d}_{ij} \tag{5.18}$$

$$(\boldsymbol{\nabla}_s \boldsymbol{u})^T = \sum_{i=1}^{n} \sum_{j=1}^{m} R_{i,\boldsymbol{x}}^p(\xi) R_{j,\boldsymbol{x}}^q(\eta) \boldsymbol{d}_{ij} \tag{5.19}$$

where $\boldsymbol{d}$ are the control points of a NURBS surface approximating the displacement field.

Each of these definitions can be expressed using shape function and shape function derivative arrays and corresponding control points in a manner analogous to the two-dimensional scalar case in equations (4.11-4.14).

$$\boldsymbol{\nu} = \mathbf{R}\mathbf{v} = \mathbf{v}^T \mathbf{R}^T \tag{5.20}$$

$$(\boldsymbol{\nabla}_s \boldsymbol{\nu})^T = \mathbf{B}\mathbf{v} = \mathbf{v}^T \mathbf{B}^T \tag{5.21}$$

$$\boldsymbol{u}(x, y) = \mathbf{R}\boldsymbol{d} \tag{5.22}$$

$$(\boldsymbol{\nabla}_s \boldsymbol{u})^T = \mathbf{B}\boldsymbol{d} \tag{5.23}$$

where $\mathbf{R}$ and $\mathbf{B}$ are the shape function and shape function derivative arrays for two-dimensional vector problems respectively. These arrays are made up of the components in the two-dimensional scalar shape function arrays. Therefore superscripts *sca* and *vec* will be used to distinguish between the two-dimensional scalar or vector shape functions respectively. Recall from equation (4.16) that the scalar shape function array, $\mathbf{R}^{sca}$ is

$$\begin{aligned}\mathbf{R}^{sca} = [&R^{p,q}_{1,1}(\xi,\eta), R^{p,q}_{2,1}(\xi,\eta), \ldots, R^{p,q}_{p+1,1}(\xi,\eta), R^{p,q}_{1,2}(\xi,\eta), \ldots, \\ &R^{p,q}_{p+1,2}(\xi,\eta), \ldots, R^{p,q}_{1,q+1}(\xi,\eta), \ldots, R^{p,q}_{p+1,q+1}(\xi,\eta)] \quad .\end{aligned}$$

The vector shape function array, $\mathbf{R}^{vec}$ and derivative shape function array, $\mathbf{B}^{vec}$ can be written as

$$\mathbf{R}^{vec} = \begin{bmatrix} \mathsf{R}^{sca}_{1}, & 0, & \mathsf{R}^{sca}_{2}, & 0, & \mathsf{R}^{sca}_{3}, & 0, & \ldots & \mathsf{R}^{sca}_{(p+1)(q+1)}, & 0 \\ 0, & \mathsf{R}^{sca}_{1}, & 0, & \mathsf{R}^{sca}_{2}, & 0, & \mathsf{R}^{sca}_{3}, & \ldots & 0, & \mathsf{R}^{sca}_{(p+1)(q+1)} \end{bmatrix} \quad . \tag{5.24}$$

and

$$\mathbf{B}^{vec} = \begin{bmatrix} \mathsf{R}^{sca}_{1,x}, & 0, & \mathsf{R}^{sca}_{2,x}, & 0, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),x}, & 0 \\ 0, & \mathsf{R}^{sca}_{1,y}, & 0, & \mathsf{R}^{sca}_{2,y}, & \ldots & 0, & \mathsf{R}^{sca}_{(p+1)*(q+1),y} \\ \mathsf{R}^{sca}_{1,y}, & \mathsf{R}^{sca}_{1,x}, & \mathsf{R}^{sca}_{2,y}, & \mathsf{R}^{sca}_{2,x}, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),y}, & \mathsf{R}^{sca}_{(p+1)*(q+1),x} \end{bmatrix} \quad . \tag{5.25}$$

A summary of the differences between the one-dimensional and the two-dimensional scalar and vector shape function arrays can be found in Appendix A.

By substituting equations (5.20-5.23) into the weak form, equation (5.15), and because $\boldsymbol{\nu}(x,y)$ is arbitrary

$$\int_\Omega \mathbf{B}^T \boldsymbol{D}\mathbf{B}\boldsymbol{d}d\Omega - \int_\Omega \mathbf{R}^T \bar{\boldsymbol{t}} d\Gamma = \int_\Omega \mathbf{R}^T \boldsymbol{b}(x,y) d\Omega \qquad \forall \boldsymbol{\nu} \quad . \tag{5.26}$$

This can be expresses as a system of linear equations by forming

$$\underbrace{\int_\Omega \mathbf{B}^T \boldsymbol{D}\mathbf{B} d\Omega}_{\boldsymbol{K}} \boldsymbol{d} = \underbrace{\int_\Omega \mathbf{R}^T \bar{\boldsymbol{t}} d\Gamma + \int_\Omega \mathbf{R}^T \boldsymbol{b}(x,y) d\Omega}_{\boldsymbol{f}} \qquad \forall \boldsymbol{\nu} \quad . \tag{5.27}$$

5.1.4 Implementation

Two-dimensional vector-valued problems are implemented in an almost identical way to two-dimensional scalar problems as described in Section 4.1.4. It follows the same procedure as described in Figure 4.2. The differences occur in the basis function arrays as

described in the previous section, applying the traction boundary condition and in the connectivity array. There are two connectivity arrays in two-dimensional vector problems; one to link nodes within an element and the other to link the degrees of freedom within an element. For a scalar problem, these arrays are the same, but for a vector-valued problem, there are two degrees of freedom for each node and therefore the connectivity arrays are different. The array connecting nodes within an element is the same as the scalar connectivity array. The DOF connectivity array can be constructed in a similar manner, but includes both DOFs of each node. The DOF naming convention of a two-dimensional vector problem domain is described in Figure 5.2.

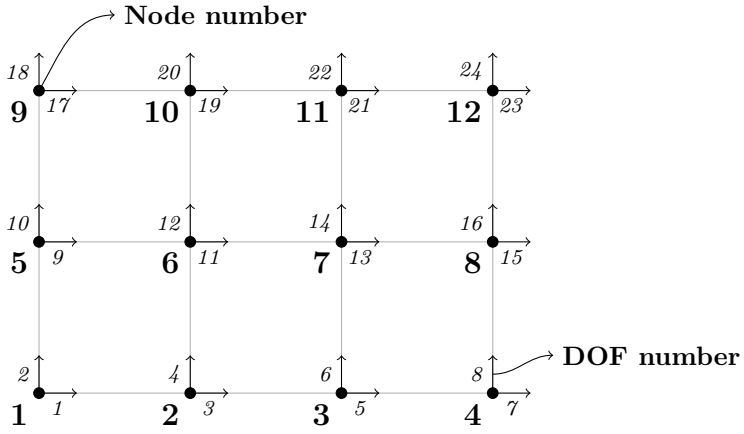


Figure 5.2 Diagram illustrating DOF labeling conventions.

The bigger bold numbers define the nodal numbers and the smaller italic numbers are the degree of freedom numbers. As in the scalar problems, each row of the connectivity arrays defines the nodes associated with a new element, labeled from left to right, bottom to top. The NURBS domain definitions and refinement procedures are the same for the two-dimensional scalar case. The boundary conditions are also labeled in the similar way to the two-dimensional scalar problems, although care should be taken to ensure that the vectors refer to the degree of freedom numbers and not the node numbers. In addition to the essential and free degree of freedom vectors, the traction degree of freedom numbers are defined in a vector `n_traction`. Similar to the the two-dimensional scalar case on page 93, `n_traction` refers to the global degree of freedoms (DOFs) that have traction conditions. `n_traction_side` is the vector that defines the side of the element on which the traction is applied by labeling the elemental nodes of the element associated with that particular side.

Additional to the boundary conditions vectors above, `asmSelect` is the vector that indicates which elemental degree of freedom values have a traction applied to them. This takes into account the direction of the traction. `asmSelect` can be calculated from `n_traction_side`, depending on whether the traction is in the x-or y-direction as

$$\texttt{asmSelect} = 2 \cdot \texttt{n\_traction\_side} - 1 \qquad for\ traction\ in\ x-direction \tag{5.28}$$

$$\texttt{asmSelect} = 2 \cdot \texttt{n\_traction\_side} \qquad for\ traction\ in\ y-direction. \tag{5.29}$$

If the traction was in the x-direction on the left elemental edge, `asmSelect` $= [1, 7, 13]$ and so forth. Figure 5.3 for $p = q = 2$ elemental degree of freedom labeling conventions.

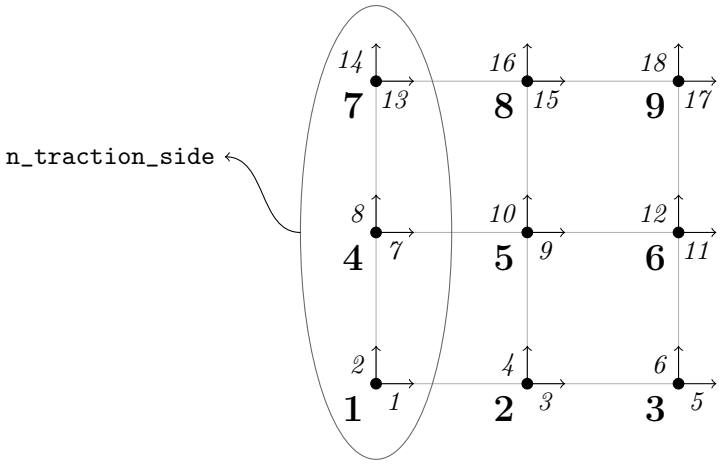


Figure 5.3 Elemental degree of freedom labeling

`el_traction` indicates which elements have a traction applied to an edge in the element. For this problem, all the elements on the left edge are represented in this vector. The assembly procedure for the two-dimensional vector-valued stiffness matrix and force vector are shown in Algorithm 8. For the sake of clarity, the traction contribution which is inserted into line 32, is described in Algorithm 9.

Algorithm 8 Procedure for evaluating stiffness matrix and force vector for two-dimensional vector problems

Initialize $\boldsymbol{K}$, $\boldsymbol{f}$ and $\boldsymbol{f}_{traction}$
`el=0`
for `elM=1:el_totalM` **do**
  for `elN=1:el_totalN` **do**
    `el=el+1`
    Initialize $\boldsymbol{K}^e$, $\boldsymbol{f}^e$ and $\boldsymbol{f}_N^e$
    Define elemental node-connectivity array, `asm` and DOF-connectivity array, `asm2`
    Select elemental control points in each direction,
    `x=Pvec(asm,1)`, `y=Pvec(asm,2)`
    Set i, j to select element domains, `i = elN+p`, `j = elM+q`
    for $gp2 = 1 : \#GaussPoints$ **do**
      for $gp1 = 1 : \#GaussPoints$ **do**
        Calculate parametric coordinates, using equation (3.26)
        Compute shape function arrays at point (ξ, η),
        $\mathbf{R}^{sca} = [R^{p,q}_{1,1}(\xi,\eta), R^{p,q}_{2,1}(\xi,\eta), \ldots, R^{p}_{p+1}(\xi)q+1]$
        Compute derivative of shape function arrays in each direction,
        $\mathbf{R}^{sca}_{\xi} = [R^{p}_{1,\xi}(\xi)R^{q}_{1}(\eta), R^{p}_{2,\xi}(\xi)R^{q}_{1}(\eta), \ldots, R^{p}_{p+1,\xi}(\xi)R^{q}_{q+1}(\eta)]$
        $\mathbf{R}^{sca}_{\eta} = [R^{p}_{1}(\xi)R^{q}_{1,\eta}(\eta), R^{p}_{2}(\xi)R^{q}_{1,\eta}(\eta), \ldots, R^{p}_{p+1}(\xi)R^{q}_{q+1,\eta}(\eta)]$
        Construct $\mathbf{R}^{vec}$ using equation (5.24)
        Compute parametric Jacobian, $J_\xi = ||\mathbf{R}^{sca}_{\boldsymbol{\xi}}\boldsymbol{x}||$
        Compute isoparametric Jacobian, $J_{\bar{\xi}} = \frac{1}{2}(\xi_{i+1} - \xi_i)(\eta_{j+1} - \eta_j)$
        Compute derivative shape function array, $\mathbf{B}^{sca} = \mathbf{R}^{sca}_{\boldsymbol{x}} = J_\xi^{-1} R^T_{\boldsymbol{\xi}}$
        Construct $\mathbf{B}^{vec}$ using equation (5.25)
        $\mathbf{R} = \mathbf{R}^{vec}$, $\mathbf{B} = \mathbf{B}^{vec}$
        Compute $(\bar{\xi}, \bar{\eta})$ in physical domain, (x_{source}, y_{source})
        Compute $\boldsymbol{K}^e$, $\boldsymbol{K}^e = \boldsymbol{K}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} \mathbf{B}^T \times \boldsymbol{D} \times \mathbf{B}$
        Compute $\boldsymbol{f}^e$, $\boldsymbol{f}^e = \boldsymbol{f}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} s(x_{source}, y_{source}) \times \mathbf{R}^T$
      end for
    end for
    Assemble $\boldsymbol{K}_{\texttt{asm asm}} = \boldsymbol{K}_{\texttt{asm asm}} + \boldsymbol{K}^e$
    Assemble $\boldsymbol{f}_{\texttt{asm}} = \boldsymbol{f}_{\texttt{asm}} + \boldsymbol{f}^e$
    Compute the traction contribution, $\boldsymbol{f}^N$
  end for
end for
Assemble $\boldsymbol{f}_{\texttt{n\_traction}} = \boldsymbol{f}_{\texttt{n\_traction}} + F^N_{\texttt{n\_traction}}$

Note: w_{gp1}, w_{gp2} are the Gauss point weights, not the NURBS weights.

The traction contribution to the force vector is shown in Algorithm 9. This algorithm is problem specific, if a different traction boundary were to be applied in either a different direction or on a different edge, many features would change, as explained on page 93. The only additional vector that is different to the two-dimensional scalar case, is variable `asmSelect`. If there is a traction applied in both directions, each contribution is added to the force vector consecutively.

Algorithm 9 Procedure for Neumann traction boundary conditions into the force vector for a two-dimensional vector problem that can be inserted into line 32 of 8.

1: **if** `el=el_traction` **then**
2: ▷ *Only when the current element has a traction contribution.*
3: Select relevant elemental control points, `yy=Pvec(asm(n_flux_side),2)`
4: **for** $gp = 1 : \#GaussPoints$ **do**
5: Calculate parametric coordinate, ξ using equation (3.26) and $\eta = 1$.
6: $I =$ `el_N`
7: Compute shape function array at point ξ, $\mathbf{R} = [R^p_I(\xi), R^p_{I+1}(\xi), R^p_{I+2}(\xi)]$
8: Compute derivative of $\mathbf{R}$, $\mathbf{R}_\xi = [R^p_{1,\xi}(\xi), R^p_{2,\xi}(\xi), R^p_{3,\xi}(\xi)]$
9: Compute parametric Jacobian, $J_\xi = ||\mathbf{R}_\xi \boldsymbol{yy}||$
10: Compute isoparametric Jacobian, $J_{\bar{\xi}} = \frac{1}{2}(\xi_{i+1} - \xi_i)$
11: Compute $(\bar{\xi}, \bar{\eta})$ in physical domain, (x_{source}, y_{source})
12: Compute $\boldsymbol{f}^e_N = \boldsymbol{f}^e_N + J_\xi J_{\bar{\xi}} w_{gp}(x_{source}, y_{source}) \mathbf{R}^T \bar{\boldsymbol{t}}$
13: **end for**
14: Assemble $\boldsymbol{f}^N_{\texttt{asm2}_{\texttt{asmSelect}}} = F^N_{\texttt{asm2}_{\texttt{asmSelect}}} - F^e_{traction}$
15: **end if**

Note: w_{gp1}, w_{gp2} are the Gauss point weights, not the NURBS weights.

