National Technical University of Athens

School of Mechanical Engineering

Section: Mechanical Design & Control Systems

I3GA: An Integrated, Interactive Isogeometric Analysis tool in MATLAB for 2D problems

Diploma Thesis

Dimitrios Tolis

Supervision:

Christopher Provatidis (Professor NTUA),

Ioannis Dimitriou (PhD Candidate NTUA)

Revision: Ioannis Dimitriou

Athens, October 2023

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Μηχανολόγων Μηχανικών

Τομέας: Μηχανολογικών Κατασκευών & Αυτομάτου Ελέγχου

I3GA: Ένα Ολοκληρωμένο και Διαδραστικό εργαλείο Ισογεωμετρικής Ανάλυσης στο περιβάλλον της MATLAB για δισδιάστατα προβλήματα

Διπλωματική Εργασία

Δημήτριος Τόλης

Επίβλεψη:

Χριστόφορος Προβατίδης (Καθηγητής ΕΜΠ),

Ιωάννης Δημητρίου (Υποψήφιος Διδάκτορας ΕΜΠ)

Αναθεώρηση: Ιωάννης Δημητρίου

Αθήνα, Οκτώβριος 2023

Contents

In loving memory of my grandfather,

a beautiful butterfly that I will always remember,

and to those people who struggle every day with their own demons…

To a future based on humanity and science…a better one

Acknowledgements

After a difficult period in my life, which made me much stronger, I managed to finish my thesis in a very interesting and state-of-the-art topic that of Isogeometric Analysis. Discovering the mysteries of Isogeometric Analysis, not only I expanded my horizons in aspects of Computational Mechanics and Geometry but also, I learned how to research. That process was very intriguing and made it clear that I would like to follow the path of researcher in the future…

First, I would like to thank my professor Christopher Provatidis, for his patience and chance he gave me to enter into this interesting topic. From his questions and ideas, I examined some interesting parts of Isogeometric Analysis. I hope that this thesis meets his expectations.

Next, I would like to thank my colleague and mentor Ioannis Dimitriou. His numerous ideas, observations, and questions were important in order to dive deep into the topic and complete my thesis. His help was precious and pivotal. His passion and countless hours of discussion not only was driving me out during strange times for my life, but also taught me that a true scientist lives for his research and do not live from it. I am thankful that I met this man and I hope to continue cooperate with him in academic and personal level.

I would like also thank people who helped me during my studies at NTUA and especially my two friends Michalis Tsagaris and Matthaios Chantzopoulos. Without them the school would not have been enjoyable. I wish them to have a good life and I hope to keep them in mine because their spirit and mentality is rare.

I would like to thank my family for the patience, the ethics that taught me and their constant support throughout the years of my life. Special thanks also to my friends and relatives that helped me in this period.

Finally, I would like to thank my grandpa Dimitrios Trigazis, not only for been so caring with me but also for teaching me the principles of studying and loving science in every aspect of human life. I hope that he watches me and continues to be on my side from “above” and someday he will be proud for his grandson…

Abstract

Nowadays, the most common way of dealing with engineering problems (elasticity, heat transfer, etc.) is to firstly model the problem within a CAD system and then use Finite element method. However, this method shows inevitable errors because the model's geometry is approximated and does not follow exact geometry with accuracy. In order to eliminate these problems, the method of Isogeometric analysis has been used in the last 15 years. The concept behind isogeometric analysis is that the model's mesh is the same as the computational; thus, the analysis process is faster and exact. However, codes that provide an integrated CAD/CAE environment do not exist. The goal of this thesis is the creation of a GUI tool in MATLAB environment where the user can create an initial geometry, interacts freely with it, and then executes isogeometric analysis using NURBS basis functions. The program was designed to solve 2D elasticity and heat transfer problem, not only using Isogeometric analysis with NURBS but also with Bézier elements via Bézier extraction method. The latter is a method that decomposes the initial geometry into C^0- elements. Furthermore, refinement methods were coded in order to study the mesh accuracy. H- and p- refinements were programmed in which the knots are inserted immediately in knot vectors or interactively on surface. Moreover, the code runs with different forms (constant value or function) of boundary conditions Dirichlet or Neumann. The code was tested using benchmark problems where different kind of refinement were studied. For the studies it was concluded that p-refinement gives satisfying error for less control points but with higher running time than h-refinement. Finally, mesh Adaptivity was studied, where two adaptivity algorithms were programmed for a specific problem.

Περίληψη

Στην εποχή μας, η συνήθης διαδικασία προσομοίωσης ενός προβλήματος μηχανικού (ελαστικότητα, μετάδοση θερμότητας κτλ.) γίνεται με τη χρήση της μεθόδου των πεπερασμένων στοιχείων, έχοντας πρώτα μοντελοποιήσει το πρόβλημα σε ένα σύστημα CAD. Ωστόσο, η μέθοδος αυτή παρουσιάζει αναπόφευκτα σφάλματα, καθώς προσεγγίζει και δεν ακολουθεί με ακρίβεια τη γεωμετρία του μοντέλου. Για την απάλειψη αυτών των σφαλμάτων τα τελευταία χρόνια χρησιμοποιείται η μέθοδος της Ισογεωμετρικής ανάλυσης κι ιδιαίτερα στη περίπτωση που οι συναρτήσεις βάσης είναι NURBS. Το νόημα της μεθόδου είναι ότι το πλέγμα που χρησιμοποιείται για τη γεωμετρία του μοντέλου είναι το ίδιο με αυτό της ανάλυσης. Ωστόσο δεν υπάρχουν κώδικες που να δημιουργούν ένα ολοκληρωμένο περιβάλλον CAD/CAE. Στόχος της διπλωματικής είναι η δημιουργία ενός εργαλείου GUI στο περιβάλλον του MATLAB στο οποίο ο χρήστης δημιουργεί μια αρχική γεωμετρία, αλληλοεπιδρά ελεύθερα μαζί της και εν συνεχεία πραγματοποιεί την ισογεωμετρική ανάλυση χρησιμοποιώντας συναρτήσεις NURBS. Το πρόγραμμα σχεδιάστηκε να λύνει δισδιάστατα προβλήματα θερμότητας και επίπεδης ελαστικότητας, όχι μόνο με τη μέθοδο της Ισογεωμετρικής ανάλυσης χρησιμοποιώντας NURBS αλλά και με την εξαγωγή Bézier η οποία αποσυνθέτει τη γεωμετρία σε C^0 στοιχεία. Επίσης προγραμματίστηκαν διαφορετικές τεχνικές εκλέπτυνσης πλέγματος για μεγαλύτερη ακρίβεια. Δοκιμάστηκαν οι h-,p- εκλεπτύνσεις οπού οι κόμβοι εισάγονται είτε στα κομβοδιανύσματα είτε διαδραστικά κατευθείαν πάνω στη γεωμετρία. Επίσης ο κώδικάς προγραμματίστηκε να τρέχει με διαφορετικές μορφές (σταθερή τιμή ή συνάρτηση) συνοριακών συνθηκών Dirichlet ή Neumann. Σε ότι αφορά τα αποτελέσματα ο κώδικας μελετήθηκε σε κλασικά benchmark προβλήματα όπου μελετήθηκε η επίδραση διαφορετικού είδους εκλεπτύνσεων στο πλέγμα. Από τη μελέτη διαπιστώθηκε ότι η p-εκλέπτυνση δίνει αρκετά ικανοποιητικό σφάλμα για λιγότερα σημεία ελέγχου αλλά ελάχιστα μεγαλύτερο χρόνο προσομοίωσης σε σχέση με τη h-εκλέπτυνση. Τέλος, μελετήθηκε η Προσαρμοστικότητα του πλέγματος, όπου αναπτύχθηκαν δύο αλγόριθμοι προσαρμογής του πλέγματος για ένα συγκεκριμένο πρόβλημα.

Abbreviations

PDE: Partial Differential Equation

BC: Boundary Condition

FEM: Finite Element Method

FEA: Finite Element Analysis

IGA: IsoGeometric Analysis

CAD: Computer-Aided Design

CAE: Computer-Aided Engineering

NURBS: Non-Uniform Rational B-splines

1 Introduction

1.1 Motivation

Engineering problems are physical problems (e.g., solids, structures, fluids etc.); therefore, most of them are described by differential equations and in particular partial differential equations (**PDEs**). These PDEs describe the advancement of physical phenomena in a problem domain Ω and on its boundary (denoted as Γ or $\partial\Omega$). For example some problems could be the stress or thermal analysis of a structural component, the fluid flow on vessels (i.e., aerodynamic problems on airplanes, cars, ships, etc.) or even medical simulations of the human circulatory system [1]. Within the domain Ω the governing PDE is generally described by (the strong form) [2]:

$$D\big(\mathbf{u}(\mathbf{x},t)\big) = \mathbf{f}(\mathbf{x},t) \quad \text{in } \Omega \tag{1.1}$$

where $D(\cdot)$ denotes a differential operator, $\mathbf{u}(\mathbf{x},t)$ is the vector of *state* or *field* variables of problem (e.g., displacements, temperature, pressure etc.) and $\mathbf{f}(\mathbf{x},t)$ is the vector of external loads (*source*) terms that provokes the phenomena in problem domain. In boundary, the boundary conditions of PDE exist, which can take one of forms below (see fig 1.1):

Boundary Conditions (**BCs**):

- $\mathbf{u} = \bar{\mathbf{u}}$ in Γ_1 (*Dirichlet* BC)
- $\frac{\partial \mathbf{u}}{\partial \boldsymbol{n}} = \bar{\mathbf{q}}$ in Γ_2 (*Neumann* BC)
- $a\mathbf{u} + \beta \frac{\partial \mathbf{u}}{\partial \boldsymbol{n}} = \bar{\mathbf{q}}$ in Γ_3 (*Robin* BC)

where, $\bar{\mathbf{u}}, \bar{\mathbf{q}}$ are vectors of specific values, a, β are coefficients and $\boldsymbol{n}$ is the normal vector on boundary Γ.

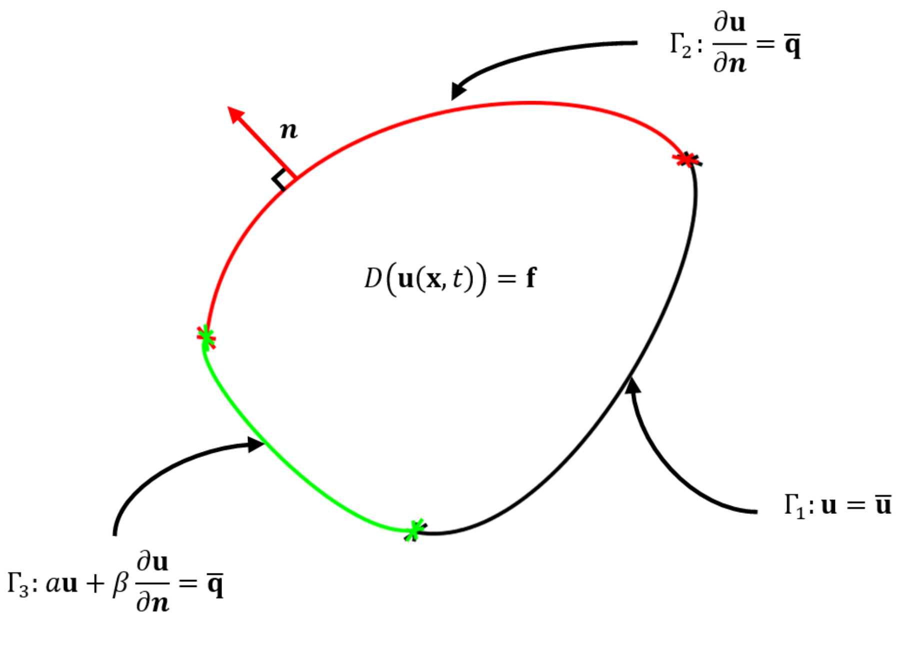

$\Gamma_2: \frac{\partial \mathbf{u}}{\partial \boldsymbol{n}} = \bar{\mathbf{q}}$
$\boldsymbol{n}$
$D\big(\mathbf{u}(\mathbf{x},t)\big) = \mathbf{f}$
$\Gamma_1: \mathbf{u} = \bar{\mathbf{u}}$
$\Gamma_3: a\mathbf{u} + \beta \frac{\partial \mathbf{u}}{\partial \boldsymbol{n}} = \bar{\mathbf{q}}$

Figure 1.1: The problem domain Ω along with boundary conditions.

In general case, PDEs have not analytical solution, as a result numerical methods have been adopted in order to give approximate but precise solutions. These numerical solutions due to problems' complexity are executed by computers. The whole analysis process is called *Computer-aided Engineering* (**CAE**). Nowadays, there is a huge variety of methods used in CAE, for example finite elements, boundary elements, collocation methods or newest ones as mesh-free, and point methods [2].

Finite element method or analysis (**FEM** or **FEA**) which was originally developed in mid- 20th century is today the prevailing method for numerical solution on PDEs [3]. The method is based on the discretization of problem domain into finite elements which all together form a computational mesh [2], on which the PDE is solved approximately. Each finite element consists of *nodal points* (or *nodes*), which form simple shapes (e.g., triangles and quadrilaterals, tetrahedrals, hexahedrals etc.) with respect to problem dimension (e.g., $\Omega \cup \Gamma \subset \mathbb{R}^n$) [2]. These shapes are usually defined mathematically in terms of interpolatory polynomials (Lagrange or Hermite) [4].

In order to use FEA, the problem needs firstly to be represented geometrically by a designer. This modelling process is commonly embodied at the computers using *Computer- aided design* (**CAD**) as a tool. At most cases, CAD systems construct the geometry by "interpolating" or "regressing" points using shape functions (i.e., polynomials), creating a *mesh*. The most common of these polynomials for creating surfaces are the *Non-uniform rational B-splines* (**NURBS**) [3], [4]. After geometry is created it needs to be imported into FEA system that solves the problem. In FEA formulation the solution space approximates the initial model space $\Omega \cup \Gamma$ with different polynomials from CAD (e.g., Hermite or Lagrange Polynomials).

However a FEM mesh is created when the CAD model is translated into an analysis-suitable geometry [5]. This process of conversion is trivial and not automatically generated. According to industry two steps are needed for this process. The first step of conversion would be to create a model in which analysis is possible (20% of total analysis time), while the second step would involve creating a model which is suitable (80% of total analysis time). This totals an 80/20 factor of conversion process versus actual analysis. Except this bottleneck, recent trends taking place in engineering analysis and high-performance computing are also demanding greater precision [5]. The classic CAD-to-FEA analysis method does not ensure exact geometrical representations. The FEA mesh is only an approximation of CAD geometry, which is viewed as "exact" [4]. This approximation can lead to errors, for example in shell buckling analysis problems solution space must be precise because they are very sensitive to geometric imperfections [4]. In fig 1.2 we can see schematically the error between FEA mesh and exact geometry. The most common technique to enhance mesh's geometrical precision is by automatic adaptive mesh refinement which is not widely used in industry because it requires access to the exact geometry, and thus it also requires continuous and automatic communication with CAD which is impractical [5]. So, the classic analysis method is both time-consuming and unfortunate to exact geometrical representations [3].

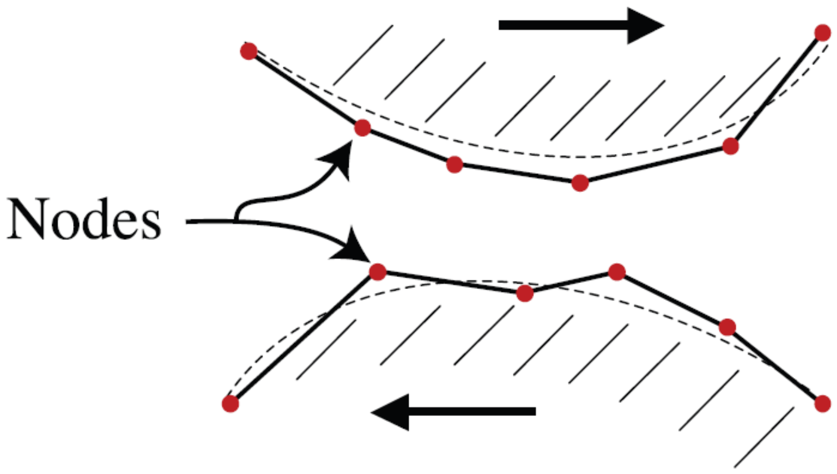


Figure 1.2: A simple representation of FEA mesh inaccuracy to exact geometry. Source: [5]

In order to address the problems of FEA, a new numerical method was proposed; the *IsoGeometric Analysis* (**IGA**) [6]. In this technique CAD-based geometries are directly employed in analysis framework without making approximations like in FEA. In other words, in IGA the same mesh which is used for construction of exact CAD-based model is also used for analysis. According to T.J. Hughes, who is the father of IGA, with this method three goals are satisfied [6]. The primary goal is to be geometrically exact on solution space no matter how coarse the discretization of mesh is. Another goal is to simplify mesh refinement by eliminating the need for communication with the CAD geometry once the initial mesh is constructed. The third goal is to integrate the mesh generation process more tightly within CAD. The founding polynomial for this procedure is the NURBS which is used widely in most CAD systems.

Even if, IGA codes are being developed the last 15 years, it is a relative new method, which has not yet been adopted in industry. In industry FEA-based frameworks are dominating (ANSYS, Altair, NASTRAN, etc.) because of the existence of a huge class of solved problems that has been examined over the 70 years that this method exists. Moreover, IGA is taught at universities in master's degree level, and there is not exist a code that is user friendly for students to use in order to examine thoroughly the method. Moreover, the problems that has been solved are limited on specific geometries and not at free-hand geometries. In addition, there is not a GUI that permits further manipulation methods of computational mesh as refinement. Consequently, to address the needs above an integrated tool is needed to be developed in order to help students or even engineers to learn or use IGA. The term "integrated" means that user can construct the geometry and then use analysis tools in same environment. MATLAB is the ideal language for the initial development of that GUI because it is a programming environment that is easy to use. MATLAB had in May 2022, more than 4.1 billion users worldwide from universities, startups and major organizations and companies [7]; thus is very popular to scientific and industry community.

1.2 Literature

A fundamental part of Isogeometric analysis is the model representation in CAD. Geometries are mostly created using methods of computational geometry. The most common methods are either parametric feature-based modelling (SolidWorks) or by using polynomial interpolation. Parametric feature-based modelling is used mostly on solid modelling and is based creating geometries according to parameters. The parameters are two; *Numerical* such as line lengths or arc radius and *Geometrical* such as tangency, parallel, etc. [8]. All parameters are associated with each other to create the final geometry. On the other hand, polynomial interpolation is used on curves, surfaces, or volumes. These methods exist approximately from 1912 with the invention of Bernstein polynomials but they are used in CAD from mid-1960s. According to prof. Christopher Provatidis at his book "Precursors of Isogeometric Analysis", the basic stations in these methods are six and they are presented in table below [2]:

CAD Interpolation Method	Year
Coons Interpolation Formula	1964
Gordon Interpolation	~1970
Bézier Interpolation	~1970
B-Spline Interpolation	1972
NURBS Interpolation	1975
Barnhill Interpolation	~1980

Table 1-1: The basic stations of CAD interpolation.

From the above, the NURBS interpolation is the most used CAD method nowadays. This method permits the exact representation of all conical surfaces (cylinders, spheres, ellipsoids, etc.), so it is suitable for the free-form surface modelling [4]. For example, CAD frameworks such as Rhinoceros [9] and Autodesk's Maya [10] provide a NURBS-based modelling system. However, NURBS modelling have a few drawbacks. For instance, in order to construct a complex geometry like the hand in fig. 1.3a below they are needed 7 *patches* ("surfaces")*,* so the need for data is relatively high. By the term "data" we mean the *control points* of the patches where at NURBS are superfluous and exist only to satisfy topological contains without adding additional information about geometry (see fig. 1.4). Furthermore, it can be seen in fig. 1.3b, at the green square region, that the two NURBS surfaces do not create the desired continuous surface and a gap exists, something that is common in NURBS formulation. All the above reasons are enough to search other geometrical representations. In 2003 Sederberg introduces T-splines which is a generalization of NURBS.

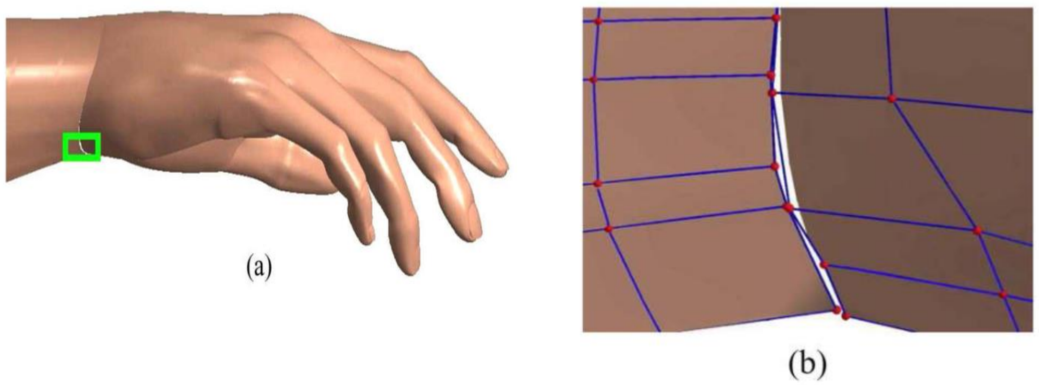


Figure 1.3: (a) The hand produces by NURBS surfaces. (b) The gap that exists between surfaces in green area. Source: [4].

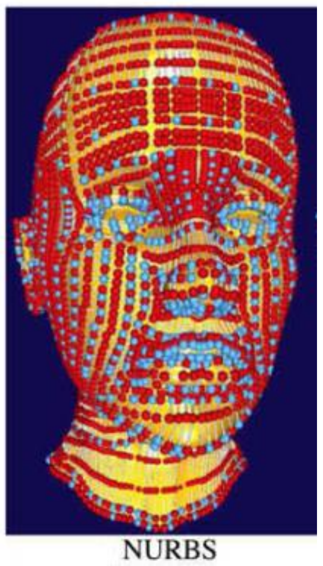


Figure 1.4: A head NURBS model with 4800 control points. The red control points are the ones that do not add extra information. Source: [4].

T-splines combine the advantages of NURBS as smoothness across patch and exact representation of quadric surfaces along with the less information on data and creation on watertight geometries [4]. As it can be seen in fig. 1.5 the control points grid or *mesh* that creates the surface are different in NURBS and T-splines. At NURBS (fig. 1.5a) the control mesh is rectangular, meaning that each control point is connected to four control points except on the boundary. This is happening because it is the result of a tensor product. On the other hand, at T-spline control mesh, the internal control points have not necessarily four adjacent control points. These control points are called T-junctions. This difference enables the local refinement property on surface, meaning that the needed information can be on focused on regions of the surface according to problems conditions. Because of their properties T-splines can create surface from a single patch without gaps as in NURBS case. Furthermore, with T-splines we can manipulate trimmed geometries which gives powerful possibilities on problem design [6].

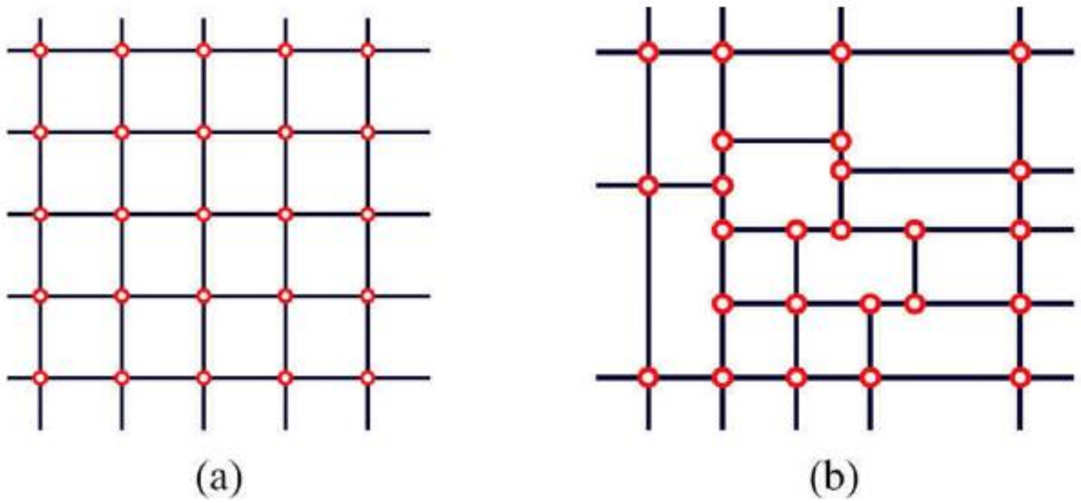


Figure 1.5: (a) NURBS control mesh. (b) T-spline control mesh with T-junctions.

T-splines is a possible CAD method that can be a solution to CAD/CAE integration problem [6]. T-splines is far more powerful than NURBS in CAD and with them more analysis-suitable geometries can be created. These analysis-suitable geometries can be used by a FEA framework in order to create trustworthy analysis models. In industry the only known framework that works this way is Autodesk's Fusion 360. In Fusion we can create T-spline surface (see fig. 1.6) and then solve a problem using FEA. However, the benefits of IGA are not used. There are many integrated frameworks using FEA (ANSYS, SolidWorks, FEATools (see fig.1.7, 1.8, etc.), but none could use IGA. On the other hand, the last 15 years many problems have been solved, and many IGA codes have been developed without importing them to an integrated environment. For instance, in field of contact mechanics FEA creates numerical errors at simulation results in comparison with IGA, which leads to more accurate and robust results using high order NURBS [11]. Also, in other areas such as fluid mechanics, thin shell problems and structural vibration IGA dominates over FEA [11]. An open-source IGA code is IGAFEM [12] which is a MATLAB suite with linear elasticity problems, plate and shell problems using NURBS and T-splines in 2D and 3D dimension. Another open-source package in MATLAB is called GeoPDEs and works as a starting point for learning Isogeometric analysis using NURBS. In 2010 Vuong et al. created ISOGAT [13] a tutorial MATLAB code that solves a specific form of Poisson equation. All the above frameworks have the disadvantage that they do not provide an integrated CAD/CAE environment and the geometry is needed to be imported externally. ISOGAT differs on previous demand because it provides a GUI, however the geometries that can be manipulated are specific and cannot change interactively. The only known integrated environment that performs IGA is the Coreform IGA [14] which is using U-splines [15], a new method of creating geometries, but it is under development and it costs.

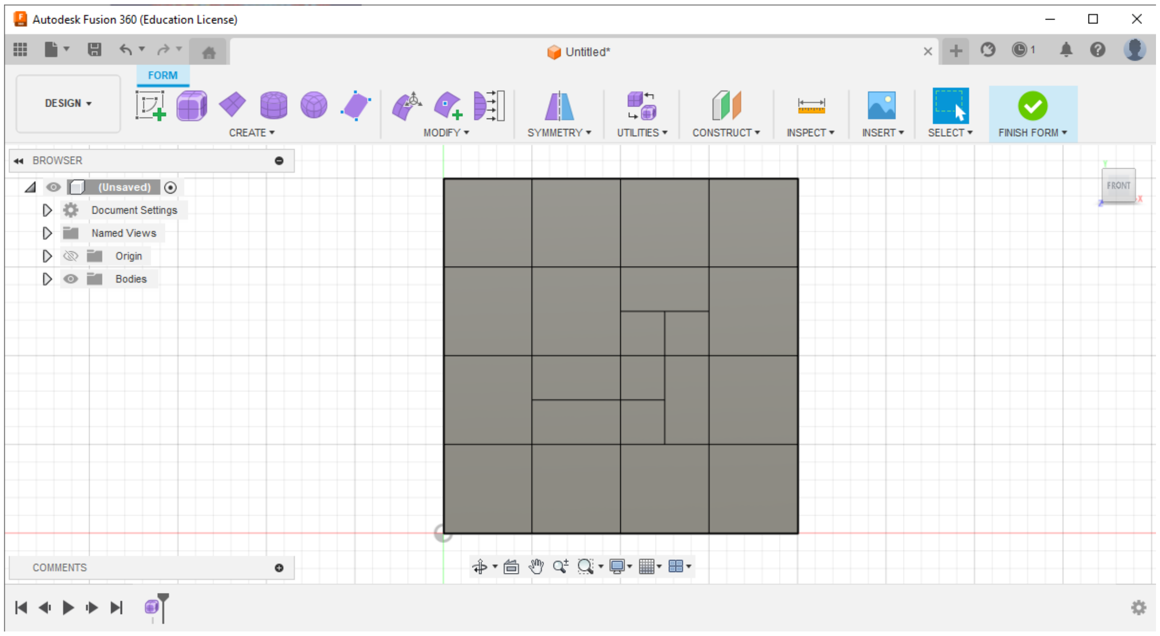
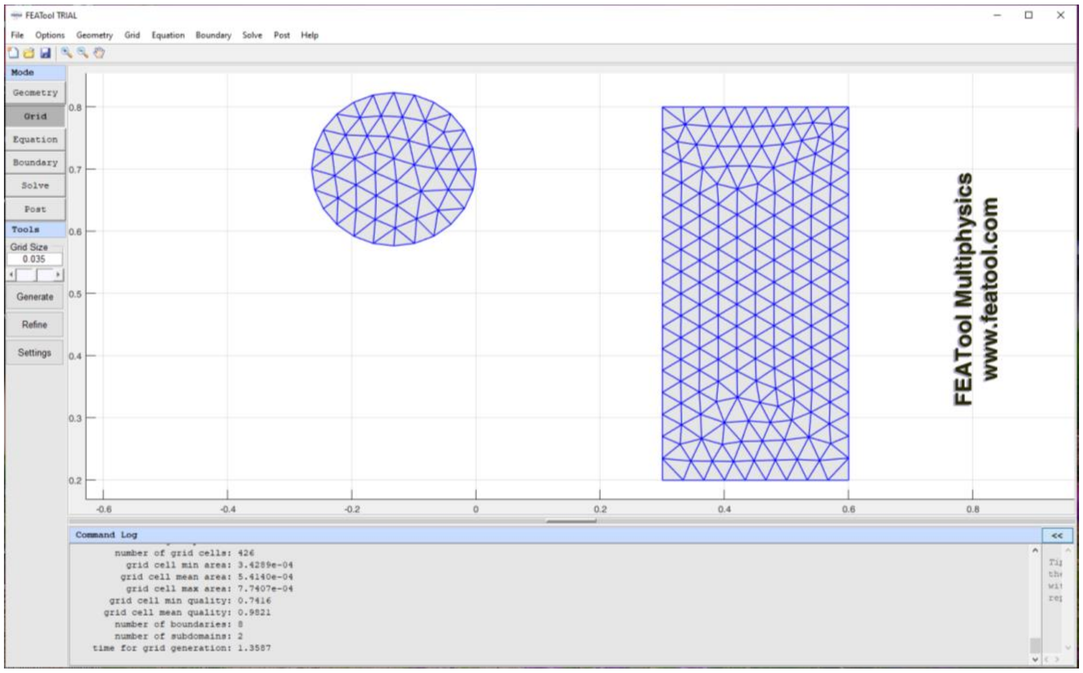


Figure 1.6: A T-spline surface in Autodesk's Fusion 360.



```
         number of grid cells: 426
           grid cell min area: 3.4289e-04
          grid cell mean area: 5.4140e-04
           grid cell max area: 7.7407e-04
        grid cell min quality: 0.7416
       grid cell mean quality: 0.9821
         number of boundaries: 8
         number of subdomains: 2
     time for grid generation: 1.3587
```

Figure 1.7: Triangular mesh generation on FEATools.

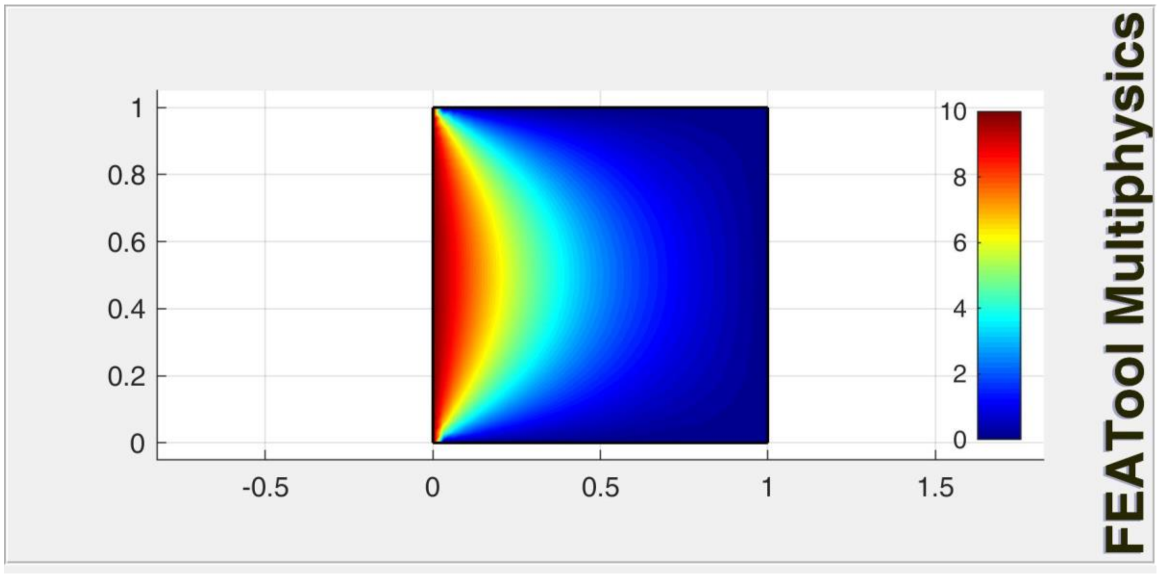


Figure 1.8: Solving a heat conduction problem on FEATools with Dirichlet boundary conditions.

1.3 Outline

On this thesis we try to solve some of the problems mentioned on previous section (1.2). Specifically, we created an integrated CAD/CAE tool where the user can interactively manipulate geometry and perform analysis using IGA. For the geometry representation NURBS will be used in the tool. The tool will provide the user with mesh refinement options, variety on boundary conditions and interactive modification on the geometries. The tool is called I3GA and constitutes an evolution of ISOGAT on use and capabilities. The I3GA runs on MATLAB 2018b and later versions.

In Chapter 2 we will describe the basics of the theory of FEA along with the basics of IGA. In Chapter 3 we will describe the crucial properties of the most notable basis functions. Basis functions are the piecewise "polynomials" that along with control points form the geometry (i.e., curve, surface and solids). In Chapter 4 we will describe some mesh refinement options along with examples. In Chapter 5 we will describe the theoretical formulation of IGA problems that will be solved along with another method which is *Bézier Extraction*. Bézier extraction is a method of decomposing the surface into C^0-continuous elements and use them into analysis with FEA codes [11]. In Chapter 6 some interesting results will be implemented using I3GA and we will introduce the reader to mesh adaptivity. Finally, we will mention some concluding remarks and mention the next step on our research after this project.

2 The Journey from FEA to IGA

The problems of the classic design to analysis (CAD-to-FEA) method which were mentioned on introduction do exist, because CAD and FEA have been developed for different purpose. CAD is for designers and FEA for engineers. Designers are focusing more on the tools which would help them to easily manipulate, visualize and construct geometries. On the other hand, engineers want tools which would be computationally fast and interpretable [16]. The tools are the different types of polynomials used for the creation of geometry or mesh. These polynomials are called shape functions in FEA and basis functions in CAD. These tools were developed independently from both communities [16]; therefore the communication between these systems was not a straightforward task, as the two areas are practically decoupled. The isogeometric concept has unified these CAD and analysis solvers (not only FEA) into a single framework, leading to elimination of cost, time-consumption and inaccuracies created by FEA mesh and probably by communication of CAD and analysis [6]. In this chapter we will present the evolution from FEA to IGA describing briefly the basic mathematical concepts behind FEA, IGA, along with an intermediate method the isoparametric finite elements.

2.1 Finite Element Method - FEM

As already been told in previous chapter the concept of finite element method is to discretize the problem domain Ω into smaller subdomains Ω^{e} (elements) in order to approximate the exact solution of PDE or function $\mathbf{u}(\mathbf{x}, t)$ by solving it "approximately" in each element [17]. Supposing that the problem domain is steady-state (time terms = 0) using the notation from [2] the unknown function is approximated by $\bar{\mathbf{u}}(\mathbf{x})$, which is a series expansion of n terms in the form:

$$\mathbf{u}(\mathbf{x}) \cong \bar{\mathbf{u}}(\mathbf{x}) = \sum_{i=1}^{n} N_i(\mathbf{x})\mathbf{a}_i = \mathbf{Na} \tag{2.1}$$

Where, n is the number of nodes, $N_i(\mathbf{x})$ are the weighting or "shape" functions prescribed by physical coordinates of each node and $\mathbf{a}_i$ are the unknown nodal parameters that have to be evaluated. These shape functions are defined usually locally for elements [17] and are based on interpolatory polynomials [4]. The nodal values $\mathbf{a}_i$ are computed by formulating a system of n equations where the strong form of eq. (1.1) (or system equations [17]) is transformed to a discretized integral form. Integrating PDE leads to its approximation element by element and then by a suitable assembly a global approximate solution is obtained [17]. One of the most notable and precise methods for obtaining this approximation is the weighted residuals method or Galerkin method [17].

The basic idea of Galerkin procedure is that PDEs can be "integrally" satisfied in $\Omega \cup \Gamma$ if firstly they are multiplied with a suitable set of linear independent arbitrary functions $W_i(x), i = 1,2,\dots,n$. Let $D(\mathbf{u}(\mathbf{x})) - \mathbf{f} = \mathbf{0} \equiv \mathbf{D}(\mathbf{u}) = \mathbf{0}$ be a system (in matrix form) of n differential equations satisfied in Ω and $\mathbf{B}(\mathbf{u}) = \mathbf{0}$ is the system of respective boundary conditions on Γ. If there is a suitable set of $W_i(\mathbf{x})$ and $\overline{W}_i(\mathbf{x})$ for $\mathbf{D}(\mathbf{u})$ and $\mathbf{B}(\mathbf{u})$ respectively then Galerkin proposes that the PDE is satisfied wherever the integral below is satisfied:

$$\int_{\Omega} \mathbf{W}^{\mathrm{T}}\mathbf{D}(\mathbf{u})\mathrm{d}\Omega + \int_{\Gamma} \overline{\mathbf{W}}^{\mathrm{T}}\mathbf{B}(\mathbf{u})\mathrm{d}\Gamma = 0 \Leftrightarrow$$

$$\Leftrightarrow \int_{\Omega} \sum_{i=0}^{n} W_i D_i(\mathbf{u})\,\mathrm{d}\Omega + \int_{\Gamma} \sum_{i=0}^{n} \overline{W}_i B_i(\mathbf{u})\,\mathrm{d}\Gamma = 0 \tag{2.2}$$

The functions $W_i(\mathbf{x})$, $\overline{W}_i(\mathbf{x})$ are chosen in order to avoid infinite integrals and also be at least C^{m-1}-continuous if m is the derivative order of $\mathbf{D}(\mathbf{u})$ or $\mathbf{B}(\mathbf{u})$ [17]. The eq. (2.2) is satisfied only in one element

Ω^e (and its respective boundary Γ^e if exists), so a proper sum of all integrals in each element should be computed in order to calculate the exact value of the unknowns $\mathbf{a}_j$. In order to achieve the correct approximation, the functions $\mathbf{W}$, $\overline{\mathbf{W}}$ is favorable to be written as an expansion of the unknowns as:

$$\mathbf{W} = \sum_{j=1}^{n} \mathbf{w}_j(\mathbf{x})\delta\mathbf{a}_j \qquad \overline{\mathbf{W}} = \sum_{j=1}^{n} \overline{\mathbf{w}}_j(\mathbf{x})\delta\mathbf{a}_j \tag{2.3}$$

Where, $\delta\mathbf{a}_j, \delta\overline{\mathbf{a}}_j$ are arbitrary parameters and $\boldsymbol{w}_j, \overline{\boldsymbol{w}}_j$ are also linear independent weighting functions. Then by inserting eq. (2.1) and (2.3) to (2.2) we have:

$$(2.2) \xRightarrow{(2.1)+(2.3)} \int_\Omega \mathbf{w}_j^{\mathrm{T}}\mathbf{D}(\mathbf{Na})\delta\mathbf{a}_j^{\mathrm{T}}\mathrm{d}\Omega + \int_\Gamma \overline{\mathbf{w}}_j^{\mathrm{T}}\mathbf{B}(\mathbf{Na})\delta\mathbf{a}_j^{\mathrm{T}}\mathrm{d}\Gamma = 0 \Rightarrow$$

$$\Rightarrow \delta\mathbf{a}_j^{\mathrm{T}}\left[\int_\Omega \mathbf{w}_j^{\mathrm{T}}\mathbf{D}(\mathbf{Na})\mathrm{d}\Omega + \int_\Gamma \overline{\mathbf{w}}_j^{\mathrm{T}}\mathbf{B}(\mathbf{Na})\mathrm{d}\Gamma\right] = 0$$

The $\delta\boldsymbol{a}_j$ are arbitrary and constant so from the above equation we have n equations to compute the unknowns $\boldsymbol{a}_j$ in form:

$$\int_\Omega \mathbf{w}_j^{\mathrm{T}}\mathbf{D}(\mathbf{Na})\mathrm{d}\Omega + \int_\Gamma \overline{\mathbf{w}}_j^{\mathrm{T}}\mathbf{B}(\mathbf{Na})\mathrm{d}\Gamma = 0, \qquad j = 1, \dots, \mathrm{n} \tag{2.4}$$

The procedure represented by eq. (2.4) is called *method of weighted residuals* or *generalized finite element method*. The term $\mathbf{D}(\mathbf{Na})$ is called residual or error and is obtained by substitution of the approximation into PDE [17] ($\boldsymbol{B}(\mathbf{Na})$ is the residual of boundary conditions). The system derived from eq. (2.4) is closer to the physical problem than the one defined by PDE as the solution in former case is becoming more smooth [17]. Alternatively, eq. (2.4) is the *weak* form of the problem. As already been told in order to realize the analysis in whole problem domain eq. (2.4) must be computed for each element and then take the whole sum for all m elements:

$$\sum_{e=1}^{m}\left(\int_\Omega \mathbf{w}_j^{\mathrm{T}}\mathbf{D}(\mathbf{Na})\mathrm{d}\Omega + \int_\Gamma \overline{\mathbf{w}}_j^{\mathrm{T}}\mathbf{B}(\mathbf{Na})\mathrm{d}\Gamma\right) = 0, \qquad j = 1, \dots, \mathrm{n} \tag{2.5}$$

Proportional to the analysis case, the weighting function could take different forms. Some of most common choices are:

Collocation methods:

In collocation methods the differential equation is satisfied in n discrete positions of problem domain Ω, without excluding the boundary in general case [2], [18]. In this condition a system of n equations is formed leading to finding the unknown parameters. The weighting function that fulfills this condition is the Dirac δ-function, Δ_i. Collocation methods are used on elliptic, parabolic or hyperbolic problems (linear or nonlinear) for example in vibration analysis of elastic rods [19]. There are two main types of collocation methods the *Point* and *Subdomain* (or *Finite Volume* method) collocation [17]:

- Point collocation: $\mathbf{w_j} = \boldsymbol{\Delta}_j$[1]. This procedure is equivalent to simply making the residual zero at n points within Ω and then integration can be satisfied.

[1] $\mathrm{w}_i \equiv \boldsymbol{\Delta}_i = \begin{cases} +\infty, & x = x_i; y = y_i; z = z_i \\ 0, & \text{otherwise} \end{cases}$ and $\int_\Omega \mathbf{w}_i \mathrm{d}\Omega = \mathbf{I}$

- Subdomain collocation: $\mathbf{w_j} = \mathbf{I}$ (the unit matrix), in each subdomain Ω_j and zero elsewhere; thus, the integrals of residuals in eq. (2.5) becomes 0 in each Ω_j.

The choice of collocation points in each method can influence the numerical solution [2], [18]. Proportional to the position of collocation points in [2], [18], we have Nodal, Orthogonal and Least-squares collocation. For more information on Collocation methods see Provatidis [2], [18] and Zienkiewicz [17].

Galerkin **or** ***Bubnov-Galerkin Method:***

In Galerkin method the used weighting functions are the shape functions $N_j(\mathrm{x})$ used for the approximation of exact solution. The shape functions give the property of "locality" on analysis procedure; thus is the most used method in approximation of problems. The shape functions can have different forms as interpolatory polynomials, splines , trigonometric functions, wavelets etc. [18].

The Bubnov-Galerkin Method for weighting functions applied to eq. (2.5) gives the most used method for numerical analysis, the *Finite Element Method*. In FEM the problem domain is discretized into a finite number of elements in which each element is connected to another via nodes. Each node constitutes a vector of unknown parameters $\mathbf{a}_j$ and a cluster of nodes forms an element or shape. The cluster of elements forms the computation mesh over the problem domain in which the approximation is done. The elements are not the same with each other (Laplacian grid) instead their shape or size differs. For example in positions where source terms affect, the elements must be smaller in order to stabilize the effect of high gradient of problem variables $\mathbf{u}$ [2] or in cases where the problem domain is irregular, the elements should be also irregular in order to be more accurate. In each element the variable $\mathbf{u}$ is described by eq. (2.1) where the shape functions are given in Cartesian coordinates of problem domain or other coordinate system proportional to element kind. Independently of elements' kind or the PDE's initial form (linear or nonlinear) the final purpose of FEM is to derive a global system of equations which can be solved numerically and find the parameters $\mathbf{a}$. In other words, the eq (2.5) can be written as:

$$(2.5) \xRightarrow{\mathbf{w}_j,\bar{\mathbf{w}}_j \to \mathbf{N}} \sum_{e=1}^{m} \left(\int_\Omega \mathbf{N}^{\mathrm{T}}\mathbf{D}(\mathbf{Na})\mathrm{d}\Omega + \int_\Gamma \mathbf{N}^{\mathrm{T}}\mathbf{B}(\mathbf{Na})\mathrm{d}\Gamma \right) = 0, \qquad j = 1, \dots, \mathrm{n} \Leftrightarrow$$

$$\Leftrightarrow \mathbf{K}_{\text{Global}}\mathbf{a} = \mathbf{f} \tag{2.6}$$

where, $\mathbf{K}_{\text{Global}} \in \mathbb{R}^{n\times n}$ is the stiffness matrix of problem domain, $\mathbf{a} \in \mathbb{R}^n$ is the vector of nodal parameters and $\mathbf{f} \in \mathbb{R}^n$ is the vector of source terms. By applying the boundary conditions then from eq. (2.6) we can find the nodal values $\mathbf{a}$. The stiffness matrix is symmetric because of shape functions that are used [17] and in most cases is positive definite. The stiffness matrix contains information on the steady properties of problem (i.e., on structural problems the stiffness matrix contains information for density, elasticity, geometry of structure etc.). The $\mathbf{K}_{\text{Global}}$ and $\mathbf{f}$ are defined as [17]: $\mathbf{K}_{\text{Global}} = \sum_{e=1}^{m} \mathbf{K}^e$ and $\mathbf{f} = \sum_{e=1}^{m} \mathbf{f}^e$. If the problem is hyperbolic or parabolic then derivatives of time exist in problem's strong form; thus, a similar analysis should be applied when except of stiffness matrix there is also mass matrix. Finally, the key difference between a linear and a nonlinear FEM problem is that the solution of eq. (2.6) is far more complex as in nonlinear case $\mathbf{K}_{\text{Global}} = \mathbf{K}_{\text{Global}}(\mathbf{a})$. In Appendix:A.1 we present an example of FEM formulation on a steady-state elasticity problem [18].

Of course, most FEM problems due to their complexity are solved with the use of computers and the calculations are numerical and not analytical. For example, the integrals in eq. (2.6) are calculated using numerical schemes (i.e., Gauss Quadrature). A traditional code-architecture of FE procedure for a structural problem is presented on fig. 2.1 below [11]. A FEM code consists of 3 stages the *pre-processing* stage where the mesh is constructed along with the definition of boundary conditions and maybe some numerical coefficients, the *processing* stage where the solution of eq. 2.6 is implemented and the *post-processing* stage where some quantities of interest (e.g., derivatives) may be computed. In this flowchart the researchers used Lagrangian shape functions and Gauss Quadrature for numerical integration.

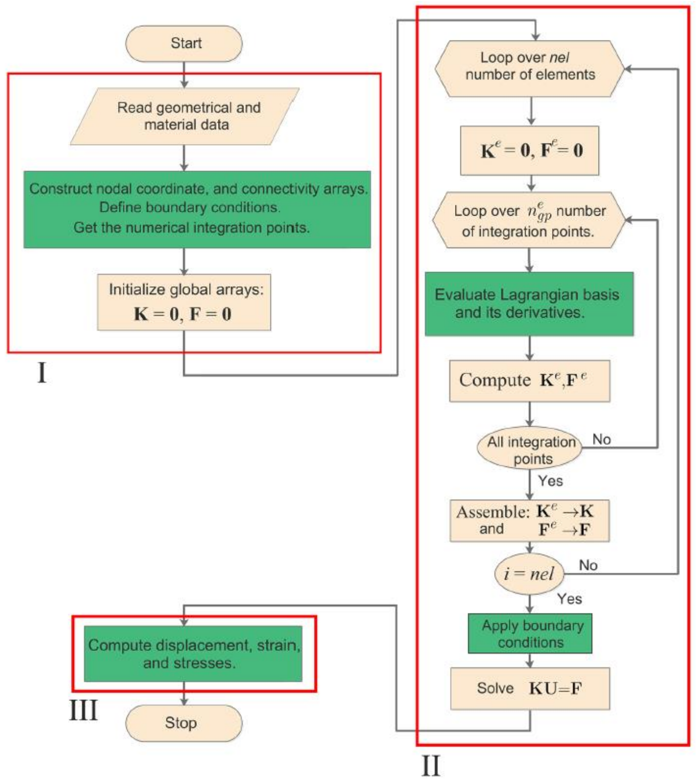

Start
Read geometrical and material data
Construct nodal coordinate, and connectivity arrays. Define boundary conditions. Get the numerical integration points.
Initialize global arrays: $\mathbf{K}=\mathbf{0}$, $\mathbf{F}=\mathbf{0}$
I
Loop over *nel* number of elements
$\mathbf{K}^e=\mathbf{0}$, $\mathbf{F}^e=\mathbf{0}$
Loop over n^e_{gp} number of integration points.
Evaluate Lagrangian basis and its derivatives.
Compute $\mathbf{K}^e,\mathbf{F}^e$
All integration points
No
Yes
Assemble: $\mathbf{K}^e\to\mathbf{K}$ and $\mathbf{F}^e\to\mathbf{F}$
$i=nel$
No
Yes
Apply boundary conditions
Solve $\mathbf{KU}=\mathbf{F}$
II
Compute displacement, strain, and stresses.
III
Stop

Figure 2.1: A flowchart of traditional FE code. The I,II,III are the 3 stages of FE code described on section 2.1. Source [11].

2.2 Isoparametric Finite Elements

The mesh generation described on previous section is not necessarily respect the initial geometry of domain and when the approximation is done it is not decent. The use of eq. (2.1) as the only starting point for FEM formulation may not be sufficient in terms of accuracy and computational cost of procedure, because the computational mesh is not necessarily close to real geometry. Geometries are in fact irregular in real world problems. The classic elements are most commonly triangles and quadrilateral elements. The triangular elements can be used in both regular or irregular geometries but the higher computational power and the lower accuracy are needed to take into consideration [20]. On the other hand, quadrilateral elements can surpass the problems of triangles except the fact that cannot be applied to irregular geometries because they maintain their orthogonality on Cartesian space. In order to address these problems and create meshes that are closer (not exact) to problem domain and the compatibility of variables $\mathbf{u}$ between neighboring elements is ensured [20], researchers have introduced the *isoparametric* element concept. References on isoparametric elements, a reader can find on [18], [21].