The control points in the y-direction are used to compute the parametric Jacobian in this question, because the traction is applied on the left edge, which is constant in the x-direction and changing along the y-direction. Because the traction is applied in the ξ-direction, the size of the traction vector's shape function array is $sz = p + 1$ and the when the Gauss points are converted into the parametric domain, η is set to one, as the traction is always applied along the $\eta = 1$ edge in this problem and the Gauss point in the ξ-direction changes depending on the quadrature point. Because the traction is in the ξ-direction, the shape functions associated with ξ are used, using the weights associated with that parametric edge of control points.

The Bubnov-Galerkin form in equation (5.27) can be represented by a system of linear equations in the form $\boldsymbol{Kd} = \boldsymbol{f}$ and the partition method, demonstrated in equation (3.32)

can be used to apply boundary conditions. However, before the free control points, $\boldsymbol{d}_F$ can be solved for, there is an additional step for this specific problem. Because there exists a pair of control points at the top left corner that must remain coincident, a penalty method is applied in order to tie the degrees of freedom together in both the x- and y-direction together for those two control points. This procedure is done for one direction at a time. For the x degree of freedom, a vector selects the degree of freedom numbers of the two nodes that need to be tied to move together. Referring to the unrefined mesh in Figure 2.39 of a domain of a plate with a centrally located hole, the degree of freedom vector for the x-direction will be $\rho_x = [19, 21]$. A penalty of $\beta = 10^{10}$ is applied to the stiffness matrix as follows

$$K_{\rho_x\ \rho_x} = K_{\rho_x\ \rho_x} + \beta \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad . \tag{5.30}$$

The same is done in the y-direction to degrees of freedom, $\rho_y = [20, 22]$ to lock the y-direction degrees of freedom of the two relevant nodes together.

$$K_{\rho_y\ \rho_y} = K_{\rho_y\ \rho_y} + \beta \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad . \tag{5.31}$$

The stiffness matrix and force vector are now ready to be used to solve for the displacement control points, $\boldsymbol{d}$. In the post-processing stage, the NURBS surface that approximates the displacements is created using $\boldsymbol{d}$ and equation (2.18). The displacement NURBS surface is then added to the original position NURBS surface to give the surface of the final position after displacement.

In order to visualize the stress and strain, they can be calculated during post-processing. The strain can be calculated by multiplying the displacement control points by the shape function derivative array for each point

$$\boldsymbol{\varepsilon}(x, y) = \mathbf{B}(x, y)\boldsymbol{d} \tag{5.32}$$

where $\mathbf{B}$ is calculated for each new point on the NURBS surface. Each entry in the strain vector associated with a new point consists of direct and shear strains. The direct and shear stresses can be calculated by using equation (5.4). These stress components can then be converted into various different stress states. The equations for different stress states are shown in Table 5.1.

Stress State	Equation
Von-Mises Stress	$\sigma_{VM} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 - \sigma_{xx}\sigma_{yy} + 3\sigma_{xy}^2}$
Principle Stress 1	$\sigma_{P1} = \left(\frac{\sigma_{xx}+\sigma_{yy}}{2}\right) + \sqrt{\left(\frac{\sigma_{xx}-\sigma_{yy}}{2}\right)^2 + \sigma_{xy}^2}$
Principle Stress 2	$\sigma_{P2} = \left(\frac{\sigma_{xx}+\sigma_{yy}}{2}\right) - \sqrt{\left(\frac{\sigma_{xx}-\sigma_{yy}}{2}\right)^2 + \sigma_{xy}^2}$

Table 5.1 *Table showing different stress state equations.*

5.1.5 Results

The results of the two-dimensional linear elasticity problem can be plotted in numerous ways. Figure 5.4 shows an exaggerated positional field.

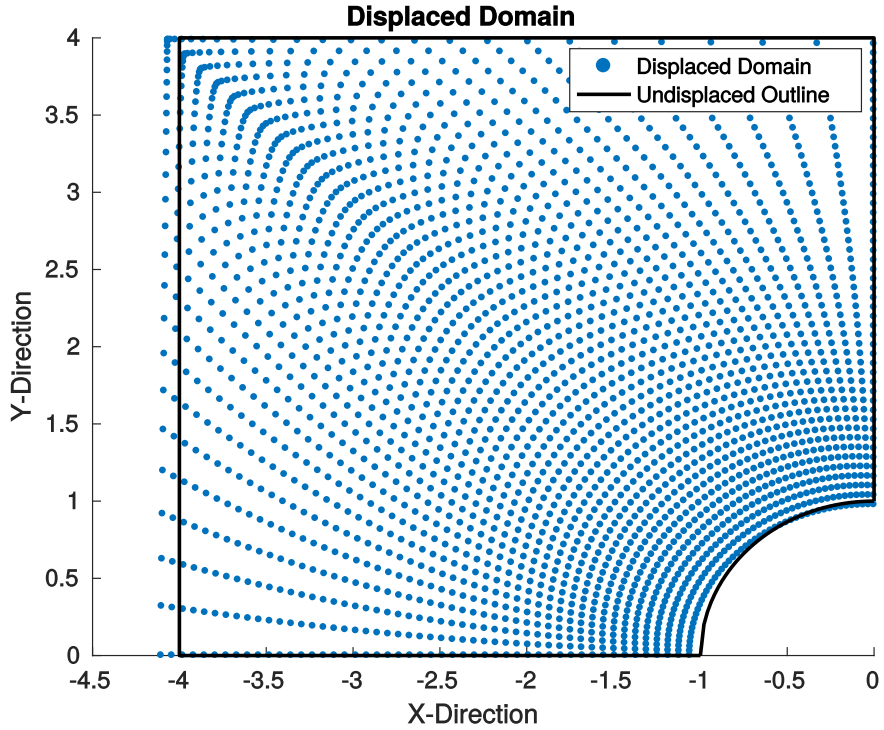


Figure 5.4 Scaled displaced position.

For some analyses, the stress field is of more use than the displacement. Because the force is in the x-direction, the most useful stress plot for this problem would be the σ_{xx} stress plot in Figure 5.5. The stress is plotted on the undeformed domain in this case.

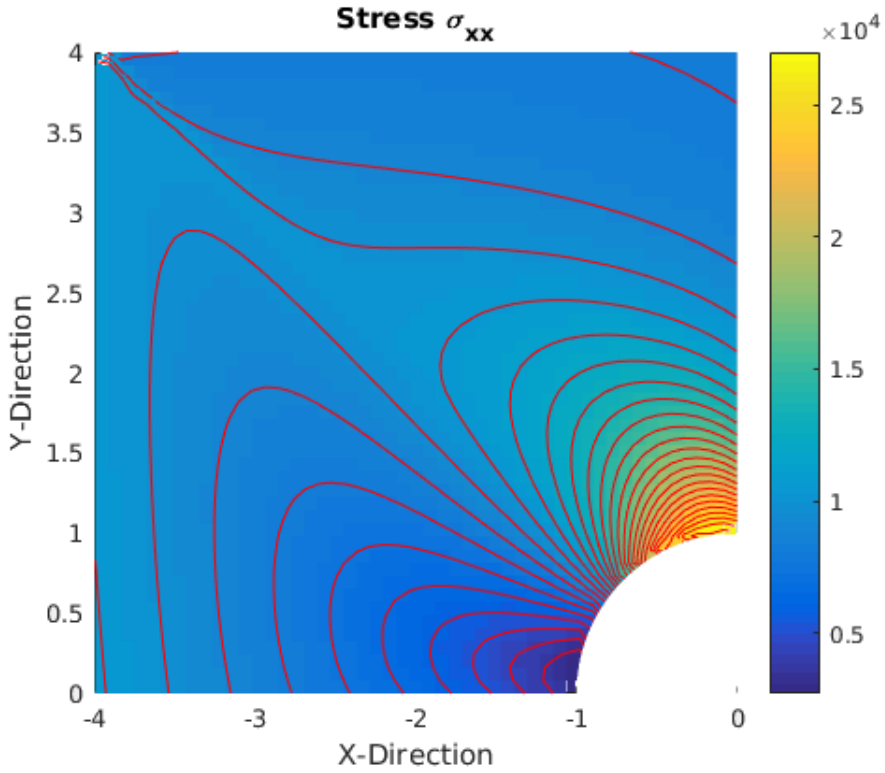


Figure 5.5 Stress, σ_{xx} on an undisplaced domain.

If the stress in the y-direction is plotted, σ_{yy}, the stress is significantly less and is shown in Figure 5.6. It is also possible to plot the von Mises stress of the problem, as well as the stresses in the principal directions, using the the equations in Table 5.1. It can also be useful to plot stress on an displaced domain. Details of this implementation can be found in the `IGA_2D_PlateHoleDisp.m` code in the digital submission of this dissertation.

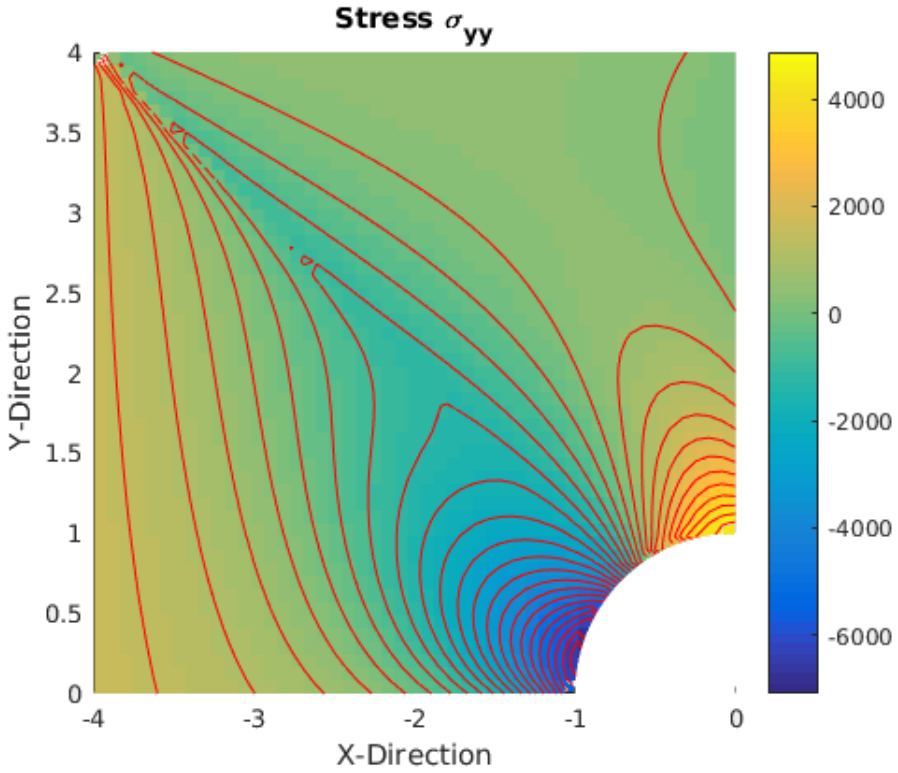


Figure 5.6 Stress, σ_{yy} on an undisplaced domain.

Because this example did not have an analytical solution as with the previous examples, it is not possible to do a L2 error analysis. A different error analysis was therefore performed based of the equation seen in [8, p. 310] as

$$e = \left| \frac{\boldsymbol{d}^T \boldsymbol{R}}{\boldsymbol{d}^T \boldsymbol{f}} \right| \tag{5.33}$$

where $\boldsymbol{R}$ is the residual of the problem as

$$\boldsymbol{R} = \boldsymbol{K}\boldsymbol{d} - \boldsymbol{f} \quad . \tag{5.34}$$

It should be noted that the error above is a measure of the numerical error of the problem and not the overall accuracy, as there is no analytical solution.

It can be seen in Figure 5.7 that for both h-refinement and p-refinement, the error decreases on a log scale. While the error is still converging, it can be seen from the decrease in gradient in both graphs, that it will converge.

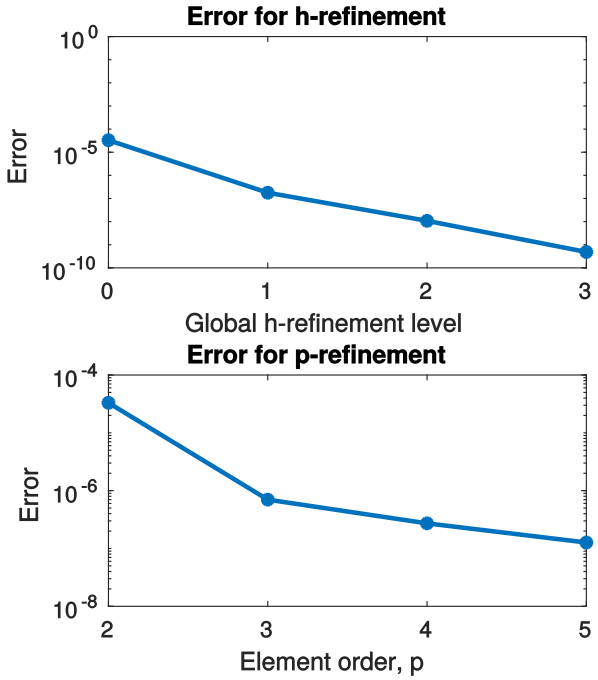


Figure 5.7 L2 error of h, p-refinement.

Chapter 6

Two Dimensional Non-Linear IGA

Isogeometric analysis can be used to solve non-linear problems. The procedure will use the Newton-Raphson Method to approximate a solution to a non-linear problem. To illustrate this procedure, the minimal surface area of a soap bubble whose boundary is a wire frame will be considered. This problem was chosen as the non-linearity comes into account with the geometry and the degree of non-linearity can be varied.

6.1 Minimal Surface of a Soap Bubble

6.1.1 Strong Form

This problem is essentially a two-dimensional scalar problem that needs to be solved iteratively until the solution converges. Minimal surfaces have a mean curvature of zero and there the strong form of this problem is

$$\nabla \cdot \left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}} \right) = 0 \tag{6.1}$$

where u is the height of the soap bubble [9] with $u = \bar{u}$ at the Dirichlet boundary, Γ_D. The bubble is defined over the domain of a unit square as

$$x \in [0, 1] \tag{6.2}$$

$$y \in [0, 1] \tag{6.3}$$

and the boundary conditions of the domain are

$$u(x,y) = A\left(\frac{1}{4} - \left(x - \frac{1}{2}\right)^2\right) \quad \text{on} \quad \Gamma_{y=0} \tag{6.4}$$

$$u(x,y) = A\left(\frac{1}{4} - \left(x - \frac{1}{2}\right)^2\right) \quad \text{on} \quad \Gamma_{y=1} \tag{6.5}$$

$$u(x,y) = 0 \quad \text{on} \quad \Gamma_{x=0} \tag{6.6}$$

$$u(x,y) = 0 \quad \text{on} \quad \Gamma_{x=1} \tag{6.7}$$

where the non-linearity can be varied by changing the value of A, to be any positive integer. $A = 1$ makes the problem the least non-linear.