Isoparametric elements use the same shape functions used for the interpolation of elemental nodal values as for the coordinates of element's nodes [18]. In other words, the basic relationships for isoparametric elements are:

Geometry: $\mathbf{x} \cong \tilde{\mathbf{x}} = \sum_{i=0}^{n} N_i \mathbf{x}_i$

Field variables: $\mathbf{u} \cong \tilde{\mathbf{u}} = \sum_{i=0}^{n} N_i \mathbf{u}_i$

Where n is the number of nodes, N_i the shape functions and $\mathbf{x}_i, \mathbf{u}_i$ are the nodal values of coordinates and field variables respectively. The isoparametric formulation begins by defining a mesh not in Cartesian (physical or global) space but on parameter (natural or intrinsic) space where the elements are regular. Let the natural coordinates (ξ, η, ζ) defined on interval $[-1,1]$. Defining shape functions on natural coordinates $N = N(\xi, \eta)$ permits integration using numerical schemes as Gaussian Quadrature, which is defined on interval $[-1,1]$ and the creation of more flexible and irregular curved elements on Cartesian space (see fig. 2.2 in 2D case). The element on parameter space is called *parent* element and on Cartesian space *global* element. The two coordinate systems for each element relate to each other using a transformation mapping $\mathbf{\Phi}$. This transformation mapping is achieved using Jacobian matrix $\mathbf{J}$ which relates the derivatives of coordinate system. A family of shape functions that is used on isoparametric elements are p^{th} order Lagrange polynomials $\mathbf{L}_i^{\mathrm{p}}(\xi)$.

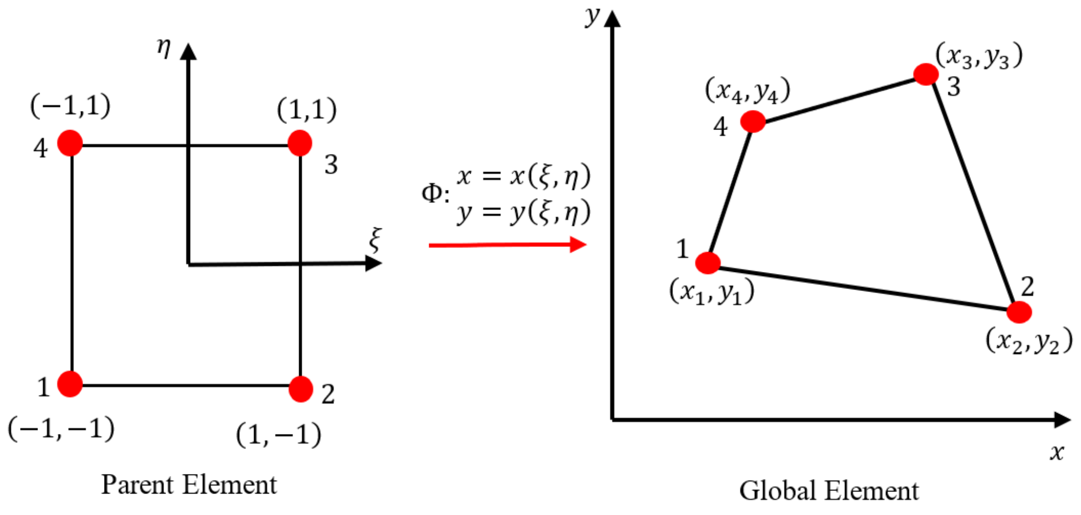


Figure 2.2: A 2-D schematic representation of an isoparametric quadrilateral element in Cartesian space and its respective element on parameter space. The two elements are related with a mapping $\boldsymbol{\Phi}$.

A Lagrange polynomial $\mathrm{L}_i^{\mathrm{p}}(\xi)$ of p^{th} order associated with node ξ_i is able to interpolate $(p+1)$ points and can be defined as:

$$\mathrm{L}_i^{\mathrm{p}}(\xi) = \prod_{k=1, k\neq i}^{p+1} \frac{\xi - \xi_k}{\xi_i - \xi_k} \tag{2.7}$$

with $k \in [1, p+1]$ and $\xi \in [-1,1]$. Lagrange polynomials have interesting properties [11] that make them suitable for FEM as:

- Kronecker's delta: $\mathrm{L}_i^{\mathrm{p}}(\xi_j) = \delta_{ij} = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$
- Partition of Unity: $\sum_{i=1}^{n} \mathrm{L}_i^{\mathrm{p}}(\xi) = 1$
- Linear Independence: $\sum_{i=1}^{n} \alpha_i \mathrm{L}_i^{\mathrm{p}}(\xi) = 0 \Leftrightarrow \alpha_k = 0, \;\; k = 1,2,\dots,n$
- $-1 \leq \mathrm{L}_i^{\mathrm{p}}(\xi) \leq 1$

Each property has its own meaning. The “Partition of Unity” and “Linear Independence” properties are “must” properties that a shape function should satisfy in order to be used in analysis. The “Kronecker's delta” property notes that Lagrange polynomials can be used as one dimensional shape functions [18]. In fact, Lagrangian shape functions have very interesting properties that permit the isoparametric formulation not only in one dimension but in more dimensions[2]\. In Appendix A.2 we will see an example of isoparametric formulation using Lagrangian shape functions. Except for the benefits of Lagrange polynomials there are also some shortcomings that will be explained in section about IGA's basis functions. Briefly Lagrangian shape functions are sensitive to higher order formulations and present only C^0-continuity across the inter-element boundary [11]. In general, higher order shape functions lead to more accurate approximations. However, Lagrangian functions are restricted mostly on orders $p \leq 2$, because they present non-smooth interpolations. The second disadvantage leads to inaccuracy during the calculation of derivative quantities as strains or stresses in structural mechanics problems. Even their problems, Lagrange polynomials are the most popular shape functions in FEA community [11] and they have been integrated in a variety of research and industrial FEA software.

2.3 Isogeometric analysis: Intuition and Notation

With *isoparametric* finite elements, the analysis approximates geometry without being exact. This leads to accuracy problems within process and a bottleneck in duration of analysis procedure as it is time-consuming to adjust a suitable mesh close to the geometry of problem domain. The geometric models of problem domains are constructed by CAD software. A possible solution to these problems could be the use of CAD representations as an analysis model. This is the concept of *isogeometric* analysis (**IGA**). In IGA the analysis is executed in a physical mesh which is the decomposition of CAD geometry [5]. In fig. 2.3 we can see the difference between a physical and a control mesh which create the former for the same surface. In FEM the control mesh has the same role as the physical mesh in IGA. Furthermore, in IGA the control points in most cases are outside the surface.

Unlike FEM, the mesh in IGA consists of surfaces or patches. Each patch is not considered as one element but the image of rectangular elements in parameter space [4]. In isoparametric formulation, the parameter space consists of one parental element which is mapped into each element of physical space in contrast to IGA where several elements of parameter space correspond to the ones in physical space. The mapping is taken place using again the Jacobian matrix. The patches are constructed by linear combination of piecewise polynomials consisting of suitable basis functions along with the points of control mesh. The term basis function has a similar role as shape functions in analysis [5], however they have different properties permitting the exact and smooth designing of geometrical objects.

[2] We will see in next section how a shape function can be extended to further dimensions.

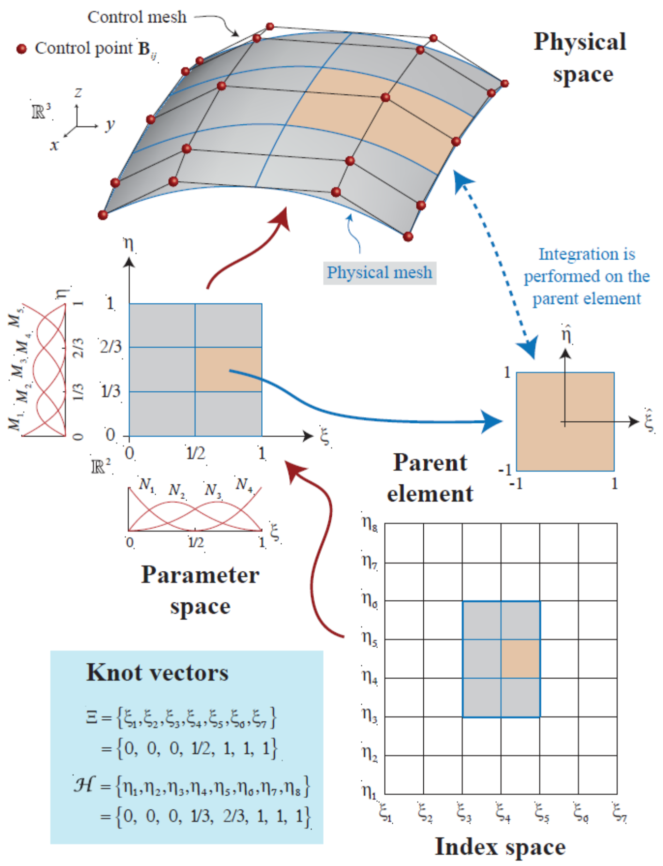

Control mesh
Control point $\mathbf{B}_{ij}$
Physical space
Physical mesh
Integration is performed on the parent element
Parent element
Parameter space
Index space
Knot vectors
$\Xi = \{\xi_1,\xi_2,\xi_3,\xi_4,\xi_5,\xi_6,\xi_7\} = \{0, 0, 0, 1/2, 1, 1, 1\}$
$\mathcal{H} = \{\eta_1,\eta_2,\eta_3,\eta_4,\eta_5,\eta_6,\eta_7,\eta_8\} = \{0, 0, 0, 1/3, 2/3, 1, 1, 1\}$

Figure 2.3: Schematic representation of isogeometric concept. A patch is depicted on physical space with both physical and control mesh. Physical space is the image of parameter space in which basis functions are defined constructing the surface in physical space. Parameter space is constructed from knot vectors and the index space shows the support of basis functions from knots. Source:[5].

Each patch is decomposed into knot vectors [5]. The knot vector is a nondecreasing set of $\xi_i \in \mathbb{R}$ in each direction of parameter space that define the space where the basis functions are defined. The knot vector can be written as:

$$\Xi = \{\xi_1, \xi_2, \dots, \xi_i, \dots, \xi_{n+p+1}\}^{\mathrm{T}} \tag{2.8}$$

where, $\xi_i \in \mathbb{R}$ is the i^{th} knot, $i = 1,2,\dots,n+p+1$, n the number of basis functions comprising the spline space [3], [16] and p is the polynomial degree of basis functions. The length of knot vector is defined as $\|\Xi\| = n + p + 1$. If a knot vector has all knots equally spaced then it is called *uniform*, otherwise it is called *non-uniform*. Furthermore, in CAD industry the used basis functions commonly use knot vectors that are "open". A knot vector is *open* if its first and last knots are repeated $p + 1$ times. The main property of open knot vectors is that they are interpolatory at the boundaries of parameter space (e.g., ξ_1, ξ_{n+p+1}), something important for the proper definition of boundary conditions in problem domain [3].

In fig. 2.3 an example of the above definitions is shown. A patch in physical space is constructed by a linear combination of basis functions along with control points defined in a control mesh. The parameter space is defined from open knot vectors $\Xi = \left\{0,0,0,\frac{1}{2},1,1,1\right\}^{\mathrm{T}}$ and $\mathbf{H} = \left\{0,0,0,\frac{1}{3},\frac{2}{3},1,1,1\right\}^{\mathrm{T}}$ and each element is defined in the interval of different knots. For example, the yellow element in figure spans in $\boldsymbol{\Xi_r} \otimes \boldsymbol{H_r} = \left[\frac{1}{2},1\right] \times \left[\frac{1}{3},\frac{2}{3}\right]$. Each knot vector (in each direction) defines a set of basis functions. Because of the repetition of knots in knot vectors, the parameter space is not suitable to present the support of basis functions.

Therefore, in IGA was introduced the concept of *index* space, which is a space where all knots are plotted at equally space intervals and each line is labelled with its index value [4], [22]. The knot vectors are selected based on the number of necessary elements and the continuity level across the element boundary [4], [5]. Each type of basis function is constructed differently after selecting the appropriate knot vector which values are positive and can vary from 0 to 1. In next chapter we will present the main basis functions used in IGA.

The general architecture of a code in IGA and FEM do not change a lot except the fact that instead of computing shape functions, we evaluate the basis functions at different values of parameter space. This can be seen in fig. 2.4 where we see the basic algorithm of an IGA code. When comparing with fig. 2.1 all steps are the same except one.

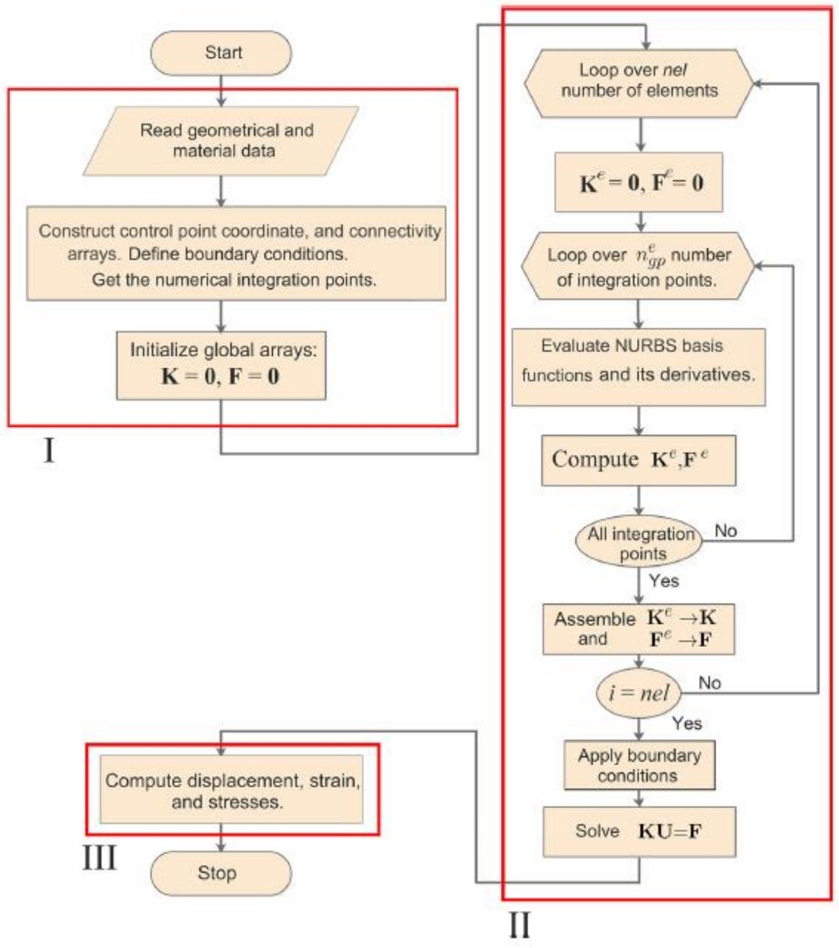

Start
Read geometrical and material data
Construct control point coordinate, and connectivity arrays. Define boundary conditions. Get the numerical integration points.
Initialize global arrays: $\mathbf{K}=0, \mathbf{F}=0$
I
Loop over *nel* number of elements
$\mathbf{K}^e=0, \mathbf{F}^e=0$
Loop over n_{gp}^e number of integration points.
Evaluate NURBS basis functions and its derivatives.
Compute $\mathbf{K}^e, \mathbf{F}^e$
All integration points
No
Yes
Assemble $\mathbf{K}^e \to \mathbf{K}$ and $\mathbf{F}^e \to \mathbf{F}$
$i = nel$
No
Yes
Apply boundary conditions
Solve $\mathbf{KU}=\mathbf{F}$
II
Compute displacement, strain, and stresses.
III
Stop

Figure 2.4: The basic algorithm of an IGA code. Source: [11].

In general, IGA manages better results in terms of accuracy and efficiency than FEM. IGA can be used in the analysis of many difficult engineering problems in which FEM fails. For example, in contact mechanics FEM results in faceted mesh representations of contact surfaces, leading to numerical errors in comparison with IGA where the higher continuity of IGA's basis functions leads smoother mesh representation; thus, more accurate results [23]. Another field except physical problems where IGA achieves great results is in structural shape optimization [22] because in IGA the design (geometric) and the analysis model of problem domain is practically the same in contrast to FEM method where the design needs to be done in CAD system which differs from the analysis model.

3 Basis Functions

As already been mentioned in previous chapter the surface in physical space is constructed using a mesh of control points along with a suitable piecewise polynomial function defined in parameter space. In CAD theory these functions are called basis functions and they are used because of their properties on representing geometrical objects. In these chapter we will describe the properties of the most notable basis functions, along with the function that was used in I3GA; NURBS.

3.1 Bernstein Basis Functions

Let a univariate parameter space $\xi \in [0,1]$, then a p^{th} order Bernstein basis function $B_i^p(\xi)$ is defined as [24]:

$$B_i^p(\xi) = \binom{p}{i}(1-\xi)^{p-i}\xi^i \tag{3.1}$$

with $i = 0,1,\dots,p$.

Bernstein polynomials were widely in CAD industry for the construction of curves and surfaces and are still usable in aeronautical industry. Let $\{\mathbf{P}_i \in \mathbb{R}^{\mathrm{d_s}} \middle| i = 0,1,\dots,p\}$ a set of points in $\mathrm{d_s}$-dimensional space. Then the curve defined by linear combination of these points and suitable Bernstein polynomials is the p^{th} order Bézier curve and it is defined as:

$$C(\xi) = \sum_{i=1}^{p+1} B_i^p(\xi)\mathbf{P}_i = \sum_{i=1}^{p+1} B_i^p(\xi)\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{\mathrm{d_s}i} \end{bmatrix} \tag{3.2}$$

The $\mathbf{P}_i$ with $i = 1,\dots,p+1$ is called *Control Point* (**CP**). The properties of Bézier curve are the following:

1. Non-Negativity: $B_i^p(\xi) \geq 0 \forall i, p: \xi \in [0,1]$
2. Partition of Unity: $\sum_{i=0}^{p} B_i^p(\xi) = 1$
3. $B_0^p(0) = B_p^p(1) = 1$
4. $B_i^p(\xi)$ attains exactly one maximum in the interval [0,1], at $\xi = \frac{i}{p}$.
5. Symmetry: For any p, the set of polynomials $\{B_i^p(\xi)\}$ is symmetric with respect to $\xi = \frac{1}{2}$.
6. Recursive definition: $B_i^p(\xi) = (1-\xi)B_i^{p-1}(\xi) + \xi B_{i-1}^{p-1}(\xi)$
7. $B_i^p(\xi) \equiv 0$ if $i < 0$ or $i > p$
8. Derivatives: $B_i^{p\prime}(\xi) = \frac{\mathrm{d}B_i^p(\xi)}{\mathrm{d}\xi} = p\left(B_{i-1}^{p-1}(\xi) - B_i^{p-1}(\xi)\right)$

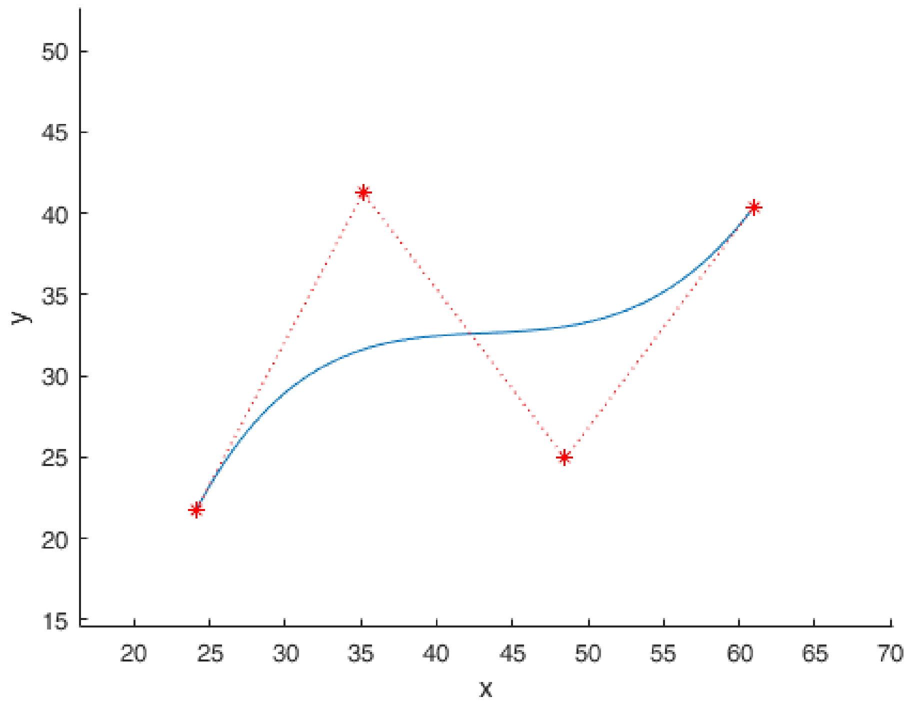


Figure 3.1: A typical cubic Bézier curve. The red line is called control polygon.

Bézier curve is practically a univariate polynomial of p^{th} degree. This formulation has a lot of shortcomings that make them an inadequate choice in use of professional CAD frameworks, however we mention them for historic reasons and because they are used in a specific form in IGA. Their main disadvantages are [24]:

- The numerical processes concerning these curves are inefficient because these curves have a relative high degree (number of control points -1) and high degree polynomials are numerically unstable.
- These curves have not the local support property. The local support property implies that each basis function is nonzero in only limited number of subintervals $[\xi_i, \xi_{i+1}]$ and not in the entire knot vector.
- Continuity of these curves depends on control point's position, as a result there is difficulty in maintaining continuity while changing the position.
- Redundant data need to be stored in memory during numerical processing using these curves.

3.2 B-Splines

B-splines are basis functions which form curves, surfaces or volumes that address the disadvantages of Bernstein polynomials. These curves are constructed from piecewise polynomials and are defined in a parametric space by a knot vector Ξ. These knot vectors are open and non-uniform in most cases in CAD applications. There are many definitions of B-splines basis functions [24] but the most computationally efficient is the Cox-De Boor recurrence formula. Thus, a basis function N_i^p of a given p degree is defined as:

$$N_i^p(\xi) = \begin{cases} H_{\xi_i}(\xi) - H_{\xi_{i+1}}(\xi), \qquad p = 0 \\ \dfrac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \dfrac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi), \quad p \in \mathbb{N}^* \end{cases} \tag{3.3}$$

Where $H_{\xi_i}(\xi)$ is the Heaviside function activated at $\xi = \xi_i$, $\xi_i \in \Xi$ is a knot. B-splines basis functions have interesting properties as [3], [4], [11]:

- Non-Negativity: $N_i^p(\xi) \geq 0,\ \forall \xi \in [\xi_1, \xi_{n+p+1}]$
- Partition of Unity: $\sum_{i=1}^{n} N_i^p(\xi) = 1$
- Linear Independence: $\sum_{i=1}^{n} \alpha_i N_i^p(\xi) = 0 \Leftrightarrow \alpha_k = 0, \quad k = 1,2,\dots,n$
- Compact or Local support: $\{\xi | N_i^p(\xi) > 0\} \subset [\,\xi_i, \xi_{i+p+1}]$
- Continuity: The basis functions are C^∞-continuous meaning that basis functions of degree p have infinitely continuous derivatives. However, the continuity at knots is C^{p-k} where k is knot's multiplicity. If $k = p$ the basis function is C^0-continuous and interpolatory at knot's location.

Below an example of B-splines continuity is depicted for quadratic basis functions $(p = 2)$ and knot vector $\Xi = \{0\ 0\ 0\ 1\ 2\ 3\ 4\ 4\ 5\ 5\ 5\}^{\mathrm{T}}$ or $\{0\ 0\ 0\ 0.2\ 0.4\ 0.6\ 0.8\ 0.8\ 1\ 1\ 1\}^{\mathrm{T}}$. This knot vector results in $(11 - 2 - 1 = 8)$ basis functions (see fig. 3.2). Each basis function corresponds to different color. The analytical form of these basis function is calculated in Appendix B.

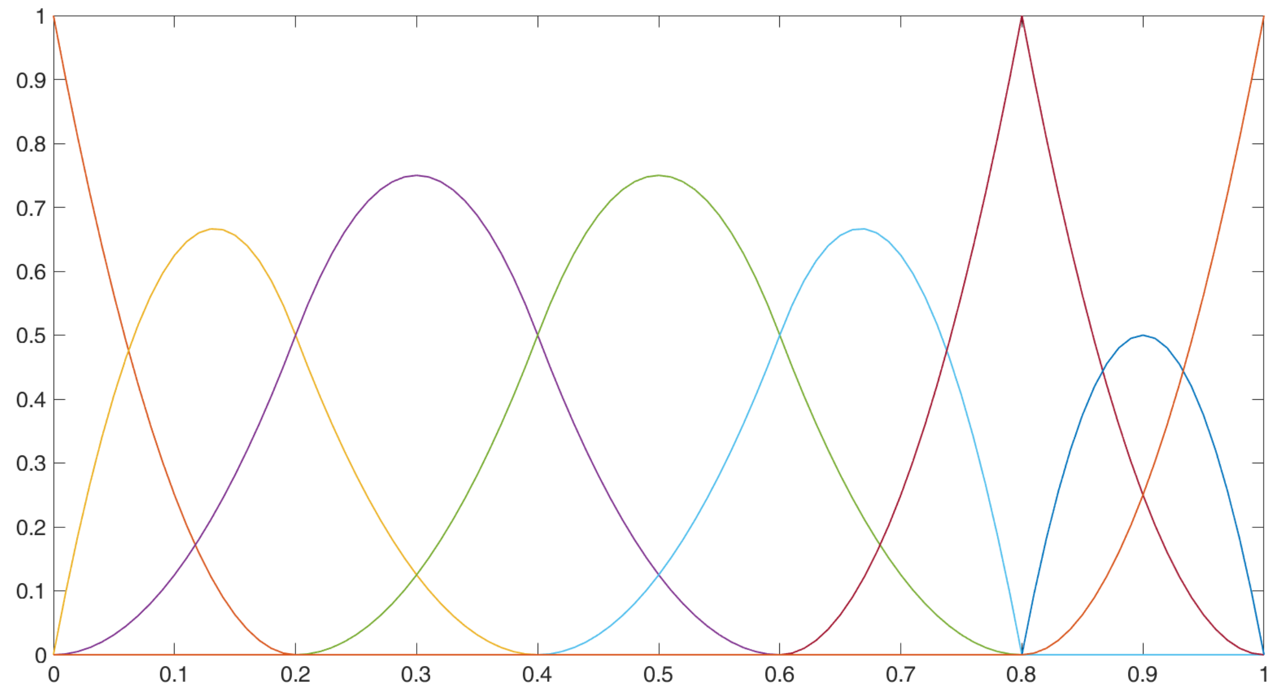


Figure 3.2: Quadratic B-spline basis functions for $\Xi = \{0\ 0\ 0\ 0.2\ 0.4\ 0.6\ 0.8\ 0.8\ 1\ 1\ 1\}^{\mathrm{T}}$. The plots were done in MATLAB using NURBS toolbox by M.Spink

From figure above we can see that the basis functions are interpolatory at $\xi = 0$ and $\xi = 1$ due to the fact Ξ is open knot vector. Except $\xi = 0.8$ basis functions are $C^{2-1} = C^1$-continuous at other knots. At $\xi = 0.8$ basis function is C^0-continuous and interpolatory because the multiplicity of this knot is equal to the degree of basis function. These properties pass to the B-spline-based geometries (curves, surfaces, and volumes). In this special case where at knot 0.8 basis function takes its maximum value; the basis function and the knot are in one-to-one correspondence but in general this is not the case. The locations in parameter space in which basis functions are associated (gaining their maximum value) are called *anchors*. The anchors t_i are addressed by the order of basis functions as:

$$t_i = \begin{cases} \xi_{i+\frac{(p+1)}{2}}, & p = 2c+1, c \in \mathbb{N} \\ \frac{1}{2}\left(\xi_{i+\left(\frac{p}{2}\right)} + \xi_{i+\left(\frac{p}{2}\right)+1}\right), & p = 2c, c \in \mathbb{N} \end{cases}$$

According to this definition, the anchors for this knot vector $\Xi = \{0\ 0\ 0\ 0.2\ 0.4\ 0.6\ 0.8\ 0.8\ 1\ 1\ 1\}^{\mathrm{T}}$ are $\mathbf{T} = \{0\ 0.1\ 0.3\ 0.5\ 0.7\ 0.8\ 0.9\ 1\}^{\mathrm{T}}$. We can observe $t_6 = \xi_7 = \xi_8 = 0.8$; therefore, one-to-one correspondence between knots and basis functions exist at this point.

The derivatives of B-splines basis functions can be calculated by the relationships below [24]:

- *First Derivative:*

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

- κ^{th} order Derivative:

$$N_i^{p(\kappa)}(\xi) = p\left(\frac{N_i^{p-1(\kappa-1)}(\xi)}{\xi_{i+p} - \xi_i} - \frac{N_{i+1}^{p-1(\kappa-1)}(\xi)}{\xi_{i+p+1} - \xi_{i+1}}\right)$$

or

$$N_i^{p(\kappa)}(\xi) = \frac{p!}{(p-\kappa)!}\sum_{j=0}^{\kappa} a_{\kappa,j} N_{i+j}^{p-\kappa}(\xi)$$

with $a_{0,0} = 1$

$$a_{\kappa,0} = \frac{a_{\kappa-1,0}}{\xi_{i+p-\kappa+1} - \xi_i}$$

$$a_{\kappa,j} = \frac{a_{\kappa-1,j} - a_{\kappa-1,j-1}}{\xi_{i+p+j-\kappa+1} - \xi_{i+j}}, \qquad j = 1, \dots, \kappa - 1$$

$$a_{\kappa,\kappa} = -\frac{a_{\kappa-1,\kappa-1}}{\xi_{i+p+1} - \xi_{i+\kappa}}$$

with $\kappa \leq p$

B-splines Curves

The simplest geometry that can be constructed from linear combination of B-spline basis functions is the B-spline curve. This curve is built using a set of control points $\{\mathbf{P}_i \in \mathbb{R}^{\mathrm{d_s}} | i = 0,1,\dots,p\}$ along with basis functions and is defined as:

$$C(\xi) = \sum_{i=1}^{n} N_i^p(\xi)\mathbf{P}_i = \sum_{i=1}^{n} N_i^p(\xi)\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{\mathrm{d_s}i} \end{bmatrix} \tag{3.4}$$

Where n is number of basis functions (or control points) and p is the degree of the curve. The *order* of a B-spline curve is denoted as k and it is $k = p + 1$. The number of basis functions it is the same as control points and it is calculated as: $n = \|\Xi\| - k$. In fig. 3.2 an example of a quadratic B-spline curve (cyan color) is presented based on knot vector $\Xi = \{0\ 0\ 0\ 0.2\ 0.4\ 0.6\ 0.8\ 0.8\ 1\ 1\ 1\}^{\mathrm{T}}$.

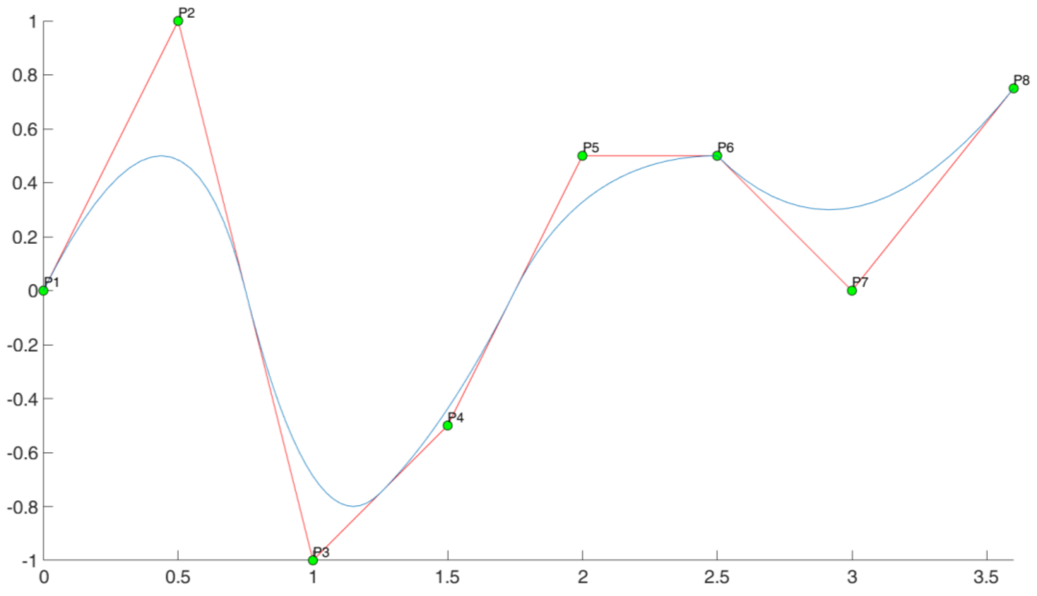


Figure 3.3: A quadratic B-spline curve.

The red piecewise curve is called *control polygon*. From figure the curve inherits the continuity properties of respective basis functions. At ends the curve is interpolatory because the knot vector is open. Curve is also interpolatory and C^0-continuous at $\xi = 0.8$ because the knot's multiplicity is 2. Everywhere else the curve is C^1-continuous. Furthermore, B-spline curves except for the properties of their basis functions, they have also interesting properties as [4], [25]:

- Affine invariance: An affine transformation $\boldsymbol{F}$ as rotations or translation in space is applied to the curve if it is applied to its control points, (3.4) $\overset{\boldsymbol{F}}{\Rightarrow} \boldsymbol{F}\big(C(\xi)\big) = \boldsymbol{F}\big(\sum_{i=1}^{n} N_i^p(\xi)\mathbf{P}_i\big) = \sum_{i=1}^{n} N_i^p(\xi)\boldsymbol{F}(\mathbf{P}_i)$.
- Convex hull: The curve lies in convex hull created by its control polygon (see the green areas in fig. 3.4).
- Variation diminishing: A curve cannot cross any straight line (or affine hyperplane) more often than its control polygon. This property gives the advantage that in higher degree the curves preserve their monotony and do not show oscillating phenomena as Lagrange shape functions in FEA (see fig.3.5).

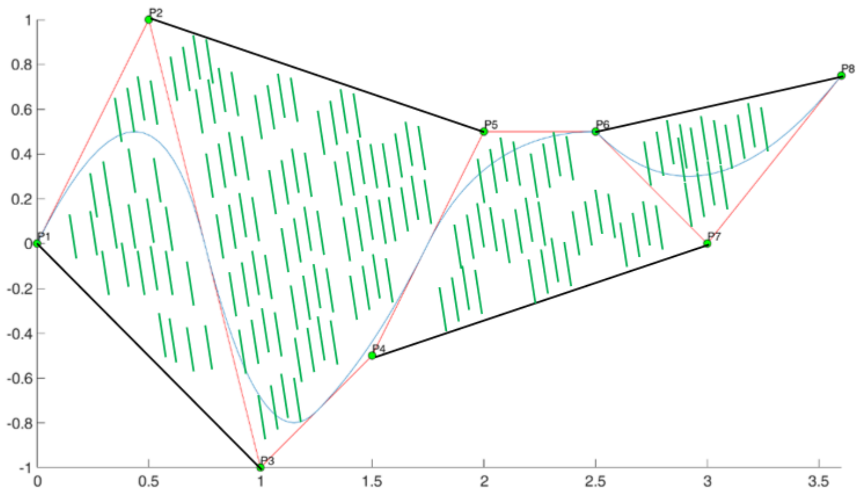


Figure 3.4: The convex hull property for a quadratic B-spline curve.

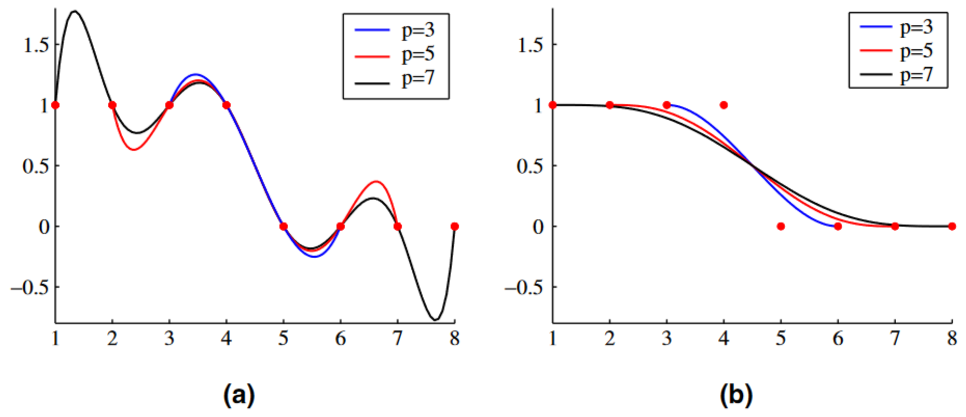
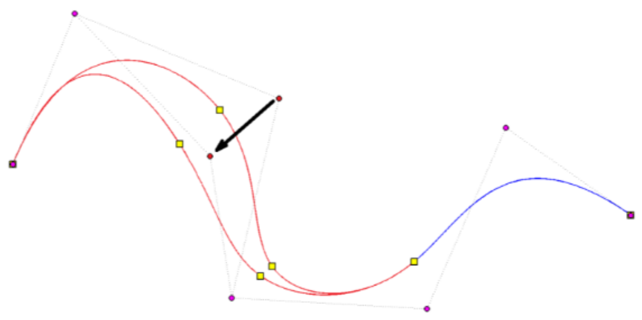
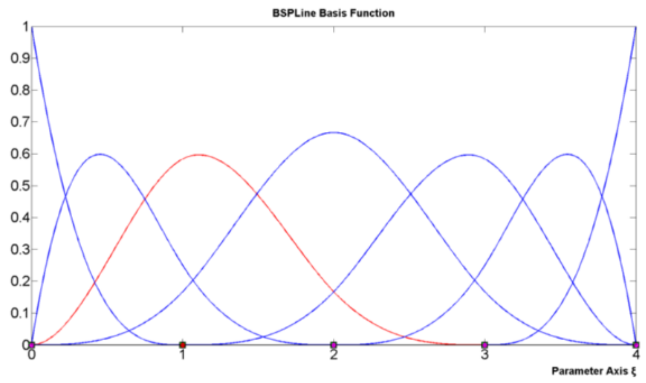


Figure 3.5: (a) Lagrange oscillations in discontinuous data and higher degree. (b) Variation diminishing property. Source: [16].

Another important property of B-splines is the local control property on curve. The compact support property tells us practically that $N_i^p(\xi) > 0, \xi \in [\xi_i, \xi_{i+p+1})$. When $\xi \notin [\xi_i, \xi_{i+p+1}) \Rightarrow N_i^p(\xi) = 0$. So, the basis function is active only in a specific region on parameter space. But every control point corresponds to a basis functions. So, moving a control point to another position in physical space do not change the whole curve as it happens on Bézier curves, but it changes the part of the curve where basis function is active [26]. That gives local control on curve, something important when a geometry is manipulated. That property can be seen on fig. 3.6 whereby changing the 3rd control points it is affected only the part of curve which depends on $N_3^p(\xi)$.



(a)



(b)

Figure 3.6: Local support property on a quadratic B-spline curve. (a) The physical space, moving the 3rd control point (b) The corresponding basis function in parameter space affecting only a specific part of the curve. Source: [26].

B-splines Surfaces

The theory of B-splines in curves is extended to surfaces using the tensor product operation between basis functions on different directions. A surface element in parametric space has the form $\Xi \otimes H = [\xi_i, \xi_{i+1}] \times [\eta_j, \eta_{j+1}]$. The extended element which supports the basis according to local support property is $\Xi_{ext} \otimes H_{ext} = [\xi_i, \xi_{i+p+1}] \times [\eta_j, \eta_{j+q+1}]$ where, p, q are the degrees of basis functions in two parametric directions. Then the basis functions that is supported by $\Xi_{ext} \otimes H_{ext}$ are the tensor product of basis function in two different parametric directions.

$$N_{i,j}^{p,q}(\xi,\eta) = N_i^p(\xi) N_j^q(\eta) \tag{3.5}$$

Where, $N_i^p(\xi), N_j^q(\eta)$ are the basis functions in different directions. Given a control net $\mathbf{P}_{ij}$ with $i = 1,2,\dots,n$ and $j = 1,2,\dots,m$ according to eq. (3.5) the surface $S(\xi,\eta)$ will be defined as:

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

$$\Leftrightarrow S(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} N_i^p(\xi) N_j^q(\eta)\,\mathbf{P}_{ij} \tag{3.6}$$

The B-spline surfaces have the same properties as B-splines curves. Below it is shown an example of B-spline surface using I3GA. The surface is a square with side length a = 1 m. In this example we have knot vectors $\mathbf{\Xi} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$ and $\mathbf{H} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$. According to these open knot vectors the degree of basis functions per parametric direction will be $p + 1 = 3 \Leftrightarrow p = 2$. Having only knot vectors we can calculate the basis functions. In order to acquire a surface, we also need a set $\{\mathbf{P}_{ij}\}$ of control points. Given the control points net of table 3.1 then the surface is shown on fig. 3.7.

Control Points	
0.000	0.000
0.500	0.000
1.000	0.000
0.000	0.500
0.500	0.500
1.000	0.500
0.000	1.000
0.500	1.000
1.000	1.000

Table 3-1: Control Points for the example surface.

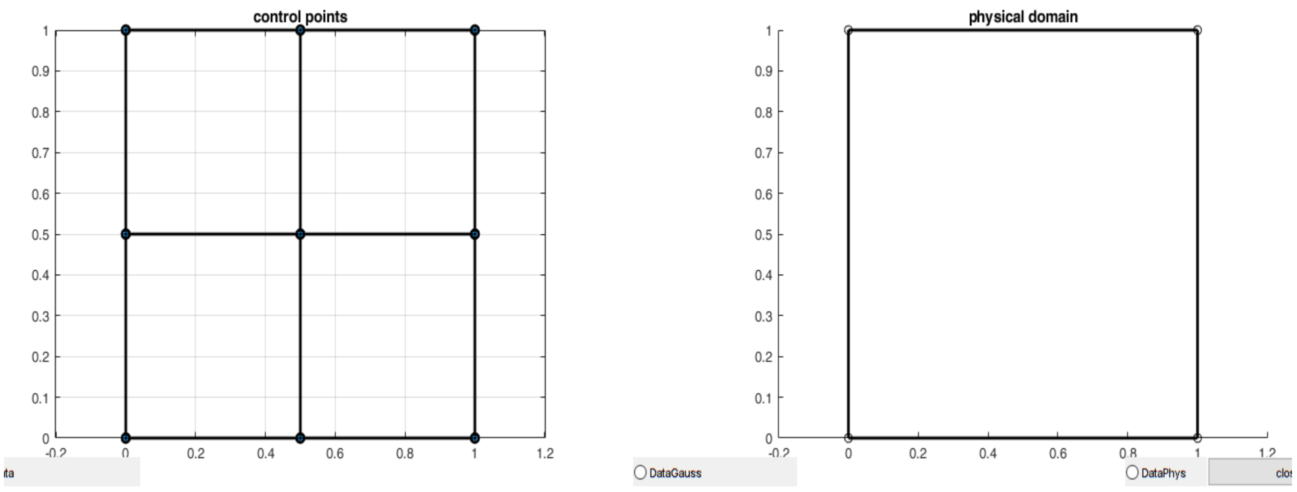


Figure 3.7: The control mesh and the example surface in I3GA.

The number of control points is in correspondence with the number of selected control points (9) of table 3.1 because according to knot vectors the number of control points per direction need to be $n + p + 1 = 6 \Rightarrow n = 3$. So, a 3×3 grid is needed to be given. The surface is C^1-continuous in every internal knot and at the boundary of control mesh every control point is interpolatory to surface because the knot vectors are open. The basis functions for this case are shown in figure below. They are 9 as the number of control points.

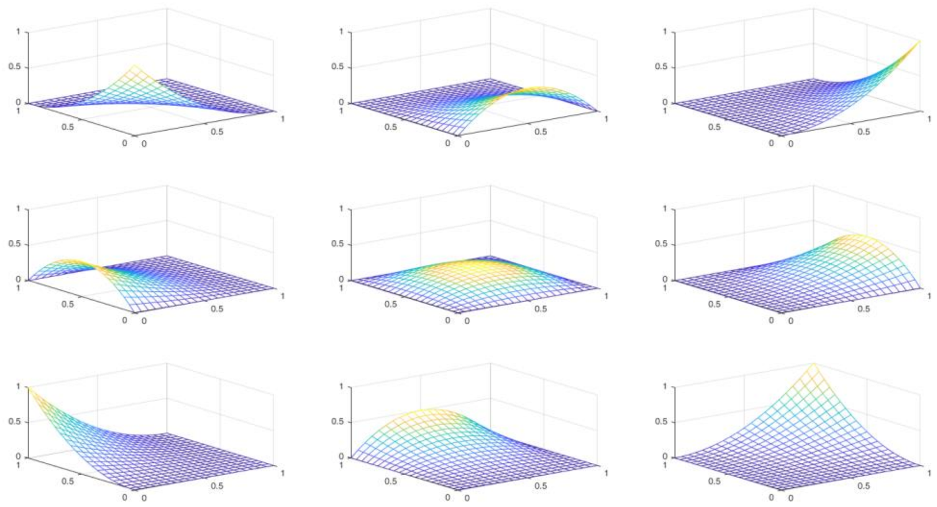


Figure 3.8:Basis functions for the example case.

As we can see every basis functions is activated near the corresponding control point where they take their maximum value. The local support property is proven also on B-spline surfaces, because of the tensor product property. If we move only one control point to a specific area of the surface then only this area will be affected as it is shown in fig. 3.9 because the corresponding basis function will be unaffected. The fact that the surface is of degree 2 can be proven from the curve that is created on the left side of square. I3GA has been programmed in order to manipulate the geometry interactively. We can simply change the geometry by clicking and moving with a mouse the control point to a specific position.

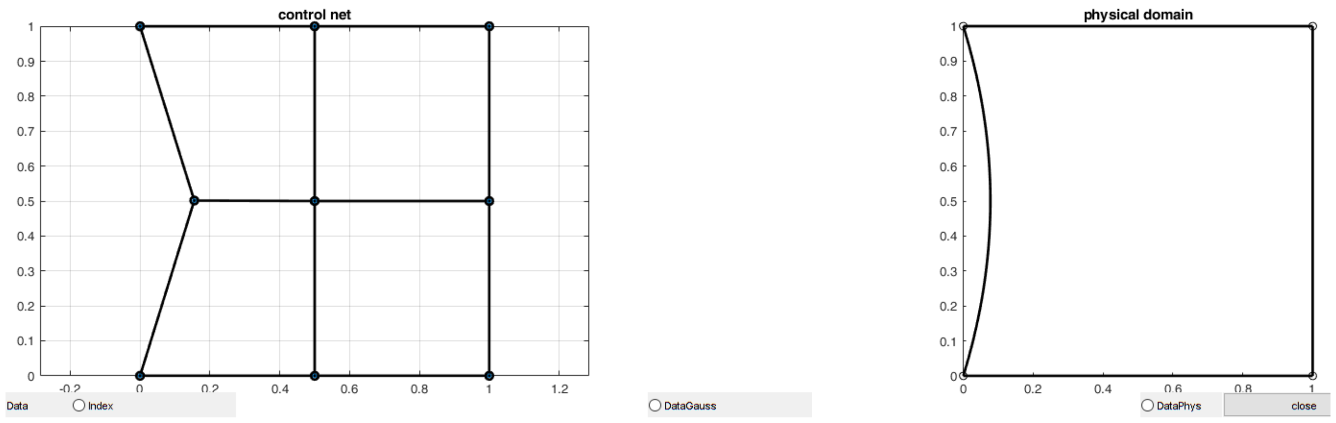


Figure 3.9: Moving only one control point, only a specific region is changed.

The theory of B-spline surface is extended also to volumes, where we will have got 3 knot vectors per parametric direction. If the degree of functions in three directions is noted as p, q, r respectively the function will be calculated by the tensor product:

$$N_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) = N_i^p(\xi)N_j^q(\eta)\, N_k^r(\zeta) \tag{3.7}$$

And given control point cloud the volume is computed as:

$$V(\xi,\eta,\zeta) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} N_i^p(\xi)N_j^q(\eta)\, N_k^r(\zeta)\, \mathbf{P}_{ijk} \tag{3.8}$$

3.3 Non-Uniform Rational B-splines (NURBS)

There are some types of curves which cannot be modelled using B-splines basis functions. These curves are quadric ones as conical sections. These curves can be constructed using a projective transformation of a B-spline curve in higher dimension. This type of basis function is called rational B-spline basis function [4], [6], [16]. Using the piecewise polynomials, we could for example approximate a circle using quadratic B-splines (see fig. 3.10). As it shown in fig. 3.10a, we project the control points $P_i^w \in \mathbb{R}^3$ of distance w_i from the origin on plane $z = 1$, where we finally have $P_i \in \mathbb{R}^2$. The control points $P_i \in \mathbb{R}^{\mathrm{d_s}}$ relate to $P_i^w \in \mathbb{R}^{\mathrm{d_s}+1}$ with:

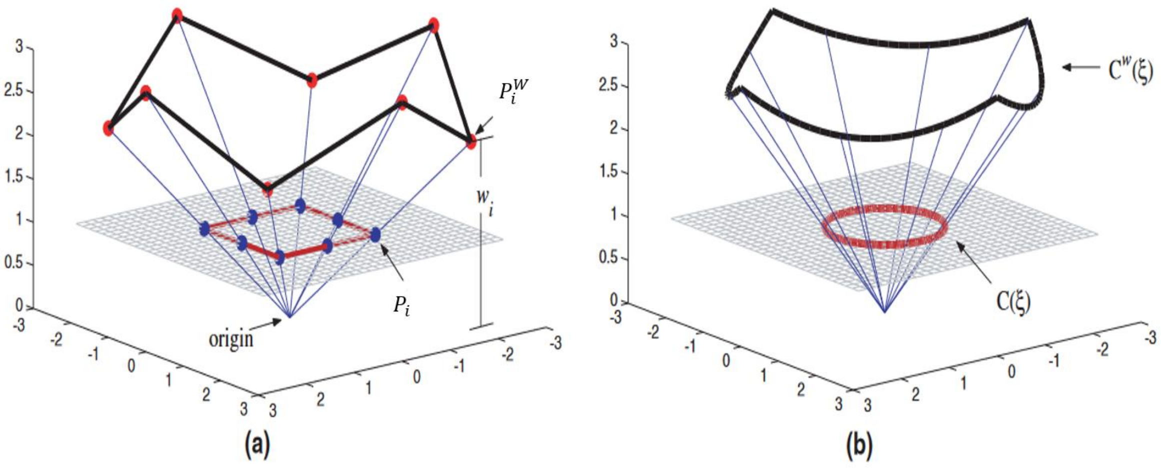


Figure 3.10: A circle constructed by the projective transformation of a B-spline curve in R^3 to R^2. (a) The control mesh. (b) The evaluated points in physical mesh. Source: [16].

$$(P_i)_j = \frac{(P_i^w)_j}{w_i}, j = 1, \ldots, \mathrm{d_s}$$

$$w_i = (P_i^w)_{\mathrm{d_s}+1}$$

Where, $(P_i)_j$ is the j^{th} component of control point vector P_i and w_i is the i^{th} *weight*. Geometrically the weight is the height of B-spline control point [3]. So, NURBS are projected to B-splines. Every major operation as refinement can be done in B-spline form and then the latter can be reprojected back to $\mathbb{R}^{\mathrm{d_s}}$ in order to acquire the NURBS form [6]. The NURBS basis functions are defined by the B-spline basis functions $N_i^p(\xi)$ in the rational form below:

$$R_i^p(\xi) = \frac{N_i^p(\xi)w_i}{W(\xi)} = \frac{N_i^p(\xi)w_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i}}^p(\xi)w_{\hat{i}}} \tag{3.9}$$

Where, $W(\xi)$ is the weighting function. The above equations can also be written in matrix form. If we define the diagonal matrix of weights as $\mathbf{W}$ then:

$$\mathbf{W} = \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_n \end{bmatrix}$$

And the column vector of B-spline basis functions $\mathbf{N}(\xi)$ then eq. (3.9) can be written in matrix form as:

$$\mathbf{R}(\xi) = \frac{1}{W(\xi)}\mathbf{W}\mathbf{N}(\xi) \tag{3.10}$$

Given a set of control points a NURBS curve is constructed as piecewise polynomial as the B-spline curve. It is defined as:

$$C(\xi) = \sum_{i=1}^{n} R_i^p(\xi)\mathbf{P}_i = \sum_{i=1}^{n} R_i^p(\xi)\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{\mathrm{d_s}i} \end{bmatrix} = \sum_{i=1}^{n} \frac{N_i^p(\xi)w_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i}}^p(\xi)w_{\hat{i}}}\begin{bmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{\mathrm{d_s}i} \end{bmatrix} \tag{3.11}$$

The NURBS inherit all the important properties of B-spline basis functions.

- Non-Negativity: $R_i^p(\xi) \geq 0,\ \forall \xi \in [\xi_1, \xi_{n+p+1}]$
- Partition of Unity: $\sum_{i=1}^{n} R_i^p(\xi) = 1$
- Linear Independence: $\sum_{i=1}^{n} \alpha_i R_i^p(\xi) = 0 \Leftrightarrow \alpha_k = 0, \quad k = 1,2,\dots,n$
- Compact or Local support: $\{\xi \,|\, R_i^p(\xi) > 0\} \subset [\,\xi_i, \xi_{i+p+1}]$
- Affine invariance
- Convex hull
- Variation diminishing property

The strong hull property on NURBS is assured only and only if the weights are non-negative. It is common the weights belong to the interval $[0,1]$. If $w_i = a \in \mathbb{R}_+^*$ for all i then $R_i^p(\xi) = N_i^p(\xi)$. This means that a NURBS curve can become a B-spline curve. Furthermore, if the knot vector has not interior knots a NURBS curve is transformed to a rational or non-rational Bézier curve [24]. In each element $[\xi_i, \xi_{i+1}]$ the curve is C^∞-continuous except the knot positions where the continuity is C^{p-m}, where m is the knot multiplicity [24]. The first derivative of a NURBS basis function which is the most important of the derivatives it is computed as:

$$\frac{d}{d\xi}R_i^p(\xi) = w_i \frac{W(\xi)N_i^{p\prime}(\xi) - W'(\xi)N_i^p(\xi)}{W^2(\xi)} \tag{3.12}$$

Where, $N_i^{p\prime}(\xi) = \frac{d}{d\xi}N_i^p(\xi)$ and $W'(\xi) = \sum_{\hat{i}=1}^{n} N_{\hat{i}}^{p\prime}(\xi)w_{\hat{i}}$.

NURBS surface

As for surfaces NURBS follow the same procedure as B-splines, they form a tensor product. A NURBS surface is defined as:

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

$$\Leftrightarrow S(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_i^p(\xi)R_j^q(\eta)\,\mathbf{P}_{ij} \tag{3.13}$$

Where, p, q is the degree of basis functions in two parametric directions. NURBS surfaces has all properties of NURBS curve except for the variation diminishing property [24]. Below it is following an example of NURBS surface using I3GA. In fact, I3GA it is a program for constructing NURBS surfaces and not B-splines. It is used before for B-splines, because the square that was displayed it is NURBS with all control point weights equal to 1. In this example we will create a circle of diameter $d = 1$ m. The knot vectors for this case are $\mathbf{\Xi} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$ and $\mathbf{H} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$, so the degree is $p, q = 2$ for the two directions. The control points and their respective weights are shown in table below. The result is shown on fig. 3.11 and the respective basis functions on fig. 3.12. As we can see in figure 3.12 each basis function influences a rectangle area of $[\xi_i, \xi_{i+p+1}] \times [\eta_j, \eta_{j+q+1}]$. The local support property still is proven on NURBS surfaces as we can see in fig. 3.13. It is obvious that if we change the weights the geometry changes. That can be seen on figures 3.14 and 3.15 where we set the control point weight equals to $w_i = 0.5$. Changing the weights can have significant effect on a NURBS entity.

Control Points		Weights
-0.3536	0.3536	1.000
-0.7071	0.000	0.7071
-0.3536	-0.3536	1.000
0.000	0.7071	0.7071
0.000	0.000	1.000
0.000	-0.7071	0.7071
0.3536	0.3536	1.000
0.7071	0.000	0.7071
0.3536	-0.3536	1.000

Table 3-2: Control points and weights for the circle of diameter $d = 1m$.

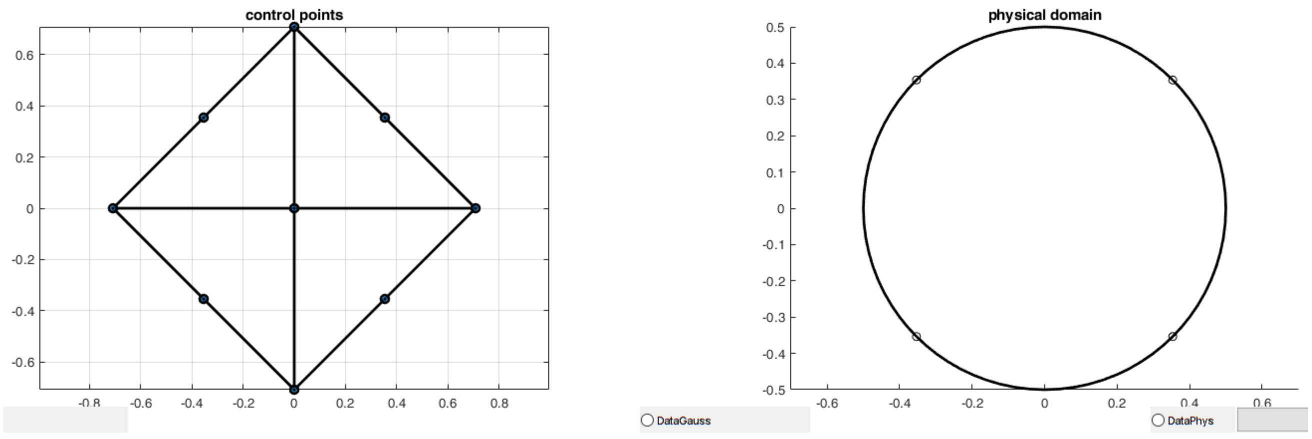


Figure 3.11: A circle surface created from NURBS.

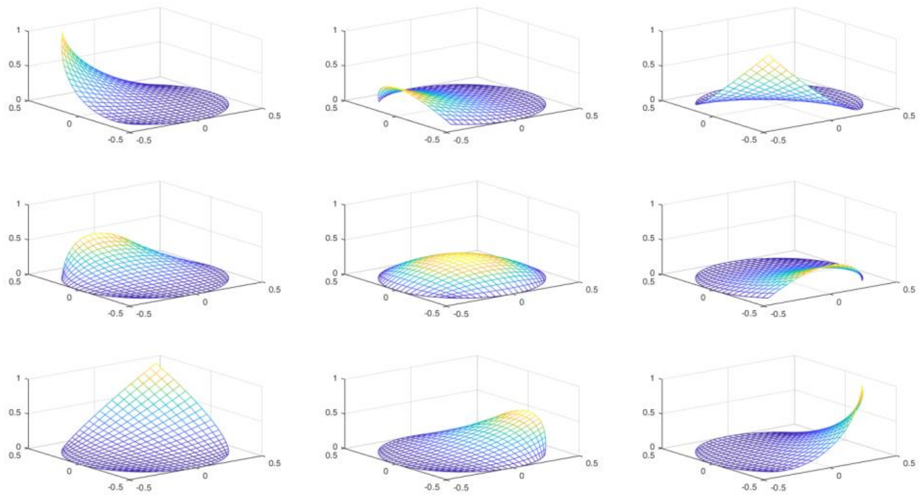


Figure 3.12: The basis function of NURBS surface.

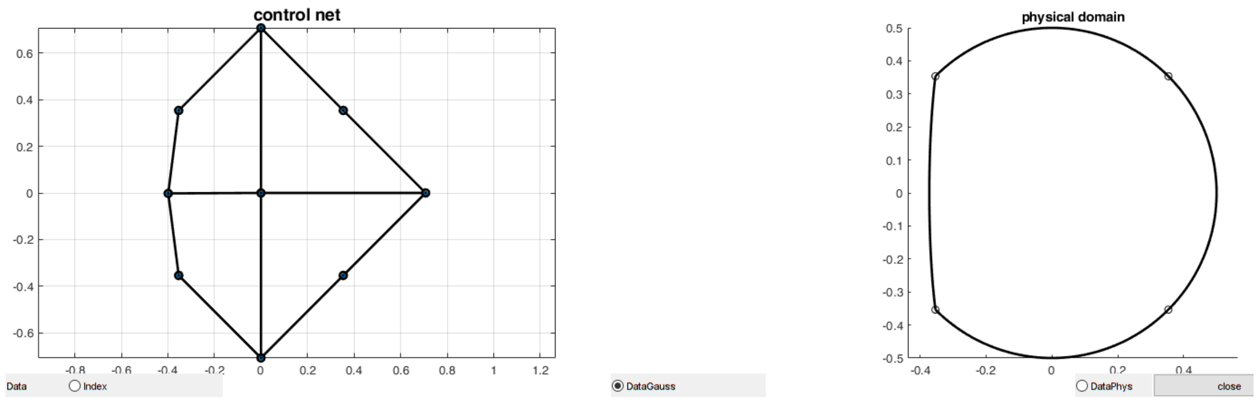


Figure 3.13: The local support property of NURBS after moving a specific point to another position.

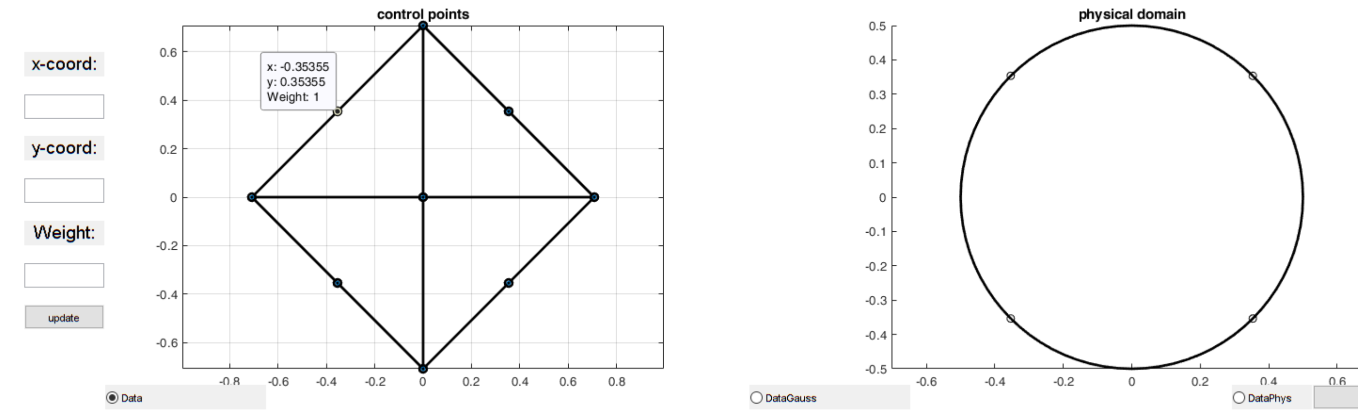


Figure 3.14: The surface before changing the control point in I3GA. In I3GA we can select a control point and change its position and weight on our own will.

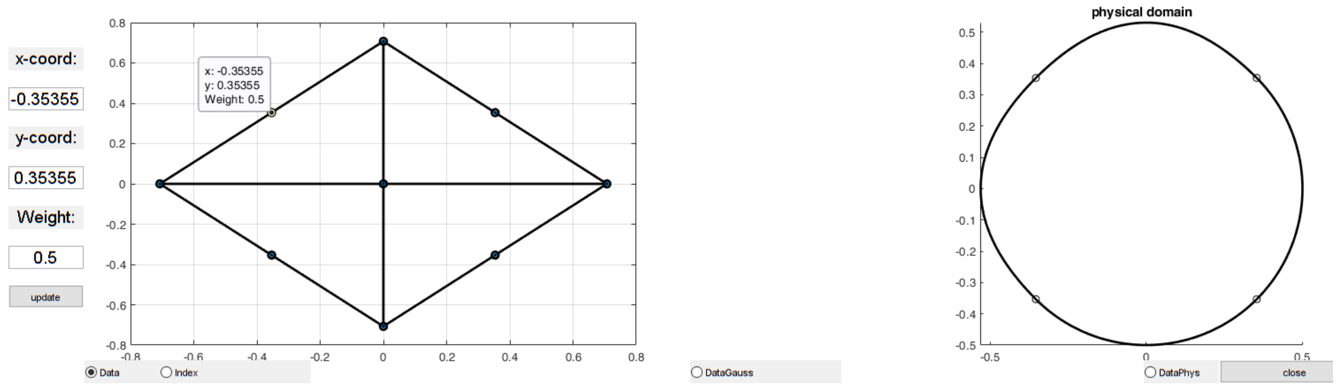


Figure 3.15: The control mesh and the physical domain of the curve after changing the weight of control point from 1 to 0.5.

3.4 T-splines

NURBS basis functions is a widely used tool in CAD and IGA community, but they have some limitations. As already been written in Introduction, NURBS cannot mostly achieve smooth representation when two or more NURBS patches need to form a continuous surface. In previous sections it is showed that if the knot vectors are open and clamped, then in the boundaries of the curve or the surface, the continuity is C^0 because the multiplicity of knots in the boundaries (0 or 1) is $p+1$. As a result, if two NURBS surfaces are coincide their representation is not smooth. In case which the NURBS do not have common boundary it cannot achieve merging without topological violation of the two surfaces [16]. This creates surfaces with gaps or overlaps. Furthermore, NURBS show another drawback, that of global refinement. Refinement process will be explained in the next chapter, but its main concept is that with refinement we divide an element into more elements in order to add extra information for analysis, without changing the parameterization of physical domain. In NURBS, more information is added from the needed one. These shortcomings can be dealt with using a new kind of spline, the T-spline.

In order to explain T-splines first we need to explain the *Point Base*-splines (PB-splines). In previous section we saw that NURBS knot vector is defined for all basis functions. However, from local support property it is showed that a basis function is only active on the interval $[\xi_i, \xi_{i+p+1}]$. So, instead of defining a global knot vector we define local knot vectors $\Xi_i^{loc} = \{\xi_{i+j}\}_{j=0}^{p+1}$. Each basis functions will be defined now from a local knot vector of length $p+2$ [16]. The basis functions are now called "blending" functions because properties as linear independency are not valid here [6]. If we now define a set of arbitrary local knot vectors with each knot vector being independent in its element from others, then we can define a blending function for each element. This set of blending functions composes the PB-splines [6].

PB-splines

Let suppose that we are in $\mathbb{R}^2$, and we want to create a surface using PB-splines. If we define some local knot vectors of arbitrary length m_a then we can collect all knot vectors to a set $\boldsymbol{\Xi_\alpha}$ for all parametric directions as:

$$\boldsymbol{\Xi_\alpha} = \{\boldsymbol{\Xi_\alpha^l}\}_{\boldsymbol{l=1}}^{\boldsymbol{2}} \tag{3.14}$$

and

$$\boldsymbol{\Xi_\alpha^l} = \{\xi_i^l\}_{i=1}^{m_a}$$

Where l is the index of parametric direction. Each knot vector is associated to a blending function $N_a^l(\xi^l)$[3] with degree $p_a = m_a - 2$. The blending function for the bivariate case defined by $\boldsymbol{\Xi}_{\boldsymbol{\alpha}}$ is given as:

$$B_a(\xi,\eta) \equiv B_a(\xi^1,\xi^2) = \prod_{l=1}^{2} N_a^l(\xi^l) \tag{3.15}$$

The support of these can be defined as:

$$\mathbf{D}_{\boldsymbol{a}} = \otimes_{\boldsymbol{l=1}}^{\boldsymbol{2}} \left[\xi_1^l, \xi_{m_a}^l\right]$$

If we collect all blending functions to a set A the irregular space in which PB-spline is created is defined as:

$$\mathbf{D} = \bigcup_{\alpha\in A} \mathbf{D}_{\boldsymbol{a}} \tag{3.16}$$

$\mathbf{D}$ is defined in order that $\sum_{\alpha\in A} B_a(\xi,\eta) > 0$ [4]. The blending functions can acquire a rational form in $\mathbf{D}$ if we define them as:

$$R_a(\xi,\eta) = \frac{B_\alpha(\xi,\eta)}{\sum_{\beta\in A} B_\beta(\xi,\eta)} \tag{3.17}$$

If knot vectors are such that we can establish a basis using $R_a(\xi,\eta)$ [4], then by assigning a control point to each basis function we can define a PB-spline surface as (see fig. 3.16):

$$S(\xi,\eta) = \sum_{\alpha\in A} \mathbf{P}_\alpha R_a(\xi,\eta) \tag{3.18}$$

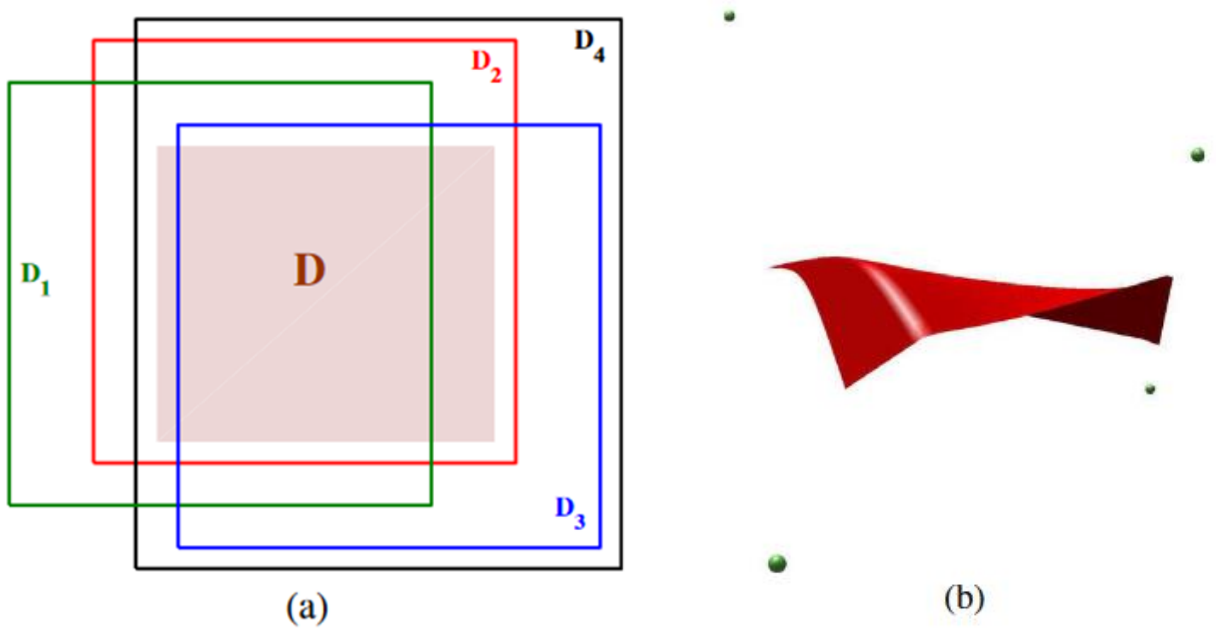


Figure 3.16: PB-spline. (a) The support space for the spline (b) PB-spline surface with four control points. Source: [4]

[3] $\xi^1 \equiv \xi$ and $\xi^2 \equiv \eta$ according to the notation in previous sections of the chapter.

From fig. 3.16 we can see that there is not an obvious ordering in control points as they form a *control cloud*. This unstructured environment ensures that some important features of NURBS still exist however there are some problems. For example, the concept of element is abolished, and refinement process is difficult to be accomplished. So, we need a set of blending functions that preserve the freedom of PB-splines along with the structure properties of NURBS. That is the T-spline technology.

T-spline and T-mesh

T-spline combine the benefits from both of NURBS and PB-splines as they are balanced between the two. In T-splines each blending function is defined from a local knot vector which has been inferred from a global "knot vector" as in NURBS. The global knot vector is a structure in index space, the *T-mesh*. The T-mesh has at least 2 dimensions, so T-splines do not form curves. A T-mesh is shown on fig. 3.17 and has been produced by a code from Ioannis Dimitriou. As we can see the T-mesh has T-junctions which are vertices connected with three edges. That do not exist in NURBS. In T-mesh the construction of local knot vectors is different in case of odd or even degree T-splines.

In odd-degree T-splines for each vertex we set an anchor $s_a = \{i, j\}$ and each one of them is associated with a control point. Then to form the local knot vector we initially set the values of s_a in the middle of knot vector . Then for horizontal (i) direction we add the right knots from the middle by recording the values of $\frac{(p+1)}{2}$ orthogonal edges to the right of the anchor in T-mesh. The same happens for the left knots. For the vertical direction (j) we do the same procedure with the difference that instead of right and left we travel up and down. Below an example is following. If we consider a T-spline of $p = 3$ then for anchor $s_a = \{3,3\}$ in fig. 3.18 the local knot vectors are $\Xi_\alpha = \{1,2\ 3,6,7\}^{\mathrm{T}}$ and $H_a = \{1,2,3,4,6\}^{\mathrm{T}}$. For even degree the differences are in the positions of anchors (where they fall at the center of each tile) and the fact that we now must record $\frac{p}{2} + 1$ values. In general, the index space plays very important role on T-splines, but in NURBS they are auxiliary.

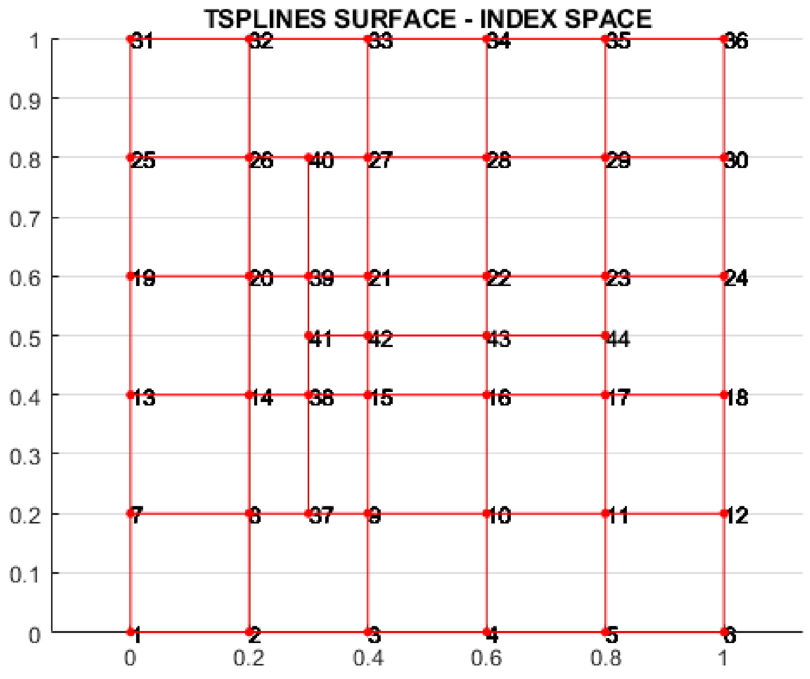


Figure 3.17: A T-mesh. Created by Ioannis Dimitriou.

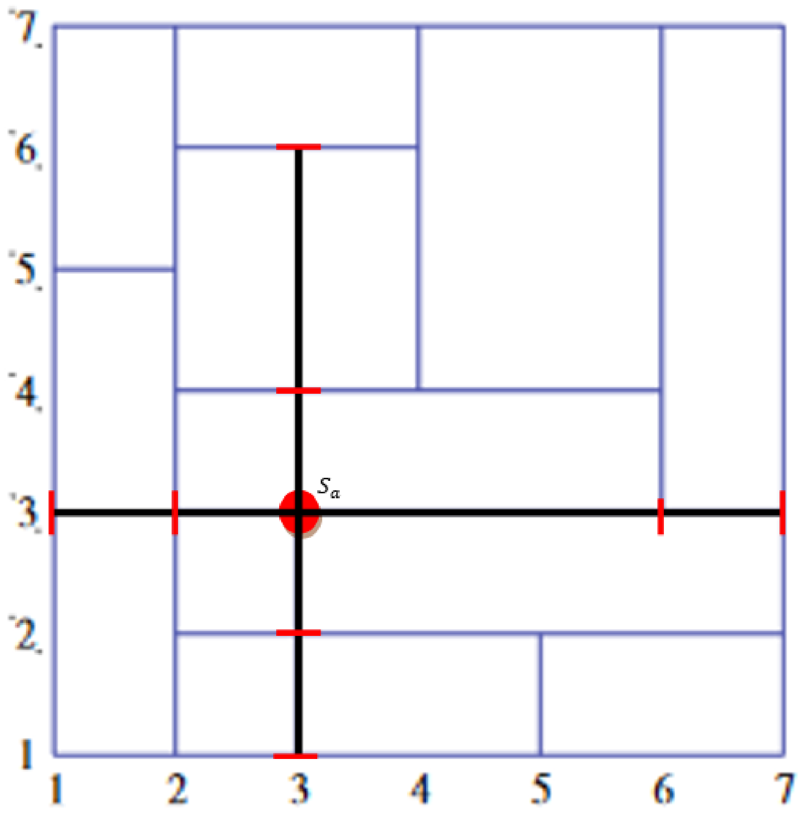


Figure 3.18: An anchor at $s_a = \{3,3\}$ from T-mesh in [4].

T-splines blending functions are like NURBS with the difference that now the blending function is not produced by a global tensor product. The tensor product happens in a local level. The blending function is defined by:

$$R_a(\xi,\eta) = \frac{w_a B_a(\xi,\eta)}{\sum_{\beta\in} w_\beta B_\beta(\xi,\eta)} \tag{3.19}$$

If we define the control point $\boldsymbol{P}_a$ for each $a \in A$ then a T-spline surface is given by:

$$S(\xi,\eta) = \sum_{\alpha\in A} \mathbf{P}_\alpha R_a(\xi,\eta)$$

T-splines have most of beneficial properties of NURBS and PB-splines but they cannot be used in analysis immediately. This happens because T-spline blending functions are not linearly independent. In order to be used in analysis, T-splines must be transformed into Analysis-Suitable T-splines [3], [27].

4 Refinement Procedures in NURBS

When we want to perform analysis to a specific geometry, we may want to add some additional information to computational mesh. For example, in fluid mechanics the mesh near boundary must be more fined in order to address boundary element phenomena and approximate better the field variables for the problem. Another example would be from CAD industry where we need to refine a mesh in order to give to a surface more degrees of freedom. In IGA there are two main ways for refining the physical mesh [16]. The first one is Knot insertion or h-refinement and the second one is called Degree elevation or p-refinement. Each method has its own advantages and disadvantages and can be combined in order to create a new method k-refinement. I3GA has both h and p refinements. H-refinement has been programmed with three different methods that will be explained. So, in this chapter we will explain the procedures for all refinement methods along with examples from code.

4.1 Knot Insertion (H-refinement)

Knot insertion is the process of inserting a new knot to index space. By inserting a new knot, the length of knot vector is increased by one; thus, a new control point must be added to the curve. This process gives more degrees of freedom in geometry without changing it [16]. Given a NURBS curve of degree p with a knot vector $\Xi = \{\xi_1, \xi_2, \dots, \xi_{n+p+1}\}^{\mathrm{T}}$ we want to add a new knot at $\bar{\xi} \in [\xi_k, \xi_{k+1}]$. The new knot vector Ξ^* will be: $\Xi^* = \{\xi_1, \xi_2, \dots, \xi_k, \bar{\xi}, \xi_{k+1}, \xi_{n+p+1}\}$. As a result, the length of knot vector will be $\|\Xi^*\| = (n+1) + p + 1$, so another control point and basis function must be added to the curve. However, not only a single control point must be added but also the positions of old control points must change in order to preserve the curve parametrically. So, if the initial set of control point is $\{P_i\}_{i=1}^{n}$ then the new set of control points $\{\bar{P}_i\}_{i=1}^{n+1}$ will be computed by the relationship below [16], [24]:

$$\bar{P}_i = a_i P_i + (1 - a_i) P_{i-1} \tag{4.1}$$

Where,

$$a_i = \begin{cases} 1, & 1 \le i \le k - p \\ \dfrac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i}, & k - p + 1 \le i \le k \\ 0, & k + 1 \le i \le n + p + 2 \end{cases}$$

Below you see an example of knot insertion to a quadratic curve of knot vector $\Xi = \{0,0,0,1,1,1\}^{\mathbf{T}}$. The inserted knot is at $\bar{\xi} = 0.5$. We can observe that the two curves are geometrically exact. The only thing that changes is the number of control points and basis functions. The continuity of the curve is also preserved after knot insertion [16]. The only case where continuity is reduced is when we add the same knot more times. The same procedure can also be repeated when we want to add multiple knots. This process is called knot refinement [24]. Knot refinement has many applications as [24]:

- Decomposition of B-spline curves (or surfaces) to Bézier counterparts.
- Merging of two or more knot vectors in order to acquire a curve with a final knot vector.
- Bring closer the curve (or surface) to control polygon (or control net respectively).

Knot refinement has some efficient algorithms that are based on eq. (4.1) but the one that is used is the algorithm of Boehm and Prautzsch [24]. The algorithm works on B-splines and not on NURBS. In order to be used for NURBS we need to transform the control points into their projective counterparts, perform knot refinement and then reproject the new control points. In fig. 4.2 we can see an example where we insert knots $\xi_1 = 0.25$ and $\xi_2 = 0.75$ in knot vector $\Xi = \{0,0,0,0.5,1,1,1\}^{\mathrm{T}}$. Now two control points must be added because we added two knots. As we can see the new (green) control points do not change the curve at all.

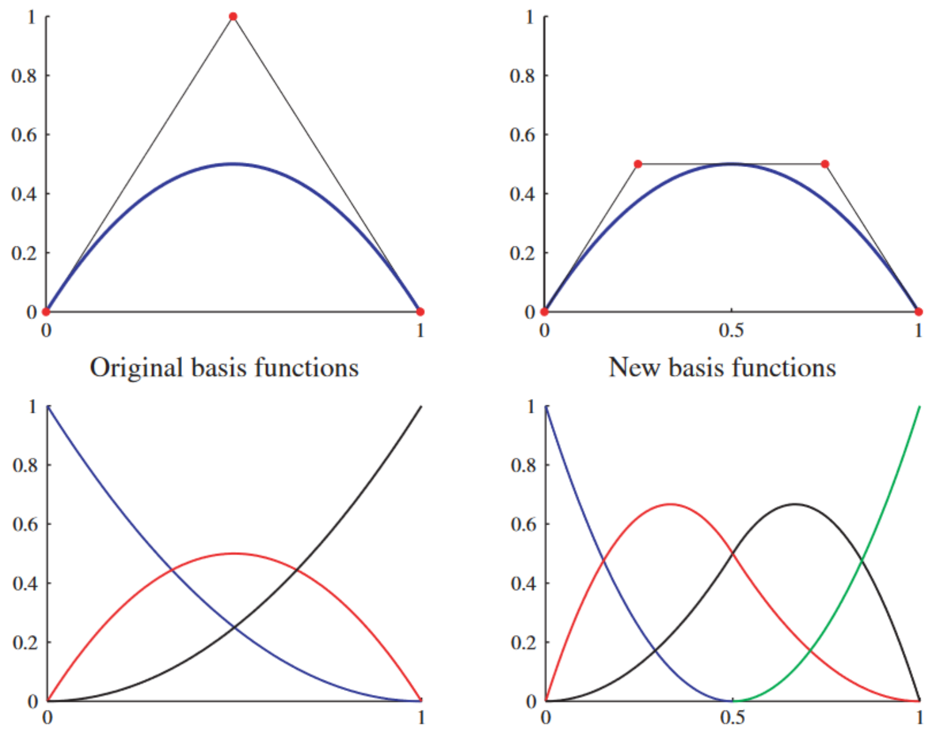


Figure 4.1: Knot insertion in curve with original knot vector $\Xi = \{0,0,0,1,1,1\}$**.** The red dots are the control points. We can also see the basis functions. Source: [16].

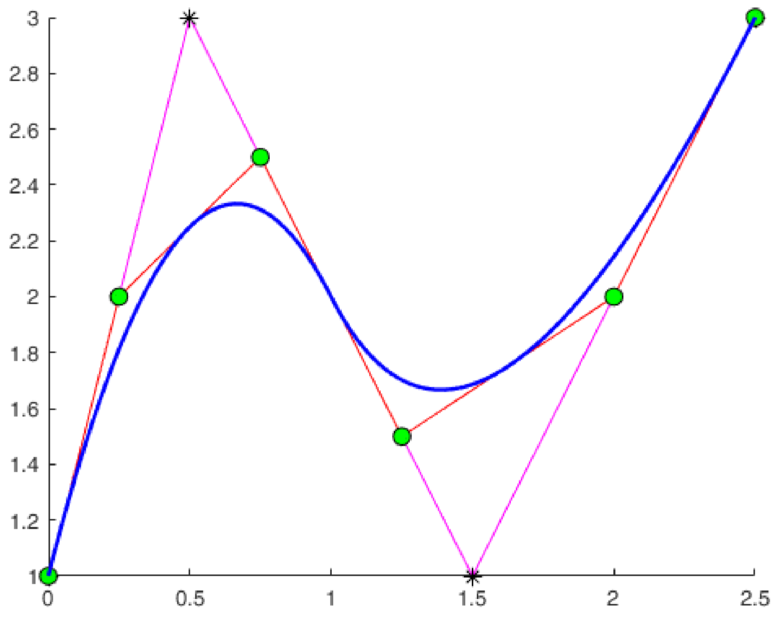


Figure 4.2: A quadratic NURBS curve with original knot vector $\Xi = \{0,0,0,0.5,1,1,1\}$. The "*" control points are the old ones and the green dots the new ones.

Knot refinement can be done also using another algorithm in matrix form. This algorithm is an improved version of a known refinement procedure known as the Oslo Algorithm [28]. This algorithm was developed from scratch at the lab and has the capability of inserting multiple knots simultaneously in both directions. Supposing that we have an original knot vector $\boldsymbol{\Xi} = \{\xi_1, \xi_2, \dots, \xi_{n+p+1}\}^{\mathbf{T}}$ and its extended form $\boldsymbol{\Xi} = \{\bar{\xi}_1, \bar{\xi}_2, \dots, \bar{\xi}_{n+m+p+1}\}^{\mathbf{T}}$ containing the new m knots. Then, the new control points $\overline{\mathbf{P}} = \{\overline{\mathbf{P}}_1, \overline{\mathbf{P}}_2, \dots, \overline{\mathbf{P}}_{n+m}\}^{\mathrm{T}}$ are formed by the original ones $\mathbf{P} = \{\mathbf{P}_1, \mathbf{P}_2, \dots, \mathbf{P}_n\}^{\mathrm{T}}$ from the relationship below:

$$\overline{\mathbf{P}} = \mathbf{T}^p\mathbf{P} \tag{4.2}$$

With $\mathbf{T}^p \in \mathbb{R}^{(\boldsymbol{n}+\boldsymbol{m})\times\boldsymbol{n}}$ be computed by:

$$T_{i,j}^0 = \begin{cases} 1, & \text{if } \bar{\xi} \in [\xi_j, \xi_{j+1}] \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

And

$$T_{i,j}^{q+1} = \frac{\bar{\xi}_{i+q} - \xi_j}{\xi_{j+q} - \xi_j} T_{i,j}^q + \frac{\xi_{j+q+1} - \bar{\xi}_{i+q}}{\xi_{j+q+1} - \xi_{j+1}} T_{i,j+1}^q \, , q = 0,1,2, \dots, p-1 \tag{4.4}$$

Using the equations above we can insert multiple knots to a knot vector. The process for using the above equations is described in flowchart below:

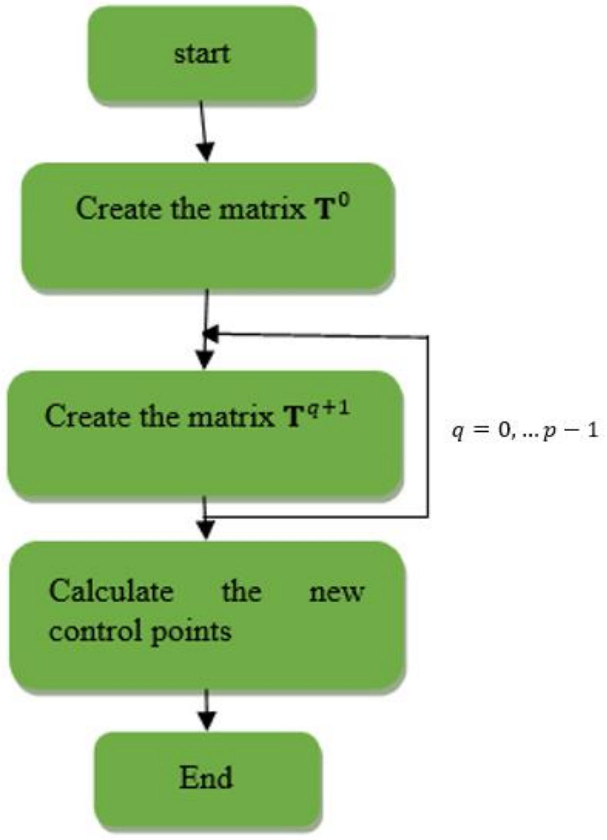

start
Create the matrix $\mathbf{T}^0$
Create the matrix $\mathbf{T}^{q+1}$
$q = 0, \dots p - 1$
Calculate the new control points
End

Figure 4.3: Flowchart that shows the knot refinement process.

An important observation in order to check the validation of the code is to check if the sum of elements in every row of T^q is 1. The polynomials defined by (4.4) are also called Kharitonov polynomials according to a report by Ioannis Dimitriou. Both Boehm's algorithm and algorithm of equation (4.2) are extended to surfaces. In case of algorithm of eq. (4.2) if we add knots to one direction we can compute the $\mathbf{T}^{\boldsymbol{p}}$ and then multiply it by the row or column in a control mesh [4]. In practice we do the process for both directions with the same manner one for the first direction and then for the second one.

In I3GA we have used the two algorithms for different occasions. In fig. 4.4 we can see the refinement menu of I3GA. In this menu 5 options for h-refinement do exist. The first 3 options take place using Boehm's algorithm and the final two using the algorithm of eq. (4.2). Specifically in the first 3 options, the new knots are inserted in positions where a knot span (element) $[\xi_i, \xi_{i+1}], \xi_i \neq \xi_{i+1}$ is divided by a specific value $n \in (0,1]$. In other words, we subdivide each element by a value n and then we import the new values in the original knot vector. For example, if the initial knot vector is $\boldsymbol{\Xi} = \{0\ 0\ 1\ 1\}^{\mathrm{T}}$ and we subdivide at $n = 0.5$, a new knot will be added in half of the unique element [0,1] and the new knot vector will be $\bar{\boldsymbol{\Xi}} = \{0\ 0\ 0.5\ 1\ 1\}^{\mathrm{T}}$. The three choices for subdivision are:

- "Refine globally at 0.5": Each knot span (of the form $[\xi_i, \xi_{i+1}], \xi_i \neq \xi_{i+1}$) is divided at half in both directions.
- "Refine globally at n": Each knot span is divided at $n \in (0,1]$ in both directions.
- "Refine globally in different directions": Each knot span is divided in both directions at $n_\xi \in (0,1]$ and $n_\eta \in (0,1]$ respectively.

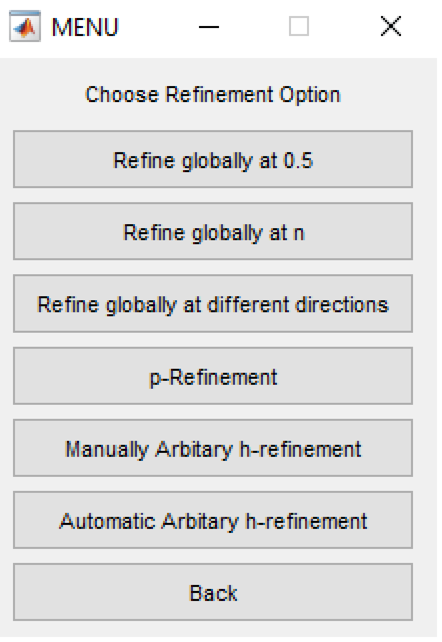

MENU
Choose Refinement Option
Refine globally at 0.5
Refine globally at n
Refine globally at different directions
p-Refinement
Manually Arbitary h-refinement
Automatic Arbitary h-refinement
Back

Figure 4.4: Refinement menu of I3GA.

Below we see examples of this type of knot refinement in L-shape with knot vectors $\boldsymbol{\Xi} = \{0\ 0\ 0\ 0.5\ 0.5\ 1\ 1\ 1\}^{\mathrm{T}}$ and $\mathbf{H} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$ in both directions. We can observe that in ξ-direction we have double knot at $\xi = 0.5$. That means at first that from ξ-direction we have two elements. In fact the number of elements in this geometry are two because in ξ-direction we have $elem_{\xi 1}$: [0,0.5] and $elem_{\xi 2}$: [0.5,1] and in η-direction $elem_{\eta 1}$: [0,1]; thus we have in total two elements $elem_1$: [0,0.5] × [0,1], $elem_2$: [0.5,1] × [0,1] (see fig. 4.6 for parameter space). The two elements are showed also in physical space where the patch is divided in two sub-patches with a diagonal line where is the knot position ($\xi = 0.5$) in physical space (see fig. 4.5). The second observation is that the continuity at $\xi = 0.5$ is C^0, so the two sub-patches are joined only with C^0 continuity at their boundary.

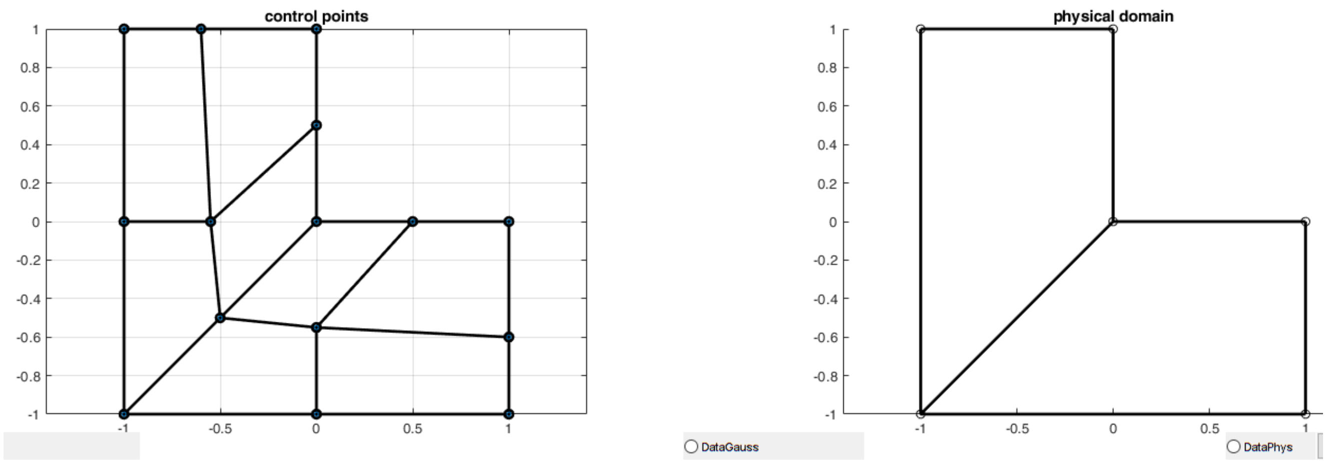


Figure 4.5: The original L-shape.

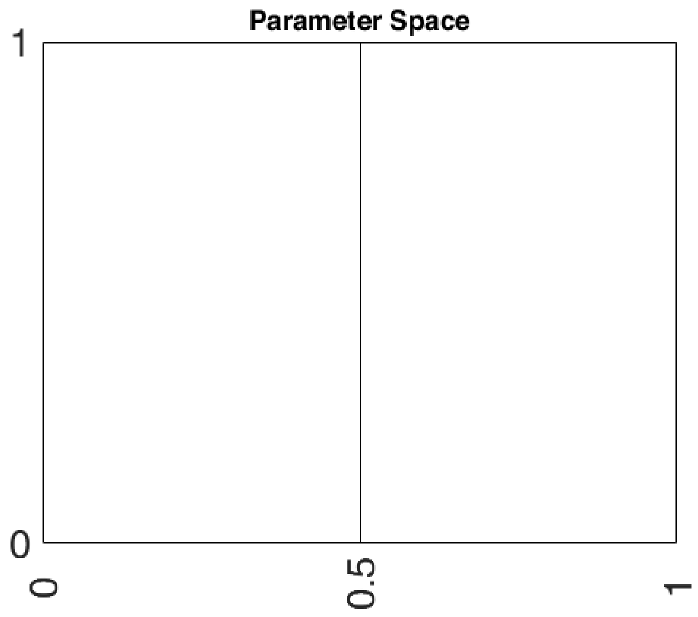


Figure 4.6: The parameter space for the L-shape.

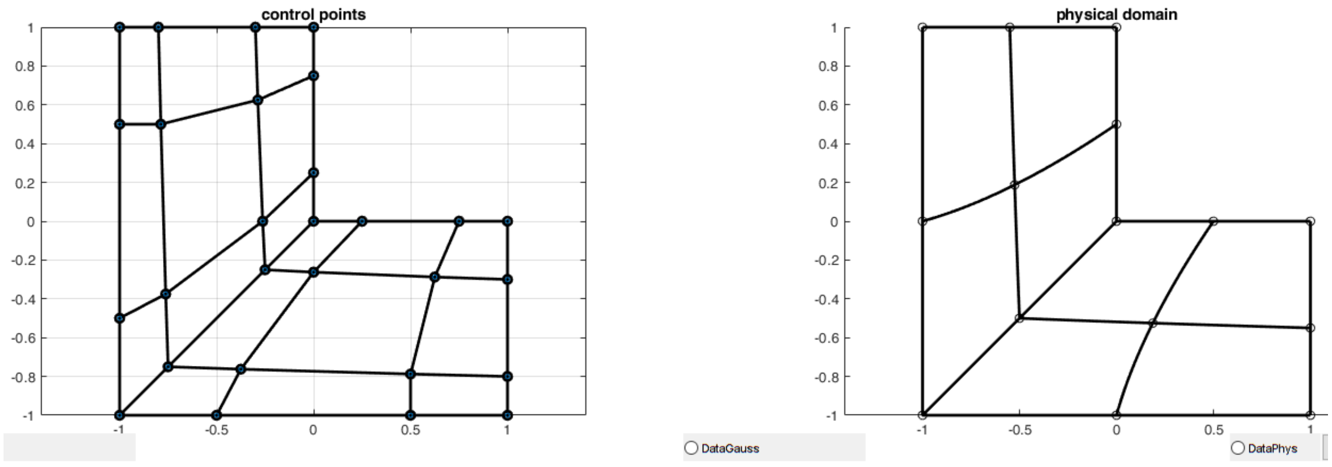


Figure 4.7: Knot refinement with subdivision at n=0.5.

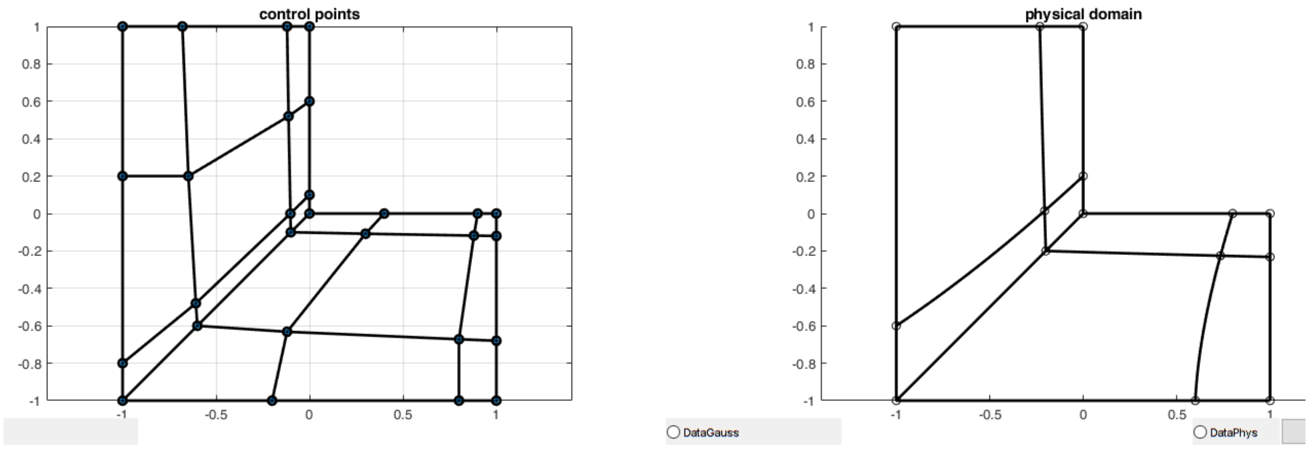


Figure 4.8: Knot refinement with subdivision at $n = 0.8$.

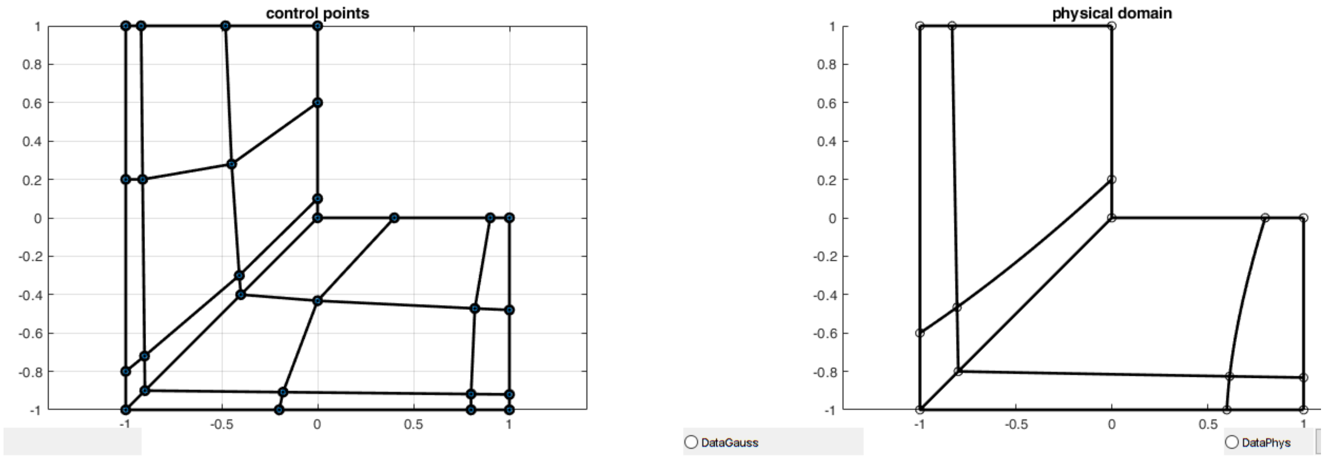


Figure 4.9: Knot refinement with subdivision at $n_\xi = 0.8$ and at $n_\eta = 0.2$.