6.1.2 Weak Form

The weak form of equation (6.1) is obtained by multiplying the equation by an weighting function, $\nu(x,y)$ and integrating across the domain. $\nu(x,y)$ is an arbitrary function with the property $\nu = 0$ for all Dirichlet boundary conditions. The weak form is

$$\int_\Omega \nu \nabla \cdot \left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}}\right) d\Omega = 0 \qquad \forall \nu \quad . \tag{6.8}$$

By applying Green's formula,

$$\int_\Gamma \nu \left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}}\right) \cdot n d\Gamma - \int_\Omega \nabla \nu \cdot \left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}}\right) d\Omega = 0 \qquad \forall \nu \tag{6.9}$$

is obtained. On the Dirichlet boundary, ν=0. Because every boundary condition is a Dirichlet boundary condition and there is no Neumann condition in this problem, the first term in the equation above falls away.

Functions $u(x,y)$ and $\nu(x,y)$, as well as their first derivatives are square integrable. Furthermore, the solution and the arbitrary test functions are assumed to satisfy the Dirichlet boundary conditions. The space of functions satisfying these conditions is denoted by

$$V_0 = \{\nu \mid \int_\Omega [\nu(x,y)]^2 \, d\Omega < \infty, \int_\Omega [\nu'(x,y)]^2 \, d\Omega < \infty, \nu(\Gamma_D) = 0\} . \tag{6.10}$$

$$V = \{u \mid \int_\Omega [u(x,y)]^2 \, d\Omega < \infty, \int_\Omega [u'(x,y)]^2 \, d\Omega < \infty, u(\Gamma_D) = \bar{u}\} . \tag{6.11}$$

where $\bar{u}$ is the specified boundary conditions.

Therefore, for all $\nu \in V_0$, there exists $u \in V$ such that $u = \bar{u}$ on Γ_D and

$$-\int_\Omega \nabla\nu \cdot \left(\frac{\nabla u}{\sqrt{1+|\nabla u|^2}}\right) d\Omega = 0 \qquad \forall \nu \in V_0 \quad . \tag{6.12}$$

6.1.3 Galerkin Form

The components of the weak form can be approximated with NURBS definitions

$$\nu = \sum_{i=1}^{n}\sum_{j=1}^{m} R_i^p(\xi) R_j^q(\eta) \mathbf{v}_{ij} \tag{6.13}$$

$$(\boldsymbol{\nabla}\nu)^T = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,\boldsymbol{x}}^p(\xi) R_{j,\boldsymbol{x}}^q(\eta) \mathbf{v}_{ij} \tag{6.14}$$

$$u(x,y) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_i^p(\xi) R_j^q(\eta) \boldsymbol{d}_{ij} \tag{6.15}$$

$$(\boldsymbol{\nabla} u)^T = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,\boldsymbol{x}}^p(\xi) R_{j,\boldsymbol{x}}^q(\eta) \boldsymbol{d}_{ij} \quad . \tag{6.16}$$

Each of these definitions can be represented by approximations in the form of shape function and derivative shape function arrays and their respective control points.

$$\nu = \mathbf{R}\mathbf{v} = \mathbf{v}^T\mathbf{R}^T \tag{6.17}$$

$$(\boldsymbol{\nabla}\nu)^T = \mathbf{B}\mathbf{v} = \mathbf{v}^T\mathbf{B}^T \tag{6.18}$$

$$u(x,y) = \mathbf{R}\boldsymbol{d} \tag{6.19}$$

$$(\boldsymbol{\nabla}u)^T = \mathbf{B}\boldsymbol{d} \tag{6.20}$$

where $\mathbf{R}$ and $\mathbf{B}$ are the shape function and derivative shape function arrays respectively and are the same arrays described in equations (4.16) and (4.17) as

$$\mathbf{R} = [R^{p,q}_{1,1}(\xi,\eta), R^{p,q}_{2,1}(\xi,\eta), \ldots, R^{p,q}_{p+1,1}(\xi,\eta), R^{p,q}_{1,2}(\xi,\eta), \ldots, R^{p,q}_{p+1,2}(\xi,\eta), \ldots, R^{p,q}_{1,q+1}(\xi,\eta), \ldots, R^{p,q}_{p+1,q+1}(\xi,\eta)]$$

and

$$\mathbf{B} = \begin{bmatrix} \mathsf{R}^{sca}_{1,x}, & \mathsf{R}^{sca}_{2,x}, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),x} \\ \mathsf{R}^{sca}_{1,y}, & \mathsf{R}^{sca}_{2,y}, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),y} \end{bmatrix} \quad .$$

These NURBS approximations can now be substituted into the weak form, equation (6.8), to yield the Bubnov-Galerkin form

$$-\int_\Omega \mathbf{v}^T\mathbf{B}^T \cdot \left(\frac{\mathbf{B}\boldsymbol{d}}{\sqrt{1+|\mathbf{B}\boldsymbol{d}|^2}}\right) d\Omega = 0 \qquad \forall\nu \quad . \tag{6.21}$$

Because $\nu(x,y)$ is arbitrary,

$$\underbrace{-\int_\Omega \mathbf{B}^T \cdot \left(\frac{\mathbf{B}\boldsymbol{d}}{\sqrt{1+|\mathbf{B}\boldsymbol{d}|^2}}\right) d\Omega}_{\boldsymbol{K}} = \underbrace{0}_{\boldsymbol{f}} \qquad \forall\nu \quad . \tag{6.22}$$

From the equation above, it can be seen that the stiffness matrix, $\boldsymbol{K}$ is a function of the control point heights, $\boldsymbol{d}$. Therefore, equation (6.22) needs to be approximated iteratively which will be discussed in the following Section 6.1.4.

6.1.4 Residual Problem and Newton-Raphson Method

In order to approximate equation (6.22), the Newton-Raphson method is used. This is done by minimizing the residual, which is defined as

$$\boldsymbol{G}(\boldsymbol{d}) = \boldsymbol{K}(\boldsymbol{d}) - \boldsymbol{f} \quad . \tag{6.23}$$

In this case, $\boldsymbol{f}$ is equal to zero and therefore the residual can be written as

$$\boldsymbol{G}(\boldsymbol{d}) = \boldsymbol{K}(\boldsymbol{d}) \quad . \tag{6.24}$$

After obtaining the residual, the Newton-Raphson method can be applied. This can be done by the following steps:

1. Guess a value for $\boldsymbol{d}_0$ which should satisfy all boundary conditions. In this case, two of the boundary conditions were projected to guess the non-prescribed nodal values:

$$\boldsymbol{d}_0 = A\Big(\frac{1}{4} - \big(x - \frac{1}{2}\big)^2\Big) \tag{6.25}$$

2. Calculate the residual, using the initial guess, $\boldsymbol{d}_0$:

$$\boldsymbol{G}(\boldsymbol{d}_0) = \boldsymbol{K}(\boldsymbol{d}_0) \tag{6.26}$$

3. Now linearize the residual with respect to the initial guess:

$$L[\boldsymbol{G}(\boldsymbol{d})]|_{\boldsymbol{d}=\boldsymbol{d}_0} = \boldsymbol{G}(\boldsymbol{d}_0) + \frac{\delta \boldsymbol{G}}{\delta \boldsymbol{d}}|_{\boldsymbol{d}_0} \cdot \boldsymbol{u} \tag{6.27}$$

Where $\frac{\delta \boldsymbol{G}}{\delta \boldsymbol{d}}$ can be calculated as follows:

$$\boldsymbol{G} = \boldsymbol{K} = \int_\Omega \mathbf{B}^T \frac{\mathbf{B}\boldsymbol{d}}{\sqrt{1 + \boldsymbol{d}^T\mathbf{B}^T\mathbf{B}\boldsymbol{d}}} d\Omega \tag{6.28}$$

$$\frac{\delta \boldsymbol{G}}{\delta \boldsymbol{d}} = \int_\Omega \frac{\mathbf{B}^T\mathbf{B}}{\sqrt{1 + \boldsymbol{d}^T\mathbf{B}^T\mathbf{B}\boldsymbol{d}}} + \mathbf{B}^T\mathbf{B}\boldsymbol{d}\Big(- \frac{1}{2}\big(1 + \boldsymbol{d}^T\mathbf{B}^T\mathbf{B}\boldsymbol{d}\big)^{-\frac{3}{2}}\big(\mathbf{B}^T\mathbf{B}\boldsymbol{d} + \boldsymbol{d}^T\mathbf{B}^T\mathbf{B}\big)\Big) d\Omega \tag{6.29}$$

$$\frac{\delta \boldsymbol{G}}{\delta \boldsymbol{d}} = \int_\Omega \frac{\mathbf{B}^T\mathbf{B}}{\sqrt{1 + \boldsymbol{d}^T\mathbf{B}^T\mathbf{B}\boldsymbol{d}}} - \frac{\mathbf{B}^T\mathbf{B}\boldsymbol{d}}{\sqrt{1 + \boldsymbol{d}^T\mathbf{B}^T\mathbf{B}\boldsymbol{d}^3}}\Big(\boldsymbol{d}^T\mathbf{B}^T\mathbf{B}\Big) d\Omega \tag{6.30}$$

4. By setting equation (6.27) equal to zero, $\boldsymbol{u}$ can be calculated by:

$$\boldsymbol{u} = -\left(\frac{\delta \boldsymbol{G}}{\delta \boldsymbol{d}}\right)^{-1} \boldsymbol{G}(\boldsymbol{d}_0) \tag{6.31}$$

5. The next approximation of the surface height control point vector, $\boldsymbol{d}_1$, can be calculated with equation (6.32):

$$\boldsymbol{d}_1 = \boldsymbol{d}_0 + \alpha \boldsymbol{u} \tag{6.32}$$

6. A new $\boldsymbol{G}(\boldsymbol{d}_1)$ is evaluated, using the new surface heights $\boldsymbol{d}_0$. This $\boldsymbol{G}(\boldsymbol{d}_1)$ is normalized (using only the free degrees of freedom) to get the normalized residual. If this residual is greater than the tolerance, return to step 2 to calculate the next guess for surface heights. If it is less than the tolerance, use the most recent approximation of the surface height control point vector as the solution.

The α in point 5 above is a damping factor to reduce the aggression of the iteration scheme and makes it more possible to approximate highly non-linear problems. $\alpha = 1$ means that no damping is implemented, α is then reduced heuristically to find the optimum damping factor to approximate highly non-linear problems.

6.1.5 Implementation

The preprocessing stage of solving the minimal surfaces problem follows the same steps to the two-dimensional scalar problem described in Section 4.1.4. The same refinement, problem variables, NURBS definitions and boundary condition steps are followed. In addition to these steps, an initial guess, $\boldsymbol{d}_0$ is chosen that satisfies the boundary conditions and is used to start the iteration procedure. For this problem an initial guess was chosen to be

$$\boldsymbol{d}_0 = A\Big(\frac{1}{4} - \big(x - \frac{1}{2}\big)^2\Big) \quad . \tag{6.33}$$

A complication of this problem is that two of the boundary conditions are represented by a inhomogeneous equation. To obtain the control points that represent this Dirichlet boundary condition exactly, the control point values need to be calculated by using the technique in Section 2.5.1.4, by matching the control points to the polynomial boundary condition function.

The main loops of the Newton-Raphson method, to obtain an approximation for $\boldsymbol{d}$, are described in Algorithm 10. The tolerance of the error, `Gn` is 1^{-6} for this problem.

Algorithm 10 Newton-Raphson method for evaluating control points.

Initialize `Gn>tol`
while `Gn>tol` **do**
  Initialize $\boldsymbol{K}$ and $\boldsymbol{DG}$
  for `elM=1:el_totalM` **do**
    for `elN=1:el_totalN` **do**
      `el=el+1`
      Initialize $\boldsymbol{K}^e$ and $\boldsymbol{DG}^e$
      for $gp2, gp1 = 1 : \#GaussPoints$ **do**
        Compute $\boldsymbol{D} = \frac{1}{\sqrt{1+\boldsymbol{d}_{\mathtt{asm}}^T(\mathbf{B}^T\mathbf{B})\boldsymbol{d}_{\mathtt{asm}}}}$
        Compute $\boldsymbol{K}^e = \boldsymbol{K}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} \mathbf{B}^T \boldsymbol{D} \mathbf{B}$
        Compute $\boldsymbol{DG}^e$
        $\boldsymbol{DG}^e = \boldsymbol{DG}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} (\mathbf{B}^T \boldsymbol{D} \mathbf{B} - \mathbf{B}^T (\boldsymbol{D}^3) \mathbf{B} \boldsymbol{d}_{\mathtt{asm}} (\boldsymbol{d}_{\mathtt{asm}}^T \mathbf{B}^T \mathbf{B}))$
      end for
      Assemble $\boldsymbol{K}_{\mathtt{asm\ asm}} = \boldsymbol{K}_{\mathtt{asm\ asm}} + \boldsymbol{K}^e$
      Assemble $\boldsymbol{DG}_{\mathtt{asm}} = \boldsymbol{DG}_{\mathtt{asm}} + \boldsymbol{DG}^e$
    end for
  end for
  $\boldsymbol{G} = \boldsymbol{K}\boldsymbol{d}$
  $\boldsymbol{u} = -\boldsymbol{DG}^{-1}\boldsymbol{G}(\boldsymbol{d}_0)$
  $\boldsymbol{d} = \boldsymbol{d} + \alpha \boldsymbol{u}$
  Initialize $\boldsymbol{K}$
  for `elM=1:el_totalM` **do**
    for `elN=1:el_totalN` **do**
      `el=el+1`
      Initialize $\boldsymbol{K}^e$
      for $gp2, gp1 = 1 : \#GaussPoints$ **do**
        Compute $\boldsymbol{D} = \frac{1}{\sqrt{1+\boldsymbol{d}_{\mathtt{asm}}^T(\mathbf{B}^T\mathbf{B})\boldsymbol{d}_{\mathtt{asm}}}}$
        Compute $\boldsymbol{K}^e = \boldsymbol{K}^e + J_\xi J_{\bar{\xi}} w_{gp1} w_{gp2} \mathbf{B}^T \boldsymbol{D} \mathbf{B}$
      end for
      Assemble $\boldsymbol{K}_{\mathtt{asm\ asm}} = \boldsymbol{K}_{\mathtt{asm\ asm}} + \boldsymbol{K}^e$
    end for
  end for
  $\boldsymbol{G} = \boldsymbol{K}\boldsymbol{d}$
  $Gn = norm(\boldsymbol{G}_{\mathtt{n\_F}})$
end while

For each iteration, the convergence, Gn is plotted. The number of possible iteration loops are also limited to prevent loops from running eternally if the error does not converge.

The NURBS vectors for the x, y-geometry is plotted as well as the NURBS vector for the field variable, height. The soap bubble can then be plotted as in the results Section 6.1.6.

6.1.6 Results

The approximate solution to the minimal surface soap bubble problem is plotted below with no refinement as a 2×1 mesh with p=q=2 with the boundary conditions having minimal non-linearity, A=1.