From the above figures we make some conclusions. The first one is that refinements are successful because the geometry (along with continuity) is preserved. The goal of knot refinement is to make a coarse mesh finer without change its parameterization. That is succeeded using Boehm's algorithm. A second observation is that from the image of knot lines in physical space we can find the knot vector in each direction. In NURBS knot refinement is a global process, so if we insert a knot, the whole patch is affected. In physical domain of fig. 4.7 to 4.9 we can see that except the initial diagonal line we have also 3 lines. Two in ξ-direction and one "L-line" in η-direction. From the position of lines, we can understand the knots inserted by knot refinement process in knot vectors. For example, the "L-line" when $n_\eta = 0.8$ is closer to the northeast edge and when $n_\eta = 0.2$ is more near at southwest edge respectively. These lines change their positions proportional to the subdivision number n. We make denser an area in physical domain just by changing the subdivision number.

The other two knot refinement options use the algorithm of eq. (4.2). In "Manually arbitrary h-refinement" we can insert multiple knots in different directions and then perform knot refinement. In "Automatic arbitrary h-refinement" we insert multiple knots automatically in each direction by inserting the start, the end knot, and the knot generation step. Below you see an example of inserting new knots at $\xi = \{0.1, 0.9\}$ and at $\eta = 0.8$ in circle of diameter $d = 1$ m and knot vectors $\boldsymbol{\Xi} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$ and $\mathbf{H} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$. As we can see the knots are inserted without changing the initial geometry, something that implies that the new control points and weights are calculated correctly. Manual h-refinement can be used in order to explicitly add a new knot in area of interest when we want to make mesh finer. This algorithm will be used in adaptivity and of course in new form of knot refinement, the knot refinement in physical domain.

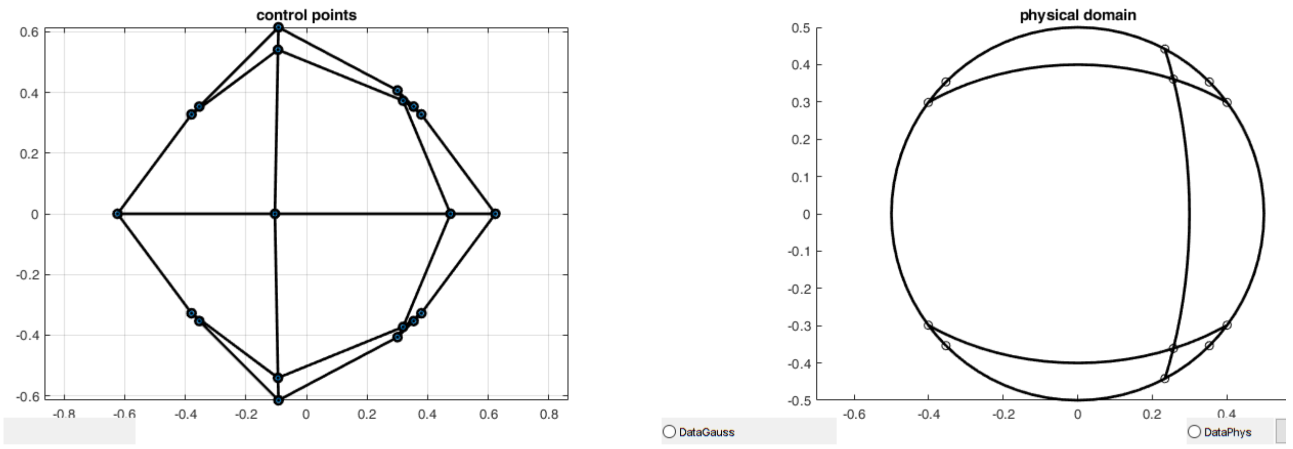


Figure 4.10: Knot refinement at knots $\xi = 0.1, 0.9$ and at $\eta = 0.8$.

Knot refinement via point inversion

Until now we saw how to insert a new knot in parameter or index space. Although most of CAD programs insert a knot by inserting a point in physical space. This is a more logical procedure because we want to make finer the mesh in physical domain and not directly in parameter space. So, by choosing a point in the physical space we want to add the corresponding knot in parameter space globally and do the "knot refinement" process. Point inversion is the procedure of finding the parameter values $\hat{\xi}, \hat{\eta}$ of a surface point $\mathbf{P}(x, y) = \boldsymbol{S}(\hat{\xi}, \hat{\eta})$. The procedure of point inversion is [24]:

1. At first, we form the vector function:

$$\boldsymbol{r}(\xi, \eta) = \boldsymbol{S}(\xi, \eta) - \mathbf{P}(x, y) \Leftrightarrow$$
$$\Leftrightarrow \boldsymbol{r}(\xi, \eta) = \boldsymbol{S}(\xi, \eta) - \mathbf{P}$$

   And two scalar functions:

$$\begin{aligned} f(\xi,\eta) &= \boldsymbol{r}(\xi,\eta)\cdot\frac{\partial \boldsymbol{S}(\xi,\eta)}{\partial \xi} = 0 \\ g(\xi,\eta) &= \boldsymbol{r}(\xi,\eta)\cdot\frac{\partial \boldsymbol{S}(\xi,\eta)}{\partial \eta} = 0 \end{aligned} \tag{4.5}$$

2. The we solve eq. (4.5). If we define as:

$$\delta_i = \begin{bmatrix} \Delta\xi \\ \Delta\eta \end{bmatrix} = \begin{bmatrix} \xi_{i+1} - \xi_i \\ \eta_{i+1} - \eta_i \end{bmatrix}$$

$$J_i = \begin{bmatrix} \left.\frac{\partial f}{\partial \xi}\right|_{(\xi_i,\eta_i)} & \left.\frac{\partial f}{\partial \eta}\right|_{(\xi_i,\eta_i)} \\ \left.\frac{\partial g}{\partial \xi}\right|_{(\xi_i,\eta_i)} & \left.\frac{\partial g}{\partial \eta}\right|_{(\xi_i,\eta_i)} \end{bmatrix} = \begin{bmatrix} \left.\left|\frac{\partial \boldsymbol{S}}{\partial \xi}\right|^2 + \boldsymbol{r}\cdot\frac{\partial^2 \boldsymbol{S}}{\partial \xi^2}\right|_{(\xi_i,\eta_i)} & \left.\frac{\partial \boldsymbol{S}}{\partial \xi}\cdot\frac{\partial \boldsymbol{S}}{\partial \eta} + \boldsymbol{r}\cdot\frac{\partial^2 \boldsymbol{S}}{\partial \xi \partial \eta}\right|_{(\xi_i,\eta_i)} \\ \left.\frac{\partial \boldsymbol{S}}{\partial \xi}\cdot\frac{\partial \boldsymbol{S}}{\partial \eta} + \boldsymbol{r}\cdot\frac{\partial^2 \boldsymbol{S}}{\partial \eta \partial \xi}\right|_{(\xi_i,\eta_i)} & \left.\left|\frac{\partial \boldsymbol{S}}{\partial \eta}\right|^2 + \boldsymbol{r}\cdot\frac{\partial^2 \boldsymbol{S}}{\partial \eta^2}\right|_{(\xi_i,\eta_i)} \end{bmatrix}$$

$$\kappa_i = -\begin{bmatrix} f(\xi_i,\eta_i) \\ g(\xi_i,\eta_i) \end{bmatrix}$$

The eq. (4.5) is solved by solving the linear system of equation for the unknown δ_i, given by the system:

$$J_i \delta_i = \kappa_i \tag{4.6}$$

With

$$\begin{aligned} \xi_{i+1} &= \Delta\xi + \xi_i \\ \eta_{i+1} &= \Delta\eta + \eta_i \end{aligned} \tag{4.7}$$

The whole process of eq. (4.6), (4.7) is the point inversion via Newton-Raphson method. In order to acquire a proper solution, we need to secure some convergence criteria. The two criteria are:

Point Coincidence:

$$|\boldsymbol{S}(\xi_i,\eta_i) - \mathbf{P}| \le \varepsilon_1$$

Zero cosine:

$$\frac{\left| \left.\frac{\partial \boldsymbol{S}}{\partial \xi}\right|_{(\xi_i,\eta_i)} \cdot (\boldsymbol{S}(\xi_i,\eta_i) - \mathbf{P}) \right|}{\left| \left.\frac{\partial \boldsymbol{S}}{\partial \xi}\right|_{(\xi_i,\eta_i)} \right| |\boldsymbol{S}(\xi_i,\eta_i) - \mathbf{P}|} \le \varepsilon_2 \quad \text{and} \quad \frac{\left| \left.\frac{\partial \boldsymbol{S}}{\partial \eta}\right|_{(\xi_i,\eta_i)} \cdot (\boldsymbol{S}(\xi_i,\eta_i) - \mathbf{P}) \right|}{\left| \left.\frac{\partial \boldsymbol{S}}{\partial \eta}\right|_{(\xi_i,\eta_i)} \right| |\boldsymbol{S}(\xi_i,\eta_i) - \mathbf{P}|} \le \varepsilon_2$$

In our case because the code is experimental, we have used only the first convergence criterion. After selecting a point and acquiring its values in parameter space we can choose the direction of doing knot refinement via the algorithm of eq. (4.2). Because of the fact Newton-Raphson is an iterative algorithm it is important the initialization step. For the needs of our code, we select as initialization the $(\xi_i, \eta_i) = (0.5, 0.5)$ after experiments. This initialization works perfectly for square or rectangle primitive shapes but has some convergence problem for other shape as L-shape or circle. This is step that could be investigated by creating an algorithm that searches the proper initialization proportional to the edge that we are near in order to take initialization from a known knot value. In figures below we see the knot refinement via point inversion in I3GA for square case.

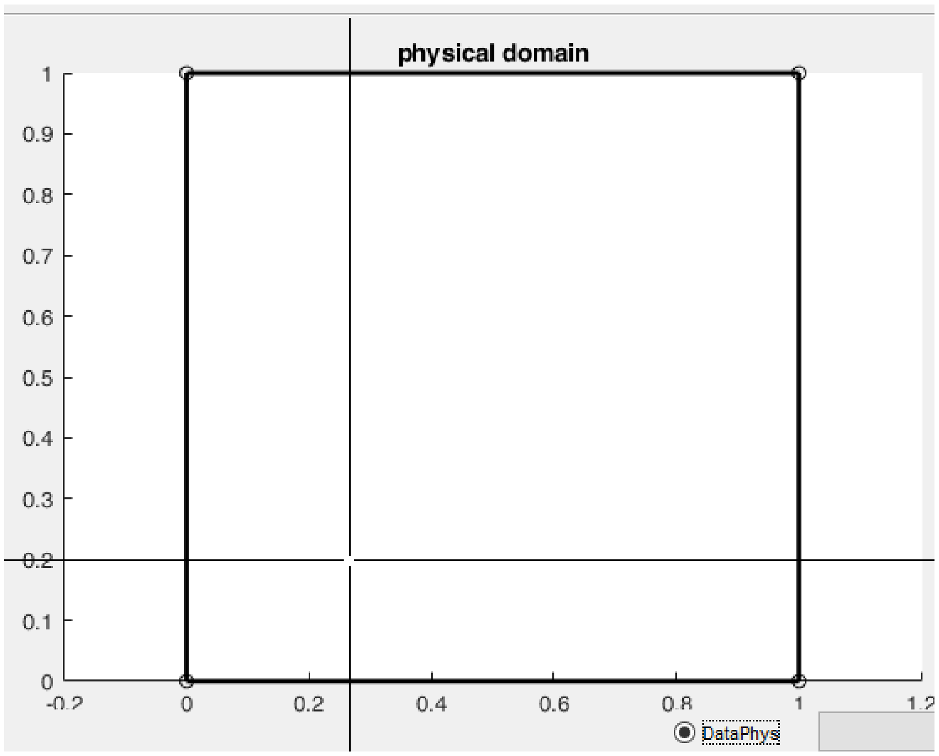


Figure 4.11: Selecting a point in physical domain for point inversion.

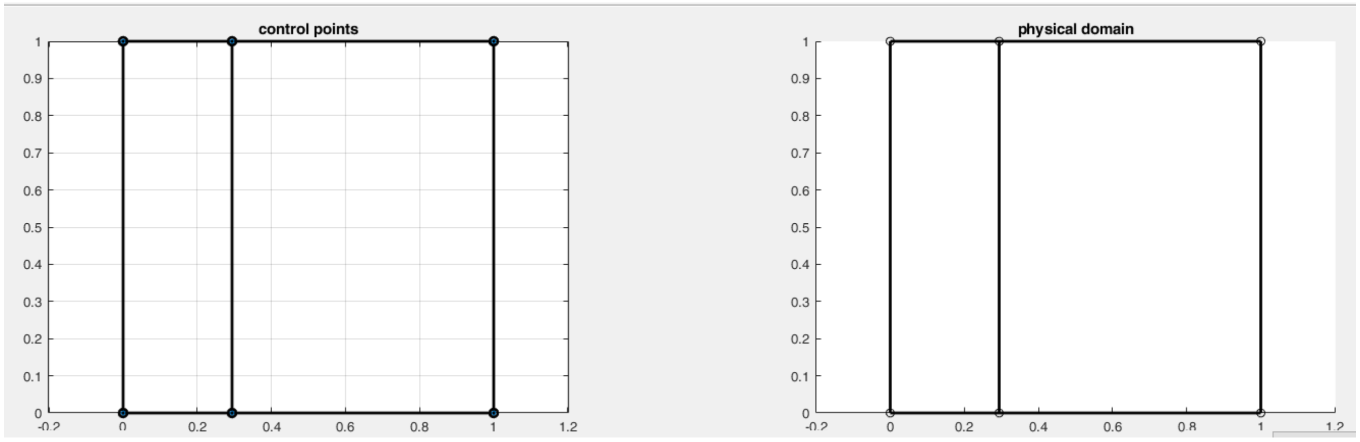


Figure 4.12: After selecting a point we perform knot refinement in ξ-direction.

In this section we saw the knot refinement process and how it was integrated in I3GA. Surely many algorithms for knot refinement. In general, with h-refinement we make the geometry more flexible by giving more freedom to its control points. The finer mesh would give more accurate results in the analysis process, although the process would be more time-consuming [26]. As we show in NURBS this is a global process, because by adding a knot we add lines to whole parameter space and as a result in physical. In T-splines, the knot refinement process is more local as, the inserted knot affects only a specific area of the parameter space.

4.2 Degree Elevation (P-refinement)

Except for knot insertion there is also another technique for refining the geometry. This technique is the degree elevation or p-refinement. In this technique the degree of basis functions is increased by increasing the multiplicity of knots in the knot vector. With the increase in degree of basis functions, control points are also increased. However, this is done without changing the curve or surface parametrically. Degree elevation is not only used as a refinement procedure to curves or surfaces but also is used for the construction of NURBS surfaces from curves [24]. For a B-spline curve the process of degree elevation algorithm presented by Piegl and Tiller [24] is described below [16], [24]:

1. First subdivide the curve into multiple Bézier curves via knot insertion [24]. The inserting knots are the same with existing knots, so we insert knots until the continuity of the curve reduced and each sub element cuts off.
2. Then in Bézier "element" we elevate the degree.
3. Finally, we remove the unnecessary knots from each segment and combine them into one B-spline curve.

In order to use this algorithm in NURBS we must project the control points in higher dimension $\mathbb{R}^{d_s+1}$, transform them into B-spline, do the process and then reproject them back in order to acquire the refined NURBS. For surfaces we perform degree elevation first for η-direction and then for ξ-direction. An example to degree elevation to a curve is shown in fig. 4.13, where the initial knot vector $\Xi = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$ it has been transformed to $\Xi^* = \{0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\}^{\mathrm{T}}$. As we can see the curve has elevated by 1 degree, so 1 control points and one basis function has been added.

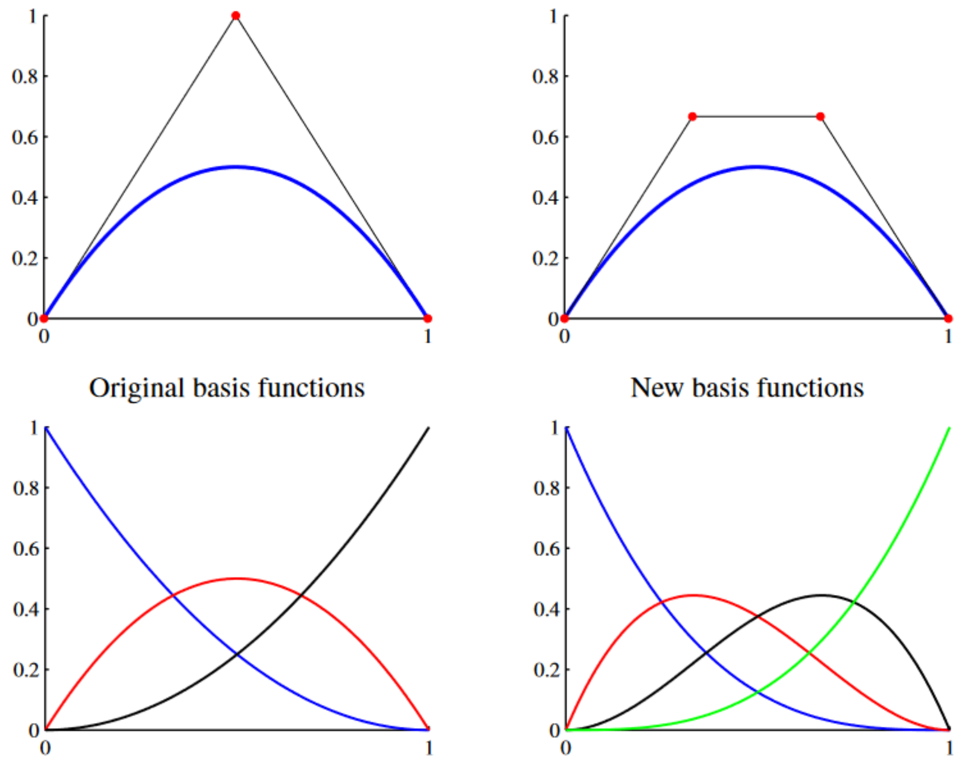


Figure 4.13: Degree elevation to a curve with knot vector $\Xi = \{0\ 0\ 0\ 1\ 1\ 1\}^T$. The red dots are the control points.

The whole process has programmed to I3GA. In I3GA we use as input how many times we want our surfaces to be elevated in each parametric direction. Below you see an example of square where the degree in each direction is 1 and it has been elevated 1 degree in each direction. The initial knot vector is $\Xi = \{0\ 0\ 1\ 1\}^{\mathrm{T}}$ for each parametric direction. After elevating the degree by 1, the surface has not been changed, even though the number of control points has been increased from 4 to 9. Not only that but the number of elements has not yet changed because the knots have just been multiplied without new ones been added. Last but not least, the process of degree elevation in IGA is called order elevation because in IGA we refer more to the order $(degree + 1)$ of basis functions and not so much about degree.

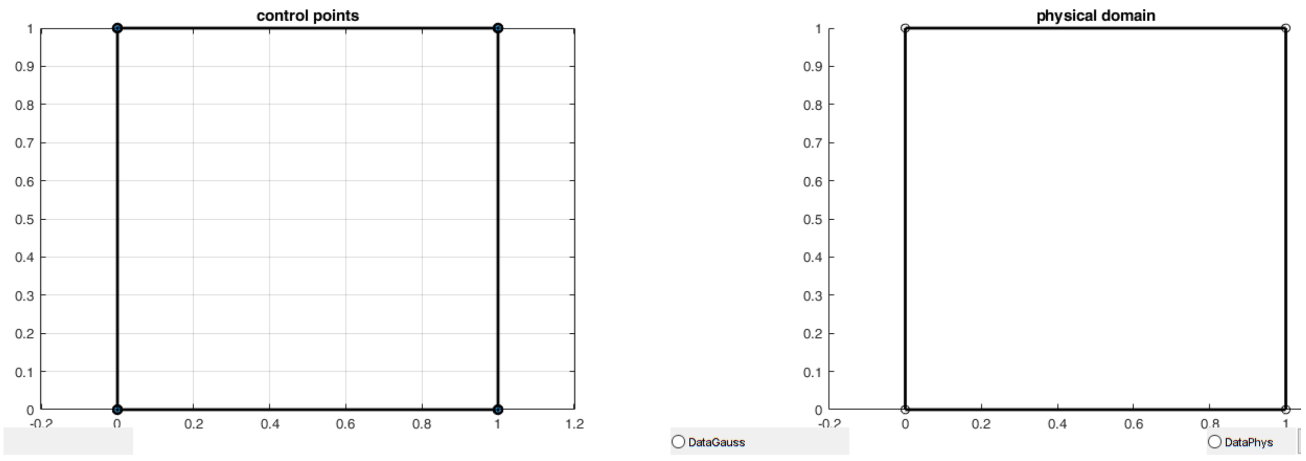


Figure 4.14: Initial square of length 1 m.

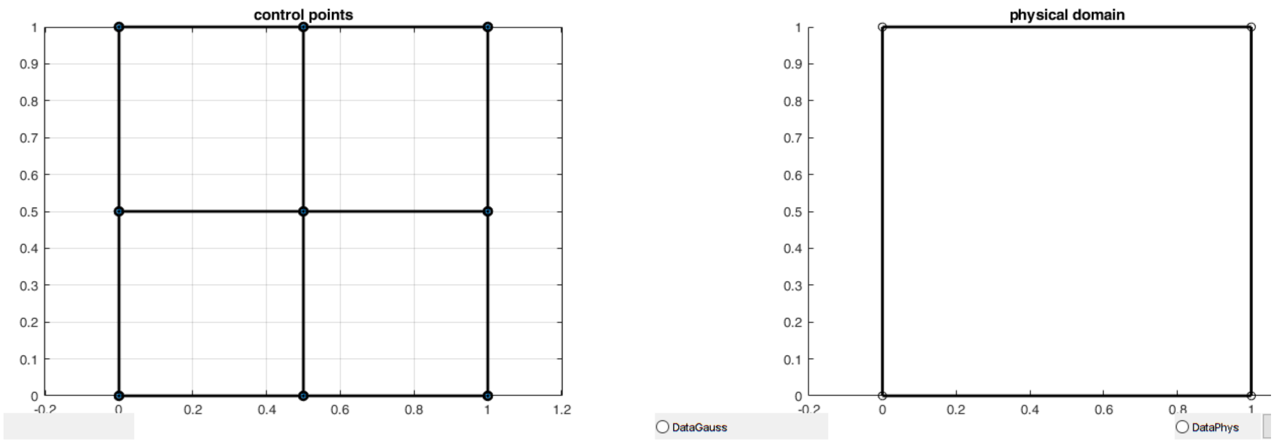


Figure 4.15: The square after degree elevation of one degree.

4.3 K-refinement

Except for the classical refinement strategies Hughes et al. [16], have been developed superior method for high-precision analysis [5]. This is the k-refinement process. In k-refinement we make more flexible the physical space by adding control points without reducing the continuity as it happens in degree (order) elevation where the C^0-continuity is preserved [5], [16]. In k-refinement we perform degree elevation to the curve or surface and then knot refinement. The reversed process does not give satisfying results. Let us give an example.

Let use a square surface with initial knot vectors $\boldsymbol{\Xi} = \mathbf{H} = \{0\ 0\ 1\ 1\}^{\mathrm{T}}$. Then after performing a knot insertion at $\xi = \eta = 0.5$, we elevate the order by one in both directions. As a result, the final knot vectors are $\boldsymbol{\Xi_1} = \mathbf{H_1} = \{0\ 0\ 0\ 0.5\ 0.5\ 1\ 1\ 1\}^{\mathrm{T}}$and the results in physical domain are shown in fig. 4.16. In contrast to the previous process, we perform k-refinement with the same steps. The final knot vectors are $\boldsymbol{\Xi_2} = \mathbf{H_2} = \{0\ 0\ 0\ 0.5\ 1\ 1\ 1\}^{\mathrm{T}}$ and the results are in fig. 4.17.

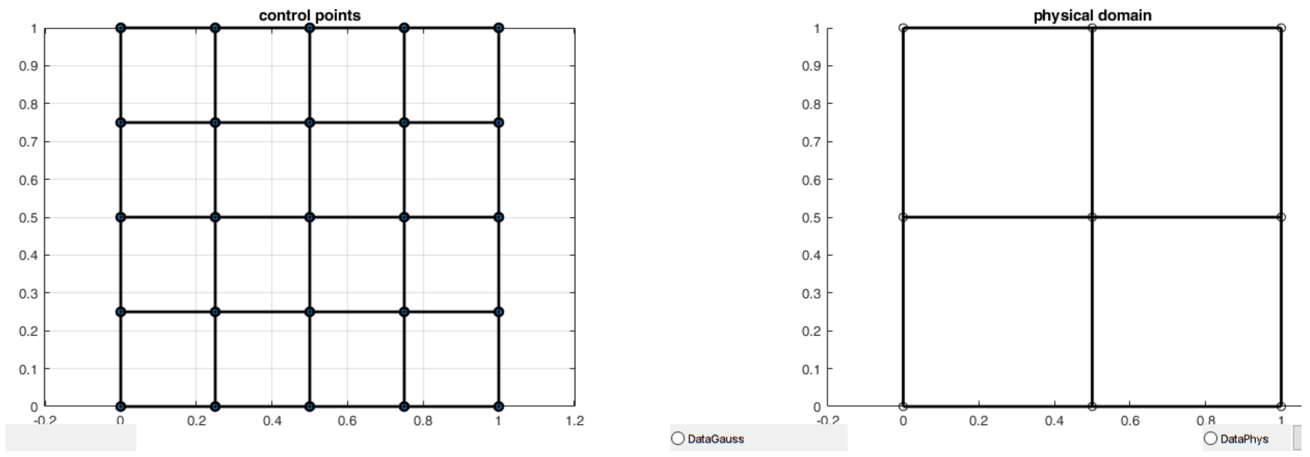


Figure 4.16:Knot refinement at $\xi = \eta = 0.5$ and then degree elevation to a square surface.

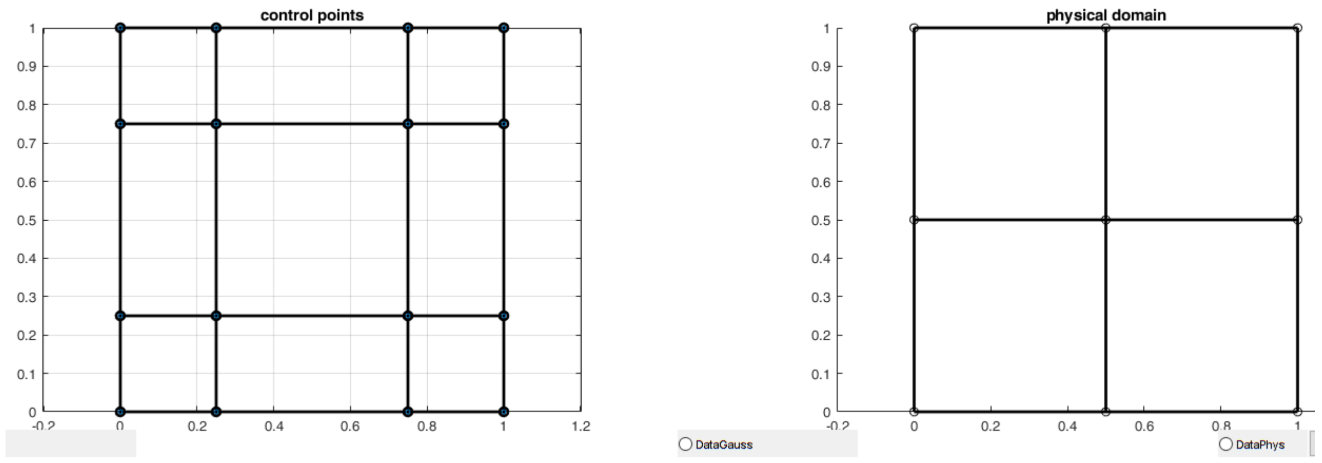


Figure 4.17: k-refinement to a square surface.

As we can see both surfaces geometrically are the same, but parametrically are different. Initially the first procedure produces 25 control points (basis functions) and the second one 16. So, k-refinement produces same geometry with less information. Moreover, the continuity on both surfaces is different. On the boundary the control points are interpolatory to both surfaces. However, in knot $\xi = \eta = 0.5$ the continuity in first case is C^0 and in k-refinement case is C^1. So, we acquire a higher order surface with k-refinements and that results in better precision during analysis process, because when a surface has higher order, the solution becomes closer to analytical. In general k-refinement is only possible if the physical mesh consist of one element as in case above, because if in initial mesh exist constraints on continuity across element boundaries these will propagate in higher orders as well [5]. As a rule of thumb, k-refinement is still effective as long as refinement is performed in order to basis functions have $p - 1$ continuous derivatives across the new element boundaries that will be created [5].

5 Isogeometric Formulation on 2D problems

After explaining the basics of IGA along with basis functions and refinement methods, it is important to explain further how we formulate problems using IGA. From Chapter 2 we saw that a typical code of Isogeometric analysis do not have many differences in relation to a Finite elements code. So, the formulation do not change much, as the only difference is in the use of basis functions. However, in FEA exists a huge knowhow in methods and codes that with the use of IGA, will be disappeared. Consequently, a new method that bridges the geometric representations (basis functions) of IGA and FEA has been developed. This method is called Bézier Extraction [3], [27], [29]. Both “pure” IGA and analysis using Bézier Extraction has been programmed in I3GA. The problems that have been dealt with are 2D thermal problems and plane stress problems. In this chapter we will present the formulation of these with the use of methods mentioned previously.

5.1 Thermal Analysis using IGA

In this section we will describe the IGA formulation in a crucial category in practical engineering, the thermal conduction problem. Specifically, we present the 2D thermal conduction problems on surfaces or solids of constant thickness. FEM and other methods mentioned in Chapter 1, dominated in numerical analysis of thermal problems. However, the problematic mesh generation and C^0-continuity of finite elements leads to the adoption of IGA. The formulation of IGA and FEA has not many differences. The only difference is that instead of using polynomial approximation for the state variables we use the model’s geometry itself for doing this. Consequently, we can apply the same formulation as in case of FEM.

Beginning the formulations, the governing equation for a 2D heat conduction problem is:

$$-\nabla \cdot (\boldsymbol{k}\nabla T) = f \tag{5.1}$$

Where, $\boldsymbol{k}$ is the thermal conductivity parameter, T the temperature and f the source term. The $\boldsymbol{k}$ is a symmetric positive define matrix and its value is prescribed by the surface material. For example, for an AISI 4320 steel the value of matrix is $\boldsymbol{k} = \begin{bmatrix} 44.5 & 0 \\ 0 & 44.5 \end{bmatrix} \frac{\mathrm{W}}{\mathrm{m°C}}$ according to a material database (MATWEB). Except for the strong form of equation, we have also the boundary conditions for the boundary Γ of problem domain Ω. Considering the boundary is decomposed in three parts $(\Gamma_1, \Gamma_2, \Gamma_3)$ proportional to the type of boundary conditions Dirichlet, Neumann and Robin then the equations in boundary conditions are:

Dirichlet Boundary Conditions

$$T = \bar{T} \text{ on } \Gamma_1$$

Neumann Boundary Conditions

$$-\boldsymbol{k}\frac{\partial T}{\partial \boldsymbol{n}} = q_n \text{ on } \Gamma_2$$

Robin Boundary Conditions

$$-\boldsymbol{k}\frac{\partial T}{\partial \boldsymbol{n}} = h(T - T_w) \text{ on } \Gamma_3$$

Where, $\bar{T}$ is the specified temperature in the boundary, q_n is the specified heat flux density on the boundary, T_w is the ambient temperature, h the convective heat transfer and $\boldsymbol{n}$ the normal on the boundary. Neumann and Robin boundary conditions are the *weak* boundary conditions as the impose on the weak form of the problem, instead of Dirichlet boundary conditions where are fixed and imposed on during solution phase. Now in order to continue the formulation we will use an approximation and the Galerkin method. Instead of FEM approximation of Chapter 2 the IGA approximation of state variable T use the geometry model (NURBS basis function) in order to be defined. Specifically, the IGA approximation of temperature field is:

$$T(x,y) \cong \sum_{i=1}^{n} R_i^{pq}(\xi,\eta) T_i \tag{5.2}$$

Where $R_i^{pq}(\xi,\eta)$ is the NURBS basis function for the surface, T_i is the temperature in control point i and n is the number of control points. After using the Galerkin method and substitute (5.1), Neumann and Robin boundary conditions and (5.2) to eq. (2.6) we have an equation of the form:

$$\boldsymbol{KT} = \boldsymbol{F} \tag{5.3}$$

Where, $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ is the global stiffness matrix, $\boldsymbol{T} \in \mathbb{R}^{n}$ is the vector of temperatures in every control point and $\boldsymbol{F} \in \mathbb{R}^{n}$ is the global vector of sources. The term global as in case of FEM means that we calculate the stiffness in every element and then we sum all the matrices in order to take the result. The elements in every $\boldsymbol{K}$ and $\boldsymbol{F}$ according to Bubnov-Galerkin method are the following:

$$K_{ij}^{e} = \int_{\Omega_e} (\boldsymbol{B}_i)^{\mathrm{T}} \boldsymbol{k} (\boldsymbol{B}_j)\, \mathrm{d}\Omega + \int_{\Gamma_3} R_i^T h R_j \mathrm{d}\Gamma \tag{5.4}$$

and

$$F_i^e = \int_{\Omega_e} q_v R_i \, \mathrm{d}\Omega - \int_{\Gamma_2} q_n R_i \mathrm{d}\Gamma + \int_{\Gamma_3} h T_w R_i \mathrm{d}\Gamma \tag{5.5}$$

Where, $\boldsymbol{B}_i = \begin{bmatrix} \frac{\partial R_i}{\partial x} & \frac{\partial R_i}{\partial y} \end{bmatrix}^{\mathrm{T}}$, R_i is the NURBS basis function in every control point of the element Ω_e. The inhomogeneous Dirichlet boundary conditions can be imposed to eq. (5.3) in source term and stiffness matrix suitably.

Gauss Quadrature

In order to compute the terms of stiffness matrix and source term vector the most notable method for executing this is the *Gauss-Legendre* Quadrature. The Gauss Quadrature is based on computing the integral in certain points (*gauss points*) in each knot span and then summing up all the results to obtain the integral. However, in order to apply this method to eq. (5.4) and eq. (5.5) we need first to change a little the above equations in more suitable form. In I3GA the equations above do not have this form because not all boundary conditions are applied. The only boundary conditions that are true, the Dirichlet and a specific form of Neumann boundary conditions. So, all the others may be omitted.

$$(5.4) \Rightarrow K_{ij}^{e} = \int_{\Omega_e} (\boldsymbol{B}_i)^{\mathrm{T}} \boldsymbol{k} (\boldsymbol{B}_j)\, \mathrm{d}\Omega$$

$$(5.5) \Rightarrow F_i^e = \int_{\Omega_e} q_v R_i \, \mathrm{d}\Omega - \int_{\Gamma_2} q_n R_i \mathrm{d}\Gamma$$

The Neumann boundary conditions are applied for the special case $q_n = 0$ so we can also omit them. In order to compute the integral for each element in physical space we need to transfer them in parameter space in order to compute them in each IGA element (knot span). In order to do this, we use the Jacobian $\mathbf{J} \in \mathbb{R}^{2\times 2}$ that connects the physical with parameter space as:

$$\mathrm{J}_{ij} = \begin{bmatrix} \frac{\partial R_1(\xi,\eta)}{\partial i} & \frac{\partial R_2(\xi,\eta)}{\partial i} & \cdots & \frac{\partial R_m(\xi,\eta)}{\partial i} \end{bmatrix} \begin{bmatrix} \mathrm{coord_{j,1}} \\ \mathrm{coord_{j,2}} \\ \vdots \\ \mathrm{coord_{j,m}} \end{bmatrix} \tag{5.6}$$

Where, $\mathrm{coord}_{\mathrm{j,k}}$ is the j coordinate (x or y) of control point k that supports the element and $\frac{\partial R_g(\xi,\eta)}{\partial i}$ is the derivative of NURBS basis about $i = \{\xi \text{ or } \eta\}$. Moreover, in integrals of eq. (5.4) and (5.5) we need the determinant of Jacobian $\det(\mathbf{J})$.

Before proceeding to the integral calculation, we need to explain the Gauss-Legendre Quadrature method. The Gauss method transforms an integral over $[-1,1] \times [-1,1]$ to a sum, meaning $\int_{-1}^{1}\int_{-1}^{1} f(x,y)\,\mathrm{d}x\mathrm{d}y \approx \sum_{i=1}^{n}\sum_{j=1}^{m} w_j\, w_i f(x_i, y_j)$, where x_i, y_i are the gauss point (roots of Legendre polynomial) in parent element $[-1,1] \times [-1,1]$, w_i, w_j are the corresponding weights. In our case after we have gone from parameter to physical space using Jacobian, then we need to go from parameter space to parent element with a suitable transformation. Let suppose that parent element spans from $[-1,1] \times [-1,1]$ and proportional to the degree of basis function a set of coordinates ξ_R, η_R corresponds to that reference element as root of Legendre polynomial. The coordinates in parameter space in the element $[\xi_i, \xi_{i+1}) \times [\eta_i, \eta_{i+1})$ and the respective weights will be [26]:

$$\xi = \frac{(\xi_{i+1} - \xi_i) \cdot \xi_R + (\xi_{i+1} + \xi_i)}{2}$$

$$w_\xi^{gp} = \frac{\xi_{i+1} - \xi_i}{2} \cdot w_\xi^R$$

And

$$\eta = \frac{(\eta_{i+1} - \eta_i) \cdot \eta_R + (\eta_{i+1} + \eta_i)}{2}$$

$$w_\eta^{gp} = \frac{\eta_{i+1} - \eta_i}{2} \cdot w_\eta^R$$

For a square with knot vectors in parameter space $\boldsymbol{\Xi} = \mathbf{H} = \{0\ 0\ 0.5\ 1\ 1\}^{\mathrm{T}}$ the parametric coordinates for gauss points per element are shown in fig. 5.1. As we can observe in each element 4 gauss points correspond because the order of basis functions is $p + 1 = 2$. So, after defining gauss integral rule the elements of stiffness matrix and source vector can be written as:

$$(5.4) \Rightarrow K_{ij}^{e} = \sum_{i=1}^{n_{GP}}\sum_{j=1}^{m_{GP}} \left(\boldsymbol{B}(\xi_i, \eta_j)\right)^{\mathrm{T}} \boldsymbol{k} \left(\boldsymbol{B}(\xi_i, \eta_j)\right) \det(\mathbf{J})\, \mathrm{w}_i^{\mathrm{GP}} \mathrm{w}_j^{\mathrm{GP}} \tag{5.7}$$

$$(5.5) \Rightarrow F_{i}^{e} = \sum_{i=1}^{n_{GP}}\sum_{j=1}^{m_{GP}} q_v R_i(\xi_i, \eta_j) \det(\mathbf{J})\, \mathrm{w}_i^{\mathrm{GP}} \mathrm{w}_j^{\mathrm{GP}} \tag{5.8}$$

Where, ξ_i, η_j are the parametric gauss coordinates, $\mathrm{w}_i^{\mathrm{GP}} \mathrm{w}_j^{\mathrm{GP}}$ are the respective weights, n_{GP}, m_{GP} is the number of gauss points in each direction. In I3GA, we can calculate gauss points for basis function degrees up to 7. In general, the algorithm of computation the elements of these matrices in IGA is shown on fig. 2.4.

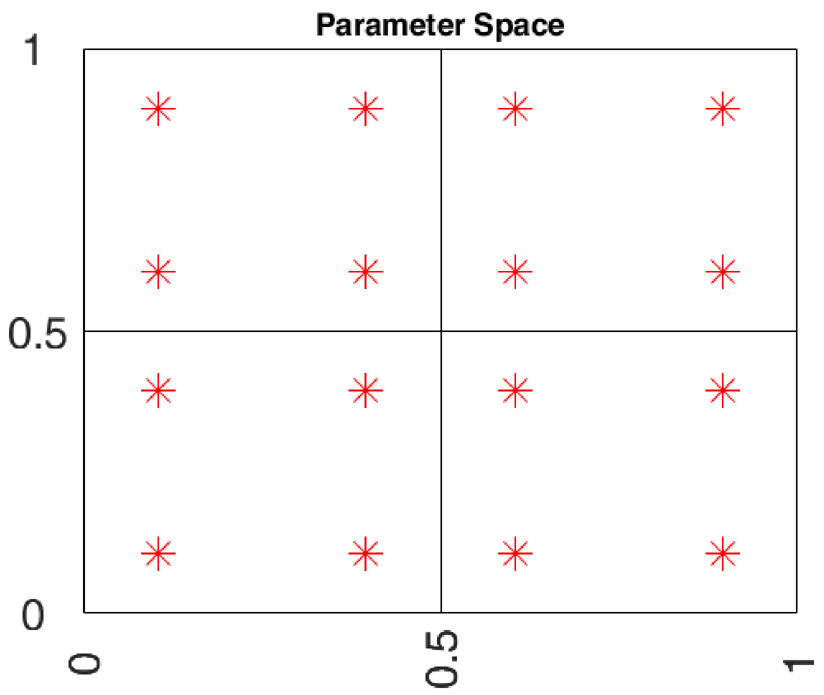


Figure 5.1: Gauss points in parameter space for a square with knot vectors $\Xi = \mathbf{H} = \{0\ 0\ 0.5\ 1\ 1\}^T$.

Dirichlet Boundary Conditions

After defining the way of computing the elements of stiffness matrix and source vector, then we need to impose the Dirichlet boundary conditions on the system of equation (5.3). The algorithm of imposing a fixed boundary condition on a degree of freedom (DoF) is:

1. Subtract from all elements of F, the value of boundary condition times the corresponding column values of K matrix in column of the DoF.
2. Zero all columns and rows of the DoF.
3. Set $K_{DoF,DoF} = 1$
4. Set F_{DoF} equal with the value of boundary conditions

This algorithm is applied in every problem with Dirichlet boundary conditions. In I3GA this algorithm is used, however it is important to show the algorithm for selecting the DoFs. This algorithm is shown in the flowchart below. The first step of the algorithm is to set the type of boundary conditions. The boundary conditions are applied to the whole side of the geometrical object. As it shown in fig. 5.3 the contour of geometrical entity is decomposed to four edges: "**d**own", "**up**", "**l**eft" and "**r**ight". Each edge is prescribed by an index number for the DoF. Each DoF corresponds to a control point because it is a thermal problem and we have only one variable. So, if we begin the numbering from down edge the DoFs are numbered as:

"Down" edge: $\mathrm{DoF} = 1{:}\,\mathrm{n}_\xi$

"Left" edge: $\mathrm{DoF} = 1{:}\,1 + \mathrm{n}_\xi(\mathrm{n}_\eta - 1)$

"Up" edge: $\mathrm{DoF} = 1 + \mathrm{n}_\xi(\mathrm{n}_\eta - 1){:}\,\mathrm{n}_\xi + \mathrm{n}_\xi(\mathrm{n}_\eta - 1)$

"Right" edge: $\mathrm{DoF} = \mathrm{n}_\xi{:}\,\mathrm{n}_\xi + \mathrm{n}_\xi(\mathrm{n}_\eta - 1)$

According to the numbering above the DoFs for each side are selected and if we have a Dirichlet condition we set it to these. Each side independently of the number of its real edges will always be "translated" in grid of 4 sides as in fig. 5.3. However, in the joints of each side (the 4 red dots in fig 5.3) two boundary conditions will contribute one from each edge. The prevailing boundary condition will be the one with the biggest absolute value.

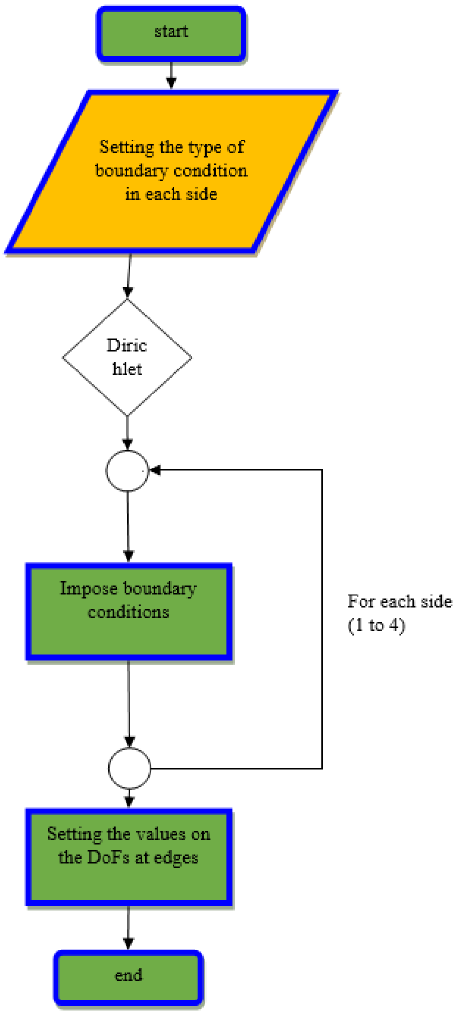

start
Setting the type of boundary condition in each side
Diric hlet
Impose boundary conditions
For each side (1 to 4)
Setting the values on the DoFs at edges
end

Figure 5.2: Flowchart of setting Dirichlet boundary conditions

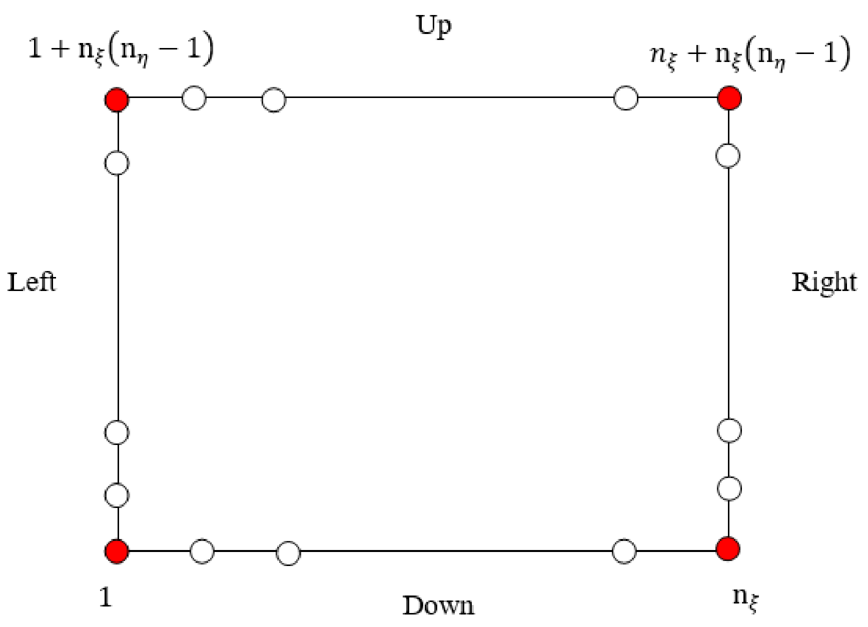


Figure 5.3: The grid for setting the boundary conditions.

5.2 Thermal Analysis using Bézier Extraction

Except for pure IGA codes, many researchers want to use existing finite element codes, combining them with IGA formulation. They do it by decomposing the geometry into C^0 Bézier elements by using the Bézier extraction operator [29]. The Bézier extraction operator is the map between piecewise Bernstein polynomial basis functions and B-spline basis functions [29]. In order to define the way for computing stiffness matrix and source vector via Bézier elements we need first to show how a NURBS curve or surface is represented by Bézier elements and then how Bézier extraction operator is computed.

In order to decompose a NURBS geometry into Bézier elements, we repeat all the interior knots of the knot vector until the multiplicity equals $p+1$. We could also repeat the knots until the multiplicity equals p for lower computational cost but then adjacent Bézier elements will share control points. The knots are repeated by a knot refinement process. After the decomposition we shall compute the Bézier extraction operator.

Let suppose that the B-spline curve (in $\mathbb{R}^{d_s+1}$) of a NURBS in $\mathbb{R}^{d_s}$ is defined by a knot vector $\boldsymbol{\Xi} = \{\xi_1, \xi_2, \dots, \xi_{n+p+1}\}^T$ and a set of control points $\mathbf{P} = \{\mathbf{P}_A\}_{A=1}^n$. If we insert the knots $\{\bar{\xi}_1, \bar{\xi}_2, \dots, \bar{\xi}_m\}$ via knot refinement such that we decompose the B-spline into Bézier elements, then for each new knot $\bar{\xi}_j, j = 1,2,\dots,m$ we define a_A^j, $A = 1,2,\dots,n+j$ to be the A^{th} alpha as prescribed in knot insertion defined in eq. (4.1). Then, by defining $\mathbf{C}^j \in \mathbb{R}^{(n+j-1)\times(n+j)}$ as:

$$\mathbf{C}^j = \begin{bmatrix} a_1 & 1-a_2 & 0 & \cdots & & & 0 \\ 0 & a_2 & 1-a_3 & 0 & \cdots & & 0 \\ 0 & 0 & a_3 & 1-a_4 & 0 & \cdots & 0 \\ \vdots & & & & & & \\ 0 & \cdots & & & 0 & a_{n+j-1} & 1-a_{n+j} \end{bmatrix} \tag{5.9}$$

We can rewrite eq. (4.1) as:

$$\overline{\mathbf{P}}^{j+1} = (C^j)^{\mathrm{T}}\overline{\mathbf{P}}^j$$

For $j = m$ the final set of control points $\overline{\mathbf{P}}^{m+1}$ defines the Bézier elements [29]. By renaming as $\overline{\mathbf{P}}^{m+1} = \mathbf{P}^{\mathrm{b}}$ and for every knot we define $\boldsymbol{C}^{\mathrm{T}} = (\boldsymbol{C}^m)^{\mathrm{T}}(\boldsymbol{C}^{m-1})^{\mathrm{T}} \ldots (\boldsymbol{C}^1)^{\mathrm{T}}$ then we have:

$$\mathbf{P}^{\mathrm{b}} = \boldsymbol{C}^{\mathrm{T}}\mathbf{P} \tag{5.10}$$

Where $\mathbf{P}$ is the control points matrix that each row contains the control point coordinates. So, $\mathbf{P} \in \mathbb{R}^{n\times \mathrm{d_s}}$ (n number of control points), $\boldsymbol{C} \in \mathbb{R}^{n\times(n+m)}$ and $\mathbf{P^b} \in \mathbb{R}^{(n+m)\times \mathrm{d_s}}$. After the Bézier decomposition which it has been done using knot refinement, the curve has not change parametrically. As a result, from final knot vector a set of Bernstein basis functions has been defined as $\mathbf{B}(\xi) = \{\mathbf{B}_A(\xi)\}_{A=1}^{n+m}$ and the final curve $\boldsymbol{C}(\xi) = (\mathbf{P}\,)^{\mathrm{T}}\mathbf{N}(\xi)$ can be written as:

$$\boldsymbol{C}(\xi) = \left(\mathbf{P}^{\mathrm{b}}\right)^{\mathrm{T}}\mathbf{B}(\xi) = (\boldsymbol{C}^{\mathrm{T}}\mathbf{P}\,)^{\mathrm{T}}\mathbf{B}(\xi) = (\mathbf{P}\,)^{\mathrm{T}}\boldsymbol{C}\mathbf{B}(\xi) \equiv (\mathbf{P}\,)^{\mathrm{T}}\mathbf{N}(\xi)$$

So, considering that $\mathbf{P}$ is arbitrary a new basis function has been developed along with a new linear operator $\boldsymbol{C}$ which is called *Bézier Extraction* operator. From the relationship above we have this equation:

$$\mathbf{N}(\xi) = \boldsymbol{C}\mathbf{B}(\xi) \tag{5.11}$$

As we show this operator is computed only by knowing the knot vector. As a result, the extraction operator it can be applied to NURBS directly because it depends only to parameterization of the curve. According to Borden by using eq. (3.10) and (5.10) a NURBS curve can be written in terms of Bernstein basis as:

$$C(\xi) = (\mathbf{P}\,)^{\mathrm{T}}\boldsymbol{R}(\xi) = \frac{1}{W(\xi)}(\mathbf{P}\,)^{\mathrm{T}}\mathbf{W}\mathbf{N}(\xi) = \frac{1}{W(\xi)}(\mathbf{P}\,)^{\mathrm{T}}\mathbf{W}\boldsymbol{C}\boldsymbol{B}(\xi) = \frac{1}{W(\xi)}(\boldsymbol{C}^{\mathrm{T}}\mathbf{W}\mathbf{P}\,)^{\mathrm{T}}\mathbf{B}(\xi) \tag{5.12}$$

Considering the weight function $W(\xi)$ and NURBS weights $\mathbf{w} = \{w_A\}_{\boldsymbol{A=1}}^{\boldsymbol{n}}$, $W(\xi)$ using again eq. (5.10) can be written as:

$$W(\xi) = \mathbf{w^T N}(\boldsymbol{\xi}) = \cdots = \left(\mathbf{w^b}\right)^{\mathbf{T}}\mathbf{B}(\boldsymbol{\xi}) = W^b(\xi) \tag{5.13}$$

Where, $\mathbf{w^b} = \boldsymbol{C}^{\mathrm{T}}\mathbf{w}$ are the Bézier basis function associated weights. If we define the diagonal matrix $\mathbf{W}^{\mathrm{b}}$ as:

$$\mathbf{W^b} = \begin{bmatrix} w_1^b & & & \\ & w_2^b & & \\ & & \ddots & \\ & & & w_{n+m}^b \end{bmatrix}$$

The Bézier control points are now computed as:

$$\mathbf{P}^{\mathrm{b}} = \left(\mathbf{W^b}\right)^{-1}\boldsymbol{C}^{\mathbf{T}}\mathbf{WP} \tag{5.14}$$

If we multiply eq. (5.14) by $\mathbf{W^b}$ it is derived that:

$$\mathbf{W^b}\mathbf{P}^{\mathrm{b}} = \boldsymbol{C}^{\mathbf{T}}\mathbf{WP} \tag{5.15}$$

Finally, the NURBS curve can have its Bézier representation form by using eq. (5.12), (5.13), (5.15) as:

$$C(\xi) = \frac{1}{W^b(\xi)}\left(\mathbf{W^b}\mathbf{P}^{\mathrm{b}}\right)^{\mathrm{T}}\mathbf{B}(\boldsymbol{\xi}) \tag{5.16}$$

Each Bézier element hat its own Bézier extraction operator [29]. If we have a NURBS surface then we have two knot vectors. So, for each Bézier element we have to compute two extraction operators and then take their tensor product in order to find the bivariate extractor operator for the element. In other words:

$$\boldsymbol{C^e} = \boldsymbol{C_\eta^j} \otimes \boldsymbol{C_\xi^k} \tag{5.17}$$

Where the tensor product for two matrices is defined as:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} A_{11}\boldsymbol{B} & A_{12}\boldsymbol{B} & \cdots \\ A_{21}\boldsymbol{B} & A_{22}\boldsymbol{B} & \\ \vdots & & \ddots \end{bmatrix}$$

So, having for each Bézier element the bivariate extraction operator along with the Bézier geometrical and parametric surface representation of NURBS (like eq. (5.16)) then we can import the latter to a FEM code. The algorithm an IGA code using Bézier extraction for a thermal problem is shown in figure below. The code has little difference with a classic FEM or IGA algorithm. The only differences are that we need to compute the extraction operators for each parametric direction in order to compute the needed basis functions and the Jacobian matrix for the formation of stiffness matrix and source vector. In other words, the equations (5.7), (5.8) are still valid we the difference that the NURBS basis function will be expressed by their Bézier counterpart.

$$(5.7) \Rightarrow K_{ij}^{e} = \sum_{i=1}^{n_{GP}} \sum_{j=1}^{m_{GP}} \left(\boldsymbol{B^{b}}(\xi_i, \eta_j)\right)^{\mathrm{T}} \boldsymbol{k} \left(\boldsymbol{B^{b}}(\xi_i, \eta_j)\right) \det(\mathbf{J^{b}})\, \mathrm{w}_i^{\mathrm{GP}} \mathrm{w}_j^{\mathrm{GP}} \tag{5.18}$$

$$(5.8) \Rightarrow F_{i}^{e} = \sum_{i=1}^{n_{GP}} \sum_{j=1}^{m_{GP}} q_v R_i^{b}(\xi_i, \eta_j) \det(\mathbf{J^{b}})\, \mathrm{w}_i^{\mathrm{GP}} \mathrm{w}_j^{\mathrm{GP}} \tag{5.19}$$

Where, $\boldsymbol{B^{b}} = \begin{bmatrix} \frac{\partial R_i^{b}}{\partial x} & \frac{\partial R_i^{b}}{\partial y} \end{bmatrix}^{\mathrm{T}}$, R_i^{b} is i^{th} NURBS basis function which is written in terms of Bézier elements and $\mathbf{J^{b}}$ being the Jacobian matrix which depends on R_i^{b}. The superscript "e" corresponds to the e^{th} element. But the elements here, are the NURBS elements which will decomposed into Bézier. The NURBS elements are in one-to-one correspondence with Bézier [29]. Thus, the same control points (basis functions) that support a NURBS element, also support the Bézier element. Consequently, we can create a mapping between the local numbering of NURBS control points and a global control point numbering [29]. That map is called IEN array and practically the superscript "e" is each row of the IEN array. That term exist also in standard FEM. The algorithm for constructing the IEN array in I3GA is:

Algorithm for IEN array:

1. Compute the number of NURBS elements in each direction.
2. Construct a matrix that each row has the limits of the intervals $[\xi_i, \xi_{i+1}]$, $[\eta_i, \eta_{i+1}]$ for each NURBS element.
3. Construct two matrices one for each parametric direction. Each matrix represents the intervals for the support of basis functions, in other words each row has the ending knots of the interval $[\xi_i . \xi_{i+p+1})$ (or $[\eta_j . \eta_{j+q+1})$ respectively).
4. Construct a matrix belonging to $\mathbb{R}^{n_\xi \times n_\eta}$ that has as elements the control points.
5. Combine all the above to complete IEN array. A control point belongs to a NURBS element if the intervals that is supported is subset of the intervals $[\xi_i, \xi_{i+1}]$ and $[\eta_i, \eta_{i+1}]$.

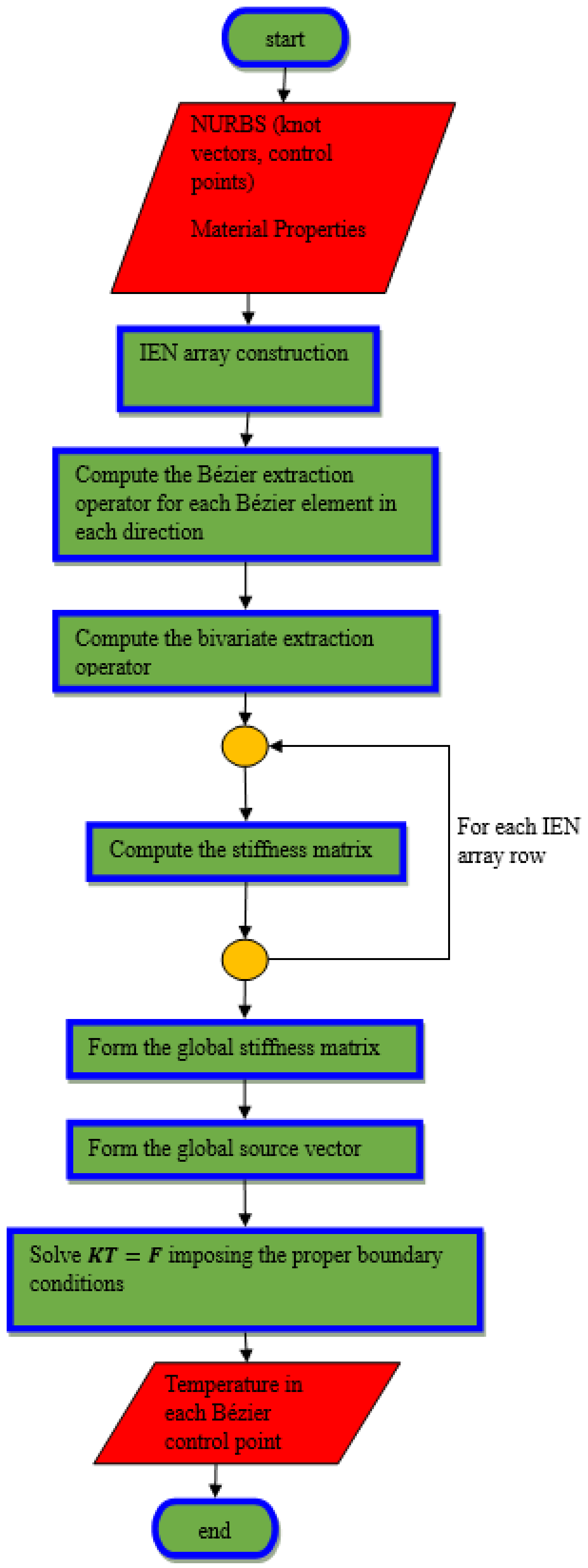

start
NURBS (knot vectors, control points)
Material Properties
IEN array construction
Compute the Bézier extraction operator for each Bézier element in each direction
Compute the bivariate extraction operator
For each IEN array row
Compute the stiffness matrix
Form the global stiffness matrix
Form the global source vector
Solve $\boldsymbol{KT} = \boldsymbol{F}$ imposing the proper boundary conditions
Temperature in each Bézier control point
end

Figure 5.4: Flowchart of IGA code using Bézier extraction.

For example, in table below you see the IEN for the geometry with the parameter space of fig. 5.1. The NURBS geometry can be seen on fig. 5.5. The Bézier control mesh along with control points is shown on fig. 5.6 and the physical mesh in fig. 5.6.

IEN array			
1	2	4	5
2	3	5	6
4	5	7	8
5	6	8	9

Table 5-1: IEN array for a square with knot vectors $\boldsymbol{\Xi} = \mathbf{H} = \{0\ 0\ 0.5\ 1\ 1\}^T$.

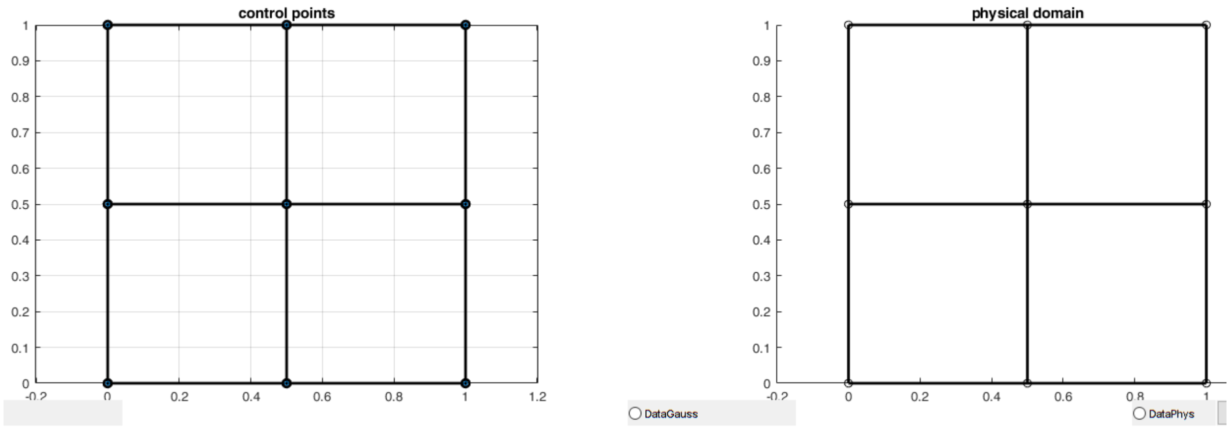


Figure 5.5: NURBS geometry with knot vectors $\boldsymbol{\Xi} = \boldsymbol{H} = \{0\ 0\ 0.5\ 1\ 1\}^T$.

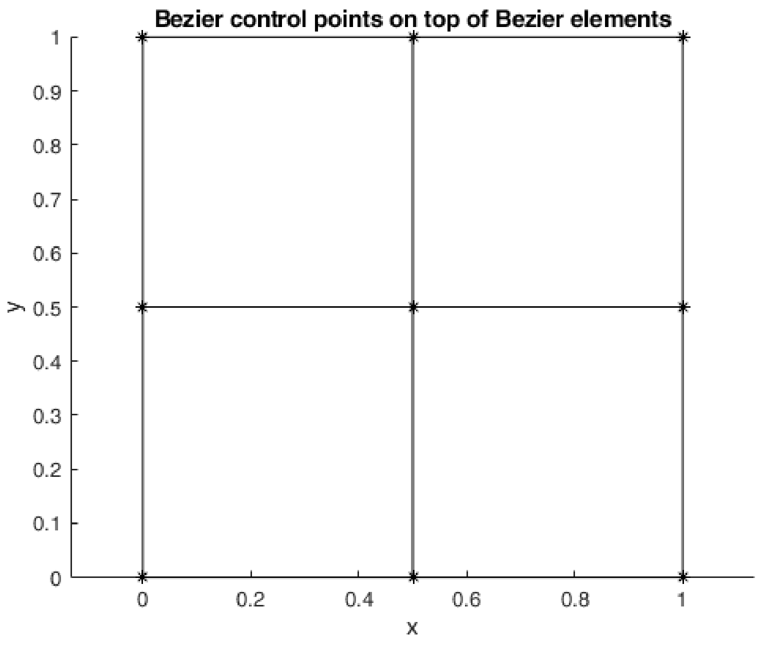


Figure 5.6: Bézier control mesh for fig. 5.5.

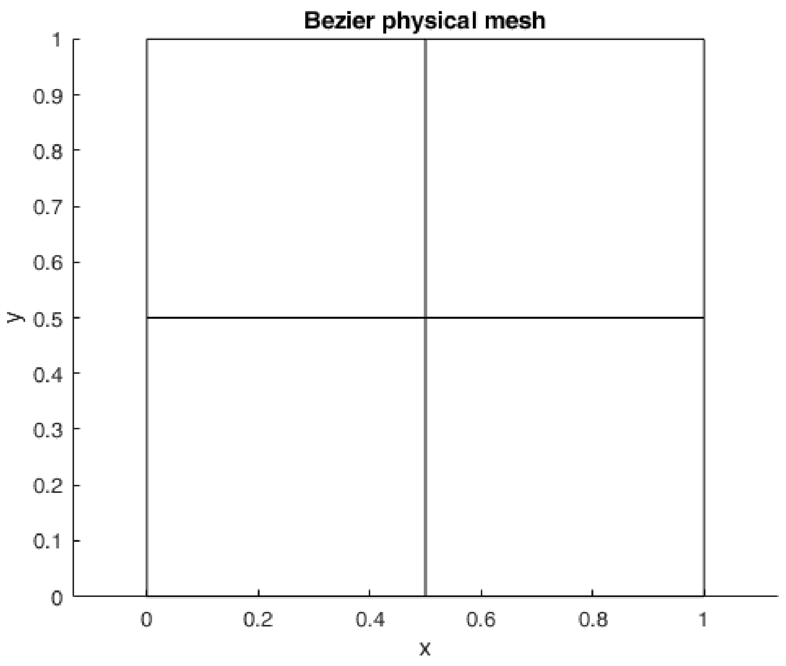


Figure 5.7: Bézier physical mesh for fig. 5.5.

Because of the geometry is first degree, the Bézier physical elements are equivalent to the NURBS physical elements. Moreover, the positions of control points are coincided in the two cases. That is not the rule. If the geometry would be more complex and with higher order, then the Bézier physical and control mesh would be different from their NURBS counterpart. This is the only case that the extraction operator for these elements is the identity matrix.

Finally, the material properties in I3GA are taken from a custom-made database that has been developed in MS Excel (see. Manual in Appendix C). This database gives thermal and elasticity material properties. For this thesis two materials were selected, and many more can be imported from the user.

5.3 Plane stress analysis using Bézier Extraction

In I3GA except for thermal problem also the plane stress problem has been solved using Bézier extraction. The plane stress problem is defined in Appendix A.2 where is formulated using isoparametric Lagrange shape functions. Both formulations and algorithms (Bézier extraction and isoparametric FEM) are equivalent and only slight differences do exist. The only differences are:

- Each control point has two DoFs, the displacements in the two directions of $\mathbb{R}^2$.
- The eq. (5.18), (5.19) about stiffness matrix and source vector are formulated with different parameters. More specifically:

$$(5.18) \Rightarrow K_{ij}^{e} = \sum_{i=1}^{n_{GP}} \sum_{j=1}^{m_{GP}} \left(\boldsymbol{B}^{\boldsymbol{b}}(\xi_i, \eta_j)\right)^{\mathrm{T}} \mathbf{D} \left(\boldsymbol{B}^{\boldsymbol{b}}(\xi_i, \eta_j)\right) \mathrm{t}\det\left(\mathbf{J}^{\mathbf{b}}\right) \mathrm{w}_i^{\mathrm{GP}} \mathrm{w}_j^{\mathrm{GP}} \tag{5.20}$$

$$(5.19) \Rightarrow F_i^{e} = \sum_{i=1}^{n_{GP}} \sum_{j=1}^{m_{GP}} q_v R_i^b(\xi_i, \eta_j) \det\left(\mathbf{J}^{\mathbf{b}}\right) \mathrm{w}_i^{\mathrm{GP}} \mathrm{w}_j^{\mathrm{GP}} \tag{5.21}$$

Where, t is the thickness and $\mathbf{D}$ is the elasticity matrix where with the use of Poisson ratio ν and Young Modulus E is defined in plane stress problems as

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

And strain-displacement matrix as $\boldsymbol{B^b}$:

$$\boldsymbol{B^b} = \begin{bmatrix} \frac{\partial R_1^b}{\partial x} & \cdots & \cdots & \frac{\partial R_m^b}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial R_1^b}{\partial y} & \cdots & \cdots & \frac{\partial R_m^b}{\partial y} \\ \frac{\partial R_1^b}{\partial y} & \cdots & \cdots & \frac{\partial R_m^b}{\partial y} & \frac{\partial R_1^b}{\partial x} & \cdots & \cdots & \frac{\partial R_m^b}{\partial x} \end{bmatrix}$$

Where, m is the final control point in every row of the IEN array.

As we can see only the computation of stiffness matrix changes, but slightly enough that does not change the algorithm of fig. 5.4. As an example, we present the square of fig. 5.5 in which we impose Dirichlet boundary conditions in right side (a shear uniform distribution with load $q = \frac{1}{16}\,\mathrm{N/m}$) and fixed boundary conditions on the left side (zero displacement in both directions). In I3GA we have the capability of inserting two types of loads; tensile and shear but in the future, we will enrich the code with other options. We can see the two different types of boundary conditions in fig. 5.8. The used material is Steel AISI 4340 and the thickness of the plate is 1 mm. Except the displacements (unit m) we also plot the stresses that are computed according to Appendix A.2. and the Von Mises equivalent stress which is defined as:

$$\sigma_e = \sqrt{\sigma_x^2 - \sigma_x\sigma_y + \sigma_y^2 + 3\tau_{xy}}$$

The units of stresses are in [GPa]. As we see in the figures below on the right side the vertical displacement is bigger and on the left side is zero in both horizontal and vertical directions. The equivalent stress is bigger near the left edge because the support is on that edge. The number of Gauss points is 25 because we have one element (Bézier element) and 25 gauss points correspond, because the degree of basis function equals to 4.

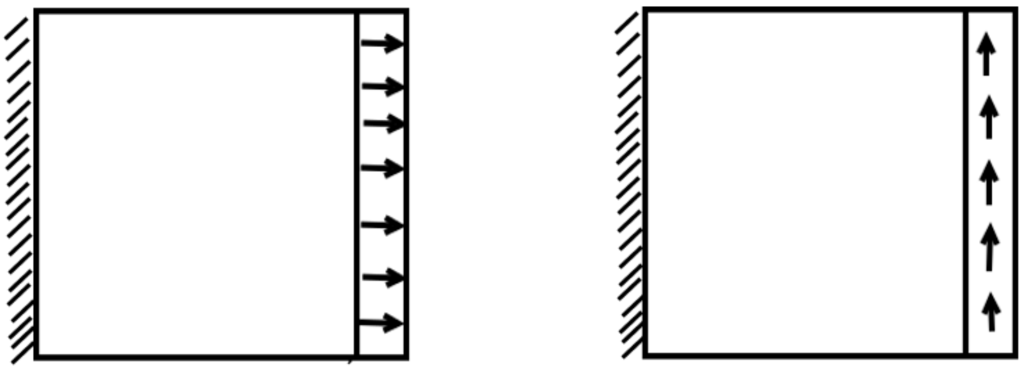


Figure 5.8: The two types of loading for plane stress demo.

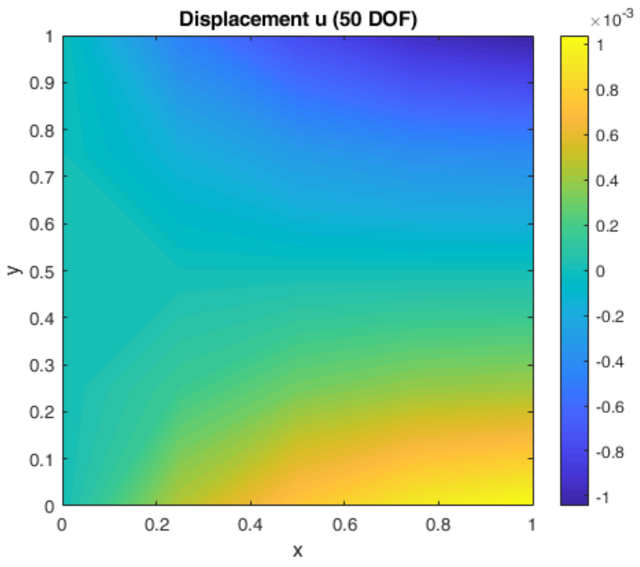


Figure 5.9: Displacement U (Parallel to x-axis)

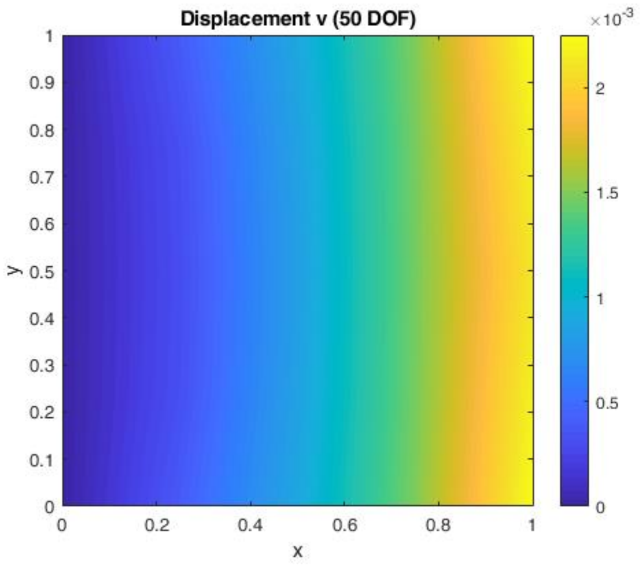


Figure 5.10: Displacement V (Parallel to y-axis)

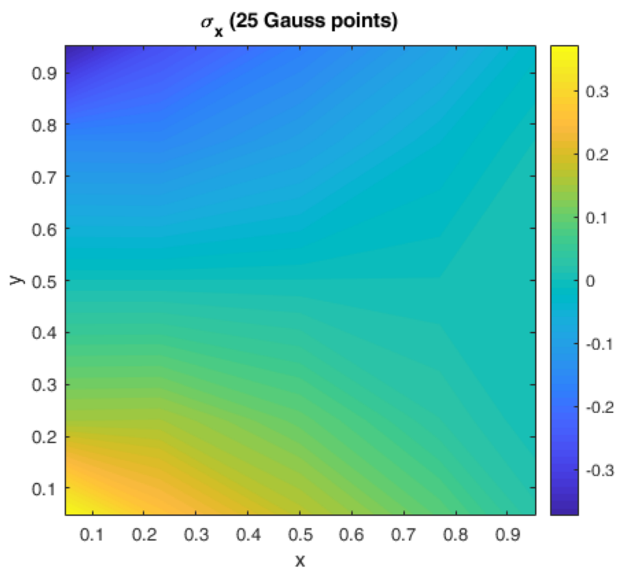


Figure 5.11: Distribution of σ_x.

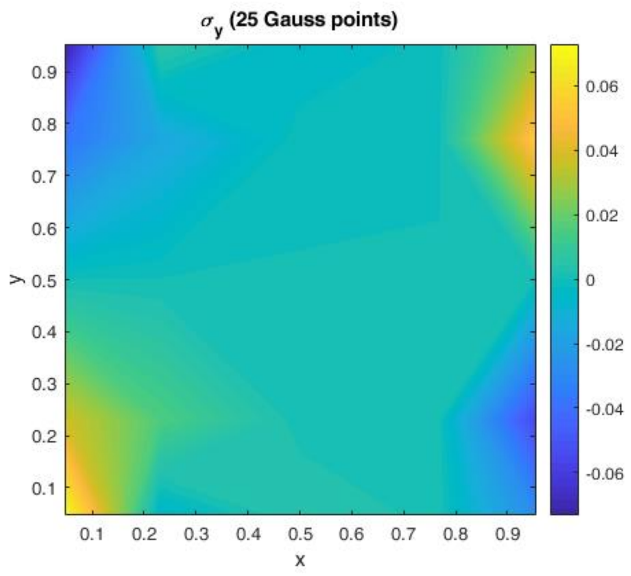


Figure 5.12: Distribution of σ_y

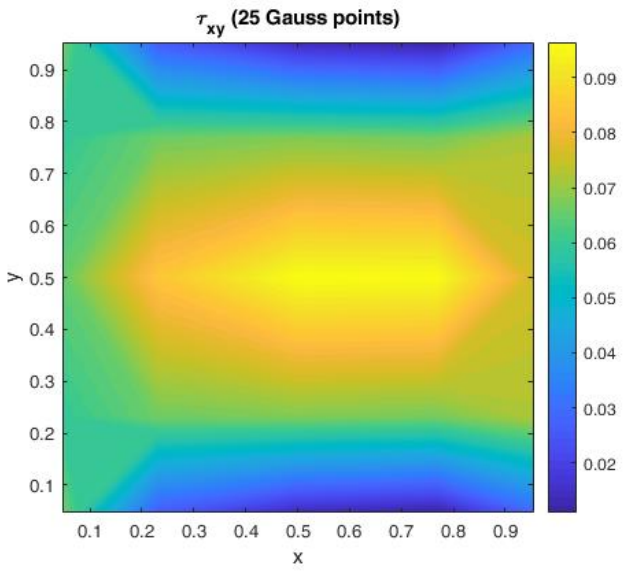


Figure 5.13: Distribution of τ_{xy}.

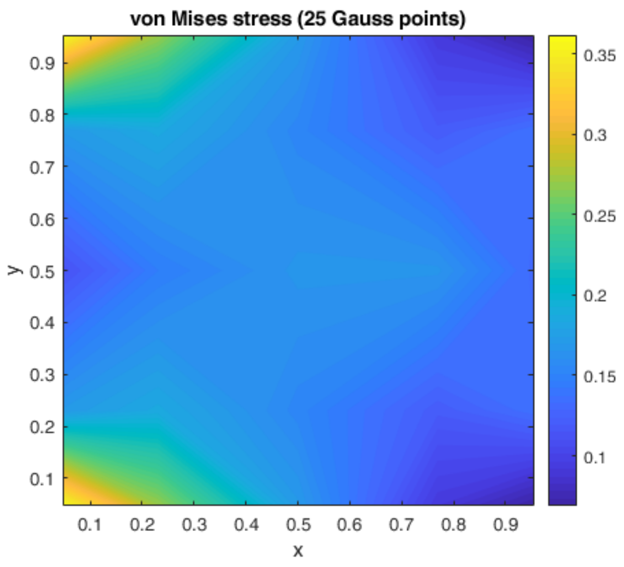


Figure 5.14:Distribution of Von Mises equivalent stress.

6 Results using I3GA

After explaining the basics for the methods used in I3GA, we proceed in running some cases in order to validate the program. We will show some results on various cases and classic finite elements benchmark tests in 2D thermal analysis. We will study the effect of different type of refinement on these cases and the different type of boundary conditions. We will also present for a specific problem, results from our adaptivity algorithms.

6.1 Annulus Benchmark Test and Refinements

In this case we examine the thermal problem on an annulus of fig. 6.1. In this problem we examine a quarter disk as part of a whole disk because the problem is axisymmetric. The inner radius is $R_{in} = 2.5$ m and the outer is $R_{out} = 10$ m. We have two types of boundary conditions; two Dirichlet $(T = \{0{,}100℃\})$ and two homogeneous Neumann boundary conditions $q_n = 0\,\frac{\mathrm{W}}{\mathrm{m}^2}$ (zero heat flux). We run this problem using both NURBS and Bézier elements for various h and p refinement cases. For Bézier extraction case we also calculate the error between the exact and the numerical solution. This error is calculated by the L2 norm as:

$$\left\|T_{approx} - T_{exact}\right\|_2 = \sqrt{\frac{\int_\Omega \left(T_{approx} - T_{exact}\right)^2 \mathrm{d}\Omega}{\int_\Omega (T_{exact})^2 \mathrm{d}\Omega}} \tag{6.1}$$

Where, T_{approx} is the numerical solution and T_{exact} is the exact solution. The evaluated points in T_{approx}, T_{exact} are evaluated on the same control points for each NURBS or Bézier element. For the computation of the integrals Gauss Quadrature is used as in previous chapter. The analytical solution of this problem according to [30] is:

$$T(x, y) = \frac{100}{\ln(4)} \ln\left(\frac{10}{\sqrt{x^2 + y^2}}\right) \tag{6.2}$$

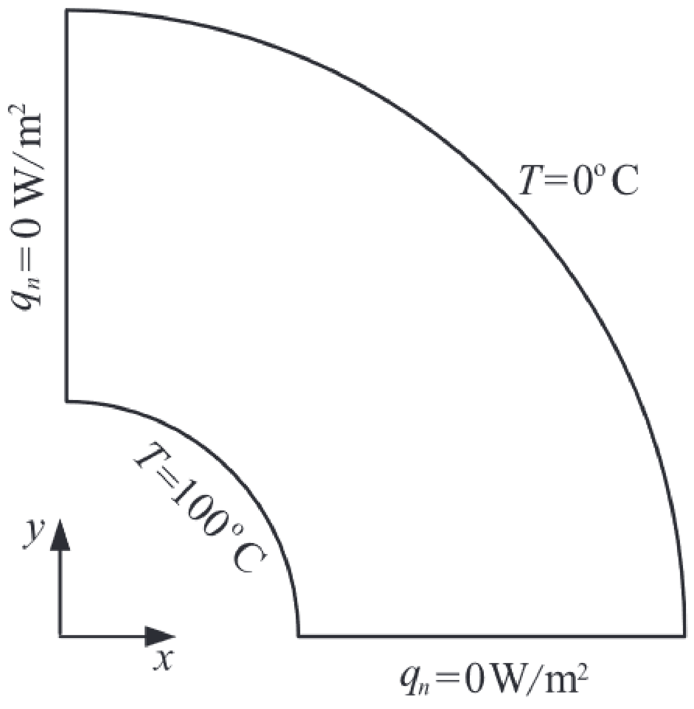


Figure 6.1: Annulus benchmark test. Source: [30].

In order to prove that the accuracy of I3GA is valid we demonstrate the same problem using one of the commercial CAD/CAE packages, SolidWorks. After model was created in SolidWorks' CAD environment, we passed it in CAE to perform the FEM analysis. In this example it was selected Steel AISI 4340 (normalized) for the simulation. After setting the boundary conditions the mesh was developed. Because of simplicity of the problem, there is no need for an adaptive mesh. A triangular mesh as this in fig .6.2 is sufficient. The mesh consists of 20834 nodes and 10675 elements. In order to design the geometry in SolidWorks the thickness was set 1 mm for this 2D analysis problem. In 2D thermal analysis the thickness is not present in the parameters of calculation. The temperature distribution across the domain is shown in fig. 6.3.

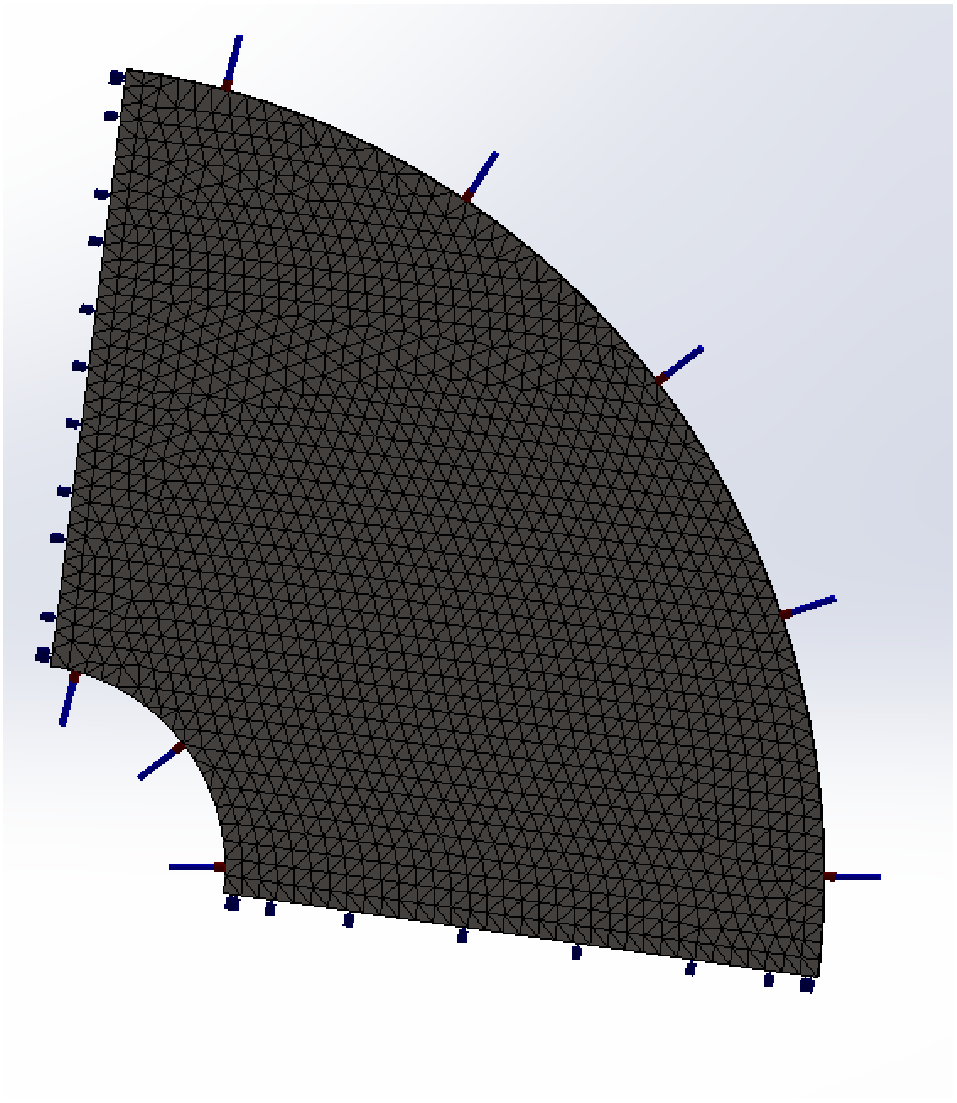


Figure 6.2: The triangular mesh of the benchmark problem.

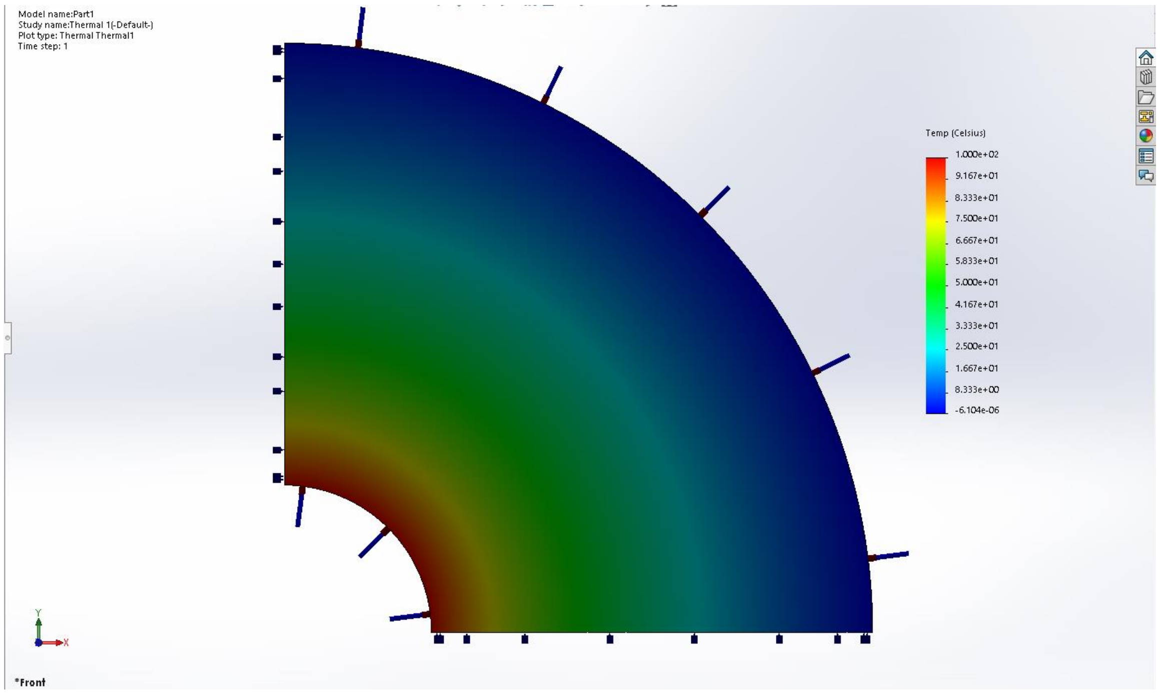

Model name:Part1
Study name:Thermal 1(-Default-)
Plot type: Thermal Thermal1
Time step: 1

Figure 6.3: FEM solution for the benchmark test.

As we can see the temperature is maximum on the inner arc and near zero (10^{-6} order) on the outer arc. Practically from the exact solution the value should be zero on the outer arc but due to numerical precision in plot, the SolidWorks does not show 0. As we can see the heat transfer is decreasing from maximum to minimum value logarithmically, which is in accordance with the theoretical results.

Below we present the results from I3GA (NURBS and Bézier elements) for the same problem. The surface is 2nd degree in every parametric direction with knot vectors $\boldsymbol{\Xi} = \mathbf{H} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$. The control mesh and the physical domain are shown on fig. 6.4 and control points weights on table 6.1. This problem has been evaluated for different knot insertion and degree elevation cases. For the h-refinement case we inserted globally knots at the half of each NURBS element for 4 times and for the p-refinement case we elevated the degree of the surfaces every time by one degree, again for 4 times (finally 6 degree). In each case during Bézier extraction method we have computed the error according to eq. (6.1). The table showing the errors and a semilogarithmic diagram of DoFs -vs- error is following below. Finally, we will present for each case, the Bézier control mesh and the physical domain in order to compare the results from the NURBS counterpart.

Control Points		Weights
0.000	2.500	1.000
0.000	6.250	1.000
0.000	10.000	1.000
2.500	2.500	0.7071
6.250	6.250	0.7071
10.000	10.000	0.7071
2.500	0.000	1.000
6.250	0.000	1.000
10.000	0.000	1.000

Table 6-1: Control points and weights for the surface of benchmark test.

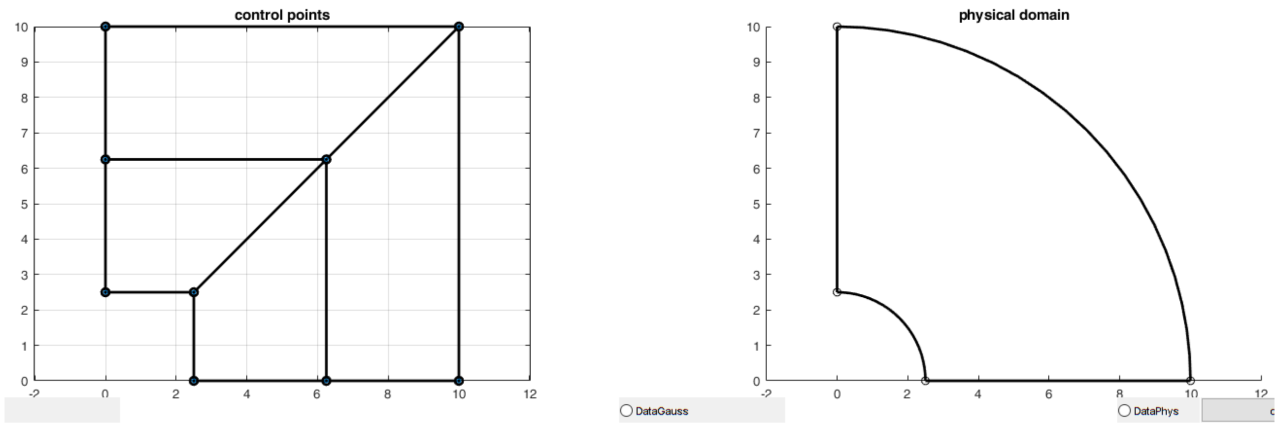


Figure 6.4: The control mesh and the physical mesh for the benchmark test.

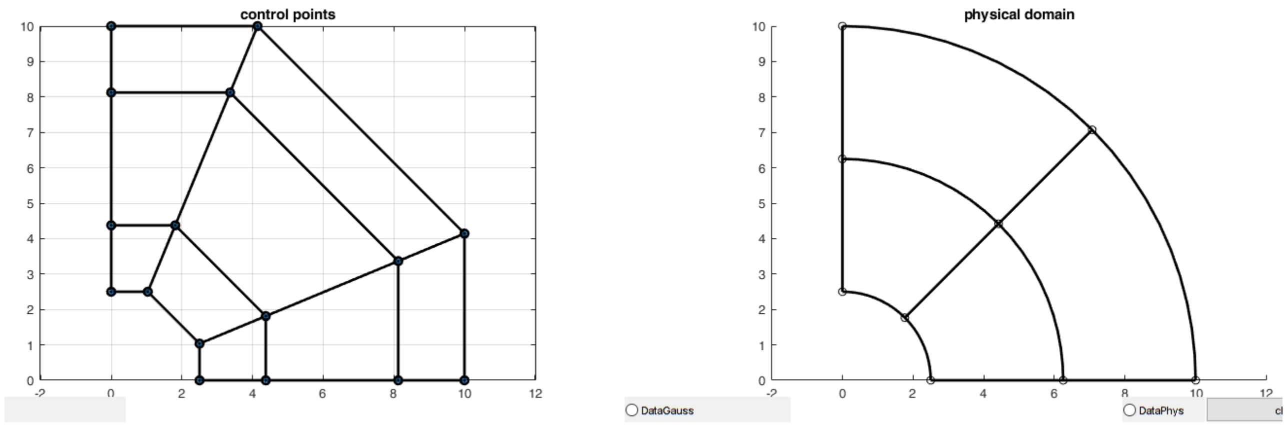


Figure 6.5: The control mesh and the physical mesh for the surface after refining globally one time. In total 16 control points.

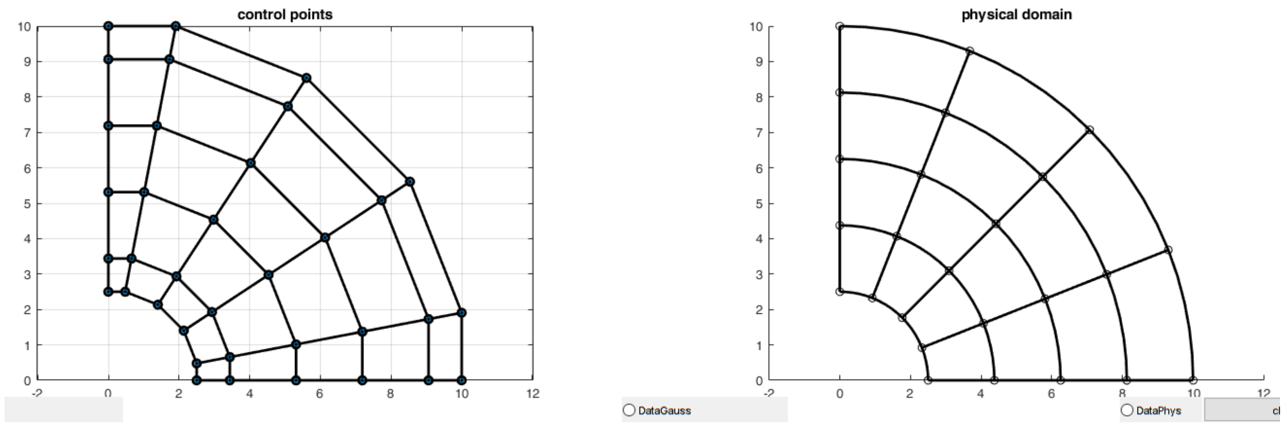


Figure 6.6: The control mesh and the physical mesh for the surface after refining globally two times. In total 36 control points.

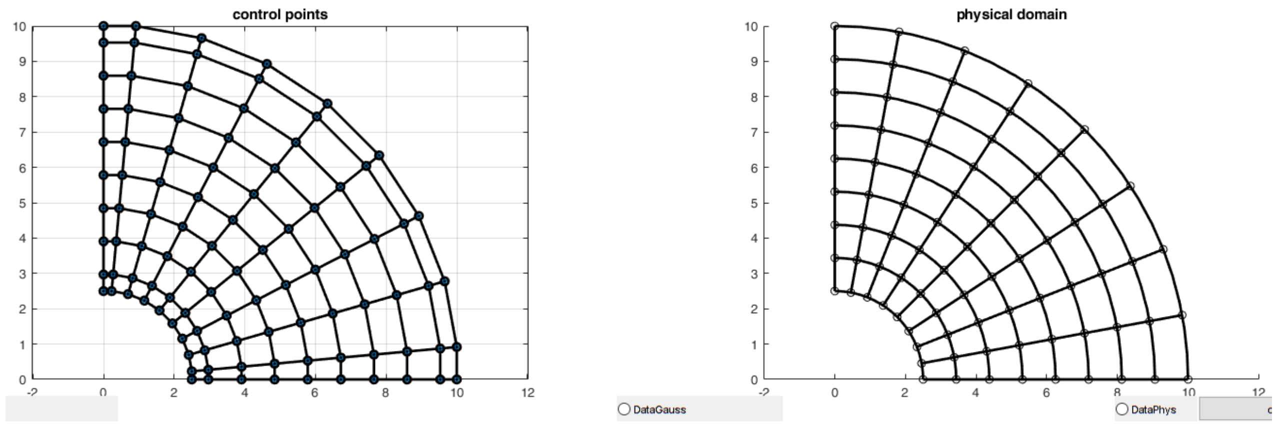


Figure 6.7: The control mesh and the physical mesh for the surface after refining globally three times. In total 100 control points.

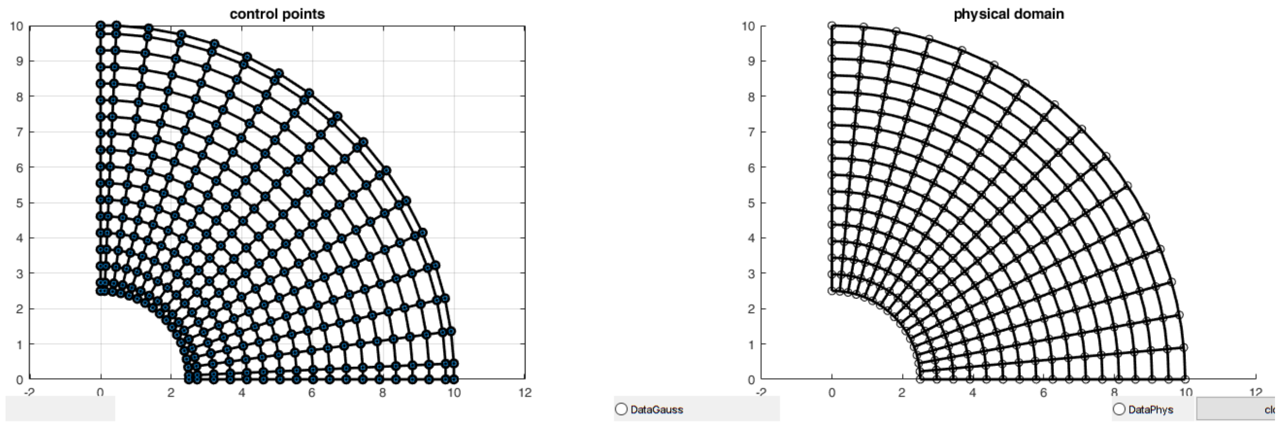


Figure 6.8: The control mesh and the physical mesh for the surface after refining globally four times. In total 324 control points.

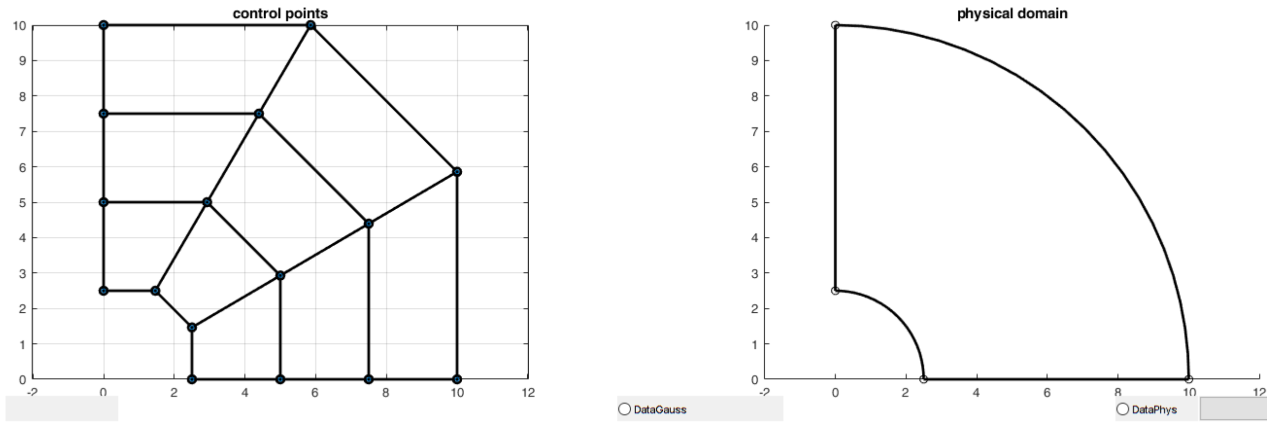


Figure 6.9: The control mesh and the physical mesh for the surface after degree elevation by one time. In total 16 control points.