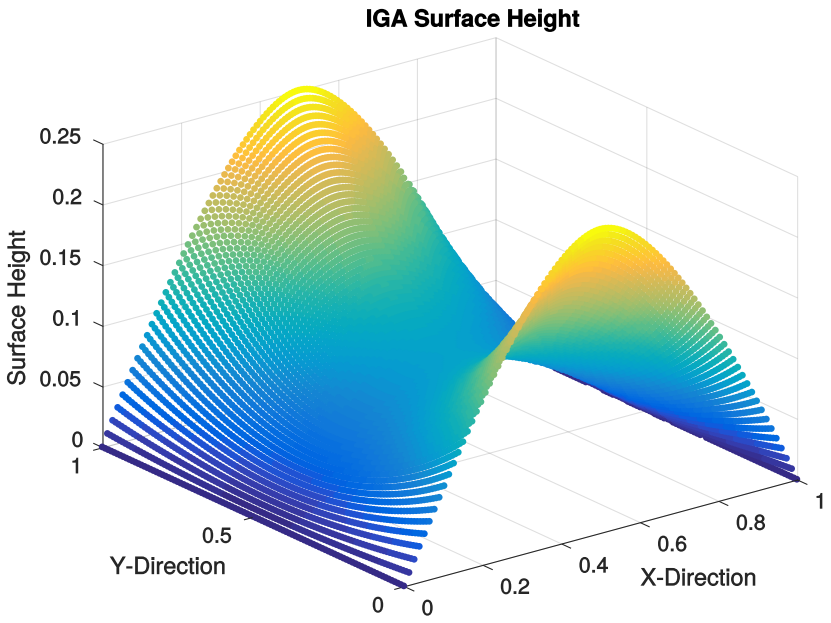


Figure 6.1 Minimal surfaces soap bubble domain with no refinement, A=1.

The number of iterations before the error, Gn, is less than the tolerance (1^{-6}), is shown in Figure 6.2.

As can be seen from the previous two figures, even for a unrefined mesh, the convergence of the rates are high and the problem can be solved accurately. The non-linearity of the problem is increased so that $A = 12$. The problem takes slightly longer to converge, but still yields accurate results in the end with no refinement necessary. The result of the plot is seen in Figure 6.3. The convergence graph of the more non-linear problem is shown in Figure 6.4.

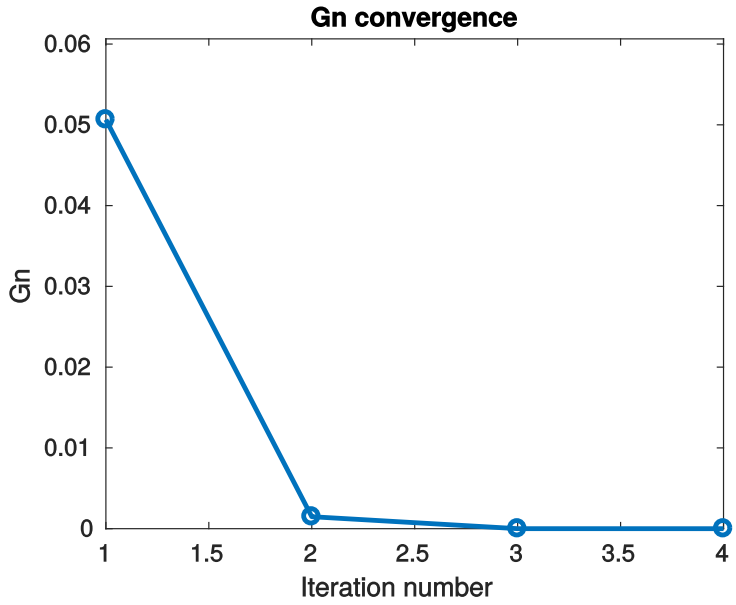


Figure 6.2 Error convergence of Soap Bubbles problem with no refinement, A=1.

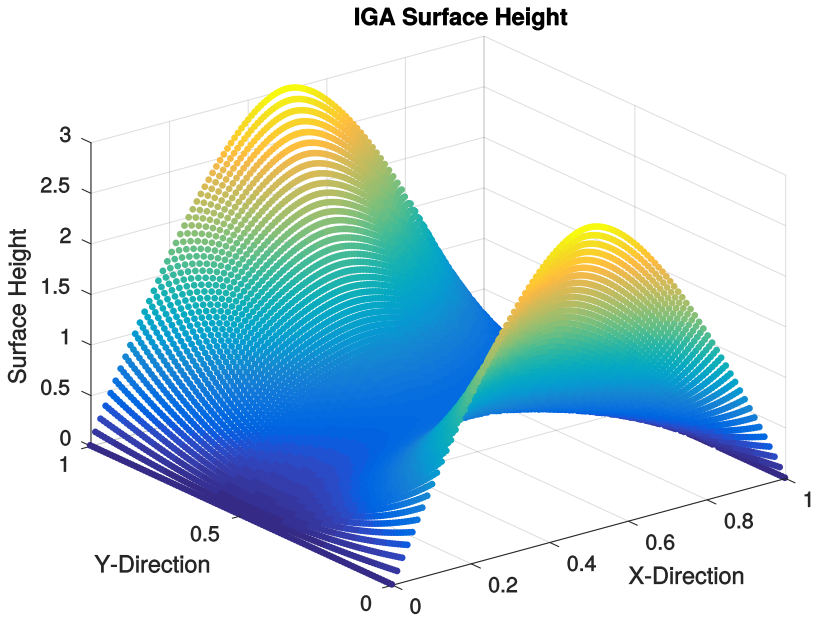


Figure 6.3 Minimal surfaces soap bubble domain with no refinement, $A = 12$.

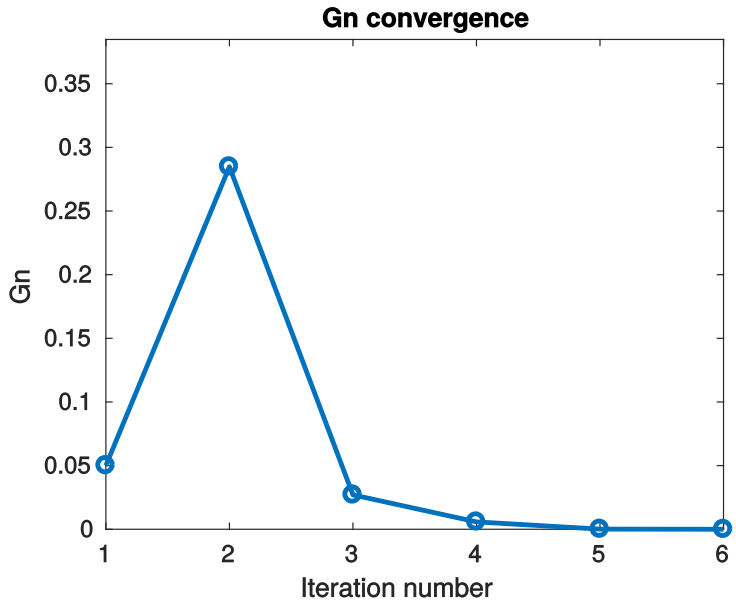


Figure 6.4 Error convergence of Soap Bubbles problem with no refinement, A=12.

Interestingly, the convergence slows down when the refinement is increased. Figure 6.5 shows the convergence of the same non-linear problem ($A = 12$) as seen in Figure 6.3, but with one level of h-refinement. The variations in the convergence in Figure 6.4 could be artificial and due to the complexity that a variation in the order produces. It is suspected that the variations would be smoothed out if damping is introduced.

p- and k-refinement does not work for highly non-linear meshes as the solution does not converge. Problems that have $A > 11$ do not converge and cannot be approximated. The convergence for a problem where $A = 11$ is shown in Figure 6.6 which has been p-refined to have orders in both direction equal to three.

A damping factor (α) was introduced in Section 6.1.5. Damping is generally used when a problem does not converge to reduce the aggression of the next approximation, however it was not needed in this work. If the problems were to fail and not converge, it could be used.

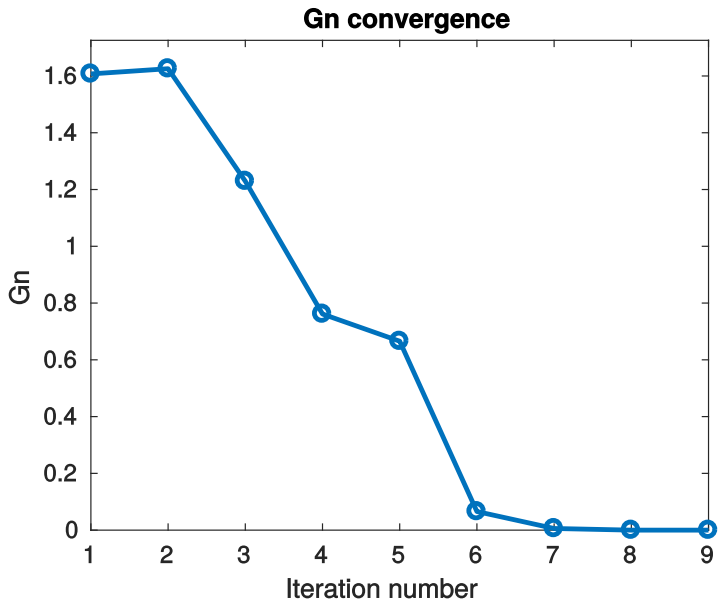


Figure 6.5 Error convergence of Soap Bubbles problem with one level of h-refinement, A=12.

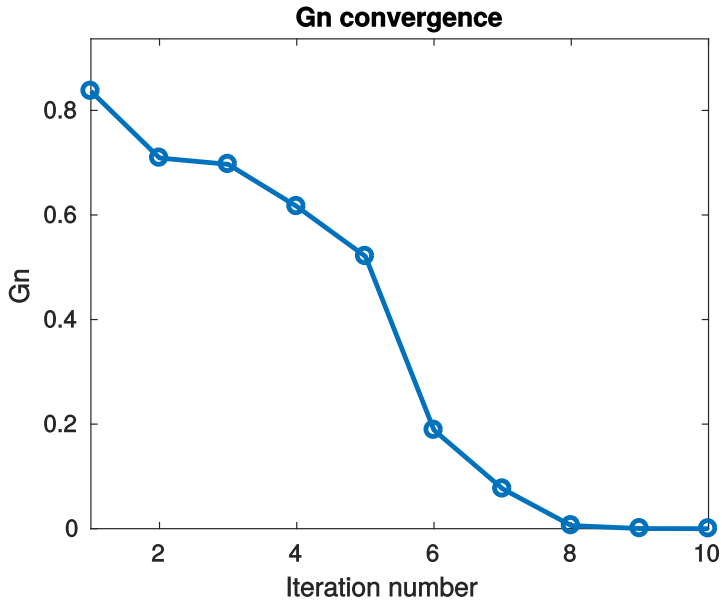


Figure 6.6 Error convergence of Soap Bubbles problem with p-refinement to p=q=3, A=11.

Chapter 7

IGA for Non-Trivial Geometries

7.1 Getting Geometries out of Rhino

Modeling - here and elsewhere is done more naturally in 3D modeling software, than by creating geometry from control points and weights. It is therefore useful to be able to extract the NURBS data from 3D modeling software, such as Rhino 3D, to then be used in the in-house code for analysis. An example of a surface extraction from Rhino 3D is shown in this section. A Python script is used to select and extract the NURBS data. The Python script below is an adaptation of the curve extraction code in [10] to extract NURBS surfaces.

```
import Rhino
import System.Guid
import scriptcontext
import rhinoscriptsyntax as rs
curve=rs.GetObject("Select Surface",rs.filter.surface)
print 'NURBS Data'
if rs.IsSurface(curve):
        points = rs.SurfacePoints(curve)
        weights = rs.SurfaceWeights(curve)
        knots = rs.SurfaceKnots(curve)
print points
print weights
print knots
print 'Done'
```

To extract the surface data, run the Python script and when prompted, select the surface that is required. Press enter and the NURBS data should be displayed in the Python

terminal. Some tidying will be required to get the points in the correct order in a matrix and exclude the text. The final lines contain the knot vectors. The first and last entries of each knot vector will need to be repeated to obtain the correct multiplicities at the end knot values to obtain a open knot vector.

The following figures show a NURBS surface drawn in Rhino 3D and the corresponding surface, drawn in the in-house code. For details of implementation, case 3 of the `NURBS_2D` file in the digital submission of this dissertation shows how the following surface was implemented.

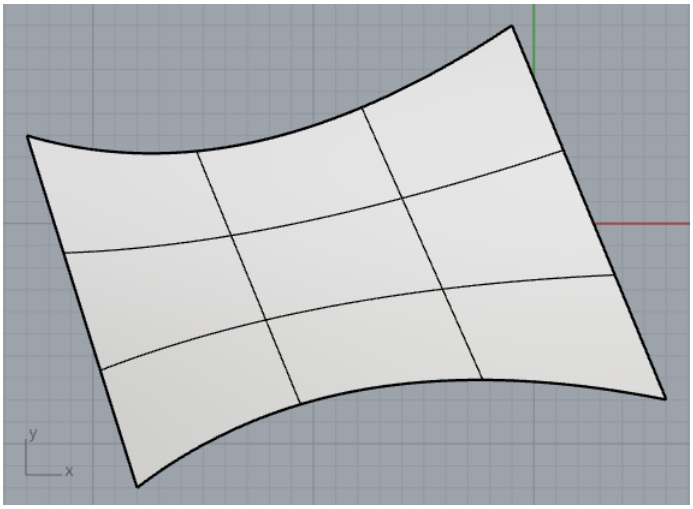


Figure 7.1 Surface generated in Rhino 3D.

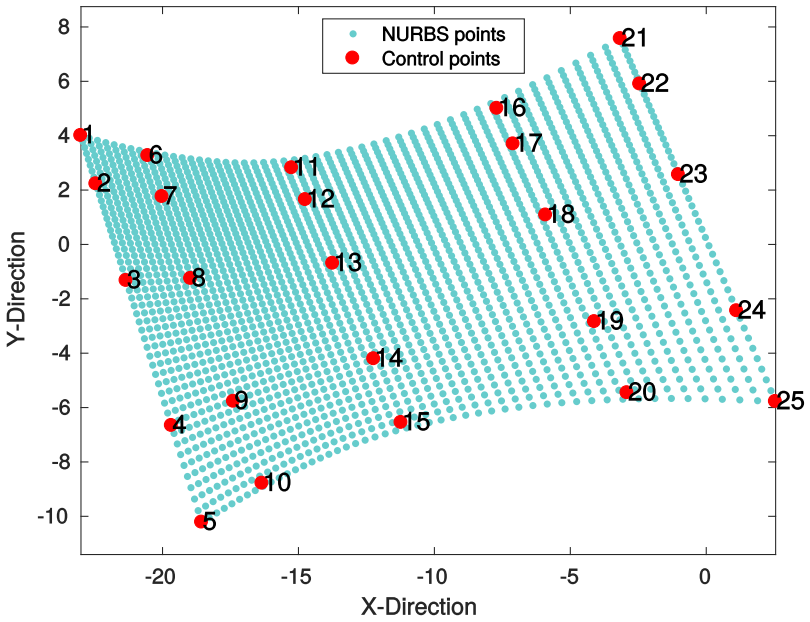


Figure 7.2 Replication of Figure 7.1 in in-house code.

Chapter 8

Recommendations and Conclusions

8.1 Recommendations

This section presents suggestions for improving the approaches adopted in this work, and on extending the study.

With regards to the in-house code, it would be useful to store the control point coordinates as objects in the control point array in MATLAB instead of each coordinate in a point being a different entry in the general control point array. This would allow the code to be used in a more modular way that allows the points to be in a one-, two- or three-dimensional space.

The code is currently set up in a simple and easy to understand way, however, it is not optimized for speed. One of the ways in which it can be optimized is by performing vectorized operations with the basis functions, instead of multiplying individual entries in for-loops. Once the speed of the code is made more efficient, it can be compared to in-house traditional FEM codes in terms of speed and computational efficiency. It would also be convenient for the code to automatically detect the direction of Neumann boundary conditions and adjust the code accordingly.

There are numerous directions in which the investigation reported here could be extended. The first option is to further the research done in extracting geometry from 3D modeling software such as Rhino 3D. Initial progress was made in extracting NURBS data of simple curves and surfaces from Rhino 3D as seen in Section 7.1, but extracting more complex surfaces, such as poly-surfaces, would require additional work. For example, Rhino 3D draws a quarter of a plate with a centrally located hole as seen in Figure 2.4b, as a

poly-surface as a square and a quarter circle. Once this data is extracted, it needs to be implemented into the IGA code. Exploratory work was done in stitching two surfaces together for a simple traction problem using a penalty method similar to equation (5.30), but this did not prove successful. [1, p. 60] suggests a method that can be used to stitch multiple surfaces together. Using the NURBS data extracted from Rhino, a multi-surface code would be useful in analyzing more complex problems.

In terms of increasing the range of problems that can be solved with this in-house code, three-dimensional solids should be considered. While creating the NURBS solids and performing basic analysis on the structure would be useful, it should be noted that most 3D modeling software represents solids using multiple surfaces. It would therefore be useful to convert these multi-surface solids from 3D modeling software into solid NURBS bodies that can be used in the in-house code.

A further direction in which to extend the study would be through the exploration of more complex non-linear problems. The non-linear problem shown in Section 6.1 introduces the approximation of solving non-linear problems in IGA and shows the fundamentals, but more complex problems such as those involving large deformations could be considered.

8.2 Summary and Conclusions

This dissertation focused on developing an in-house isogeometric analysis (IGA) code that could be used to approximate a range of problems. The project started by introducing the concept of B-splines and then NURBS (non-uniform rational B-splines), that form the additional knowledge required for IGA, above that required for traditional FEM. Key features of NURBS were discussed, including how to implement the mathematics and create NURBS curves, surfaces and solids. Refinement was discussed in this section, which forms the basis of refinement for IGA.

In the next chapter, the fundamentals and implementation of IGA were presented with two one-dimensional problems. The solution to a Poisson equation was approximated in order to introduce the framework of an IGA code. A one-dimensional radial heat conduction problem was approximated in order to demonstrate how Neumann boundary conditions are applied, for a one-dimensional problem.

The same radial heat conduction problem was then used to introduce two-dimensional scalar problems. The fundamentals of the analysis were introduced and the results compared to the analytical solution. In order to demonstrate how complex boundary conditions

are introduced, a heat conduction problem is approximated for a domain of a plate with a centrally located hole. This geometry introduces many intricacies of applying both the Neumann and Dirichlet boundary conditions. One of the key successes of this project is the detail in which the boundary condition applications have been explained, which was not found in depth in any of the literature.

A two-dimensional linear elasticity problem was used to demonstrate the implementation of a two-dimensional vector-valued problem. The application of boundary conditions was also discussed in depth.

A minimal surfaces problem was used to demonstrate a simple geometric non-linear problem. While demonstrating the ability of IGA to approximate non-linear problems, this problem also showed how to apply inhomogeneous Dirichlet boundary conditions.

The final chapter dealt with how to extract geometries from Rhino 3D, a 3D modeling software, using a Python script. This chapter serves as an introduction to what would be possible for continuing this project.

While no novel work was carried out in this dissertation, the detail in which implementation was discussed is hoped to be useful to future researchers wishing to learn the fundamentals and implementation of IGA. The digital submission of this project includes the code that was developed for this dissertation, as well as a readme file on how to use the code.

References

[1] Cottrell, JA., Hughes, T.J.R. & Bazilevs, Y., *Isogeometric Analysis towards integration of CAD and FEA*, Wiley, 2009.

[2] A. Townsend, "On the spline: A brief history of the computational curve", URL `http://www.alatown.com/spline-history-architecture/`.

[3] Hughes, T.J.R., Cottrell, J.A. & Bazilevs, Y., "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement", *Computer Methods in Applied Mechanics and Engineering*, vol. 194.

[4] Vinh Phu Nguyen, Stéphane P.A. Bordas, Timon Rabczuk & Robert N. Simpson, "An introduction to standard and enriched isogeometric analysis through a simple Matlab code", *Computers and Structures*.

[5] Les Peigl & Wayne Tiller, *The NURBS book*, Springer, 1997.

[6] James Stewart, *Calculus Concepts and Contexts*, Brookes/Cole, Cengage Learning, 4 edn., 2010, mcMaster University and University of Toronto.

[7] Fish, Jacob (Rensselaer Polytechnic Institute), USA & Belytschko, Ted (Northwestern University), USA, *A First Course in Finite Elements*, Wiley, 2007.

[8] Cook, Robert, Malkus, David, Plesha, Michael & Witt, Robert, *Concepts and applications of finite element analysis*, John Wiley & Sons, inc., 2002.

[9] Marc Calvo Schwarzwälder, *The mathematical model of a soap film. Theory of minimal surfaces*, M.Sc thesis, Facultat de Matemàtique, Universitat Polytecnica de Catalunya, 2013/2014, URL `https://upcommons.upc.edu/bitstream/handle/2099.1/22932/memoria.pdf`.

[10] "How to get data out of nurbs curves in rhino", URL `https://discourse.mcneel.com/t/how-to-get-data-out-of-nurbs-curves-in-rhino/3436`.

[11] G.E. Farin, "Nurb curves and surfaces: from projective geometry to practical use", *Choice Reviews Online*, vol. 33, (1995), pp. 33–0977,33–0977.

Appendices

Appendix A

Notation

This dissertation draws on works from [1], [4] and [5], which expectedly all use their own notation. This section aims on clarifying the notation used in this dissertation and settle on a common naming convention. First, the symbols are clarified, after which the basis function and tensor notation will be explained. Because this dissertation does not work with 3D NURBS, it will not be extended to in this explanation, however, where it is mentioned, the notation will be explained.

A.1 Symbols

NURBSs and B-splines describes geometric structures, such as curves, surfaces or solids. Each of these structures respectively relate to a one-, two- or three-dimensional parametric shape. These geometric structures can be described in a one-, two- or three-dimensional physical space. This dissertation does not deal with solids, however it does deal with curves and surfaces in a three-dimensional space.

Symbol	Coding variable	Name	Description
ξ, η	`xi,eta`	Parametric coordinates	Corresponds to coordinates in different directions in the parametric domain.
$\bar{\xi}, \bar{\eta}$	`Xi_bar, Eta_bar`	Isoparametric coordinates	Points in the isoparametric domain running from -1 to 1 in each direction. The Gauss points exist in this domain.
Ξ,$\mathscr{H}$	`Xi,Eta`	Knot vectors	Knot vectors indicating element boundaries in respective directions in the parametric domain.
p, q	`p,q`	Order of basis functions	Order of basis functions in respective ξ and η directions.
n, m	`n,m`	Number of basis functions	Number of basis functions in each direction across domain which corresponds to the number of control points in each direction.
$\boldsymbol{P}_{ijk}/\boldsymbol{Q}_{ijk}$	`P,Q`	Control Points	Control points are points in the physical space that map the parametric domain to the physical domain to give a curve (1D parametric domain) with notation $\boldsymbol{P}_i$, surface (2D parametric domain) with notation $\boldsymbol{P}_{ij}$ or solid (3D parametric domain) with notation $\boldsymbol{P}_{ijk}$ in 1D space, 2D space, 3D space. $\boldsymbol{Q}$ is used when new control points are being calculated in refinement. $\boldsymbol{P}/\boldsymbol{Q}$ refers to the whole array of control points, not just single entries.
w_{ijk}	`w`	Weights	These are the weights associated with each control point. Each weight corresponds to a control point. $\boldsymbol{w}$ refers to the whole array of control points, not just single entries.
$\boldsymbol{P}^w_{ijkl}$	`P_w`	Projected control points	Each entry of P_{ijk} is multiplied by its associated weight (w_{ijk}), with the final column being the actual weight. $\boldsymbol{P}^w/\boldsymbol{Q}^w$ refers to the whole array of control points, not just single entries.
J_ξ	`JXi/JEta`	Parametric Jacobian	Jacobian to convert the parametric space to the physical space.
$J_{\bar{\xi}}$	`JXi_bar/ JEta_bar`	Isoparametric Jacobian	Jacobian to convert the isoparametric space to the parametric space.

Table A.1 *Table of Symbols*

A.2 Basis Functions

The following notation helps distinguish B-spline basis functions from NURBS basis functions. Both basis functions are recursive and NURBS basis functions are themselves functions of B-spline basis functions, as well as the control point weights. NURBS basis functions collapse to B-spline basis functions when all the weights are equal.

The B-spline basis function is called the Cox-de Boor recursion formula [4] and can be defined as:

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi) \tag{A.1}$$

where when p=0:

$$N_i^{p=0}(\xi) = \begin{cases} 1, & \text{if } \xi_i \geq \xi > \xi_{i+1} \\ 0, & \text{otherwise.} \end{cases} \tag{A.2}$$

Because there is no exponentiation in B-splines, NURBS and IGA, the superscript of N will refer to the order used to calculate the basis function. The N refers to the B-spline basis function. NURBS basis functions will be denoted by R. The i in the subscript refers to the shape function number which can range from 1 to n in ξ direction. For any subsequent directions, when creating a surface, the following notation will be used: j refers to the shape function number which can range from 1 to a maximum of m in the η direction. Similarly, q would respond to the order of the basis functions. The order of basis functions (p,q), the number of basis functions (n,m) and the knot vectors (Ξ,$\mathscr{H}$) can therefore be different in different parametric directions (ξ, η). The basis functions in the η direction can then be defined as:

$$N_j^q(\eta) = \frac{\eta - \eta_j}{\eta_{j+q} - \eta_j} N_j^{q-1}(\eta) + \frac{\eta_{j+q+1} - \eta}{\eta_{j+q+1} - \eta_{j+1}} N_{j+1}^{q-1}(\eta) \tag{A.3}$$

where when q=0:

$$N_i^{q=0}(\eta) = \begin{cases} 1, & \text{if } \eta_j \geq \eta > \eta_{j+1} \\ 0, & \text{otherwise.} \end{cases} \tag{A.4}$$

Derivatives of basis functions can be indicated by the subscript notation as defined

$$N_{i,\xi}^p(\xi) = \frac{dN_i^p(\xi)}{d\xi} \tag{A.5}$$

where N is used to represent B-spline basis functions.

The NURBS basis functions are defined below in (A.6) and (A.7). Where the symbol N is reserved for B-spline basis functions, R is used for NURBS basis functions

$$R_i^p(\xi) = \frac{N_i^p(\xi) w_i}{W(\xi)} \tag{A.6}$$

where

$$W(\xi) = \sum_{j=1}^{n} N_j^p(\xi) w_j \quad . \tag{A.7}$$

Basis functions in two-dimensional NURBS is a product of the two basis functions in each direction. The bivariate basis function is defined as

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_i^p(\xi) N_j^q(\eta) w_{ij}}{\sum_{\hat{i}=1}^{n} \sum_{\hat{j}=1}^{m} N_{\hat{i}}^p(\xi) N_{\hat{j}}^q(\eta) w_{\hat{i}\hat{j}}} \quad . \tag{A.8}$$

A.2.1 Arrays

Quantities comprised of multiple components such as tensors, vectors, matrices and arrays are described using bold font (eg. $\mathbf{R}$). Each individual scalar component in the tensor, vector, matrix or array is denoted by the same letter, but in a non-bold font (eg. R_{ij}) with subscripts to indicate the position in the tensor, vector, matrix or array. The exception to this rule is that $\boldsymbol{P}_i$, $\boldsymbol{P}_{ij}$ and $\boldsymbol{P}_{ijk}$ is still written in bold as it refers to a coordinate space and not a scalar value. Although $\boldsymbol{P}_i$ is a scalar value, it was still decided to use a bold font for a single entry.

Tensors, vectors, matrices and arrays will also always be denoted by a sans-serif font, while a serif font is reserved for basis functions. For example, the shape function array $\mathbf{R}$ is made up on basis functions $R_i^p(\xi)$. For a one dimensional NURBS curve, the shape function array is defined as

$$\mathbf{R} = [R_1^p(\xi), R_2^p(\xi), R_3^p(\xi)] \quad . \tag{A.9}$$

When dealing with higher order structures, the superscripts *sca* and *vec* are used to indicate scalar or vector field values. For a two dimensional NURBS surface, the scalar valued shape function array is defined as

$$\mathbf{R}^{sca} = [R^{p,q}_{1,1}(\xi,\eta), R^{p,q}_{2,1}(\xi,\eta), \ldots, R^{p,q}_{p+1,1}(\xi,\eta), R^{p,q}_{1,2}(\xi,\eta), \ldots, R^{p,q}_{p+1,2}(\xi,\eta), \ldots, R^{p,q}_{1,q+1}(\xi,\eta), \ldots, R^{p,q}_{p+1,q+1}(\xi,\eta)] \quad .$$

It is important to note that $\mathbf{R}^{sca}$ is a first order tensor (aka. same shape as a vector), not a matrix. Each component in $\mathbf{R}^{sca}$ is associated to a particular elemental control point. The components of $\mathbf{R}^{sca}$ will be used in future equations. Either of the above can be written in index notation as seen in (A.10). For the one dimensional NURBS equation, $j = 0$ and $R^{p,q}_{i,j}(\xi,\eta)$ collapses to the univariate basis function, $R^{p}_{i}(\xi)$.

$$\mathsf{R}_{i+(p+1)(j-1)} = R^{p,q}_{i,j}(\xi,\eta) \quad . \tag{A.10}$$

For a two dimensional NURBS curve, the vector valued shape function array (where the field parameter a vector, eg. displacement in x and y) is made up of components in equation (4.16) and is defined as follows:

$$\mathbf{R}^{vec} = \begin{bmatrix} \mathsf{R}^{sca}_{1}, & 0, & \mathsf{R}^{sca}_{2}, & 0, & \mathsf{R}^{sca}_{3}, & 0, & \ldots & \mathsf{R}^{sca}_{(p+1)(q+1)}, & 0 \\ 0, & \mathsf{R}^{sca}_{1}, & 0, & \mathsf{R}^{sca}_{2}, & 0, & \mathsf{R}^{sca}_{3}, & \ldots & 0, & \mathsf{R}^{sca}_{(p+1)(q+1)} \end{bmatrix} \quad . \tag{A.11}$$

The next important matrix to introduce is the $\mathbf{B}$ matrix. The $\mathbf{B}$ matrix is the matrix of derivatives of the shape functions. In one dimensional NURBS curve, the $\mathbf{B}$ matrix is simply a first order array of the derivatives of the shape functions

$$\mathbf{B} = \begin{bmatrix} R^{p}_{1,x}, & R^{p}_{2,x}, & \ldots & R^{p}_{(p+1),x} \end{bmatrix} \quad . \tag{A.12}$$

The $\mathbf{B}$ matrix for two dimensional NURBS surfaces for scalar value problems is defined

$$\mathbf{B}^{sca} = \begin{bmatrix} \mathsf{R}^{sca}_{1,x}, & \mathsf{R}^{sca}_{2,x}, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),x} \\ \mathsf{R}^{sca}_{1,y}, & \mathsf{R}^{sca}_{2,y}, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),y} \end{bmatrix} \quad . \tag{A.13}$$