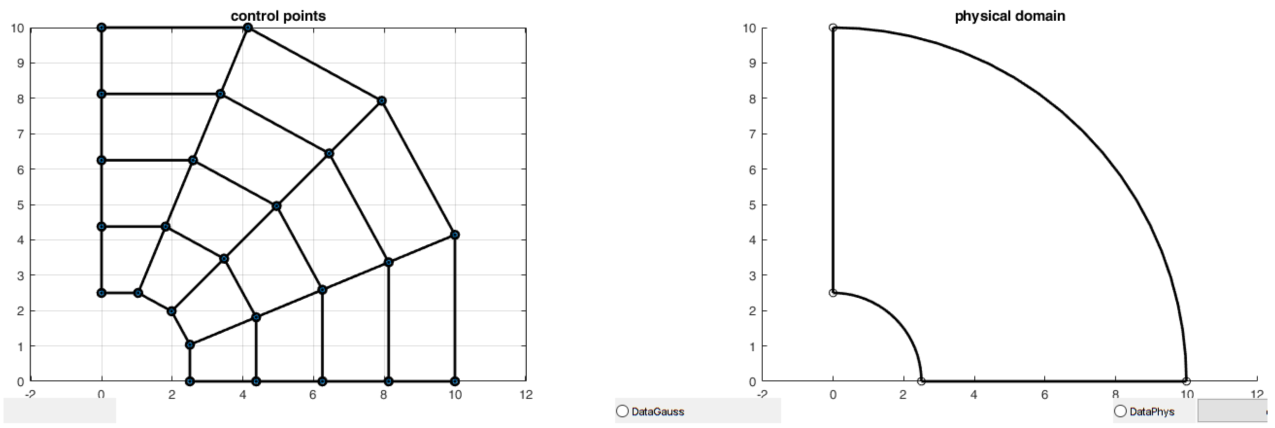


Figure 6.10: The control mesh and the physical mesh for the surface after degree elevation by two times. In total 25 control points.

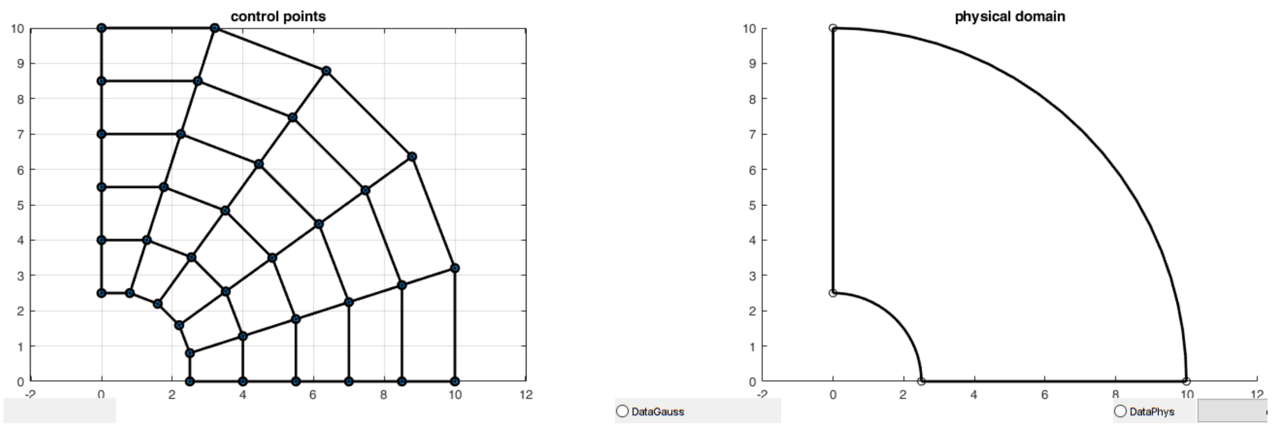


Figure 6.11: The control mesh and the physical mesh for the surface after degree elevation by three times. In total 36 control points.

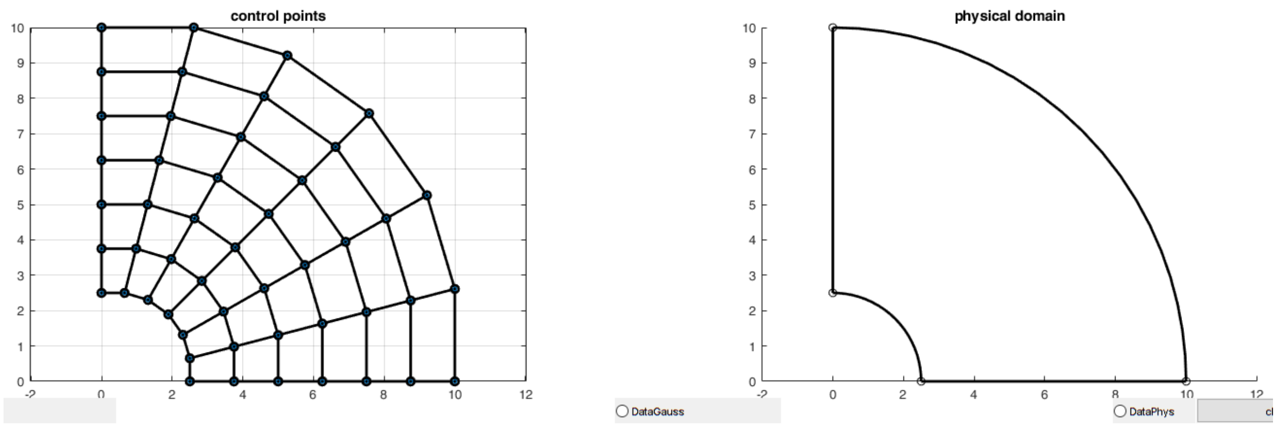
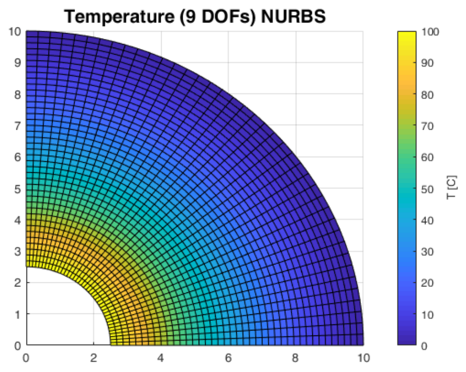
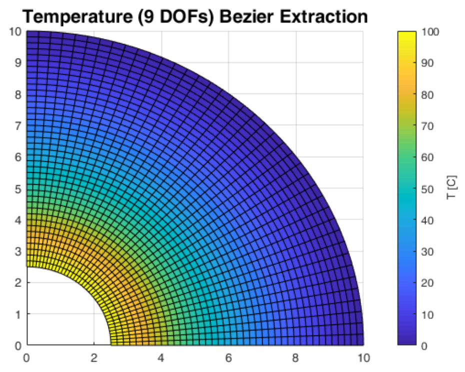
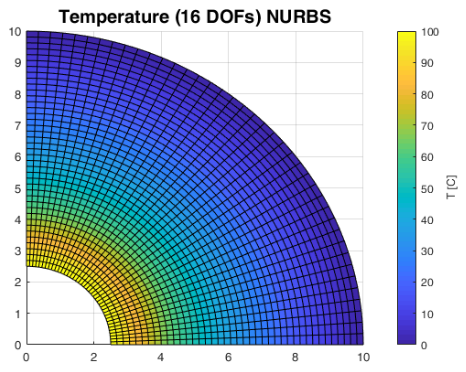
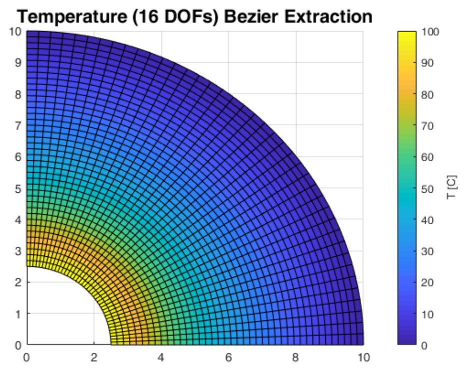


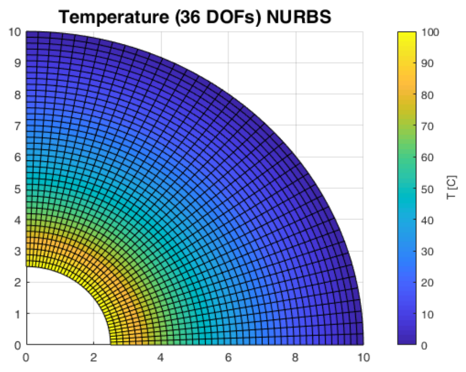
Figure 6.12: The control mesh and the physical mesh for the surface after degree elevation by four times. In total 49 control points.

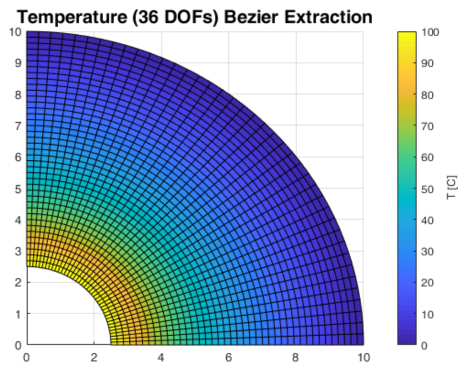

Temperature (9 DOFs) NURBS
T [C]

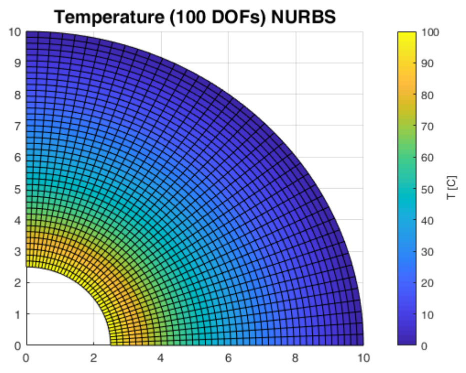

Temperature (9 DOFs) Bezier Extraction
T [C]

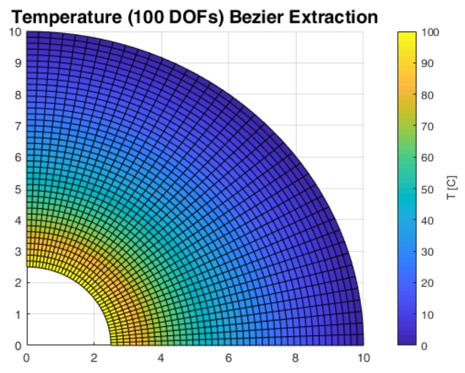

Temperature (16 DOFs) NURBS
T [C]


Temperature (16 DOFs) Bezier Extraction
T [C]


Temperature (36 DOFs) NURBS
T [C]


Temperature (36 DOFs) Bezier Extraction
T [C]


Temperature (100 DOFs) NURBS
T [C]


Temperature (100 DOFs) Bezier Extraction
T [C]

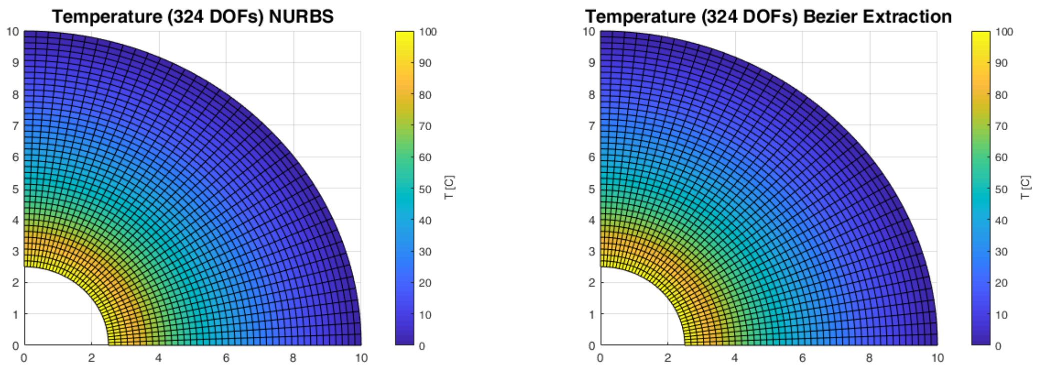
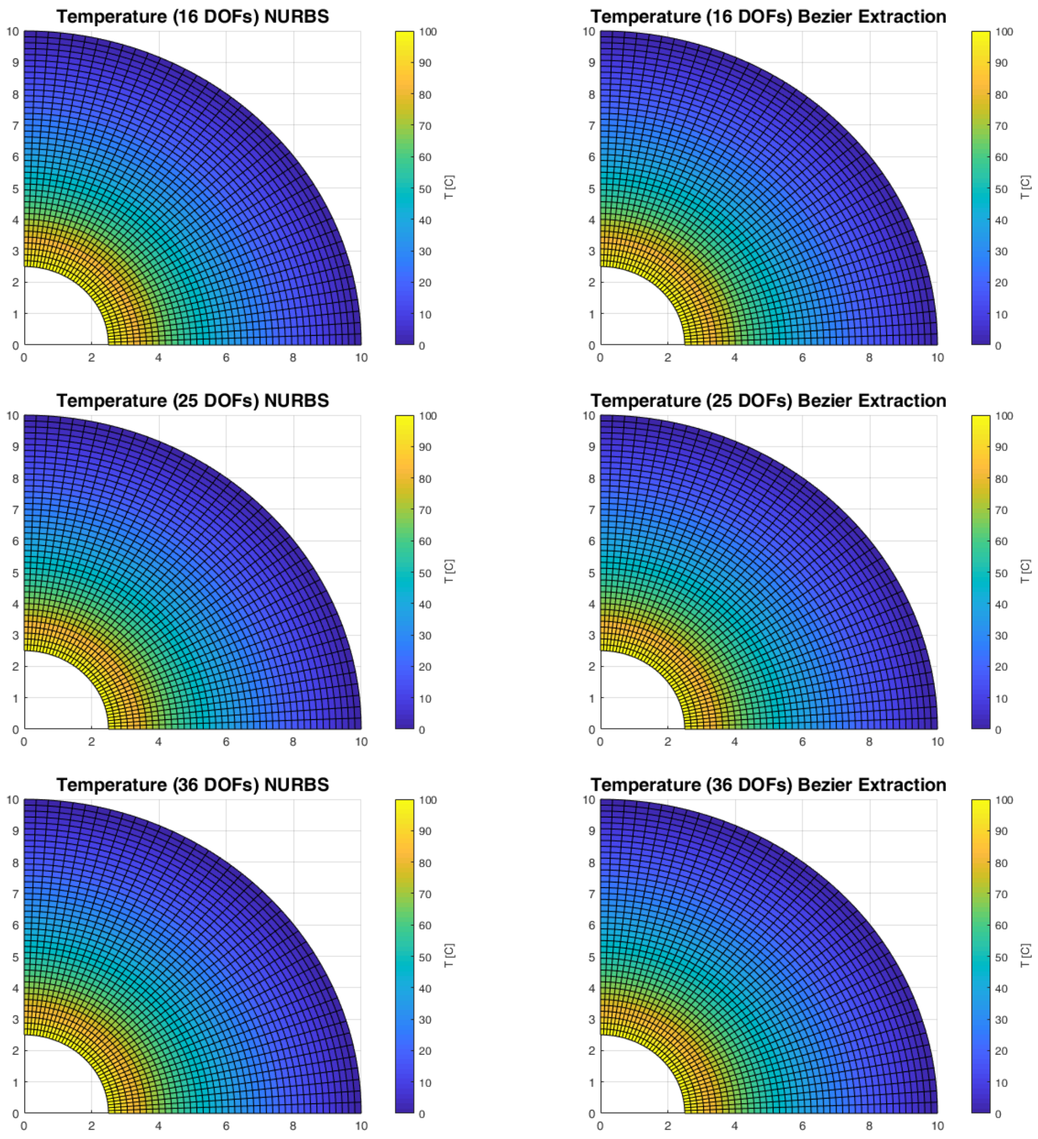


Figure 6.13: Temperature distribution on h-refinement cases using IGA and Bézier extraction.



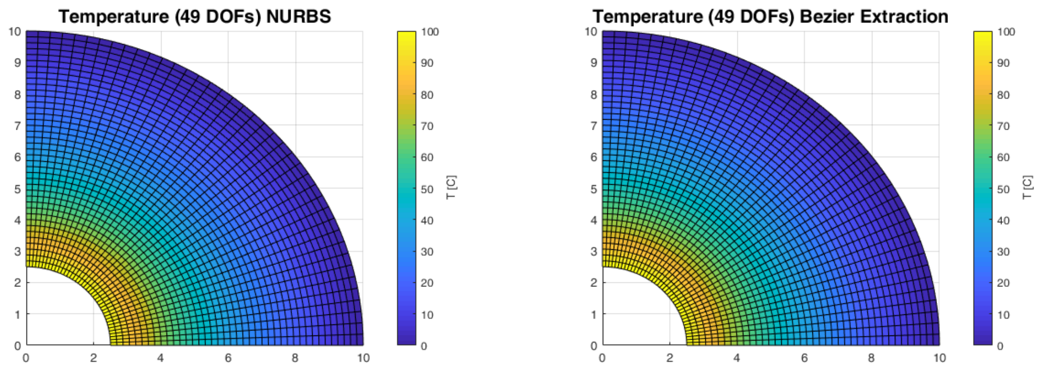
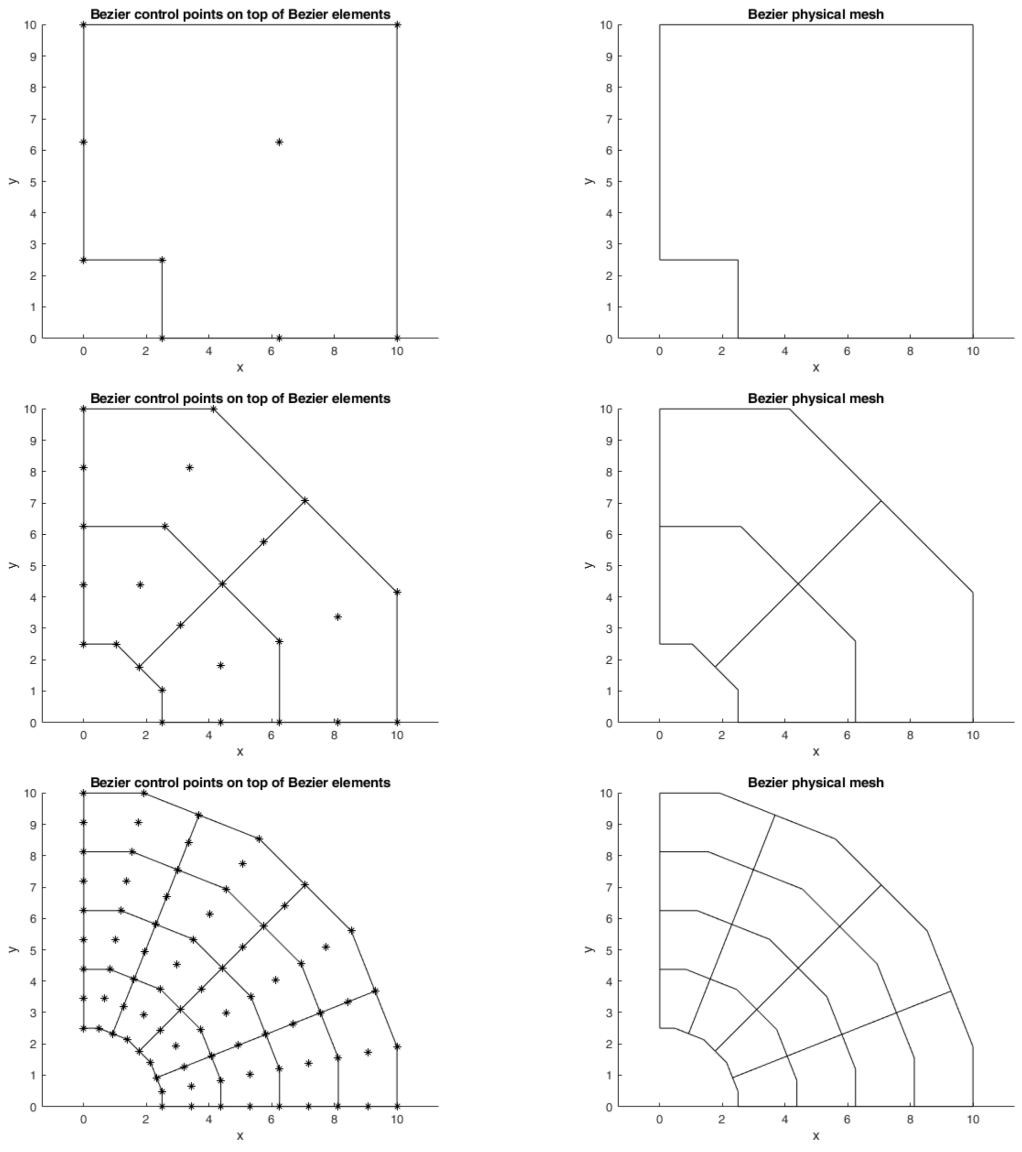


Figure 6.14: Temperature distribution on p-refinement cases using IGA and Bézier extraction.


Bezier control points on top of Bezier elements
Bezier physical mesh
Bezier control points on top of Bezier elements
Bezier physical mesh
Bezier control points on top of Bezier elements
Bezier physical mesh
x
y

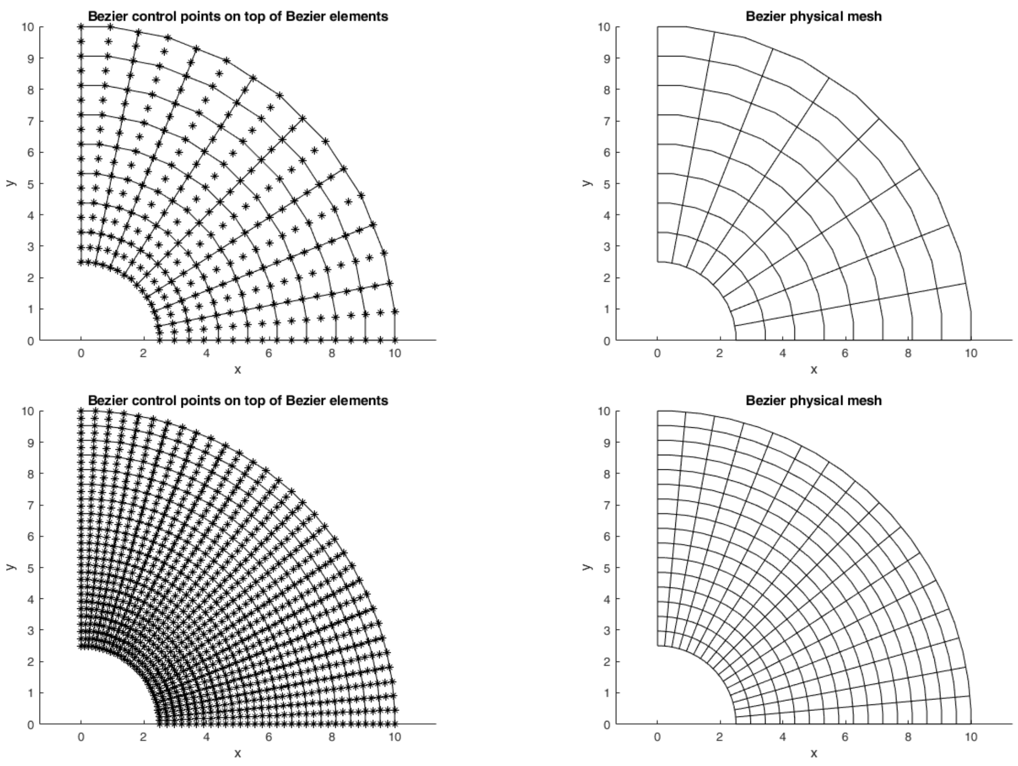
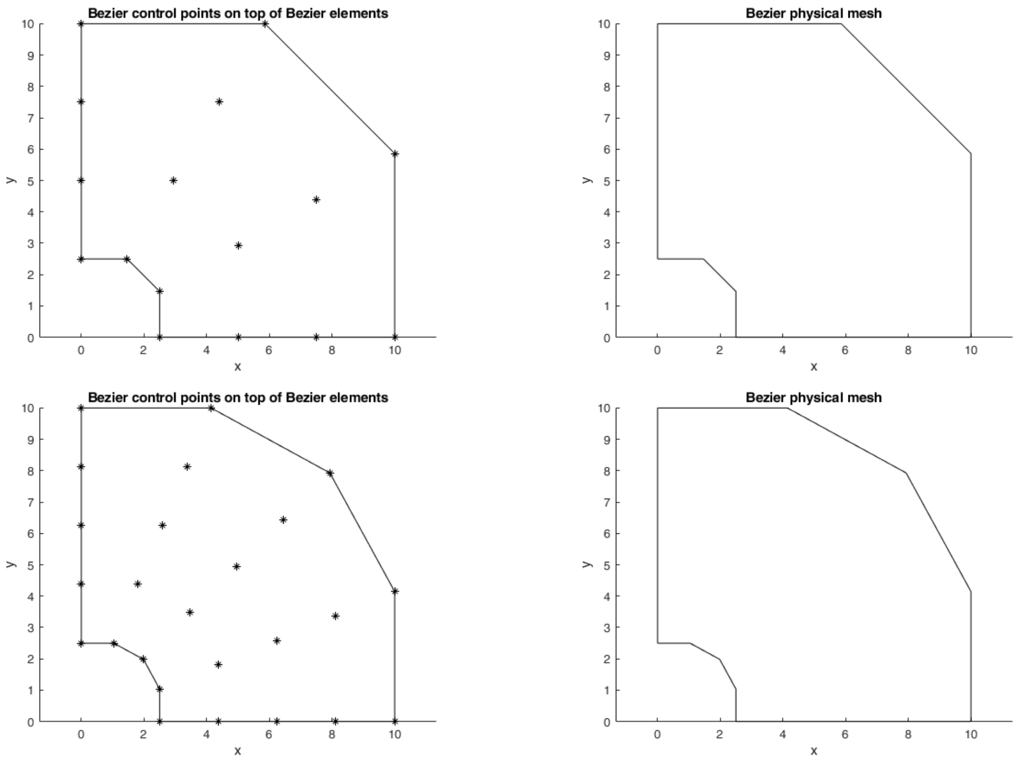
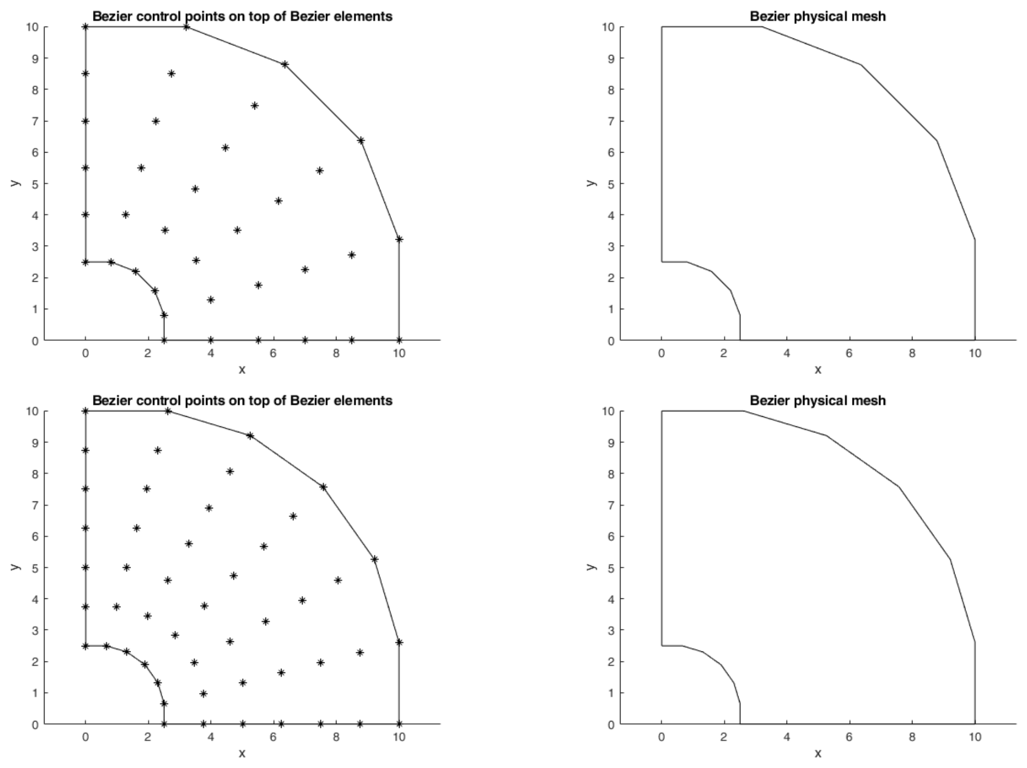


Figure 6.15: Bézier control and physical mesh on h-refinement cases.




Bezier control points on top of Bezier elements
Bezier physical mesh
Bezier control points on top of Bezier elements
Bezier physical mesh
x
y
Figure 6.16: Bézier control and physical mesh on p-refinement cases.

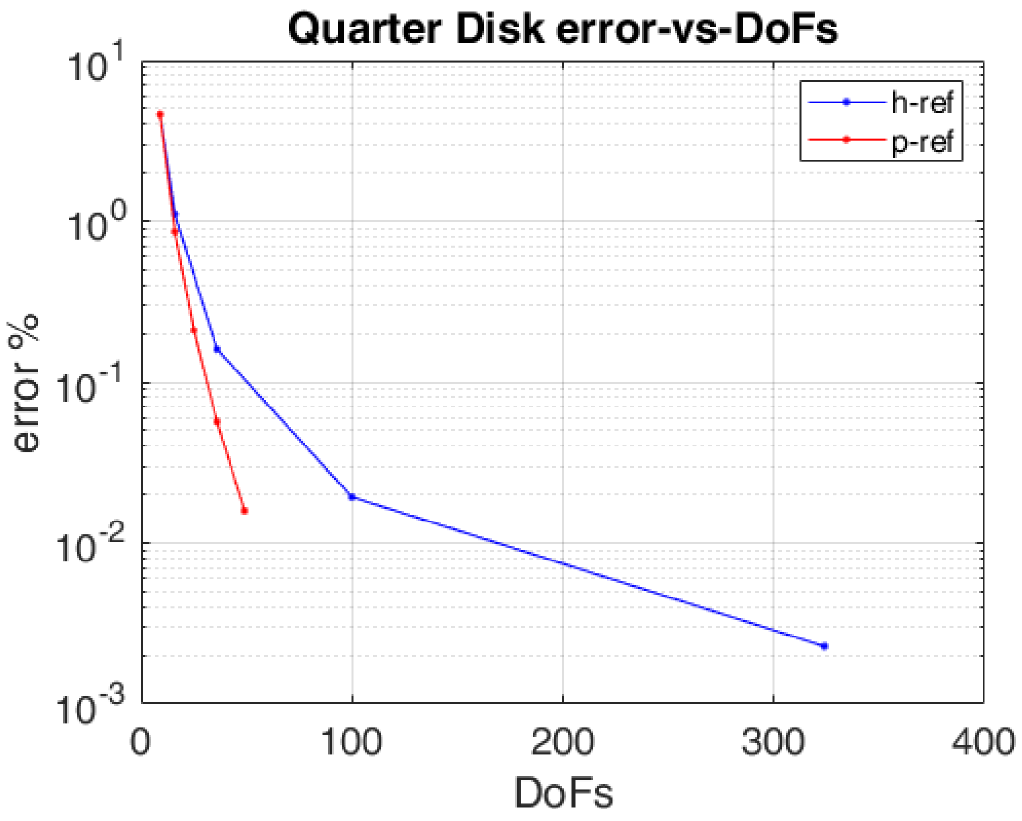


Figure 6.17: Error of Bézier extraction method-vs- degrees of freedom for different cases in h and p refinement (MATLAB).

h-refinement		p-refinement	
DoFs	error	DoFs	error
9	0.0462	9 $(p, q = 2)$	0.0462
16	0.0111	16 $(p, q = 3)$	0.0086
36	0.0016	25 $(p, q = 4)$	0.0021
100	1.93E-04	36 $(p, q = 5)$	5.67E-04
324	2.28E-05	49 $(p, q = 6)$	1.59E-04

Table 6-2: Table with DoFs and errors for all cases in Bézier extraction method. The p, q are the basis functions degrees in ξ and η direction respectively.

Conclusions

At first the analysis algorithms using IGA NURBS and IGA Bézier extraction are validated. As we can see in both figures 6.13 and 6.14, the results for every case are the same with the distribution in FEM analysis on fig. 6.3. Although the number of nodes in FEA is way much larger than the number of control points in IGA cases. That shows the superiority of IGA vs FEA, in which we use the geometric mesh for computational space, something that gives precise results with less information (control points).

The results from h-refinement are the same for both IGA NURBS and Bézier extraction. Even when the number of control points is increasing, the results remain unchanged. H-refinement shows its potential in more complex problems with different types of boundary conditions. However, as we can see in fig.6.17 and table 6.2 the overall error is decreasing as long as the number of control points increases. This is a logical outcome. In correspondence with h-refinement in p-refinement the error also decreases while the surface degree increases. Then the analysis is becoming more accurate. We can see that with p-refinement the number of DoFs for $p, q = 6$ becomes 49 with an error of 0.016% in contrast to h-refinement in which at 100 DoFs the number becomes 0.019%. The higher continuity due to degree elevation gives more accurate results comparing with knot insertion process.

We can also observe that the NURBS control and physical mesh have more similarities than differences with their Bézier counterparts. If we see figures 6.4 to 6.8 and fig. 6.15, we can also notice that with more refined geometry we take more elements in both cases. Each Bézier element share 9 control points with NURBS and Bézier element. The same also happens in p-refinement where the physical domain has only one element in both NURBS and Bézier extraction case. The number of control points in corresponding cases is also the same. The only difference that the two methods have is their representation where Bézier elements have C^0-continuity instead of NURBS elements which have higher continuity.

6.2 Square with Non-Uniform Dirichlet conditions Benchmark Test

Another case that was executed by using I3GA is the square with Non-Uniform Dirichlet boundary conditions that is shown in fig. 6.18.

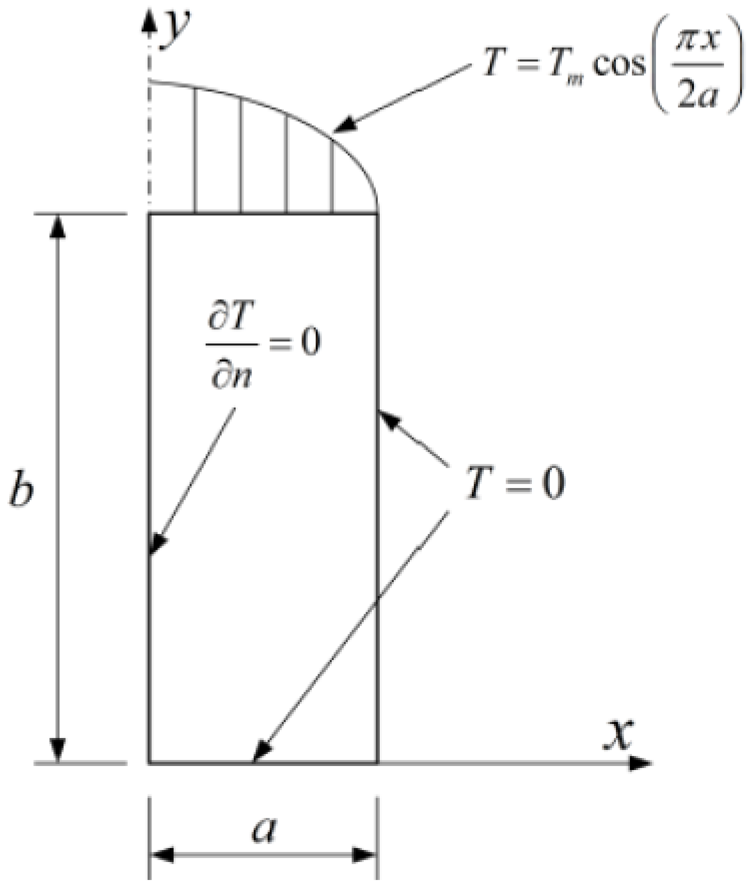

y
$T = T_m \cos\left(\frac{\pi x}{2a}\right)$
$\frac{\partial T}{\partial n} = 0$
b
$T = 0$
x
a

Figure 6.18: Square with Non-Uniform Dirichlet Boundary conditions. Source [27]

Considering zero source term and the boundary conditions of fig. 6.18 the theoretical solution of the problem is:

$$T(x,y) = T_m \left(\frac{\sinh\left(\frac{\pi y}{2a}\right)}{\sinh\left(\frac{\pi b}{2a}\right)} \right) \cos\left(\frac{\pi x}{2a}\right) \tag{6.3}$$

Where, T_m is the maximum temperature. For this problem we consider $a = 1, b = 1$ and $T_m = 100$ ℃. The geometry of this surface it is shown in next figure. The knot vectors in each direction are $\boldsymbol{\Xi} = \mathbf{H} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$ and all weights are equal to 1.

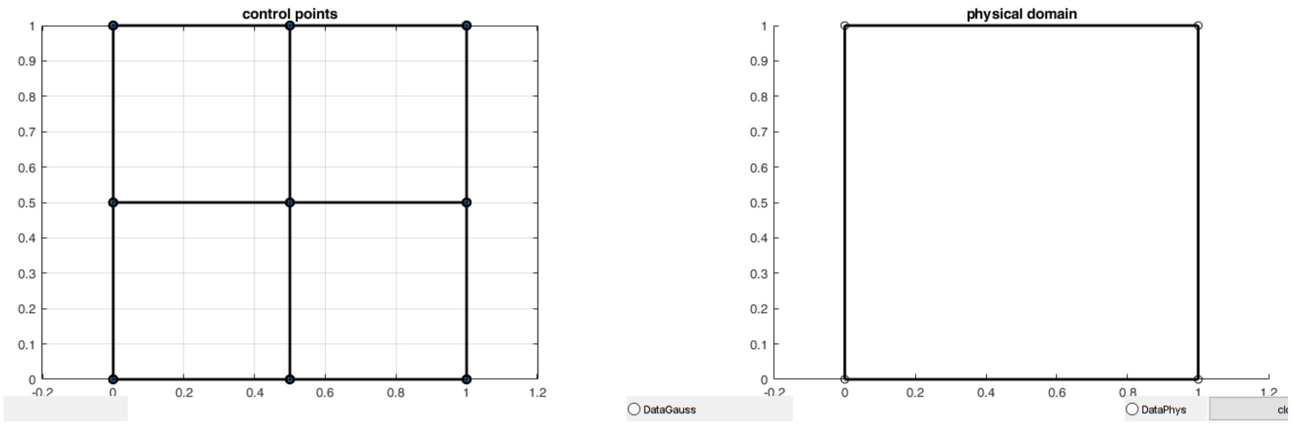


Figure 6.19: Geometry for the problem of fig. 6.18

We solve this problem again using IGA NURBS and Bézier extraction. The boundary conditions on the “up” side of the square are imposed differently from the boundary conditions in I3GA. So, another version of I3GA was implemented for dealing with this problem. Until now we impose the values of Dirichlet boundary conditions directly to the unknown control variables parameters T_i calculated by eq. (5.3). In this case, since the temperature follows a distribution to the upper side, we need to find the T_i that corresponds to this side. In order to calculate them we need to find “x-coordinate” of control points in the side and then calculate the corresponding basis functions [27]. After that, we calculate the basis functions at these points and then we solve the system between the basis function values, the corresponding distribution values and the control variables. In other words, we solve the system below:

$$\boldsymbol{T} = \boldsymbol{N}_{\boldsymbol{mat}}^{-1}\boldsymbol{q} \tag{6.4}$$

Where, $\boldsymbol{T} \in \mathbb{R}^{\boldsymbol{m}}$ is the control variable vector, $\boldsymbol{N}_{\boldsymbol{mat}} \in \mathbb{R}^{\boldsymbol{m}\times\boldsymbol{m}}$ is the basis function matrix, $\boldsymbol{q} \in \mathbb{R}^{\boldsymbol{m}}$ is the vector with elements, the distribution values at x-coordinate of upper side control points and m is the number of control points in the upper side. After calculate the control variables we impose them directly to system (5.3) as with the other boundary conditions. The results of analysis are following below:

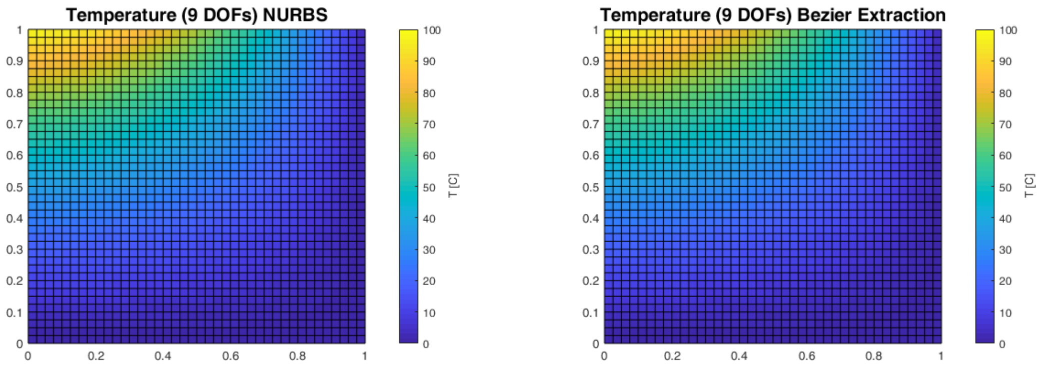


Figure 6.20: Results of square problem with non-uniform boundary conditions.

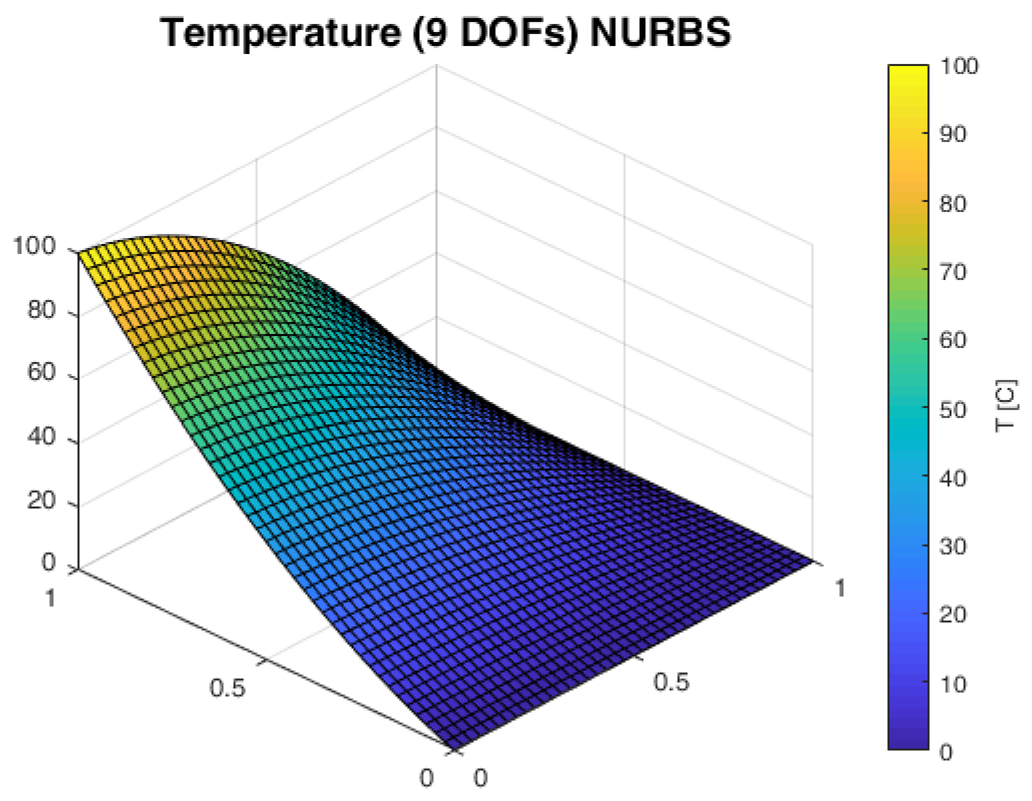


Figure 6.21: Isometric view of temperature distribution of IGA NURBS analysis case.

In both IGA NURBS and Bézier extraction the temperature distribution are the same. The maximum value is near the left edge and the minimum value is on the right edge of the upper side. This distribution is imposed by the cosine of boundary conditions. In "isometric view" of distribution (fig. 6.21) we have a better view of the effect of cosine has on the upper side. The overall error of Bézier extraction analysis was calculated equal to 2.63 %. Due to the amount of control points that were used for the first approximation the overall error seems quite expected.

6.3 Cases with sources

In this section we study cases where we impose a source term on the governing equation. The three cases that we study are the C0 and C1 L-shapes and the circular disk.

L-shapes

The two L-shapes control mesh and their physical mesh is shown in fig. 6.22 and 6.23. The L-shape of fig. 6.22 has 15 control points (C0) and the other one 12 (C1). Although the physical space in two L-shapes seems the same it is not. First, the knot vectors are different. In case of fig. 6.22 the knot vectors are $\boldsymbol{\Xi} = \{0\ 0\ 0\ 0.5\ 0.5\ 1\ 1\ 1\}^{\mathrm{T}}$ and $\mathbf{H} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$ while in case of fig. 6.23 the knot vectors are $\boldsymbol{\Xi} = \{0\ 0\ 0\ 0.5\ 1\ 1\ 1\}^{\mathrm{T}}$ and $\mathbf{H} = \{0\ 0\ 0\ 1\ 1\ 1\}^{\mathrm{T}}$. In ξ-direction in the first case we have a double knot at 0.5 in contrast to the other case. So, the continuity in first case is C^0 at $\xi = 0.5$ and in the other C^1 because the overall degree of the surface in both directions equals to 2. Let name the first case L-shape C0 and the second case L-shape C1. The diagonal line in physical space represents the position where $\xi = 0.5$. The control points in C1 case seem to be 10 in number but this happens because two of them are coincide in order to create the internal C^1-continuity.

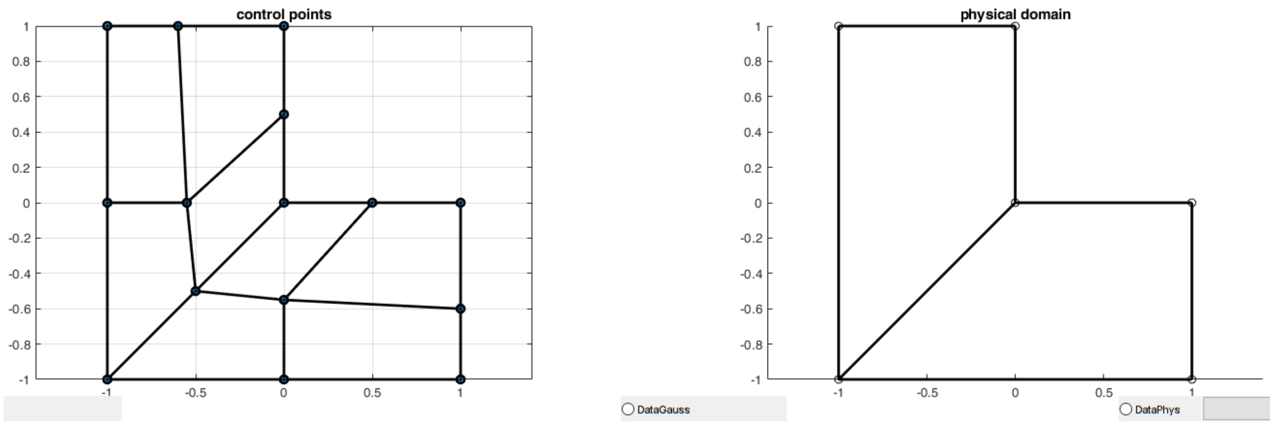


Figure 6.22: L2-shape with C0 internal boundary.

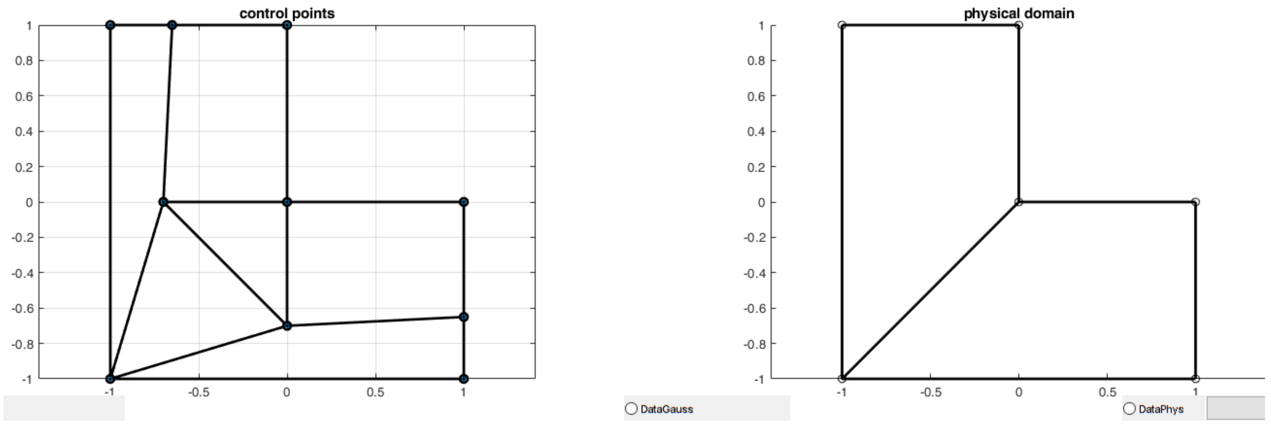


Figure 6.23: L2-shape with C1 internal boundary.

Before performing the analysis, we refine the geometries near their internal boundary because near boundaries the mesh must be finer. More specifically we add these knots in ξ-direction $\{0.4, 0.41, 0.42, \ldots, 0.49, 0.51, 0.52, \ldots, 0.6\}$ and in η $\{0.1, 0.2, 0.3, \ldots, 0.9\}$. The figures that show the final mesh are fig. 6.22, 6.23. The source term for both cases is the $f = 2\pi^2 \sin(\pi x) \sin(\pi y)$ and the boundary conditions are homogeneous Dirichlet on each side. The IGA NURBS results are following below.