The $\mathbf{B}$ matrix for two dimensional NURBS surfaces for vector value problems is defined

$$\mathbf{B}^{vec} = \begin{bmatrix} \mathsf{R}^{sca}_{1,x}, & 0, & \mathsf{R}^{sca}_{2,x}, & 0, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),x}, & 0 \\ 0, & \mathsf{R}^{sca}_{1,y}, & 0, & \mathsf{R}^{sca}_{2,y}, & \ldots & 0, & \mathsf{R}^{sca}_{(p+1)*(q+1),y} \\ \mathsf{R}^{sca}_{1,y}, & \mathsf{R}^{sca}_{1,x}, & \mathsf{R}^{sca}_{2,y}, & \mathsf{R}^{sca}_{2,x}, & \ldots & \mathsf{R}^{sca}_{(p+1)*(q+1),y}, & \mathsf{R}^{sca}_{(p+1)*(q+1),x} \end{bmatrix} \quad . \tag{A.14}$$

Appendix B

Projective Transformations

While B-splines can create a large range of geometries, they struggle to create conic shapes. It is however possible to create NURBS curves in $\mathbb{R}^d$ by projecting a B-spline curve in $\mathbb{R}^{d+1}$ onto a plane. Recall from page 12 that the projective NURBS weights are made up of the NURBS control points and weights as

$$\boldsymbol{P}_i^w = [\boldsymbol{P}_i w_i,\ w_i] \quad .$$

where $\boldsymbol{P}$ are the $\mathbb{R}^d$ control points, with the weights, $\boldsymbol{w}$ as the d+1 dimension. A projective curve $\boldsymbol{C}^w(\xi)$ can be created from the B-spline formulations as seen in equation (2.3)

$$\boldsymbol{C}^w(\xi) = \sum_{i=1}^{n} N_i^p(\xi)\boldsymbol{P}_i^w \quad .$$

In order to get a NURBS curve, $\boldsymbol{C}(\xi)$, the above projective curve, $\boldsymbol{C}^w(\xi)$, is projected onto a plane, w=1, where w is the d+1 dimension. This is done by solving for a weighting function, W, so that

$$\boldsymbol{C}^w(\xi)W = \boldsymbol{C}(\xi) \quad . \tag{B.1}$$

To show that equation (2.14)

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

holds, the curves $\boldsymbol{C}^w(\xi)$ and $\boldsymbol{C}(\xi)$ are separated into their different dimensions, assuming d=2. The superscript of $\boldsymbol{P}^{d=1}$ indicates the dimension of the component. The projective curve,

$$\boldsymbol{C}^w = \left(\sum_{i=1}^{n} N_i^p(\xi) P_i^{d=1} w_i,\ \sum_{i=1}^{n} N_i^p(\xi) P_i^{d=1} w_i,\ \sum_{i=1}^{n} N_i^p(\xi) w_i \right) \tag{B.2}$$

is divided by W to give

$$\boldsymbol{C}(\xi) = \left(\frac{\sum_{i=1}^{n} N_i^p(\xi) P_i^{d=1}}{\sum_{\hat{i}=1}^{n} N_{\hat{i}}^p(\xi) w_{\hat{i}}},\ \frac{\sum_{i=1}^{n} N_i^p(\xi) P_i^{d=1}}{\sum_{\hat{i}=1}^{n} N_{\hat{i}}^p(\xi) w_{\hat{i}}},\ 1 \right) \tag{B.3}$$

in order for the d+1 entry of $\boldsymbol{C}(\xi)$ to equal 1.

Therefore, the NURBS curve can be calculated using the B-spline basis functions, along with the NURBS control points and weights. Rearrange equation (B.3) to get the NURBS definition as seen in equation (2.12)

$$\boldsymbol{C}(\xi) = \sum_{i=1}^{n} \frac{N_i^p(\xi) w_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i}}^p(\xi) w_{\hat{i}}} \boldsymbol{P}_i \quad .$$

The projection of control points can be explained more visually in Figure B.1 below. If a NURBS circle of unit radius, $\boldsymbol{C}(\xi)$ is described on the w=1 plane as seen below, the projected B-spline curve, $\boldsymbol{C}^w(\xi)$, also exists. The projected curve is scaled by a magnitude of three. This is done to visualize the curve better, as well as prove that the projected curve can be scaled by any amount without affected the NURBS curve. This property is also shown mathematically. If the NURBS weights are scaled by constant, γ, and because γ is multiplied by every entry in the sum in the denominator, it can be taken out of the sum to cancel with the γ in the numerator to give the same curve as seen in equation (2.12).

$$\boldsymbol{C'}(\xi) = \sum_{i=1}^{n} \frac{N_i^p(\xi) w_i \gamma}{\sum_{\hat{i}=1}^{n} N_{\hat{i}}^p(\xi) w_{\hat{i}} \gamma} \boldsymbol{P}_i = \sum_{i=1}^{n} \frac{N_i^p(\xi) w_i \gamma}{\gamma \sum_{\hat{i}=1}^{n} N_{\hat{i}}^p(\xi) w_{\hat{i}}} \boldsymbol{P}_i = \boldsymbol{C}(\xi) \quad .$$

The projective B-spline curve is therefore projected on to the w=1 plane through lines from the projective curve through the origin as seen in Figure B.1.

The projected control points, $\boldsymbol{P}^w$ are used in NURBS refinement. This is the level of understanding required to use projected weights in this dissertation. For more information, see [11].

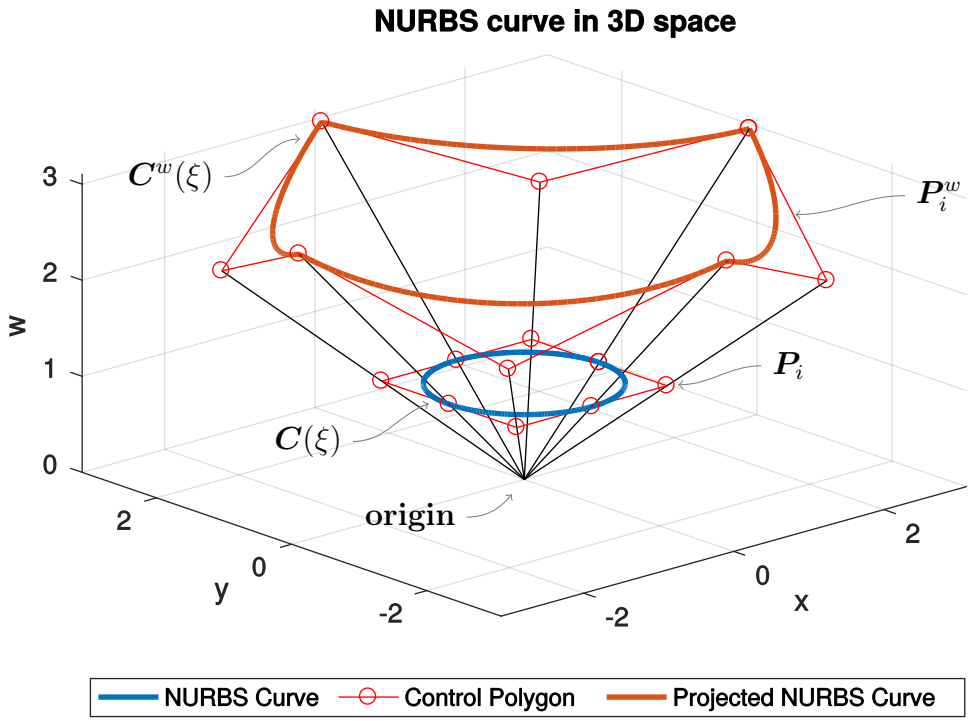


Figure B.1 $\mathbb{R}^{d+1}$ B-spline curve, $\boldsymbol{C}^w(\xi)$, transformed projectively to a $\mathbb{R}^d$ NURBS curve, $\boldsymbol{C}(\xi)$. Adapted from [1, p. 49].

Appendix C

List of Code

C.1 Main Codes

List of all codes in the order of complexity. The list starts with the simplest code progressing towards more complex codes.

Code Name	Code Description	Features
NURBS_1D.m	One dimensional NURBS curve code for various different shapes, in different physical dimensions.	To illustrate how to draw a one dimensional NURBS curve in any physical space.
NURBS_2D.m	Two dimensional NURBS curve code for various different shapes, in different physical dimensions.	To illustrate how to draw a two dimensional NURBS curve in any physical space.
IGA_1D_Nguyen.m	One dimensional scalar IGA code, solving the differential equation in the Nguyen paper.	Simple 1D problem used to compare IGA results to the results found in Nguyen.
IGA_1D_ Radial-Temperature.m	One dimensional scalar IGA code, solving a radial heat problem.	Used to compare 1D and 2D radial temperature problems and verify results.
IGA_1D_AdvecDiff.m	One dimensional scalar IGA code, solving an advection-diffusion problem.	Used to examine the effects of refinement and the stability of the code compared to traditional FEM.
IGA_1D_ Gradient.m	One dimensional scalar IGA code, solving a steep gradient problem.	Used to illustrate the effect of p-refinement and compare answers to the problem in Nguyen.

IGA_2D_ Scalar_Grid.m	Two dimensional scalar IGA code, solving a generic 2D scalar differential equation for a simple domain.	A simple 2D scalar code, used to get the basics correct before moving on to more complex geometries and problems.
IGA_2D_ Radial-Temperature.m	Two dimensional scalar IGA code, solving a radial heat problem. The same problem is solved in the IGA_1D_RadialTemperature.m code.	Used to verify resutls, compared to the 1D version. Also used to solve a simple temperature problem over a simple domain. Illustrates the effect of h- , k- and p-refinement.
IGA_2D_ Hole-PlateTemp.m	Two dimensional scalar IGA code, solving a temperature problem over a complex "Hole in Plate" domain.	Used to compare results against a traditional FEM code of the same problem and solve a 2D scalar problem over a complex domain. Illustrates the effect of h- and p-refinement.
IGA_2D_Vector.m	Two dimensional vector IGA code, solving a generic 2D vector differential equation for a simple domain.	A simple 2D vector code, used to get the basic concepts of 2D vector working before moving on to more complex geometries and problems. This code illustrates how to account for different traction direction of different edges and what needs to change in each case.
IGA_2D_ HolePlate-Disp.m	Two dimensional vector IGA code, solving a displacement problem over a complex "Hole in Plate" domain.	A more complex linear elasticity code, used to verify results and illustrate a 2D vector code on a more complex geometry.
IGA_2D_ Cook-Membrane.m	Two dimensional vector IGA code, solving a standard Cook Membrane displacement problem.	Used to compare results to a standard benchmark test.
IGA_2D_ SoapBubbles.m	Two dimensional scalar IGA code, solving a non-linear minimal surfaces problem, using Newton Raphson method.	To illustrate that IGA can also be used for non-linear problems in a very similar way to traditional FEM.

C.2 Function Files

Function files in alphabetical order.

Code Name	Code Description
basis_function_B.m	Function that calculates the given B-spline basis function.
basis_function_deriv_B.m	Function that calculates the derivative of a given B-spline basis function.
basis_function_deriv_R.m	Function that calculates the derivative of a given NURBS basis function.
basis_function_R.m	Function that calculates the NURBS basis function.
create_NURBS_1Dsca.m	Function that creates a one dimensional NURBS curve from a given control polygon in 1D, 2D or 3D space.
create_NURBS_2Dsca.m	Function that creates a two dimensional NURBS (scalar or vector) surface from a given control polygon in2D or 3D space.
gaussQuad.m	Outputs the required Gauss points and weights for a selected quadrature order.
hrefinement.m	Performs h-refinement on 1D NURBS curve.
hrefinementEta.m	Performs h-refinement on 2D NURBS surface in the Eta direction.
hrefinementXi.m	Performs h-refinement on 2D NURBS surface in the Xi direction.
krefinement1D.m	Performs k-refinement on 1D NURBS curve.
krefinement2D.m	Performs k-refinement on 2D NURBS surface.
prefinement1D.m	Performs p-refinement on 1D NURBS curve.
prefinement2D.m	Performs p-refinement on 2D NURBS surface.
xi_plot.m	Gives plotting points for the 1D control points so that it looks good when plotted with the NURBS curve. Holds no significant relevance to the NURBS curve; it is purely for aesthetics.