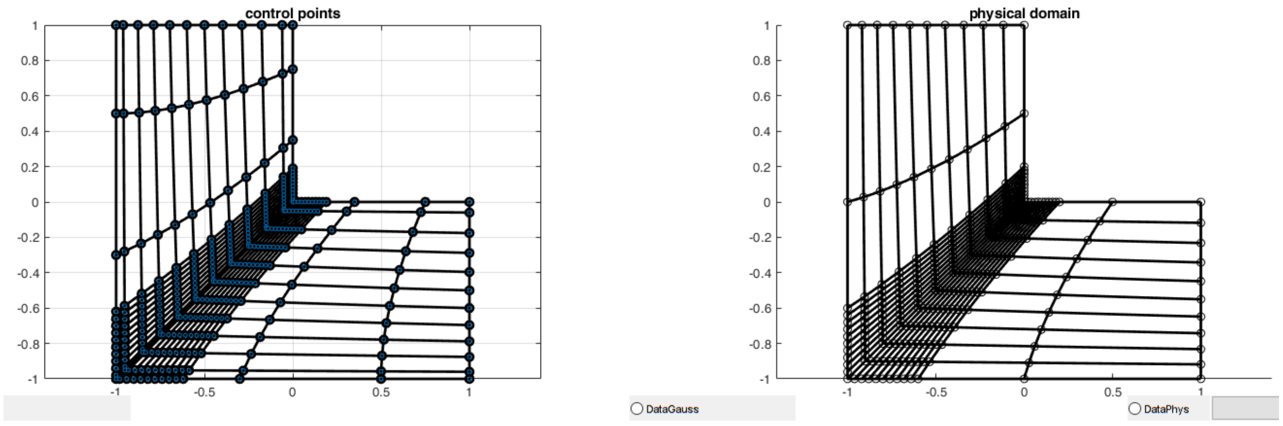


Figure 6.24: L2-shape C0 final mesh after knot refinement.

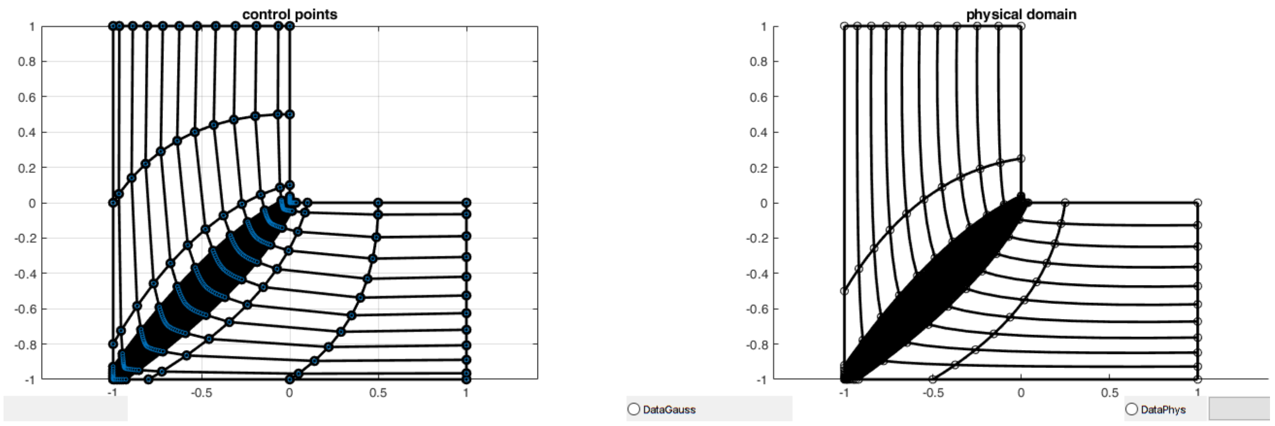


Figure 6.25: L2-shape C1 final mesh after knot refinement.

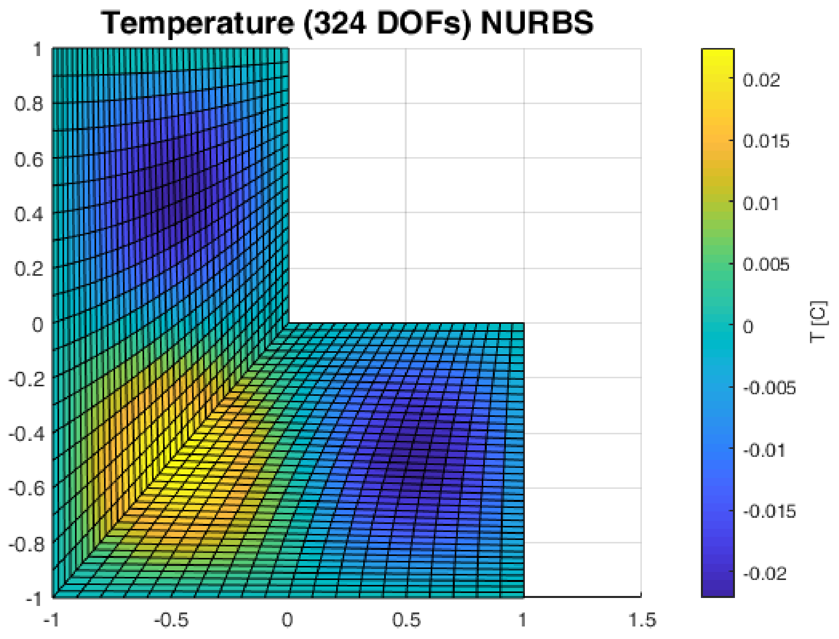


Figure 6.26: L2-shape C0 IGA NURBS results.

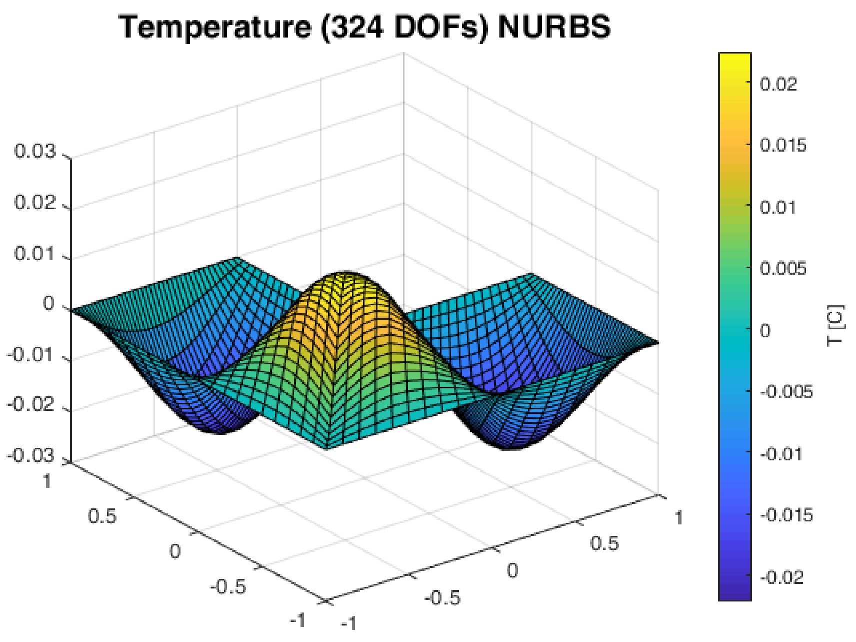


Figure 6.27: L2-shape C0 IGA NURBS results (isometric view).

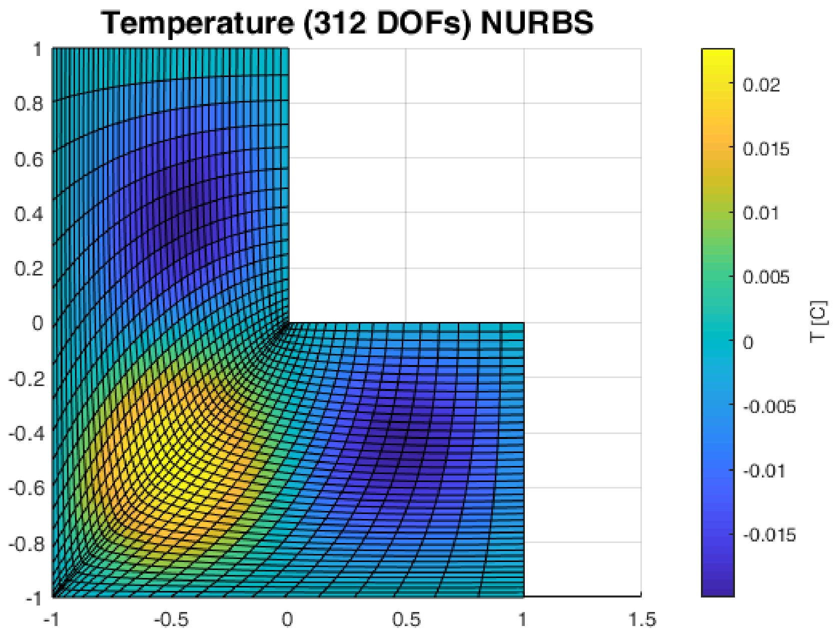


Figure 6.28: L2-shape C1 IGA NURBS results.

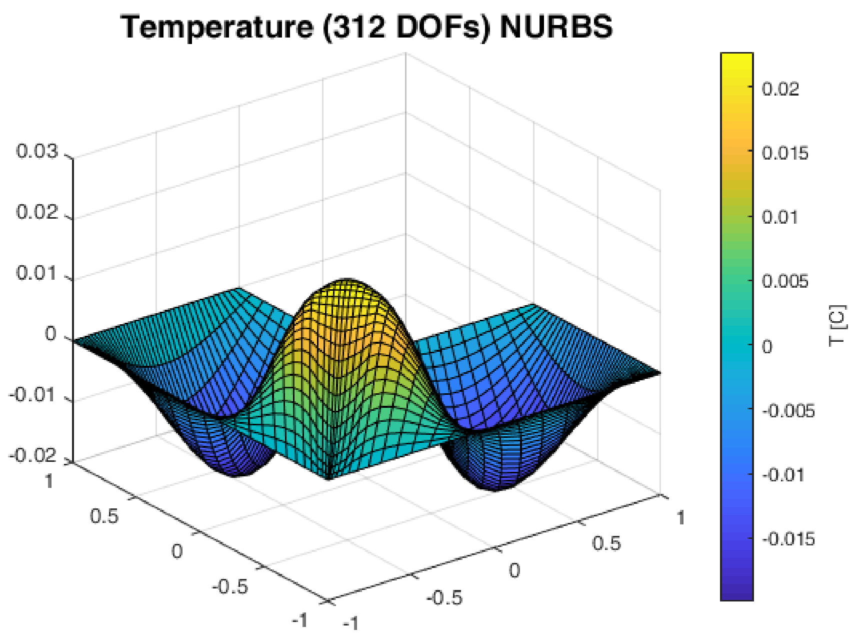


Figure 6.29: L2-shape C1 IGA NURBS results (isometric view).

From fig. 6.24 and 6.25 we see that the physical mesh is different because the control point distribution along with continuity at $\xi = 0.5$ differs. In C1 two control points are coincident with other in order to create the C^1-continuity. That is the reason that in C1 we have less DoFs than in C0. Moreover, the curvature in isolines[4] that are shown in physical space is also different because of the different continuity. Another difference exist also in the results, where the curvature of grid lines follows the lines of the respective shape in physical space. The distribution of temperature is following a trigonometric form as it is implied by the source term.

<u>Circular Disk</u>

In this case we have a circular surface which was refined until it has acquired 100 DoFs and then was solved using both IGA NURBS and Bézier extraction. The boundary conditions were again homogeneous Dirichlet and the source term follows a sinus function of the form $f = \sin\left(10\sqrt{x^2+y^2}\right)$. Below the results follow along with the control mesh and physical space.

[4] Lines of constant ξ and η.

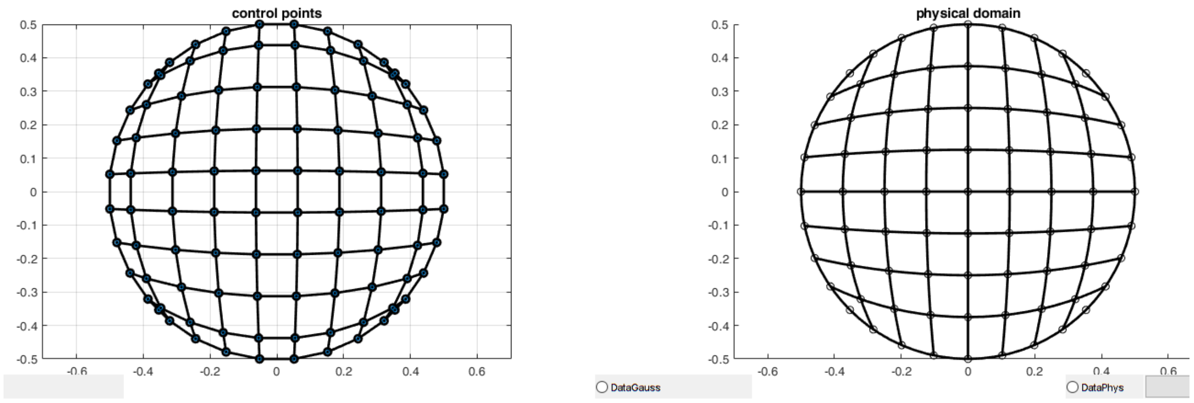


Figure 6.30: The control mesh and physical domain of a disk.

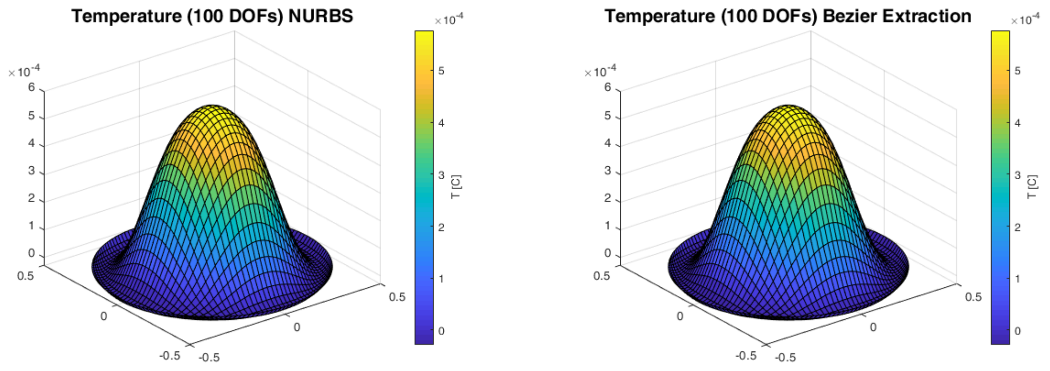


Figure 6.31: The results from IGA NURBS and Bézier extraction.

6.4 Effect of refinement on code execution time

The calculation time in analysis is significant so in this section we present a study on the effect of refinement methods on the analysis time while I3GA tool was used. Specifically, we use a benchmark test, using different refinement procedures in order to accomplish the study. The benchmark test is a square surface with length 1 m and Dirichlet and Neumann boundary conditions. The inhomogeneous Dirichlet boundary conditions are $T = 60℃$ on the "left" side and $T = 20℃$ on the "right" side. The homogeneous Neumann boundary conditions will be applied to "down" and "up" edges.

The initial surface is a square with knot vectors $\boldsymbol{\Xi} = \mathbf{H} = \{0\ 0\ 1\ 1\}^{\mathrm{T}}$. We have applied four refinement cases. These are:

- 2-h: Two h-refinements at the half of each NURBS element.
- 2-p: Two p-refinements by elevating the degree in total 2 times at both parametric directions. The final degree of surface is three.
- k: One k-refinement by applying one p-refinement for one degree and the one h-refinement at the half of the elements.
- h-p: One h-refinement at the half of the elements and then we elevate the degree by one globally.

The figures with the control mesh and physical domain for all cases are following below along with the results. We also present a bar graph with the execution times for all methods.

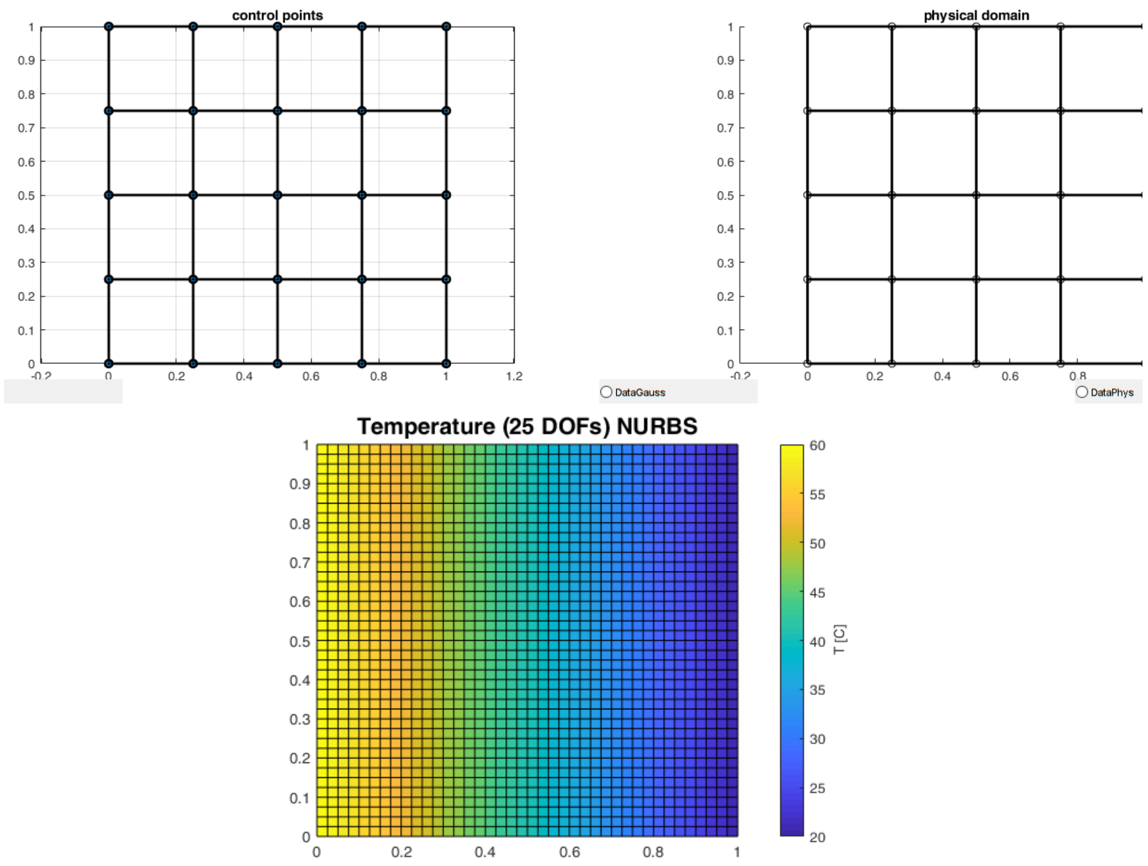
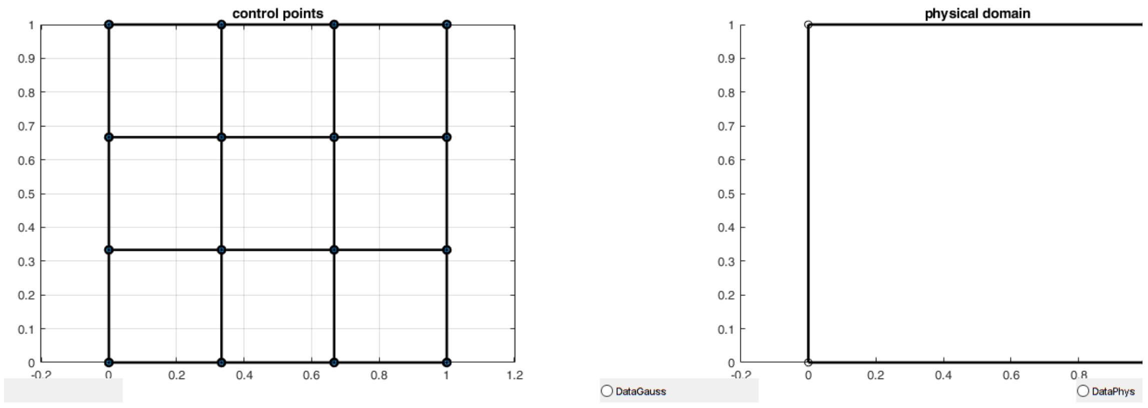


Figure 6.32:The surface and the results for 2-h method.



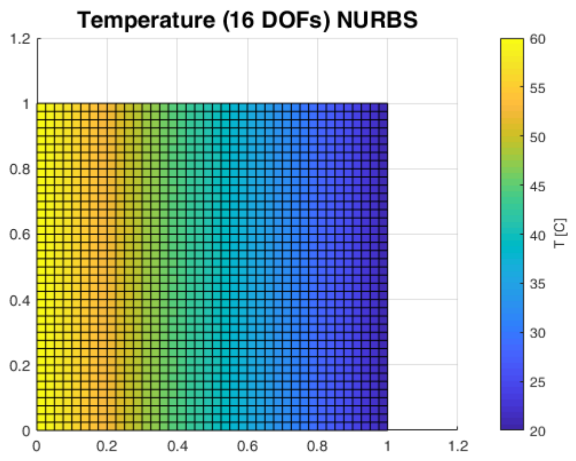


Figure 6.33: The surface and the results for 2-p method.

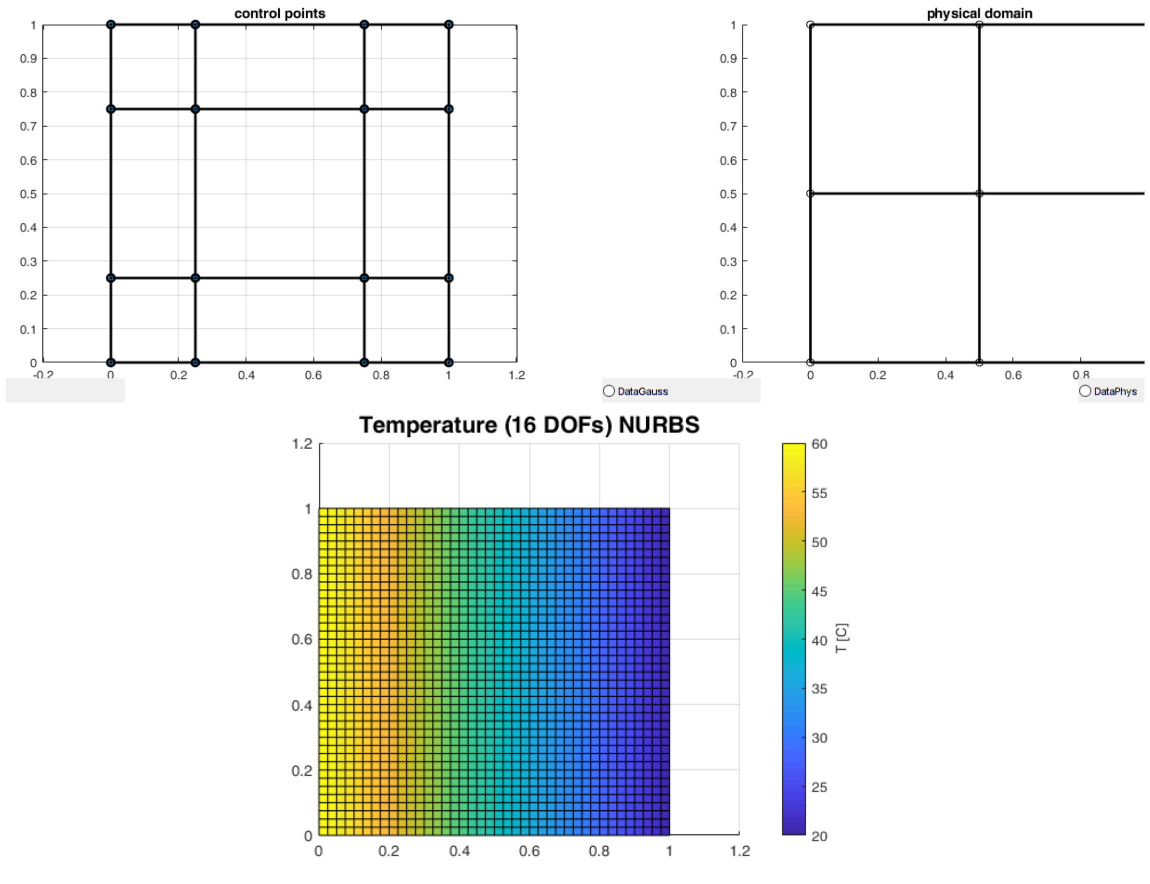


Figure 6.34: The surface and the results for k method.

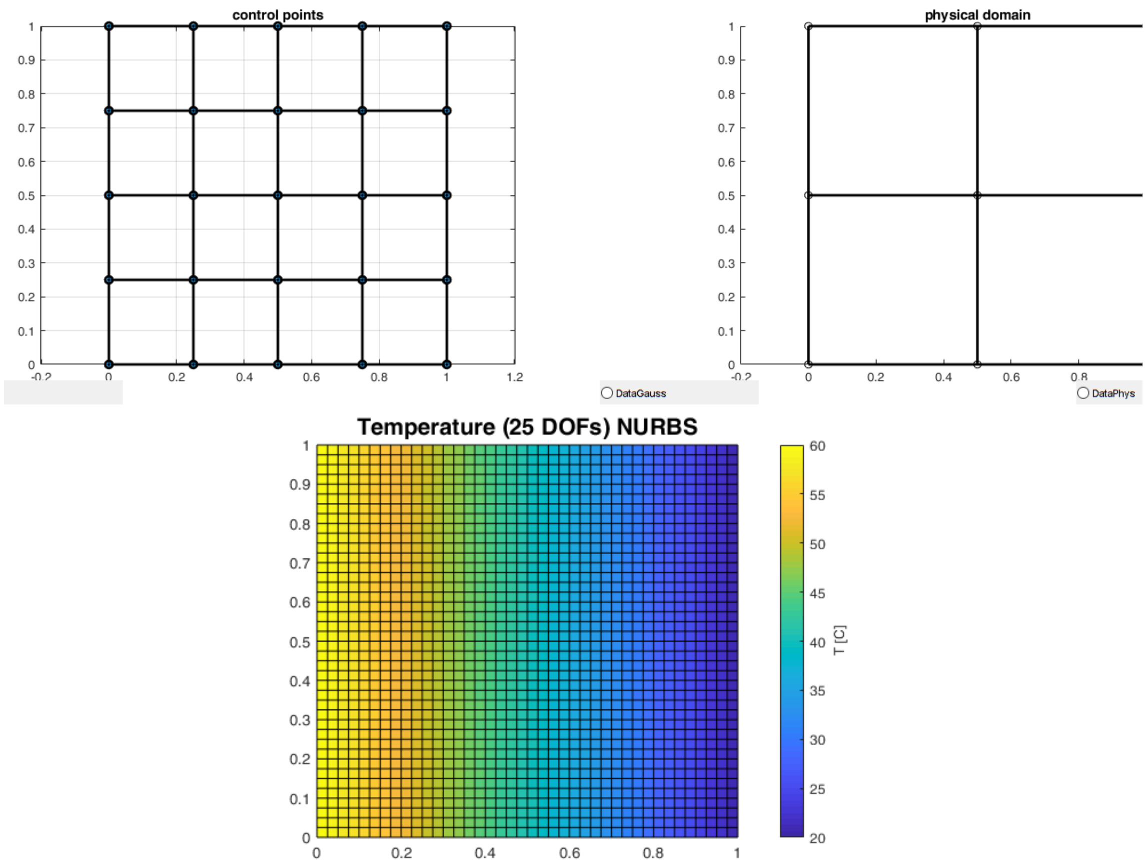


Figure 6.35: The surface and the results for h-p method.

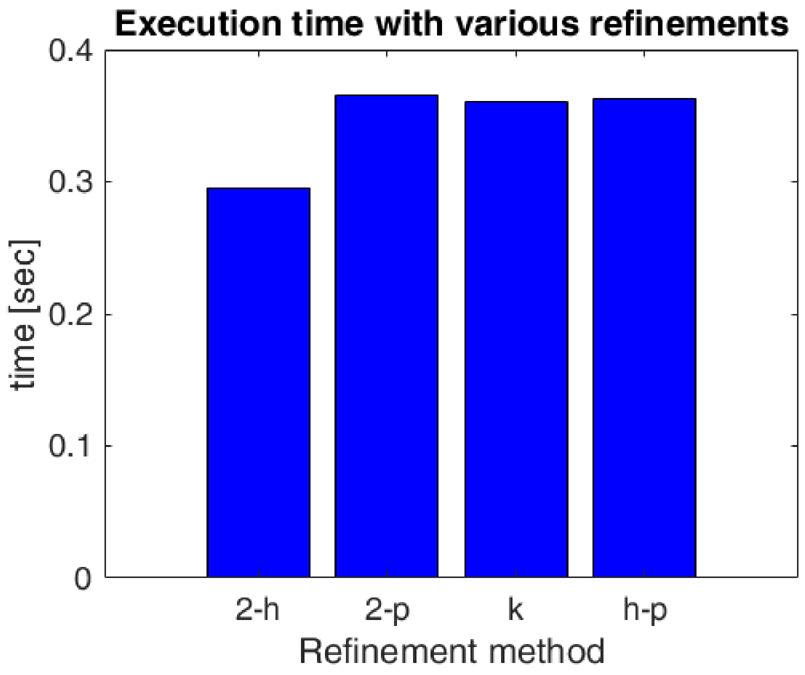


Figure 6.36: Execution time for different methods.

As we see in all the methods, the results are the same in terms of temperature distribution. However, the number of control points and elements have difference in each case:

- (2-h): 25 control points; 16 elements; 1st degree per direction.
- (2-p): 16 control points; 1 element; 3rd degree per direction.
- (k): 16 control points; 4 elements; 2nd degree per direction.
- (h-p): 25 control points; 4 elements; 2nd degree per direction.

From the above data and fig. 6.36 we may conclude that the degree of the surface is more significant than the number of elements in the surface. This happens because the surface degree defines the number of Gauss points. So, the execution time increases in accordance with the polynomial degree. Three of the four methods seem to have practically the same time results, but in problems with more complex geometries the k-refinement method is the most efficient according to Hughes [16]. The h-refinement method seems faster but as we showed in theory is less accurate.

6.5 Adaptivity

In industry and in Academic community there is an important need. The need is how a proper mesh can be built automatically for a problem in order to be accurate enough and computationally fast. There are certain areas in a model that need more information (elements) such as the boundaries of problem domain or near external sources. On the other hand, there are areas that do not need elements and the mesh could be coarser without affecting the analysis. The process of creating automatically the mesh with the adequate information for a model is called *Adaptivity*.

In I3GA two adaptivity examples are included for a specific problem case. The problem is a square with length equal to 1 m which has inhomogeneous Dirichlet boundary conditions. We have $T = 10℃$ on the "up" edge and $T = 0℃$ on all the other edges. The knot vectors for each direction are $\boldsymbol{\Xi} = \mathbf{H} = \{0\ 0\ 1\ 1\}^{\mathrm{T}}$. The problem was also solved using FEA in SolidWorks. The mesh and the solution are shown on fig. 6.37 and 6.38. As we can see on the upper boundary the temperature is maximum and the heat is transmitted in a way such that the temperature becomes zero on the other boundaries.

Let us return to adaptivity examples. A flowchart[5] for our current implementation is shown in fig. 6.39.

In order to create an adaptivity code we need a criterion, that controls how many times refinement should be done in order to be accurate enough during analysis. It is not a simple process to find a general criterion for each case. A criterion could be for example the error of solution between the computed and the exact solution in each element. Because this is the first adaptivity implementation in I3GA we have adopted a very simple criterion for this problem. Because of problems' symmetry as it is shown in solution space of fig. 6.38., after the 3rd refinement (see fig. 6.39) we calculate the temperature values at three points on the symmetry axis, which is parallel to y-axis. After we solve the problem once more, we check if the new three temperatures are different from the ones calculated in the previous iteration. Their comparing takes place for a specific error tolerance ($\varepsilon = 10^{-2}$). If the new values are near enough to the previous ones, the algorithm stops. In any other case the refinement continues. After the problem converges, we receive the results.

The difference between the two examples is the refinement algorithm. In the first example we apply h-refinement at the half of each NURBS element. In the second example we perform h-refinement at the half of specific NURBS elements. These specific elements are the elements that are near the edge that we have applied the non-zero Dirichlet boundary condition. In the case of "Example 2" we have chosen to perform refinement between the "up" edge and the axis parallel to x-axis that passes through the center of gravity.

[5] There is a typo in "if-statement" and the term is $\|T^{old} - T^{new}\|_{L1} < \varepsilon$

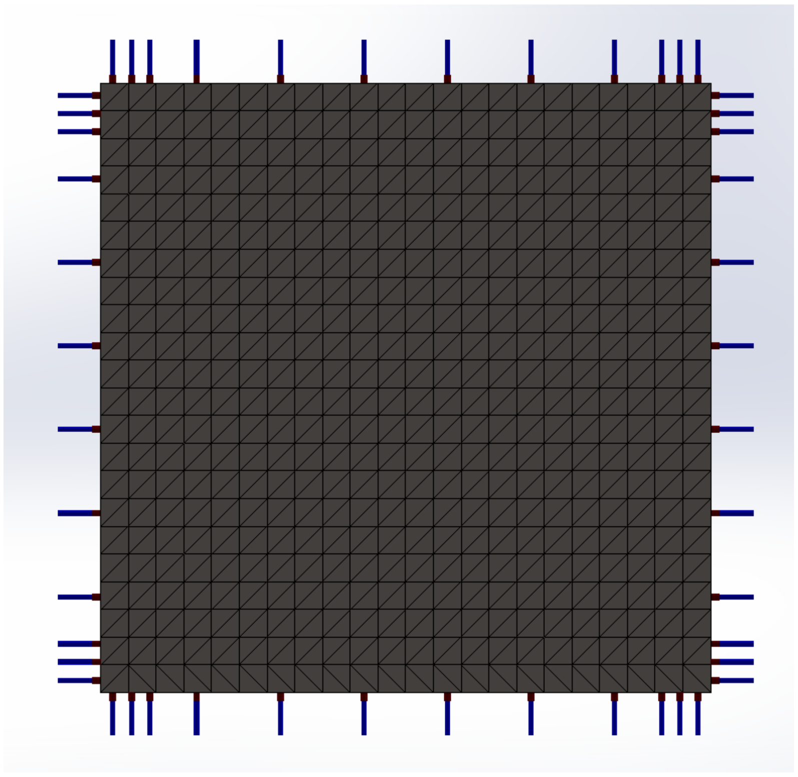


Figure 6.37: The mesh for the problem domain.

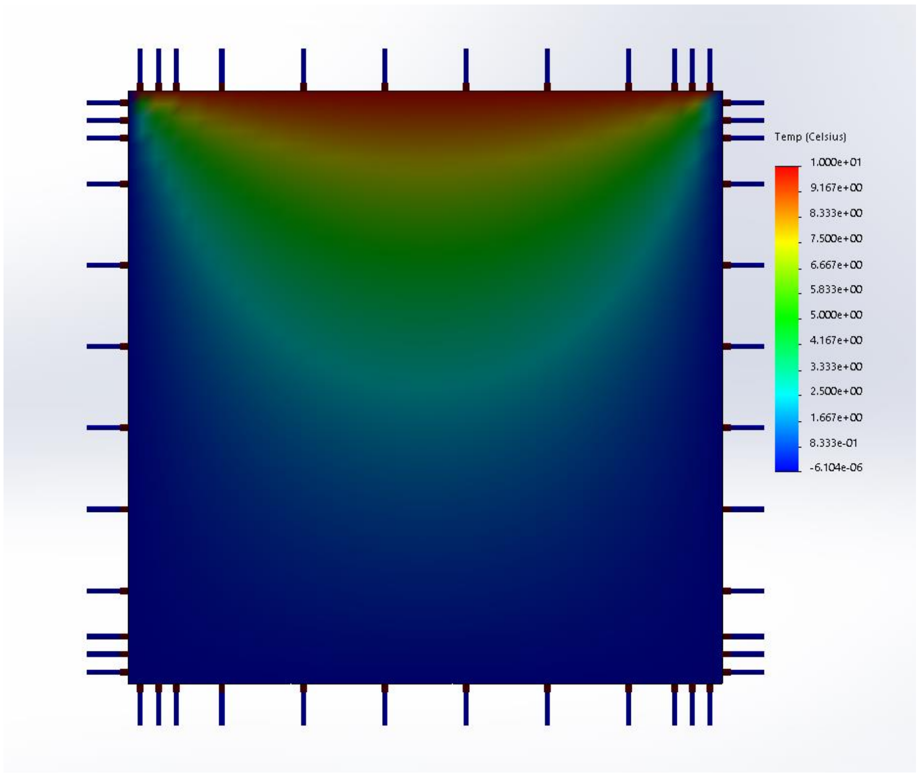


Figure 6.38:The distribution of temperature for thermal problem.

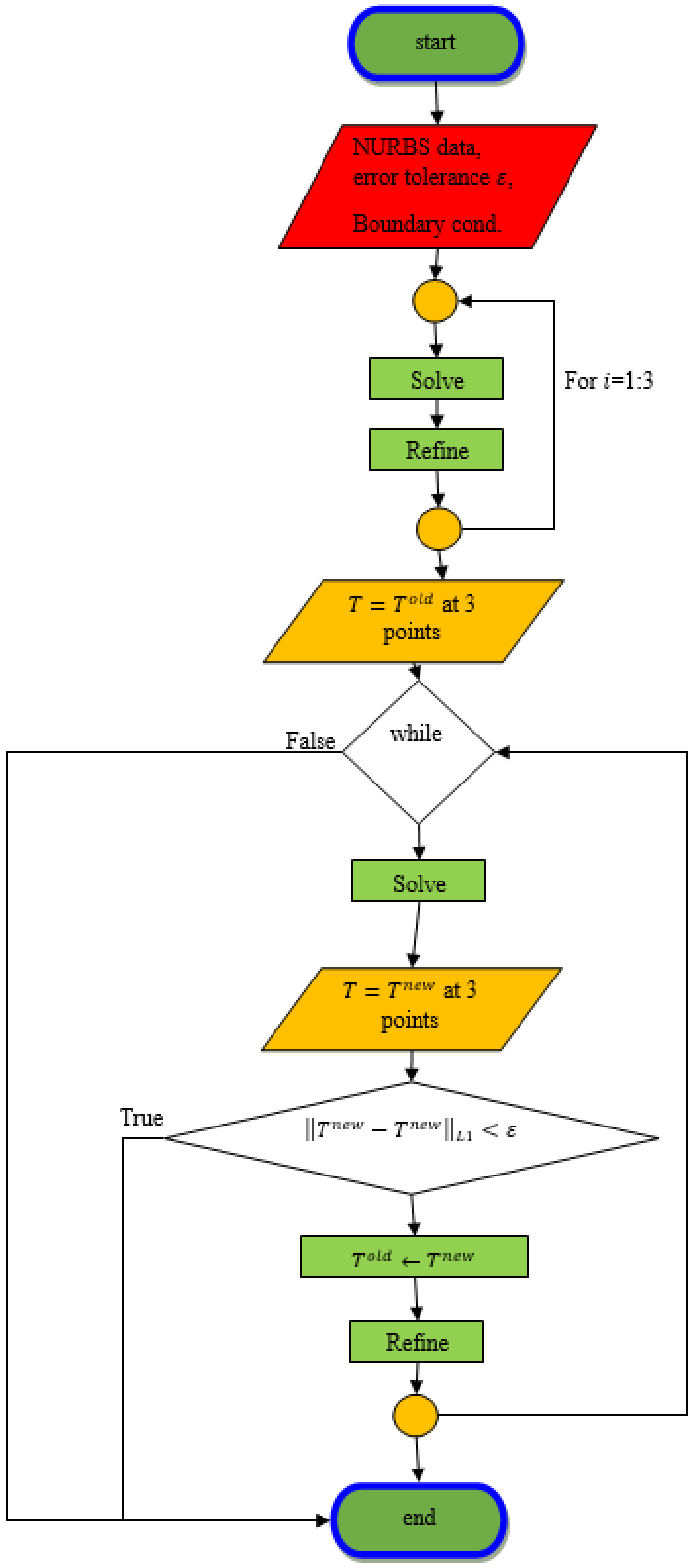
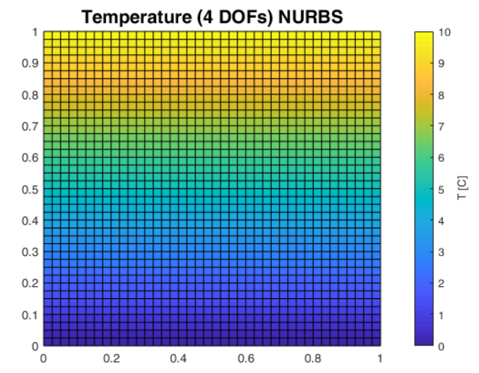
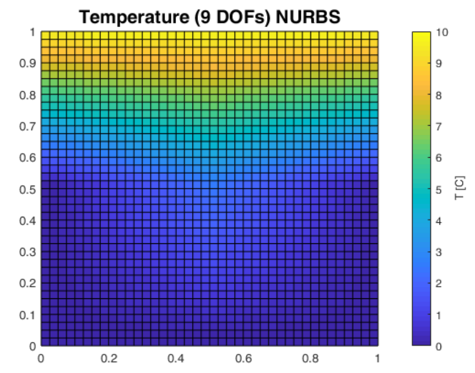

start
NURBS data,
error tolerance ε,
Boundary cond.
Solve
Refine
For i=1:3
$T = T^{old}$ at 3 points
while
False
Solve
$T = T^{new}$ at 3 points
True
$\|T^{new} - T^{new}\|_{L1} < \varepsilon$
$T^{old} \leftarrow T^{new}$
Refine
end

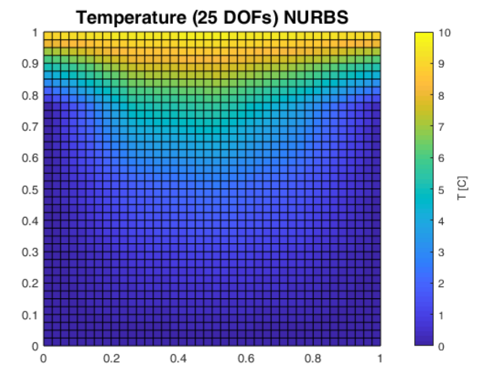
Figure 6.39: Flowchart of adaptivity code in I3GA.

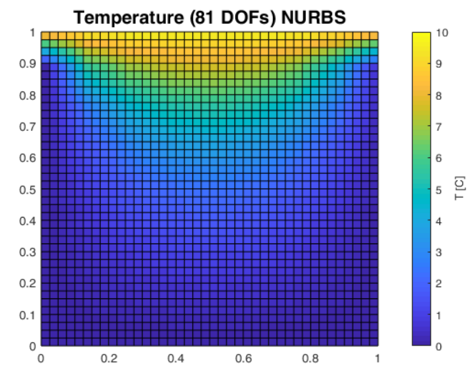
Finally, we have performed a study in order to see the effect of the surface degree in adaptivity. In this study we examined for each example, 4 adaptivity cases, in which the initial mesh has different degree of surface. We examined surfaces from degree equal to 1 up to 4. The distributions of temperature along with the final mesh for each example are following below.

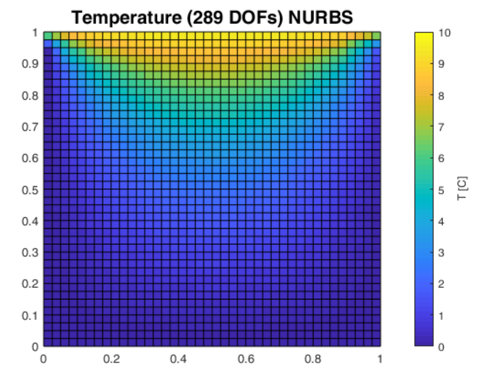
Example 1

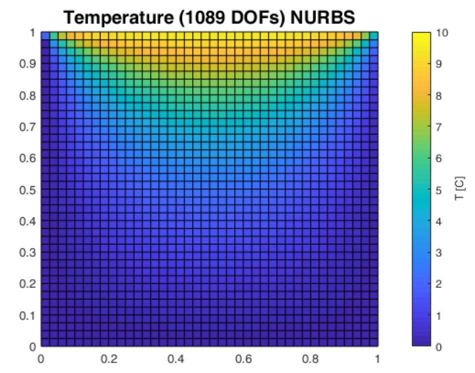












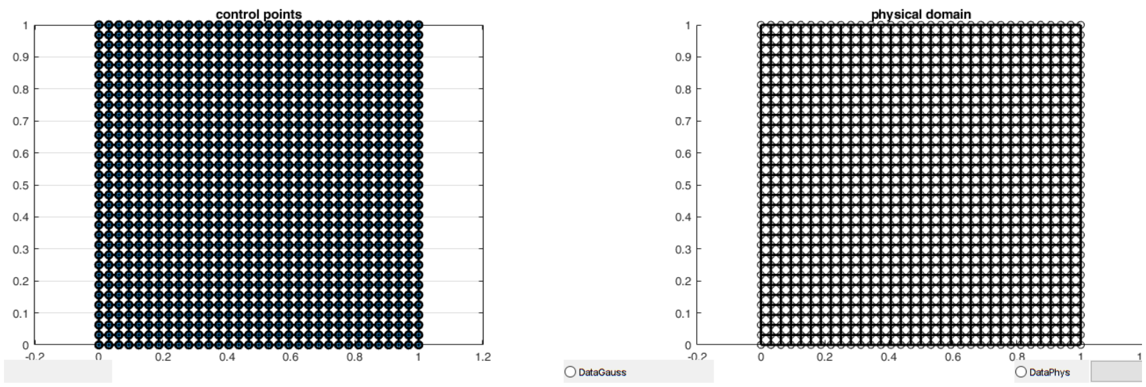
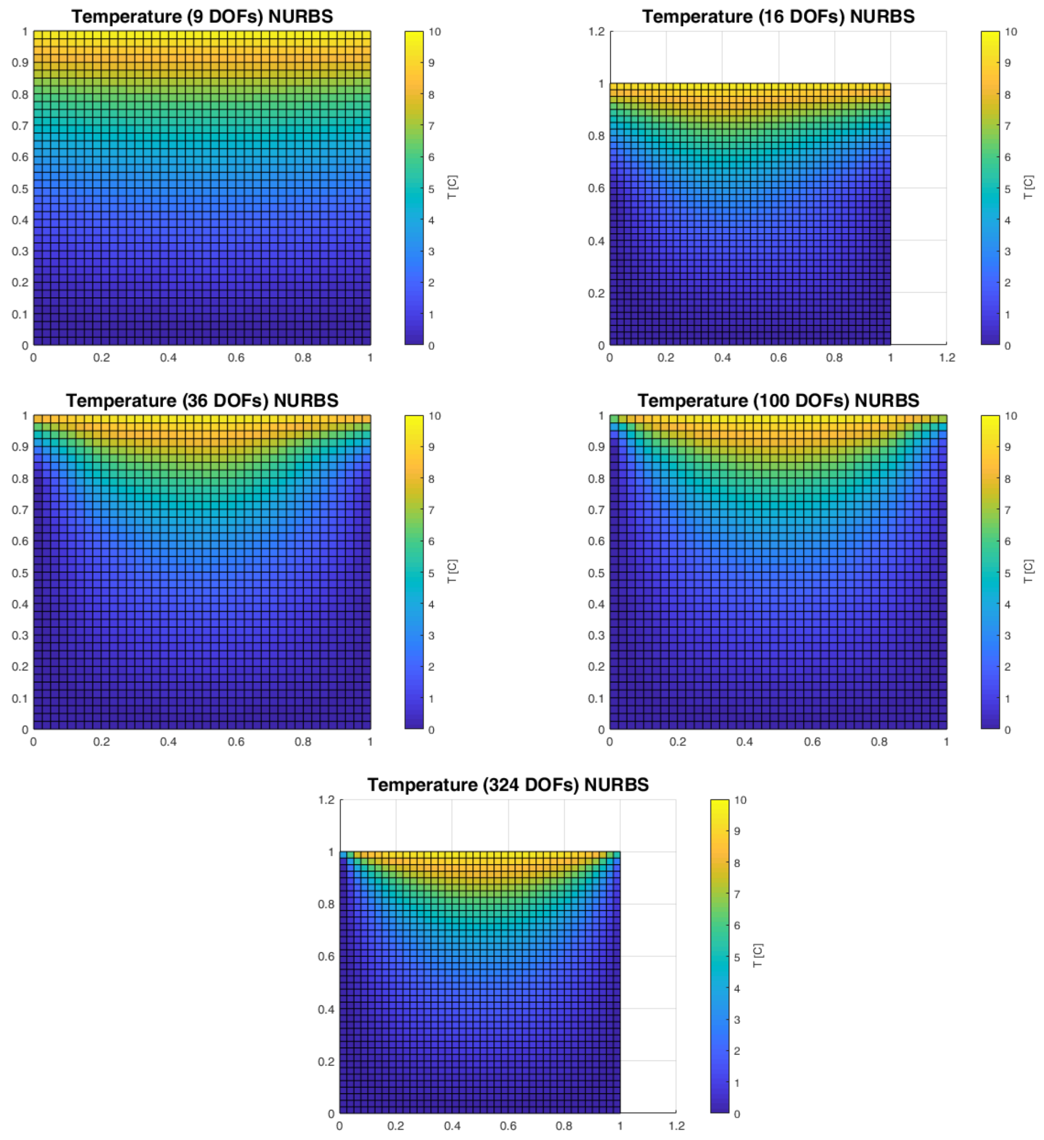


Figure 6.40: Temperature distributions from example 1 of adaptivity and final mesh for surface of degree 1.