Appendix D

Sample Code

D.1 One-Dimensional Heat Conduction Code

```
%%1D IGA Radial Temperature code
%%Heidi Burger
clear all
% close all

%% Notes
%  This code is the same as the 2D temperature code. It is a 2D dimensional
%  doughnut shape, that can be modelled as a one dimensional problem.

%% Setup

%Refinement Details:
    refinementlevel = 0;  %h-refinement level
    p_target = 3;         %Target basis function order
    refinementType = 0;   %refinementType=0 for no order refinement
                          %refinementType=1 for p-refinement
                          %refinementType=2 for k-refinement

%Problem Boundaries
    xs = 1;                         %Domain start value
    xe = 4;                         %Domain end value
    source = @(x) -10;              %Source equation
    k = 10;                         %Conductance
    flux = 0;                       %Flux at Neumann condition
    funcExact = @(x) (xs^2/2)*(source(x)/k)*log(x)+xe.^2/4*source(x)/k-xs.^2/2*source(x)/k*log(xe)-x.^2/4*source
        (x)/k;      %Exact Solutioin for a outer edge temperature of 0.

%NURBS Variables
    p=2;                                %Basis function order
    xi_s=0;
    xi_e=1;
    xi_s_vec=ones(1,p)*xi_s;
    xi_e_vec=ones(1,p)*xi_e;
```

```
33
34      el_num=2;
35      middleXi=linspace(xi_s,xi_e,el_num+1);
36      Xi=[xi_s_vec middleXi xi_e_vec];    %Knot vector
37
38      n=length(Xi)-p-1;                   %Number of basis functions across domain and number of control points
39      P = [xs xs+(xe-xs)*linspace(1/(p*el_num),(p*el_num-1)/(p*el_num),n-2) xe]';    %Control points
40      w = [ 1 1 1 1]';                    %Control point weights
41
42  %Plotting Variables
43      ppoints = 50;                       %Number of plotting points
44      a = min(Xi);                        %Lower bound of Xi-vector
45      b = max(Xi);                        %Upper bound of Xi-vector
46      xi_vec = linspace(a,b,ppoints);     %Parametric vector for curve
47      [w_gp,Xi_bar]=gaussQuad(5);         %Gauss points
48
49  %% Refinement
50
51
52  %p- or k-refinement
53      if refinementType==1
54          [p,Xi,n,P,w] = prefinement1D(p,p_target,Xi,n,P,w);
55      else
56          if refinementType==2
57          [p,Xi,n,P,w] = krefinement1D(p,p_target,Xi,n,P,w);
58          else
59          end
60      end
61  %h-refinement
62      for refine=1:refinementlevel
63          [Xi,n,P,w] = hrefinement(p,Xi,n,P,w);
64      end
65
66      el_num=n-p;
67
68  %% Boundary Conditions
69  %Essential (dirichlet) nodes
70      n_E = [n];                  %Essential degrees of freedom (dofs)
71      d_E = [0]';                 %Dirichet conditions
72  %Free nodes
73      n_F = linspace(1,n,n);
74      n_F(n_E) = [];              %Free dofs
75  %Neumann nodes
76      n_flux=1;                %Elemental nodes on which flux occur. For [3], the flux is on the right side of
            the element
77
78  %% Main
79  %Initialise
80      K=zeros(n);                 %Initialise K
81      F=zeros(n,1);               %Initialise F
82  %Elemental Loops
83      for el=1:el_num
84          asm=el:el+p;
85
86          sz = length(asm);       %Number of basis functions in element
87          x = P(asm);             %Control points for element
```

```
88
89          K_e = zeros(sz);        %Initialise elemental K
90          F_e = zeros(sz,1);      %Initialise elemental F
91          F_eN= 0;                %Initialise elemental flux contribution F
92          i = el+p;               %Basis function number and element selector.
93          %Gauss Loop
94          for gp=1:length(Xi_bar)
95
96              xi = 0.5*( (Xi(i+1)-Xi(i))*Xi_bar(gp) + Xi(i+1) + Xi(i) );        %Parametric domain
97
98              R_dXi = zeros(1,sz);            %Derivative of Basis Functions Matrix
99              R = zeros(1,sz);                %Basis Functions Matrix
100             for nr=1:sz
101                 I = nr+el-1;
102                 R_dXi(nr) = basis_function_deriv_R(p,Xi,I,n,xi,w);
103                 R(nr) = basis_function_R(p,Xi,I,n,xi,w);
104             end
105
106             JXi = (R_dXi*x);                %Parametric jacobian
107             JXi_bar = 0.5*(Xi(i+1)-Xi(i));  %Isoparametric jacobian
108             B = JXi\R_dXi';                 %Derivatives of basis functions wrt x
109
110             x_source = create_NURBS_1Dsca(p,Xi,n,xi,w,P);           %Coordinate for Gauss point in physical
                    domain to be able to calculate the source.
111
112             K_e = K_e + JXi*JXi_bar*w_gp(gp)*k*x_source*(B*B');
113             F_e = F_e + JXi*JXi_bar*w_gp(gp)*source(x_source)*x_source*R';
114
115         end
116
117         K(asm,asm) = K(asm,asm) + K_e;
118         F(asm)     = F(asm) + F_e;
119
120         %Neumann Contribution.
121         if el==1           %Flux occurs on firstt element.
122             sz = length(asm);
123             x = P(asm);    %Control points for element
124             for gp=1:length(Xi_bar)
125
126                 xi = 0.5*[(Xi(i+1)-Xi(i))*Xi_bar(gp) + Xi(i+1) + Xi(i)];        %Parametric domain
127
128                 R_dXi = zeros(1,sz);        %Derivative of Basis Functions Matrix
129                 R = zeros(1,sz);            %Basis Functions Matrix
130                 for nr=1:sz
131                     I = nr+el-1;
132                     R_dXi(nr) = basis_function_deriv_R(p,Xi,I,n,xi,w);
133                 R(nr) = basis_function_R(p,Xi,I,n,xi,w);
134                 end
135
136                 JXi = (R_dXi*x);                %Parametric jacobian
137                 JXi_bar = 0.5*(Xi(i+1)-Xi(i));  %Isoparametric jacobian
138
139                 x_source = create_NURBS_1Dsca(p,Xi,n,xi,w,P);           %Coordinate for Gauss point in physical
                        domain if flux is a function of the spatial position.
140
141                 F_eN = F_eN + JXi*JXi_bar*w_gp(gp)*flux*R(n_flux)';
```

```
            end
        end
        F(asm(n_flux))=F(asm(n_flux))+F_eN;
    end

%% Solve
%Partion Method
    K_E=K(n_E,n_E);
    K_EF= K(n_E,n_F);
    K_F=K(n_F,n_F);

    F_E = F(n_E);
    F_F = F(n_F);

    d_F=K_F\(F_F-K_EF'*d_E);    %Solve for unknown d nodes
    d=zeros(n,1);
    d(n_E)=d_E;
    d(n_F)=d_F;            %Control polygon for field variables

%% Post Processing
%Draw field value NURBS vector
    c_vec = create_NURBS_1Dsca(p,Xi,n,xi_vec,w,d);
%Draw spatial NURBS vector
    x_vec = create_NURBS_1Dsca(p,Xi,n,xi_vec,w,P);

%% Plot
figure(1)
%     plot(x_vec,c_vec)
%     hold on
%     plot(x_vec,funcExact(x_vec),'k--','LineWidth',2)
%     plot(P,d,'ro-')
%     hold off
%     title('IGA Temperature Solution')
%     xlabel('Radial Direction')
%     ylabel('Temperature')
%     legend('IGA Solution','Exact Solution','Field Variable Control Polygon','Location','southoutside','
     Orientation','horizontal')
%     axis equal

    subplot(1,2,1)
    plot(x_vec,c_vec)
    hold on
    plot(x_vec,funcExact(x_vec),'k--','LineWidth',2)
    plot(P,d,'ro-')
    title('IGA Temperature Solution')
    xlabel('Radial Direction')
    ylabel('Temperature')
    legend('IGA Solution','Exact Solution','Field Variable Control Polygon')
    axis([1 4 -3.5 0])

subplot(1,2,2)
    plot(x_vec,c_vec-funcExact(x_vec))
    title('Error in Temp')
    xlabel('Radial Direction')
    ylabel('Temperature Difference')
    legend('Error')
```

```
%     axis([1 4 -1.5e-3 1.5e-3])


%% Error Analysis

    L2error=0;

    for el=1:el_num
        asm=el:el+p;

        sz = length(asm);       %Number of basis functions in element
        x = P(asm);             %Control points for element

        i = el+p;               %Basis function number and element selector.
        %Gauss Loop
        for gp=1:length(Xi_bar)

            xi = 0.5*( (Xi(i+1)-Xi(i))*Xi_bar(gp) + Xi(i+1) + Xi(i) );         %Parametric domain

            R_dXi = zeros(1,sz);            %Derivative of Basis Functions Matrix
            R = zeros(1,sz);                %Basis Functions Matrix
            for nr=1:sz
                I = nr+el-1;
                R_dXi(nr) = basis_function_deriv_R(p,Xi,I,n,xi,w);
                R(nr) = basis_function_R(p,Xi,I,n,xi,w);
            end

            JXi = (R_dXi*x);                %Parametric jacobian
            JXi_bar = 0.5*(Xi(i+1)-Xi(i));  %Isoparametric jacobian
            B = JXi\R_dXi';                 %Derivatives of basis functions wrt x

            x_source = create_NURBS_1Dsca(p,Xi,n,xi,w,P);           %Coordinate for Gauss point in physical
                 domain to be able to calculate the source.
            uex= funcExact(x_source);
            uh = create_NURBS_1Dsca(p,Xi,n,xi,w,d);

            L2error = L2error + JXi*JXi_bar*w_gp(gp)*((uex-uh).^2);

        end
    end
    L2error
```

D.2 Two-Dimensional Heat Conduction Code

```
%%2D IGA Hole in Plate Temperature code
%%Heidi Burger
clear all
% close all

%% Notes
%  There are also different refinement options. Change "refinementlevel"
%  for h-refinement and set the p_target, q_target and choose refinementType
%  for p-refinement.
```

```
%  This code can be compared to the FEM Hole in Plate Temperature code

%% Setup

%Refinement Details
    refinementlevel = 0;    %h-refinement level
    p_target = 3;           %Target basis function order for the Xi direction
    q_target = 3;           %Target basis function order for the Eta direction
    refinementType = 0;     %refinementType=0 for no order refinement
                            %refinementType=1 for p-refinement

%Problem Boundaries
    inner_radius = 1;               %Radial start value
    outer_edge = 4;                 %Edge end value
    k = 1 ;                 %Conductance (W/m^2/K)
    D=[k 0;0 k];
    source=@(x,y) k*(2.*inner_radius./sqrt(x.^2+y.^2)-4);      %Source term

    flux=@(x,y) k*2*outer_edge.*(inner_radius./sqrt(x.^2+y.^2)-1);                    %Flux function
    analytical=@(x,y) x.^2 +y.^2 -2.*inner_radius.*sqrt(x.^2+y.^2) +inner_radius.^2;

%NURBS Variables
    p = 2;
    q = 2;
    xi_s = 0;
    xi_e = 1;
    eta_s = 0;
    eta_e = 1;
    xs_vec = ones(1,p)*xi_s;
    ys_vec = ones(1,q)*eta_s;
    xe_vec = ones(1,p)*xi_e;
    ye_vec = ones(1,q)*eta_e;
    Xi = [xs_vec linspace(xi_s,xi_e,3) xe_vec];         %Knot Vector in Xi direction
    Eta = [ys_vec linspace(eta_s,eta_e,2) ye_vec];         %Knot Vector in Eta direction

    n = length(Xi)-p-1;           %Number of basis functions for Xi
    m = length(Eta)-q-1;          %Number of basis functions for EtaD

    P=[-inner_radius 0,                       -((outer_edge-inner_radius)/2+inner_radius) 0,     -outer_edge 0
       -inner_radius inner_radius*(sqrt(2)-1),      -((outer_edge-inner_radius)/2+inner_radius) inner_radius
            *0.75,  -outer_edge outer_edge
       -inner_radius*(sqrt(2)-1) inner_radius,     -inner_radius*0.75 ((outer_edge-inner_radius)/2+inner_radius
            ),  -outer_edge outer_edge
       0 inner_radius,                        0 ((outer_edge-inner_radius)/2+inner_radius),      0 outer_edge];

    w=[ 1 1 1
        (1+1/sqrt(2))/2 1 1
        (1+1/sqrt(2))/2 1 1
        1 1 1];
    dim = 2;

%Plotting Variables
    ppoints = 100;                  %Number of plotting points
    a = min(Xi);                      %Lower bound of Xi-vector
    b = max(Xi);                      %Upper bound of Xi-vector
    aa = min(Eta);                    %Lower bound of Eta-vector
```

```
    bb = max(Eta);                    %Upper bound of Eta-vector
    xi_vec = linspace(a,b,ppoints); %Parametric vector for curve in Xi direction
    eta_vec = linspace(aa,bb,ppoints); %Parametric vector for curve in Eta direction
    [w_gp,Xi_bar,Eta_bar] = gaussQuad(5); %Gauss points

%% Refinement
%h-refinement
    for refine=1:refinementlevel
        [Eta,m,P,w]=hrefinementEta(q,Eta,n,m,P,w,dim);
    end     % End of refine=refinementlevels loop
    for refine=1:refinementlevel
        [Xi,n,P,w]=hrefinementXi(p,Xi,n,m,P,w,dim);
    end     % End of refine=refinementlevels loop

%p-refinement
%     if refinementType==1
%         [Xi,Eta,p,q,n,m,P,w]=prefinement2D(Xi,Eta,p,q,n,m,P,w,dim,p_target,q_target);
%     end

    if refinementType==1
        [Xi,Eta,p,q,n,m,P,w]=prefinement2D(Xi,Eta,p,q,n,m,P,w,dim,p_target,q_target);
    else if refinementType==2
        [Xi,Eta,p,q,n,m,P,w]=krefinement2D(Xi,Eta,p,q,n,m,P,w,dim,p_target,q_target);
         end
    end

%% Code Preperation
    Pvec=zeros(n*m,dim);
    for j=1:m                   %Loop to convert B into a vector of pairs
        Pvec(n*(j-1)+1:n*j,[1:dim])=P(1:n,[dim*j-(dim-1):dim*j]);
        wvec(n*(j-1)+1:n*j,1)=w(1:n,j);
    end

    nn = n*m;                   %Total number of nodes in the domain
    el_numN = n-p;              %Total number of elements in the xi-direction
    el_numM = m-q;              %Total number of elements in the eta-direction
    el_tot = el_numN*el_numM;   %Total number of elements in the domain

%Connectivity Array
    NODE=zeros(el_tot,(p+1)*(q+1));
    c=0;
    numNrow=0;
    for i=1:el_tot
        numNrow=numNrow+1;
        c=c+1;
        node=zeros(1,(p+1)*(q+1));
        for j=1:q+1
            node(j*(p+1)-p:j*(p+1))=(c+(j-1)*(n)):((c+(j-1)*(n))+p);
        end
        NODE(i,:)=node;
        if rem(numNrow,el_numN)==0
            c=c+p;
            numNrow=0;
        end
    end

```

```
%% Boundary Conditions
%Essential (dirichlet) nodes
    n_E=[1:n n*m-n+1:n*m-n/2+1];
    d_E=zeros(size(n_E))';
    count=n;
    for i=1:length(Pvec)
        if Pvec(i,1)==-outer_edge
            count=count+1;
            d_E(count)=analytical(Pvec(i,1),Pvec(i,2));
        end
    end
%Free nodes
    n_F = linspace(1,nn,nn);
    n_F(n_E) = [];              %Free dofs
%Neumann nodes
    n_flux=zeros(size(n*m-n+2:n*m-n/2+1));      %Dof numbers that contain flux conditions.
    count=0;
    for i=1:length(Pvec)
        if Pvec(i,2)==outer_edge
            count=count+1;
            n_flux(count)=[i];
        end
    end
    n_flux_side=(p+1)*(q+1)-p:(p+1)*(q+1);
    el_flux=el_numM*el_numN-(n/2+1-p)+1:el_numM*el_numN;  %Elements which have flux sides.

%% Main
%Initialise Matrices
    K=zeros(nn,nn);             %Initialise global stiffness matrix
    F=zeros(nn,1);              %Initialise global force matrix
    F_flux=zeros(nn,1);         %Initialise global flux foce matrix
    d=zeros(nn,1);              %Initialise global displacement matrix
    szN=p+1;                    %Number of basis functions in element in xi direction
    szM=q+1;                    %Number of basis functions in element in eta direction
    sz = p+1;       %Number of basis functions required for each element for the flux component

%Elemental Loop
    el=0;
    for elM=1:el_numM                   %eta direction elemental loop
        for elN=1:el_numN               %xi direction elemental loop
            el=el+1;                    %Global element number
            asm=NODE(el,:);             %Elemental connectivity matrix

            x=Pvec(asm,1);              %x control points associated with element
            y=Pvec(asm,2);              %y control points associated with element

            K_e=zeros(szN*szM);                     %Initialise elemental stiffness matrix
            F_e=zeros(szN*szM,1);                     %Initialise elemental source force matrix
            F_eN=zeros(szN,1);                    %Initialise elemental flux force matrix
            i=elN+p;                    %Knot span selector in xi direction. Selects first and second (i+1) knot
                  of element
            j=elM+q;                    %Knot span selector in eta direction. Selects first and second (i+1)
                 knot of element

            for gp2=1:length(Eta_bar)           %Second Gauss Quadrature loop
                for gp1=1:length(Xi_bar)        %First Gauss Quadrature loop
```

```
174
175                     xi = 0.5*((Xi(i+1)-Xi(i))*Xi_bar(gp1) + Xi(i+1) + Xi(i));               %Parametric value of
                            Gauss point (xi direction)
176                     eta = 0.5*((Eta(j+1)-Eta(j))*Eta_bar(gp2) + Eta(j+1) + Eta(j));         %Parametric value of
                            Gauss point (eta direction)
177
178                     N=zeros(1,szN*szM);             %Elemental basis function matrix for xi direction
179                     N_dXi=zeros(1,szN*szM);         %Elemental basis function derivative matrix for xi direction
180                     count=0;
181                     for nr0=0:szM-1
182                         wselect=n*nr0+(elM-1)*n+(1:n); %Selects the associated row or column of weights to use
                                with that control point.
183 %                       wselect
184                         for nr=1:szN
185                             count=count+1;
186                             I = nr+elN-1;           %Shape function number in Xi direction, not global, eg. 1:3
187 %                             N(szM*nr0+nr)=basis_functR(p,Xi,I,n,xi,wvec(wselect,1));  %Shape function number
    in elemental coords, eg. 1:9 and 4:12
188 %                             N_dXi(szM*nr0+nr)=basis_funct_derivR(p,Xi,I,n,xi,wvec(wselect,1));
189                             N(count)=basis_function_R(p,Xi,I,n,xi,wvec(wselect,1));  %Shape function number in
                                    elemental coords, eg. 1:9 and 4:12
190                             N_dXi(count)=basis_function_deriv_R(p,Xi,I,n,xi,wvec(wselect,1));
191                         end
192                     end
193
194                     M=zeros(1,szN*szM);             %Elemental basis function matrix for eta direction
195                     M_dEta=zeros(1,szN*szM);        %Elemental basis function derivative matrix for eta
                            direction
196                     count=0;
197                     for nr=1:szM
198                         I = nr+elM-1;
199                         for nr0=0:szN-1
200                             count=count+1;
201                             wselect=zeros(1,m);  %Selects the associated row or column of weights to use with
                                    that control point.
202                             for tt=0:m-1
203                                 wselect(tt+1)=((nr0+1)+tt*n+(elN-1));
204                             end
205
206                             M(count)=basis_function_R(q,Eta,I,m,eta,wvec(wselect,1));        %Check if szM or m
                                    for 1:m
207                             M_dEta(count)=basis_function_deriv_R(q,Eta,I,m,eta,wvec(wselect,1));
208                         end
209                     end
210
211                     nr=0;
212                     R=zeros(size(asm));         %Vector of basis function products
213                     dR_dXi=zeros(size(asm));    %Derivative of basis function product vector wrt to xi
214                     dR_dEta=zeros(size(asm));   %Derivative of basis function product vector wrt to eta
215                     for nrG=1:szN*szM
216                         nr=nr+1;
217                         R(nr)=N(nrG)*M(nrG);
218                         dR_dXi(nr)=N_dXi(nrG)*M(nrG);
219                         dR_dEta(nr)=N(nrG)*M_dEta(nrG);
220                     end
221
```

```
222                 JXi=[dR_dXi*x, dR_dEta*x;                          %Parametric Jacobian
223                      dR_dXi*y, dR_dEta*y];
224                 JXi_bar=0.25*(Xi(i+1)-Xi(i))*(Eta(j+1)-Eta(j));     %Isoparametric jacobian
225
226                 B=([dR_dXi;dR_dEta]'/JXi)';                    %Derivatives of basis functions wrt to x and
                         y
227
228                 [c_source]=create_NURBS_2Dsca(p,q,Xi,Eta,n,m,xi,eta,w,Pvec);        %Gauss points in
                         physical domain
229
230                 K_e = K_e + JXi_bar*det(JXi)*w_gp(gp1)*w_gp(gp2)*(B'*D*B);
231                 F_e = F_e + JXi_bar*det(JXi)*w_gp(gp1)*w_gp(gp2)*source(c_source(1),c_source(2))*R';
232
233             end
234         end
235
236         %Global assembly
237         K(asm,asm) = K(asm,asm) + K_e;
238         F(asm)     = F(asm) + F_e;
239
240         %%For the Flux term
241         if ismember(el,el_flux)          %Sides on which flux term appears.% Or use: ismember(el,el_flux) %
                 elM==1
242             xx = Pvec(asm(n_flux_side),1);    %Control points for element. (Similar to x and y above)
243             yy = Pvec(asm(n_flux_side),2);    %Control points for element. (Similar to x and y above)
244             for gp=1:length(Xi_bar)
245
246                 xi = 0.5*((Xi(i+1)-Xi(i))*Xi_bar(gp) + Xi(i+1) + Xi(i));         %Parametric domain
247                 eta=1;
248
249                 N_dEta = zeros(1,sz);         %Derivative of Basis Functions Matrix
250                 N = zeros(1,sz);             %Basis Functions Matrix
251                 for nr=1:sz
252                     I = nr+elN-1;
253                     wselect=nn-n+1:nn;
254                     N(nr) = basis_function_R(p,Xi,I,n,xi,wvec(wselect));
255                     N_dEta(nr) = basis_function_deriv_R(p,Xi,I,n,xi,wvec(wselect));
256                 end
257
258                 JEta = (N_dEta*xx);                 %Parametric jacobian
259                 JEta_bar = 0.5*(Xi(i+1)-Xi(i));   %Isoparametric jacobian
260
261             %NURBS code to determine value of Gauss point in physical domain
262                 [c_source]=create_NURBS_2Dsca(p,q,Xi,Eta,n,m,xi,eta,w,Pvec);
263
264             %Flux equation
265                 F_eN = F_eN + JEta*JEta_bar*w_gp(gp)*flux(c_source(1),c_source(2))*N';      %Neumann
                         contribution to Force matrix
266
267             end
268             F_flux(asm(n_flux_side))=F_flux(asm(n_flux_side))-F_eN;
269         end
270     end
271 end
272 F(n_flux)=F(n_flux)+F_flux(n_flux);
273
```

```
%% Solve
%Partition Method
    K_E=K(n_E,n_E);
    K_EF= K(n_E,n_F);
    K_F=K(n_F,n_F);

    F_E = F(n_E);
    F_F = F(n_F);

    d_F=K_F\(F_F-K_EF'*d_E);    %Solve for unknown d nodes
    d(n_E)=d_E;
    d(n_F)=d_F;            %Control polygon for field variables

%% Post Processing
%Create NURBS vector of spatial dimensions
    c_vec = create_NURBS_2Dsca(p,q,Xi,Eta,n,m,xi_vec,eta_vec,w,Pvec);
    c_flux_vec = create_NURBS_2Dsca(p,q,Xi,Eta,n,m,linspace(xi_s,xi_e,50),linspace(eta_s,eta_e,50),w,Pvec);
%Create NURBS vector of temperature distribution
    temp_vec = create_NURBS_2Dsca(p,q,Xi,Eta,n,m,xi_vec,eta_vec,w,d);
%Create NURBS vector of temperature flux distribution
    flux_vec = create_NURBS_deriv_2Dsca(p,q,Xi,Eta,n,m,linspace(xi_s,xi_e,50),linspace(eta_s,eta_e,50),w,Pvec,d)
        ;

%% Plot

figure(1)
subplot(2,2,1)  %IGA temperature plot across domain
    scatter3(c_vec(:,1),c_vec(:,2),temp_vec,10,temp_vec,'filled')
    title('IGA Temperature Plot')
    xlabel('X')
    ylabel('Y')
    zlabel('Temperature')
    view([30 30])
    set(gca, 'DataAspectRatio', [diff(get(gca, 'XLim')) diff(get(gca, 'YLim')) diff(get(gca, 'ZLim'))])

subplot(2,2,2)  %Analytical temperature plot across domain
   temp_analytic_vec=analytical(c_vec(:,1),c_vec(:,2));
   scatter3(c_vec(:,1),c_vec(:,2),temp_analytic_vec,10,temp_analytic_vec,'filled')

    title('Analytical Temperature Plot')
    xlabel('X')
    ylabel('Y')
    zlabel('Temperature')
    view([30 30])
    set(gca, 'DataAspectRatio', [diff(get(gca, 'XLim')) diff(get(gca, 'YLim')) diff(get(gca, 'ZLim'))])

    error = -(temp_analytic_vec-temp_vec) ;
    temp_mean=mean(temp_vec);

subplot(2,2,3)  %Error (difference between analytical and IGA) plot across domain
    scatter3(c_vec(:,1),c_vec(:,2),error,10,error,'filled')
    title('Error Plot')
    xlabel('X')
    ylabel('Y')
    zlabel('Error')
    view([30 30])
```

```
    set(gca, 'DataAspectRatio', [diff(get(gca, 'XLim')) diff(get(gca, 'YLim')) diff(get(gca, 'ZLim'))])

subplot(2,2,4)  %Error (difference between analytical and IGA) plot as a percentage of the analytical solution
    across domain
%     scatter3(c_vec(:,1),c_vec(:,2),error/temp_mean,10,error,'filled')
    scatter3(c_vec(:,1),c_vec(:,2),error./temp_analytic_vec,10,error,'filled')
    title('Error Plot (%)')
    xlabel('X')
    ylabel('Y')
    zlabel('Error (%)')
    view([30 30])
    set(gca, 'DataAspectRatio', [diff(get(gca, 'XLim')) diff(get(gca, 'YLim')) diff(get(gca, 'ZLim'))])

   figure(2)   %Temperature flux plot across domain
    quiver(c_flux_vec(:,1),c_flux_vec(:,2),flux_vec(:,1),flux_vec(:,2),2)
    axis equal
    title('Flux Distribution across Domain')
    xlabel('X-Direction')
    ylabel('Y-Direction')
    zlabel('Temperature')

figure(3)   %Temperature plot across domain with contour lines
    xlin = linspace(min(c_vec(:,1)),max(c_vec(:,1)),200);
    ylin = linspace(min(c_vec(:,2)),max(c_vec(:,2)),200);
    [X,Y] = meshgrid(xlin,ylin);

    Z = griddata(c_vec(:,1),c_vec(:,2),temp_vec,X,Y,'cubic');

    s = surf(X,Y,Z);
    s(1).EdgeColor = 'none';
    view([0 90])
    set(gca, 'DataAspectRatio', [diff(get(gca, 'XLim')) diff(get(gca, 'YLim')) diff(get(gca, 'ZLim'))])
    colorbar
    hold on
    contour3(X,Y,Z+0.01,30,'Color','red');
    hold off
    axis equal
    title('Plot of Temperature')
    xlabel('X-Direction')
    ylabel('Y-Direction')
    zlabel('Temperature')

    %Draw the hole for aesthetics
    eta_circle_vec = 0;
    [uuc_vec] = create_NURBS_2Dsca(p,q,Xi,Eta,n,m,xi_vec,eta_circle_vec,w,d);
    [c_hole_vec] = create_NURBS_2Dsca(p,q,Xi,Eta,n,m,xi_vec,eta_circle_vec,w,Pvec);
    hole=c_hole_vec;
    hole = [hole; 0,0; hole(1,:)];

    p1 = patch(hole(:,1),hole(:,2),max(temp_vec)*ones(size(hole,1),1),'w');
    p1.EdgeColor = 'none';

figure(4)   %Control points plot
    plot(Pvec(:,1),Pvec(:,2),'ro','MarkerFaceColor','r')
    axis equal
    hold on
```

```
384     for i=1:length(Pvec)
385                 if i==(n*m-n/2)
386                     nNr=text(Pvec(i,1)-0.3,Pvec(i,2)+0.02,num2str(i),'FontSize',12);
387                 else
388                     nNr=text(Pvec(i,1)+0.01,Pvec(i,2)+0.02,num2str(i),'FontSize',12);
389                 end
390             end
391     hold off
392     title('Control Points plot')
393     xlabel('X-Direction')
394     ylabel('Y-Direction')
395
396 %% Error Analysis
397
398 L2error=0;
399 el=0;
400     for elM=1:el_numM                   %eta direction elemental loop
401         for elN=1:el_numN               %xi direction elemental loop
402             el=el+1;                    %Global element number
403             asm=NODE(el,:);             %Elemental connectivity matrix
404
405             x=Pvec(asm,1);              %x control points associated with element
406             y=Pvec(asm,2);              %y control points associated with element
407
408             i=elN+p;                    %Knot span selector in xi direction. Selects first and second (i+1) knot
                      of element
409             j=elM+q;                    %Knot span selector in eta direction. Selects first and second (i+1)
                     knot of element
410
411             for gp2=1:length(Eta_bar)           %Second Gauss Quadrature loop
412                 for gp1=1:length(Xi_bar)        %First Gauss Quadrature loop
413
414                     xi = 0.5*((Xi(i+1)-Xi(i))*Xi_bar(gp1) + Xi(i+1) + Xi(i));              %Parametric value of
                            Gauss point (xi direction)
415                     eta = 0.5*((Eta(j+1)-Eta(j))*Eta_bar(gp2) + Eta(j+1) + Eta(j));        %Parametric value of
                            Gauss point (eta direction)
416
417                     N=zeros(1,szN*szM);             %Elemental basis function matrix for xi direction
418                     N_dXi=zeros(1,szN*szM);         %Elemental basis function derivative matrix for xi direction
419                     count=0;
420                     for nr0=0:szM-1
421                         wselect=n*nr0+(elM-1)*n+(1:n); %Selects the associated row or column of weights to use
                               with that control point.
422                         for nr=1:szN
423                             count=count+1;
424                             I = nr+elN-1;           %Shape function number in Xi direction, not global, eg. 1:3
425                             N(count)=basis_function_R(p,Xi,I,n,xi,wvec(wselect,1));  %Shape function number in
                                   elemental coords, eg. 1:9 and 4:12
426                             N_dXi(count)=basis_function_deriv_R(p,Xi,I,n,xi,wvec(wselect,1));
427                         end
428                     end
429
430                     M=zeros(1,szN*szM);             %Elemental basis function matrix for eta direction
431                     M_dEta=zeros(1,szN*szM);        %Elemental basis function derivative matrix for eta
                           direction
432                     count=0;
```

```
                for nr=1:szM
                    I = nr+elM-1;
                    for nr0=0:szN-1
                        count=count+1;
                        wselect=zeros(1,m);  %Selects the associated row or column of weights to use with
                             that control point.
                        for tt=0:m-1
                            wselect(tt+1)=((nr0+1)+tt*n+(elN-1));
                        end

                        M(count)=basis_function_R(q,Eta,I,m,eta,wvec(wselect,1));        %Check if szM or m
                             for 1:m
                        M_dEta(count)=basis_function_deriv_R(q,Eta,I,m,eta,wvec(wselect,1));
                    end
                end

                nr=0;
                R=zeros(size(asm));         %Vector of basis function products
                dR_dXi=zeros(size(asm));    %Derivative of basis function product vector wrt to xi
                dR_dEta=zeros(size(asm));   %Derivative of basis function product vector wrt to eta
                for nrG=1:szN*szM
                    nr=nr+1;
                    R(nr)=N(nrG)*M(nrG);
                    dR_dXi(nr)=N_dXi(nrG)*M(nrG);
                    dR_dEta(nr)=N(nrG)*M_dEta(nrG);
                end

                JXi=[dR_dXi*x, dR_dEta*x;                           %Parametric Jacobian
                     dR_dXi*y, dR_dEta*y];
                JXi_bar=0.25*(Xi(i+1)-Xi(i))*(Eta(j+1)-Eta(j));     %Isoparametric jacobian

                B=([dR_dXi;dR_dEta]'/JXi)';                   %Derivatives of basis functions wrt to x and
                     y

                [c_source]=create_NURBS_2Dsca(p,q,Xi,Eta,n,m,xi,eta,w,Pvec);        %Gauss points in
                     physical domain
                uex = analytical(c_source(1),c_source(2));
                uh = create_NURBS_2Dsca(p,q,Xi,Eta,n,m,xi,eta,w,d);

                L2error = L2error + JXi_bar*det(JXi)*w_gp(gp1)*w_gp(gp2)*((uex-uh).^2);

            end
        end
    end
end
L2error
```