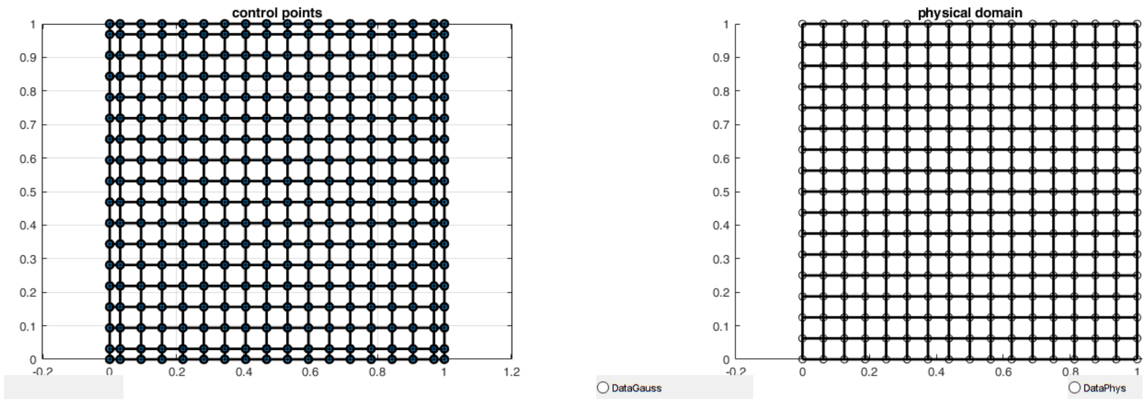
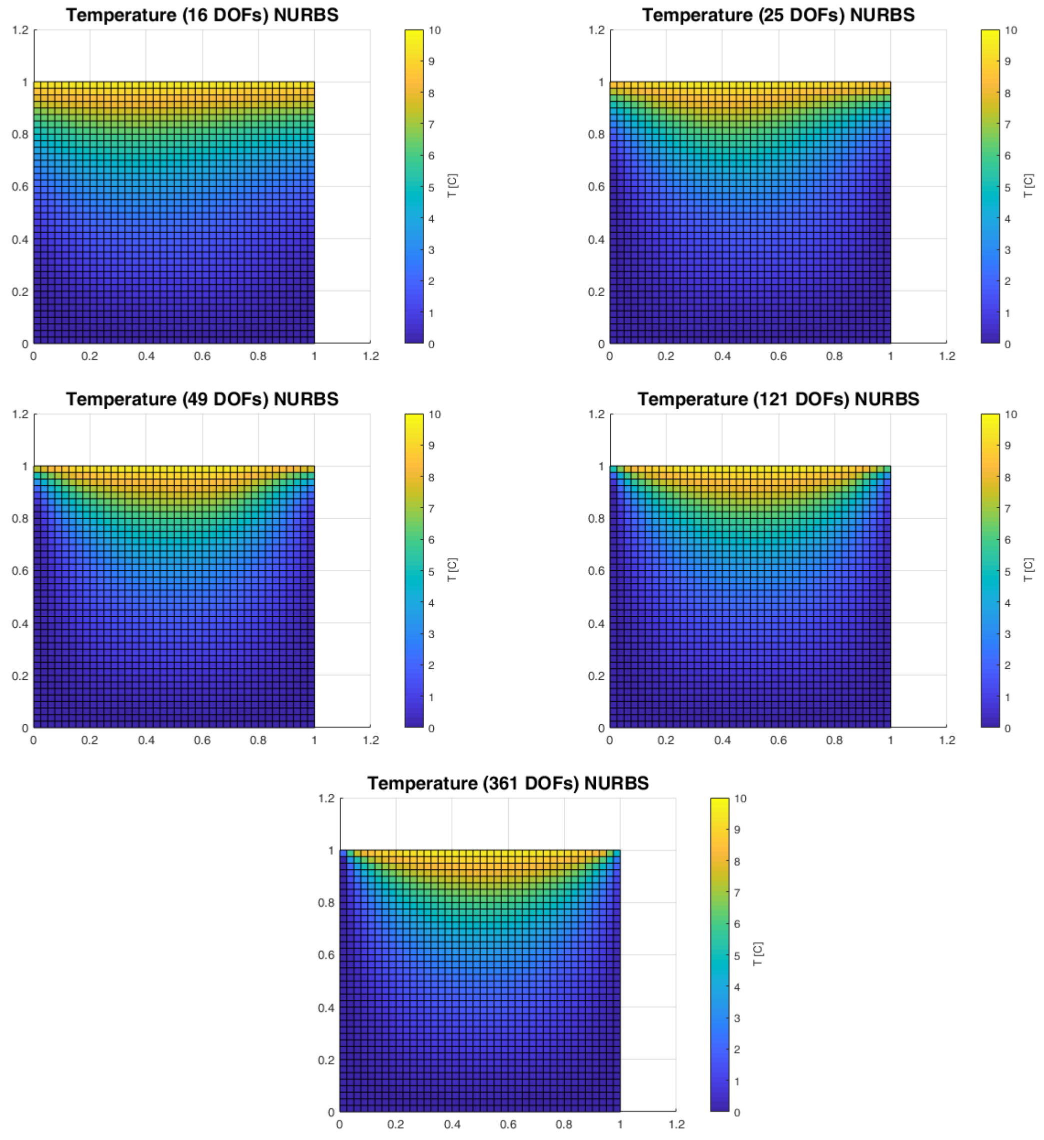


Figure 6.41: Temperature distributions from example 1 of adaptivity and final mesh for surface of degree 2.



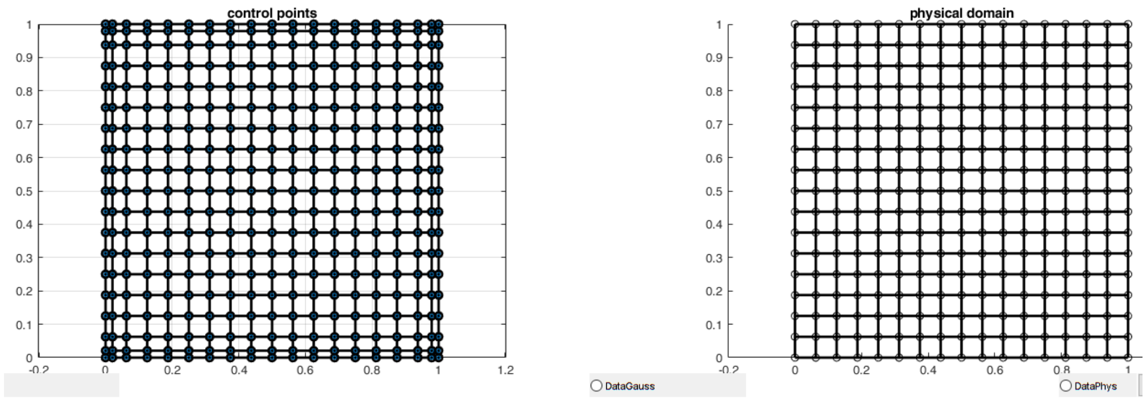
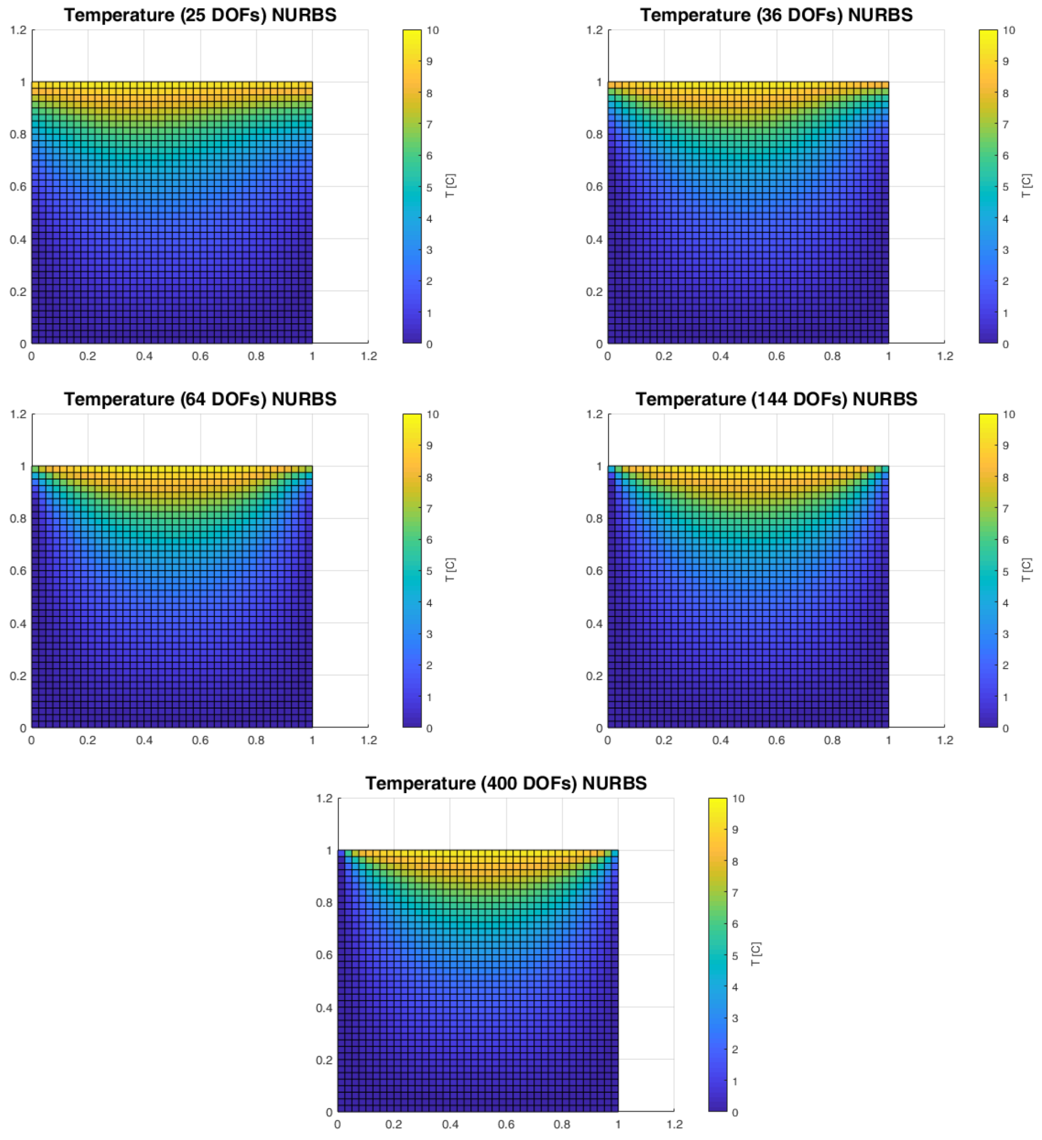


Figure 6.42: Temperature distributions from example 1 of adaptivity and final mesh for surface of degree 3.



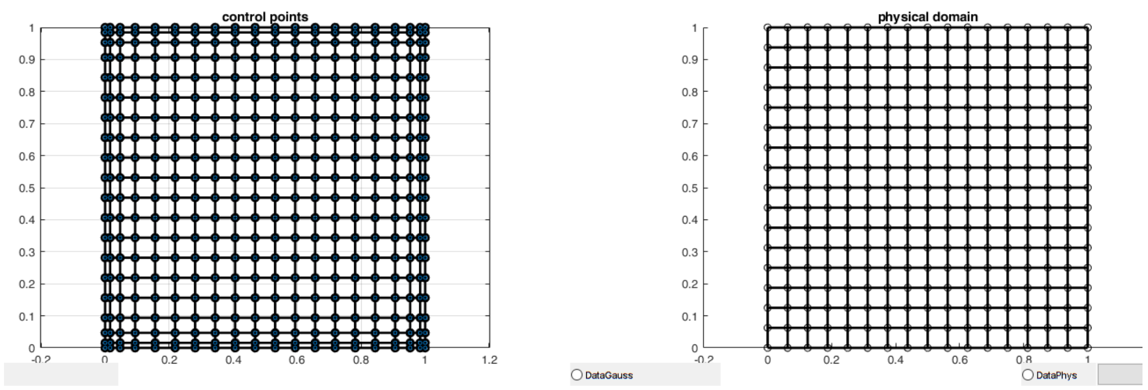
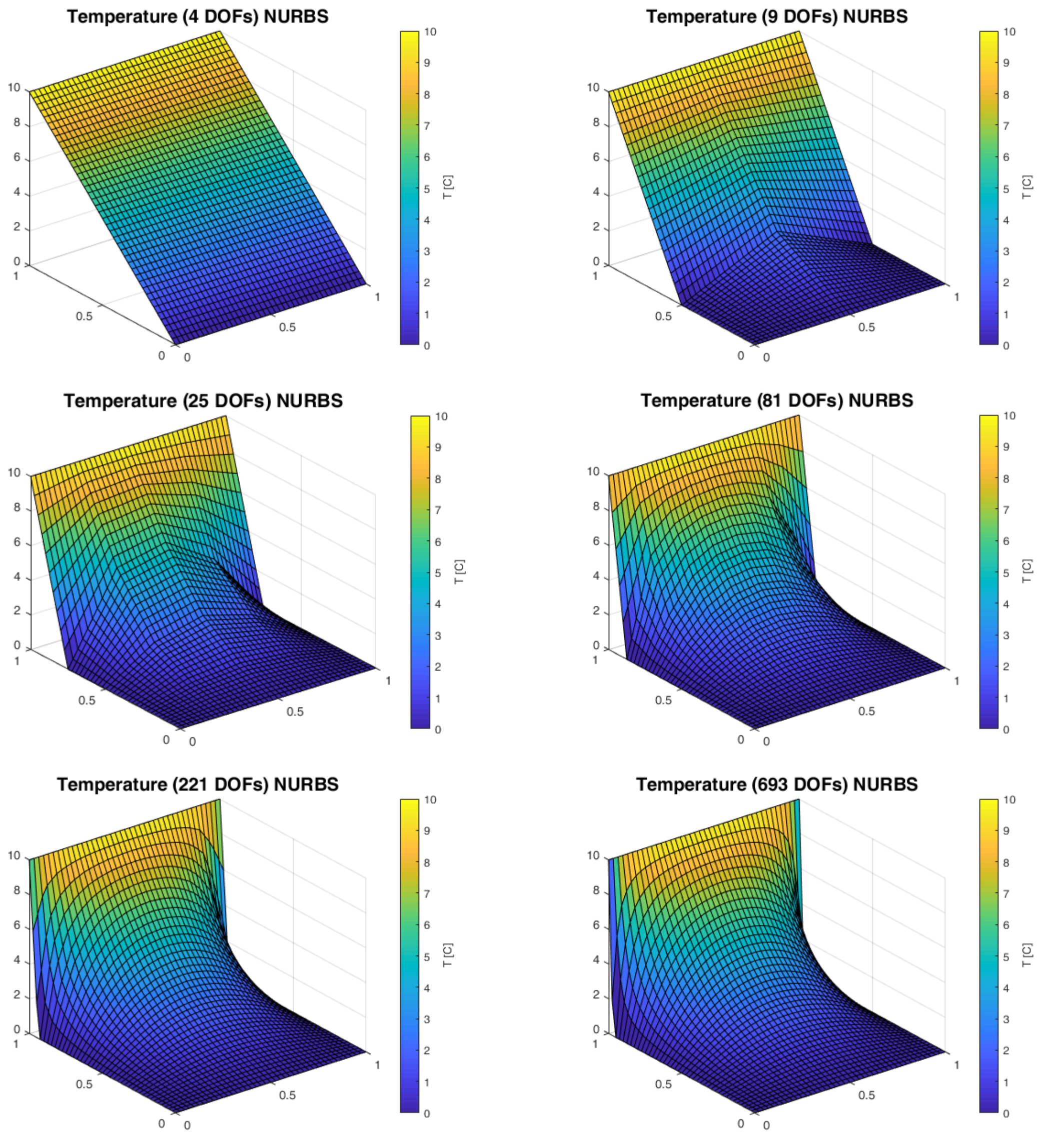


Figure 6.43: Temperature distributions from example 1 of adaptivity and final mesh for surface of degree 4.

Example 2



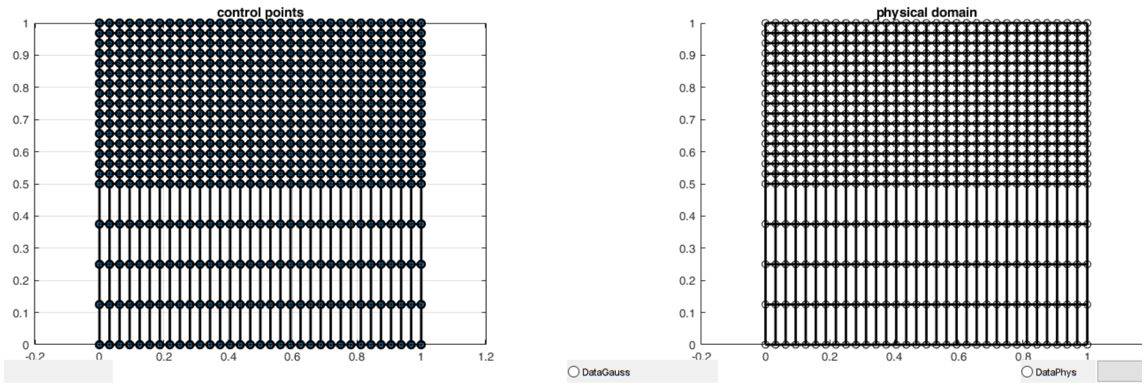
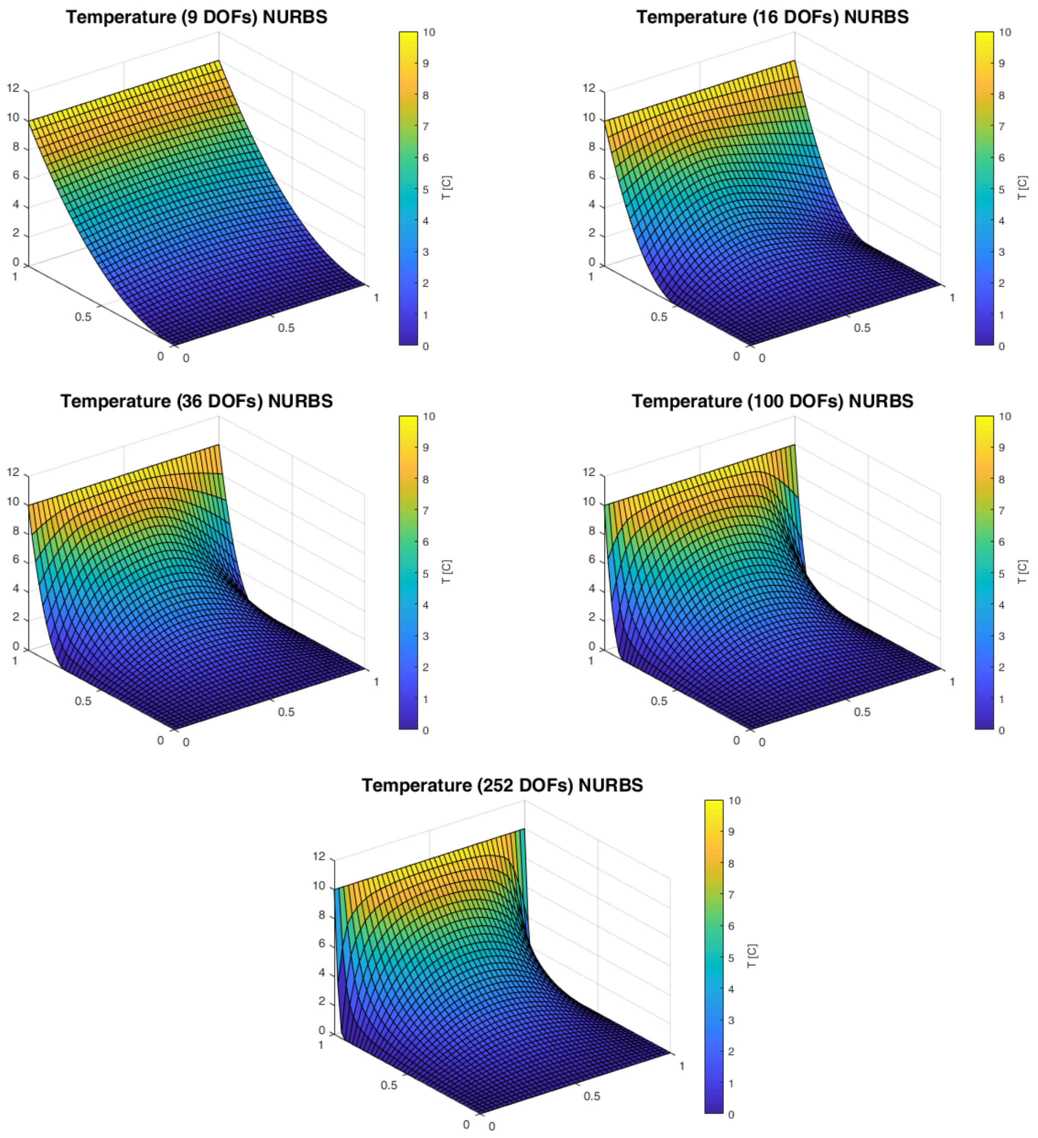


Figure 6.44: Temperature distributions from example 2 of adaptivity and final mesh for surface of degree 1.



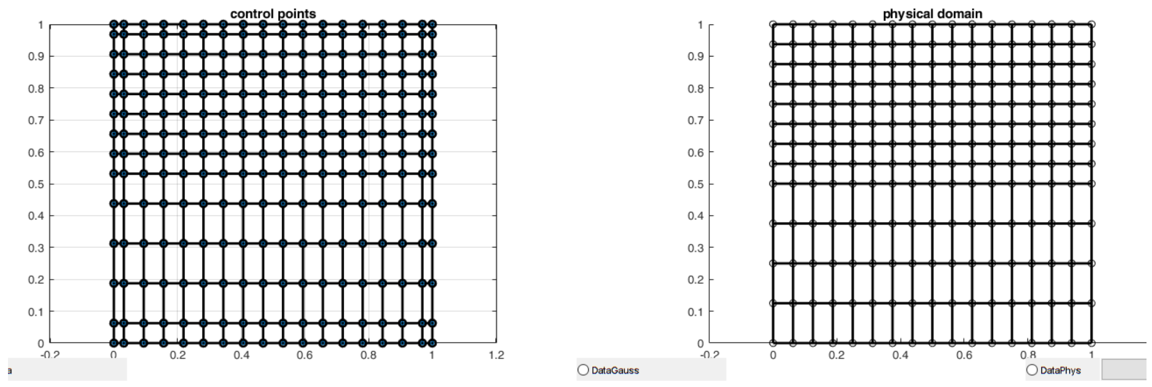
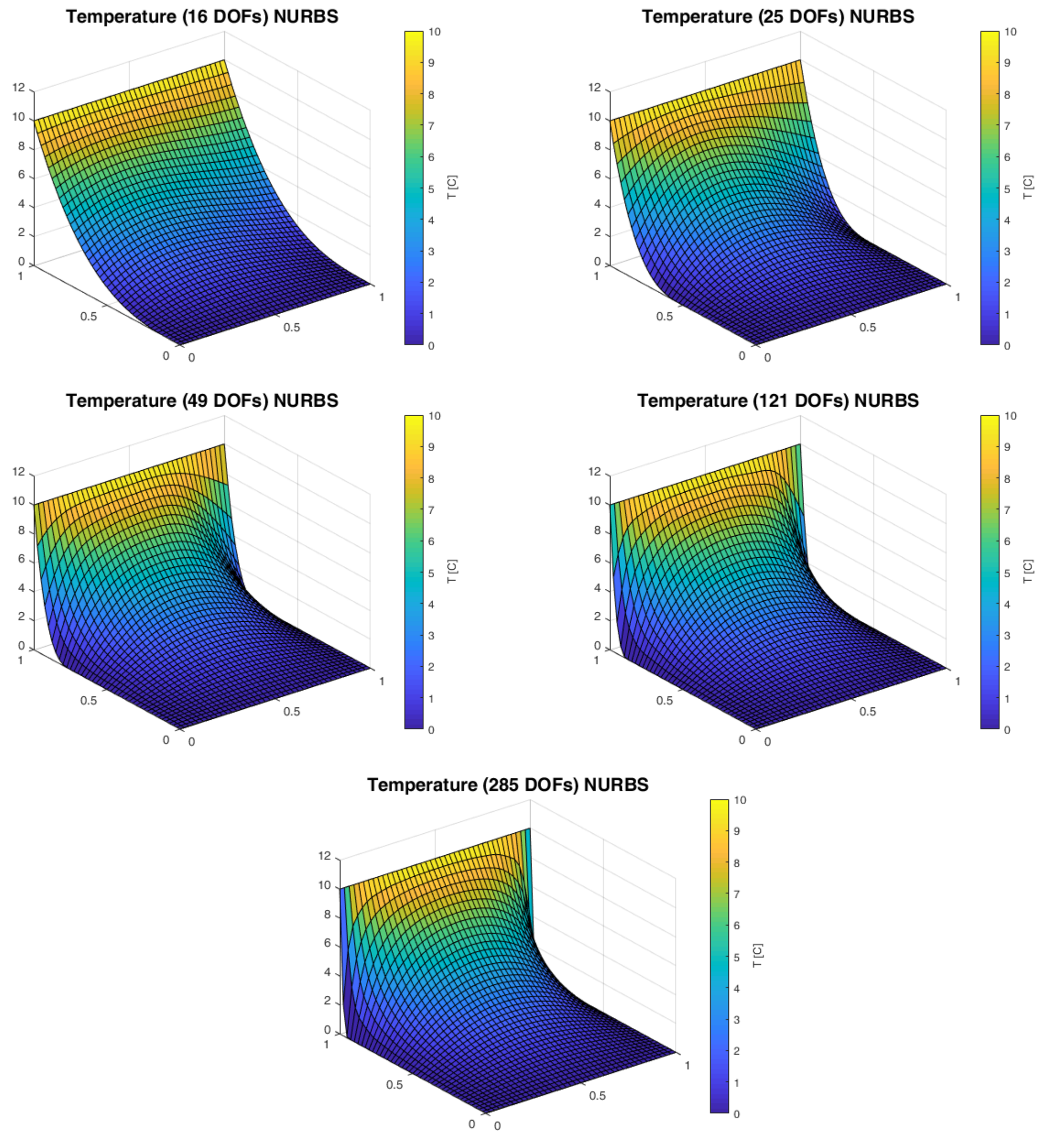


Figure 6.45: Temperature distributions from example 2 of adaptivity and final mesh for surface of degree 2.



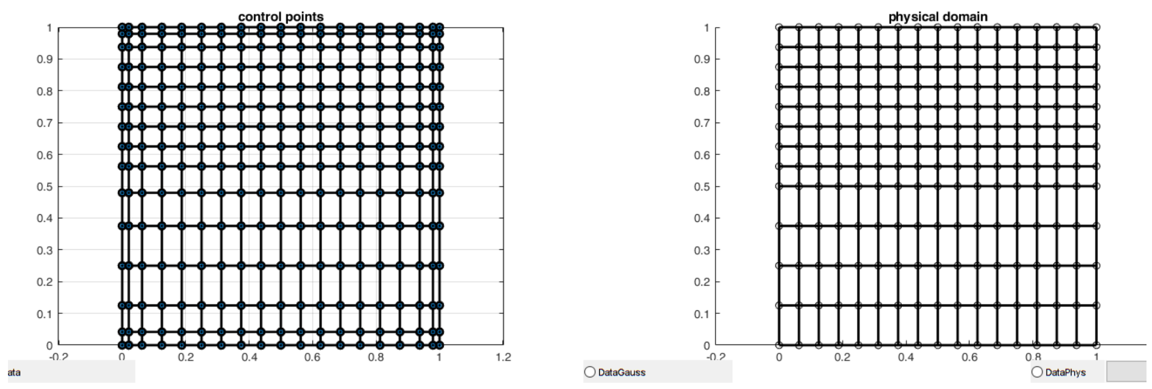
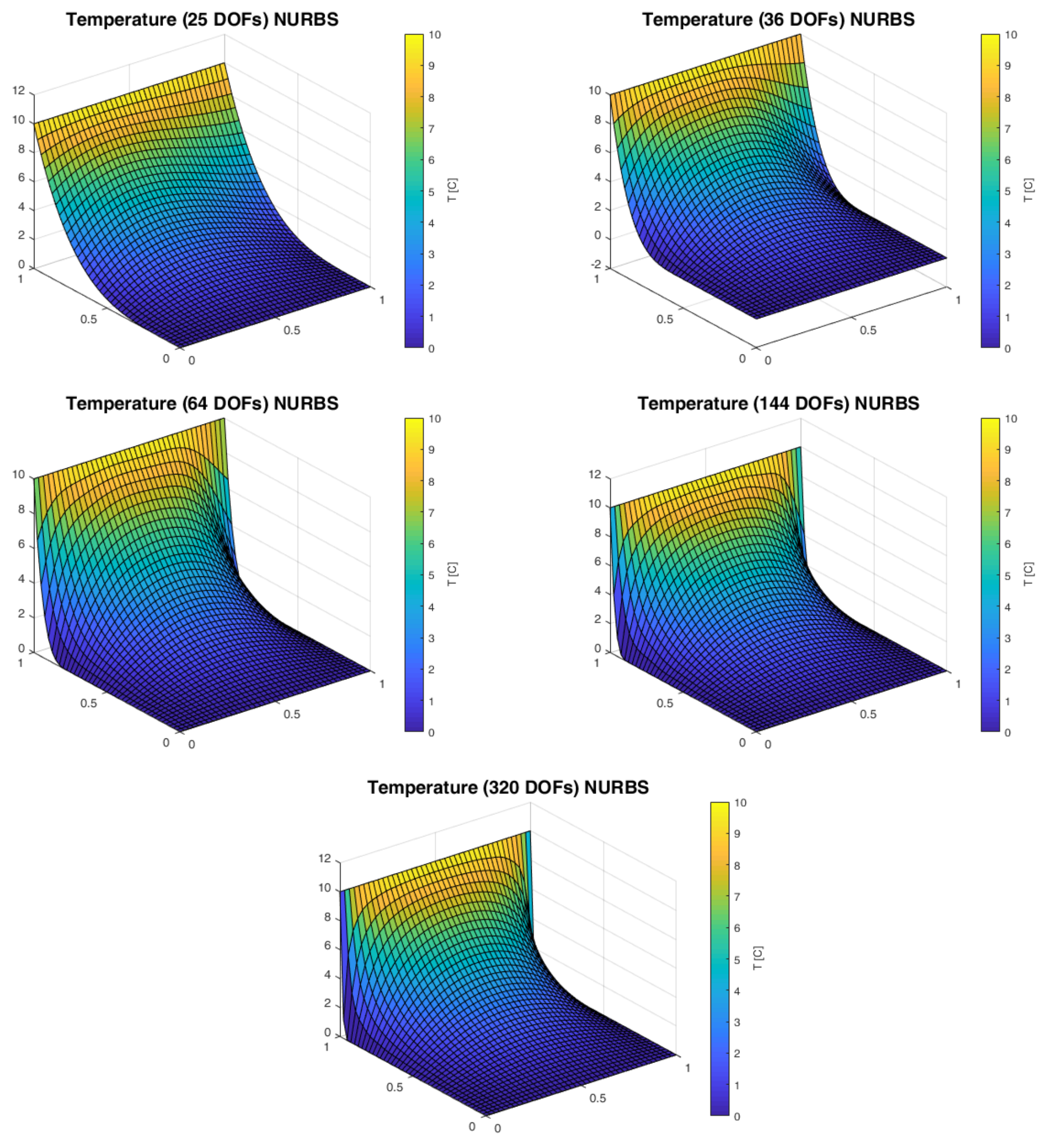


Figure 6.46: Temperature distributions from example 2 of adaptivity and final mesh for surface of degree 3.



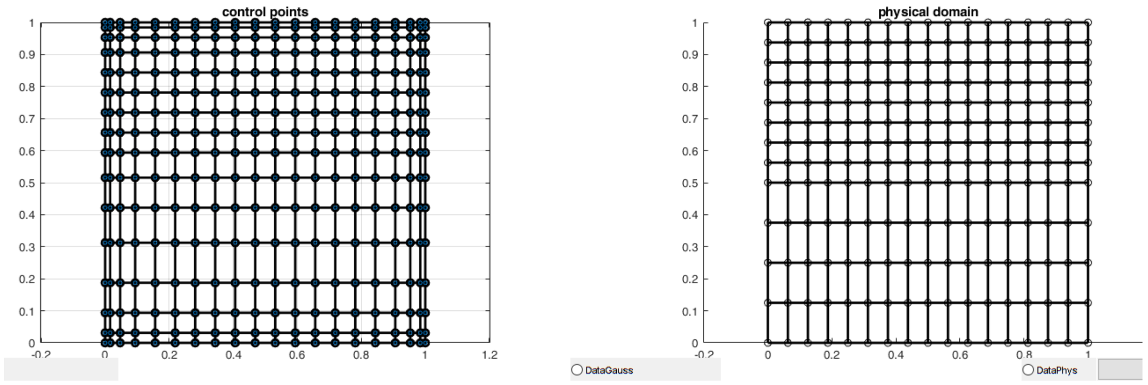


Figure 6.47: Temperature distributions from example 2 of adaptivity and final mesh for surface of degree 4.

We also show the effect of surface degree on execution time and the amount of refinement iterations during algorithm execution. Below two diagrams follow for the two examples mentioned.

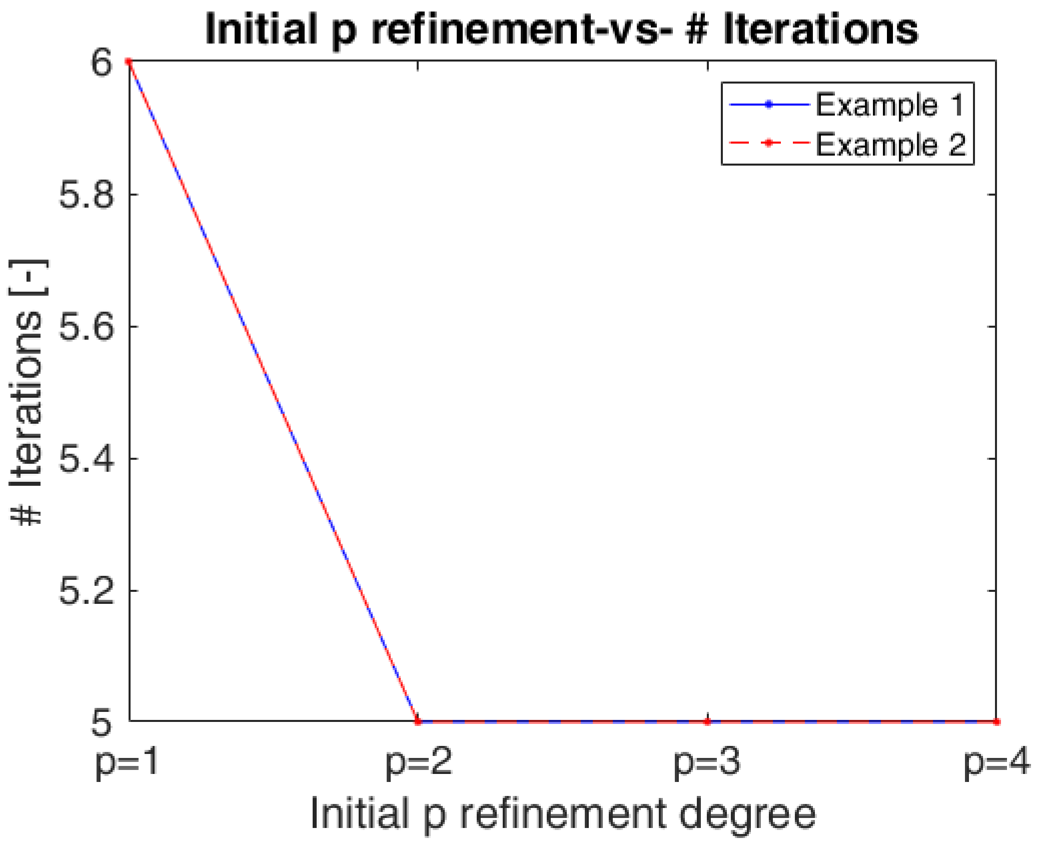


Figure 6.48:Total number of refinement iterations-vs-degree of surface.

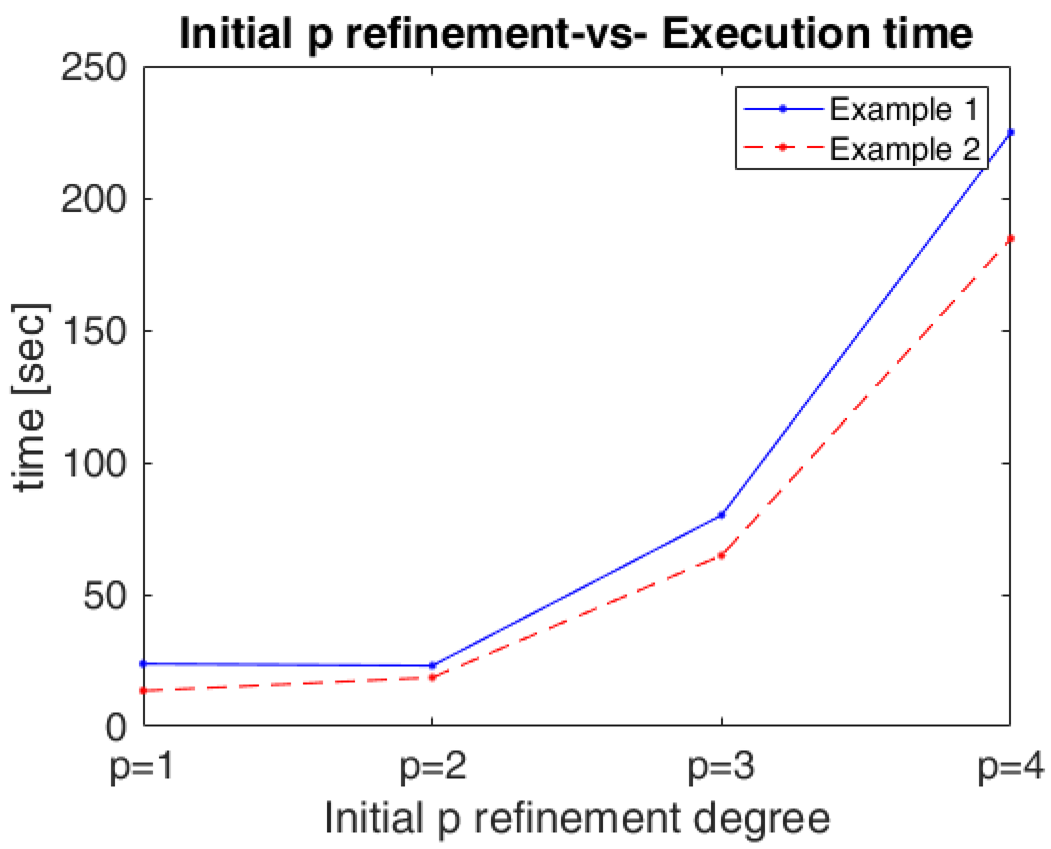


Figure 6.49: Execution time-vs-degree of surface.

In figures 6.40 to 6.47 we can see from both examples the evolution of temperature distribution with different DoFs. As long as the number of DoFs increases the results seems closer to the result from FEM. That is happening because with less NURBS elements we cannot achieve the desired accuracy. From the results we can conclude that the method (IGA) becomes more accurate over 300 DoFs approximately which are still less than the corresponding nodes in FEM mesh.

For every example, when we move on from surface degrees $p, q = 1$ to $p, q = 2$ then the final DoFs are decreasing. For instance, in "Example 1" we move on from 1089 to 324 DoFs and in "Example 2" we move on from 693 to 252 DoFs. However, as the surface degree increases the number of resulting DoFs is also increases. For instance, we see that in "Example 1" when $p, q = 4$ the DoFs are 400 and in "Example 2" they are 320. So, the increase in surface degree does not always result to a less coarse mesh. That happens only when we move from 1 to 2. Moreover, as long as the degree increases for $p, q \geq 3$ the execution time augments (see fig. 6.49). The needed time for "Example 2" is less than "Example 1" because the DoFs in "Example 2" are less. Although, the number of total knot refinement iterations is constant for $p, q \geq 2$ for both examples.

Finally, in every figure of the examples, we can see the final meshes. In "Example 2" the mesh is finer only to a specific region near the edge that is loaded in contrast to "Example 1" which is globally refines. The final meshes on both "Example 1" and "Example 2" for $p, q \geq 2$ are the same. The only difference exists in the control mesh where the control mesh is finer near the external boundary of the shape. That is happening because of the initial p-refinement before the h-refinements. With p-refinement the number of elements are not increased but only the number of control points. So, beginning from the same surface for $p, q \geq 2$, five h-refinements happen but with different starting number of control points.

The used adaptivity criterion of checking only the number of 3 values it is only a simple criterion in developing adaptivity algorithms. Also, other criteria as one mentioned have also been tested and many more can be included in the future. In future version of I3GA more complex and generic methods of Adaptivity will follow.

7 Concluding remarks & Future work

In this thesis we presented a fully integrated and interactive tool of Isogeometric analysis. We have presented in detail the used methods in refinement, knot refinement and degree elevation, the used Basis functions, the NURBS and used methods for analysis. In h-refinement we implemented our own Oslo algorithm that enables the multiple knot insertion in both parametric directions. We have seen the pure IGA formulation and the use of IGA with Bézier extraction and how problems are formulated in thermal and plane stress analysis. From the results of our examples, we have many interesting conclusions. Let us mention the effectiveness in accuracy of h and p-refinement, the effect of non-uniform boundary conditions and the internal or external sources to the problems. Finally, we examined a custom-made adaptivity formulation for a thermal problem and the effects of p-refinement in these examples.

Except for creating geometries in I3GA we have also developed algorithms that export the geometries to other formats as .iges and .iga (IGAFEM). Below we see an example of an exported I3GA geometry in Fusion 360. After opening it to another framework further manipulation can be achieved.

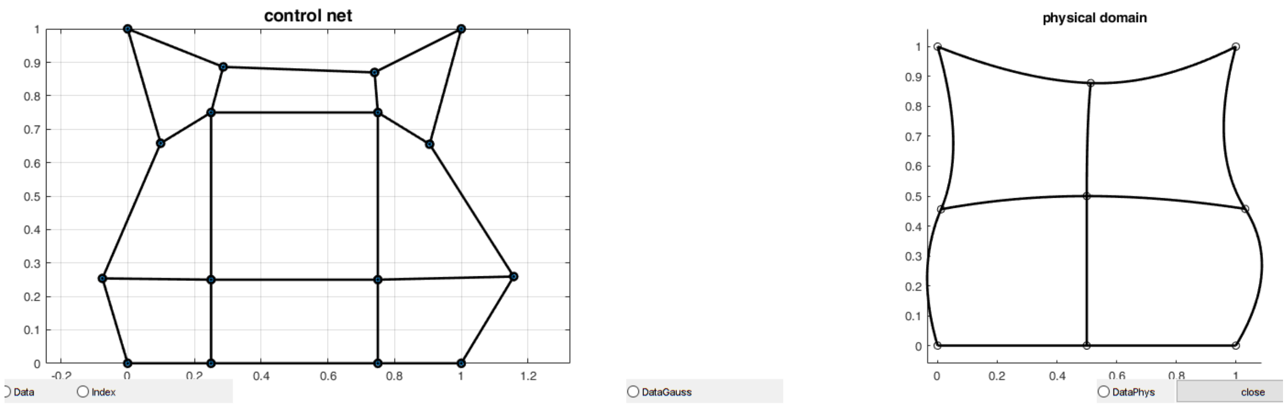


Figure 7.1: A free-hand geometry in I3GA.

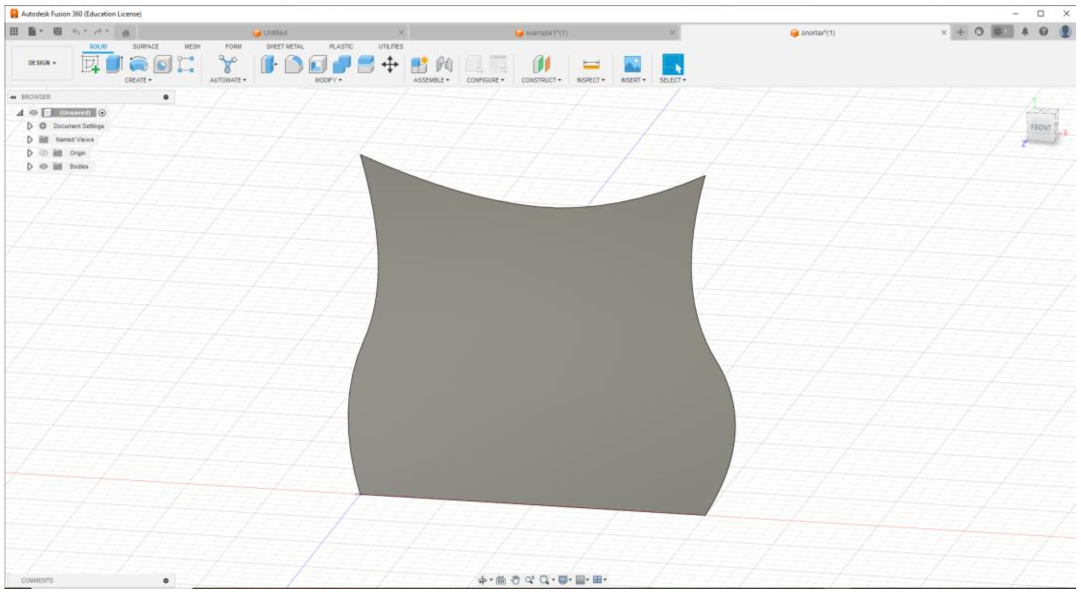


Figure 7.2: The above geometry in Fusion 360.

Someone could say that I3GA that is extension ISOGAT. The developed code has many differences from his precursor. Overall, the characteristics of I3GA that offer more control to engineer and student of IGA are listed here:

- I3GA offers a GUI with which the user can manipulate and design geometries freely. We can also move control points in specific locations or change their weights using this GUI. These capabilities do not exist in ISOGAT where the geometries are very specific and cannot be manipulated. Also, ISOGAT is not GUI oriented.
- I3GA has at least 6 refinement options and ISOGAT only 1. For h-refinement an alternative method has been programmed which is used in knot insertion both from parameter and physical space. In physical space the point insertion is interactive. Except for p-refinement we have also programmed p-refinement something that does not exist in ISOGAT.
- ISOGAT solves a specific form of thermal equation. I3GA solve a more generic form via Isogeometric analysis, where the materials are listed in a database. Moreover, in I3GA we solve thermal and plane stress equation via Bézier extraction method something that does not exist too in ISOGAT. Finally, we have programmed I3GA to take also inhomogeneous Dirichlet boundary conditions.
- With I3GA we can extract geometries to other formats in order to work with commercial packages as Autodesk's Fusion 360.

The above differences can be summarized in the table below:

Function	I3GA	ISOGAT
GUI with geometry manipulation	Yes	No
Free-hand geometry	Yes	No
Plotting geometry	Yes	Yes
Importing/Exporting/Saving geometries	Yes	No
H-refinement via element subdivision	Yes	Yes
H-refinement multiple knots	Yes	No
P-refinement	Yes	No
Insertion of point in physical space	Yes	No
Solving using NURBS with homogeneous boundary conditions	Yes	Yes
Solving using NURBS with inhomogeneous boundary conditions	Yes	No
Solving using NURBS with internal sources	Yes	Yes
Solving using NURBS with internal external	Yes	No
Solving using NURBS Bézier extraction	Yes	No

Table 7-1: Functions of I3GA and ISOGAT.

In general, I3GA offers an integrated tool for the education of IGA. However, many topics could evolve this tool. Initially I3GA could be programmed in other environments than MATLAB like in Python or C++ programming languages in order to make it even faster and opensource. We could change the kind of basis functions. LR-Bsplines [31] and T-splines are blending functions that offer the local refinement property and not global as in NURBS. Using them we could refine certain areas of the physical domain. Although a problem specifically in T-splines is difficult to program an efficient knot insertion algorithm. Furthermore, in this current form we could add some other adaptivity algorithms like the adaptivity algorithm using a posteriori error estimation. In steady heat conduction problems like in I3GA we could use adaptivity via error estimation based on temperature gradient recovery [30]. However, there are also other adaptivity techniques that could be tested using error estimation. Finally, we could also add in I3GA other problems that can be solved as acoustic problems that are based on Poisson equation.

8 List of Figures

9 List of Tables

10 References

[1] J. Wan *et al.*, “A One-dimensional Finite Element Method for Simulation-based Medical Planning for Cardiovascular Disease,” *Computer Methods in Biomechanics and Biomedical Engineering*, vol. 5, no. 3, pp. 195–206, Jan. 2002, doi: 10.1080/10255840290010670.

[2] C. G. Provatidis, *Precursors of Isogeometric Analysis: Finite Elements, Boundary Elements, and Collocation Methods*, vol. 256. in Solid Mechanics and Its Applications, vol. 256. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-030-03889-2.

[3] T. N. Nguyen, “Isogeometric Finite Element Analysis based on Bézier Extraction of NURBS and T-Splines”.

[4] Y. Bazilevs *et al.*, “Isogeometric analysis using T-splines,” *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 5–8, pp. 229–263, Jan. 2010, doi: 10.1016/j.cma.2009.02.036.

[5] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs, *ISOGEOMETRIC ANALYSIS TOWARD INTEGRATION OF CAD AND FEA*. Wiley, 2009.

[6] K. A. Johannessen, “An adaptive isogeometric finite element analysis”.

[7] “Reddit - Dive into anything.” Accessed: Sep. 19, 2023. [Online]. Available: https://embed.reddit.com/r/Python/comments/u6bcgc/why_do_people_still_pay_and_use_matlab_having/i57cwr0/?depth=2&showmore=false&embed=true&context=1&showmedia=false

[8] L. s n c S. I. per il T.- http://www.logovia.it, “SOLIDWORKS® 3D.” Accessed: Sep. 20, 2023. [Online]. Available: https://www.socoges.it/en/facilities/solidworks-3d.html

[9] R. M. & Associates, “Features,” www.rhino3d.com. Accessed: Sep. 20, 2023. [Online]. Available: https://www.rhino3d.com/features/

[10] “Maya Help | NURBS Modeling | Autodesk.” Accessed: Sep. 20, 2023. [Online]. Available: https://help.autodesk.com/view/MAYAUL/2024/ENU/?guid=GUID-735A0B9A-2180-4FB8-9A7B-68F21F306E97

[11] V. Agrawal and S. S. Gautam, “IGA: A Simplified Introduction and Implementation Details for Finite Element Users,” *J. Inst. Eng. India Ser. C*, vol. 100, no. 3, pp. 561–585, Jun. 2019, doi: 10.1007/s40032-018-0462-6.

[12] “igafem,” SourceForge. Accessed: Sep. 22, 2023. [Online]. Available: https://sourceforge.net/projects/cmcodes/

[13] A.-V. Vuong, Ch. Heinrich, and B. Simeon, “ISOGAT: A 2D tutorial MATLAB code for Isogeometric Analysis,” *Computer Aided Geometric Design*, vol. 27, no. 8, pp. 644–655, Nov. 2010, doi: 10.1016/j.cagd.2010.06.006.

[14] “Isogeometric Analysis is the next generation of FEA.” Accessed: Sep. 22, 2023. [Online]. Available: https://coreform.com/products/coreform-iga/

[15] D. C. Thomas, L. Engvall, S. K. Schmidt, K. Tew, and M. A. Scott, “U-splines: Splines over unstructured meshes,” *Computer Methods in Applied Mechanics and Engineering*, vol. 401, p. 115515, Nov. 2022, doi: 10.1016/j.cma.2022.115515.

[16] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs, “Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement,” *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 39–41, pp. 4135–4195, Oct. 2005, doi: 10.1016/j.cma.2004.10.008.

[17] O. C. Zienkiewicz and R. L. Taylor, *The finite element method*, 5th ed. Oxford ; Boston: Butterworth-Heinemann, 2000.

[18] Χ. Γ. Προβατίδης, *ΠΕΠΕΡΑΣΜΕΝΑ ΣΤΟΙΧΕΙΑ ΣΤΗΝ ΑΝΑΛΥΣΗ ΜΗΧΑΝΟΛΟΓΙΚΩΝ ΚΑΤΑΣΚΕΥΩΝ*, 2η Έκδοση. ΕΚΔΟΣΕΙΣ ΤΖΙΟΛΑ.

[19] C. Provatidis, “Free vibration analysis of elastic rods using global collocation,” *Archive of Applied Mechanics*, vol. 78, pp. 241–250, Apr. 2008, doi: 10.1007/s00419-007-0159-4.

[20] “Engineering at Alberta Courses » Isoparametric Elements.” Accessed: Jan. 14, 2023. [Online]. Available: https://engcourses-uofa.ca/books/introduction-to-solid-mechanics/finite-element-analysis/one-and-two-dimensional-isoparametric-elements-and-gauss-integration/isoparametric-elements/

[21] D. L. Logan, *A first course in the finite element method*, 5th ed. Stamford, CT: Cengage Learning, 2012.

[22] W. A. Wall, M. A. Frenzel, and C. Cyron, “Isogeometric structural shape optimization,” *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 33, pp. 2976–2988, Jun. 2008, doi: 10.1016/j.cma.2008.01.025.

[23] L. De Lorenzis, P. Wriggers, and T. J. R. Hughes, “Isogeometric contact: a review,” *GAMM-Mitteilungen*, vol. 37, no. 1, pp. 85–123, 2014, doi: 10.1002/gamm.201410005.
[24] L. Piegl and W. Tiller, *The NURBS Book*, Second. Springer-Verlag Berlin Heidelberg.
[25] D. F. Rogers, *An introduction to NURBS: with historical perspective*. San Francisco: Morgan Kaufmann Publishers, 2001.
[26] Δ. Καρράς, “Ιεραρχική Βελτίωση της Προσωμοίωσης με τη μέθοδο της Ισογεωμετρικής Ανάλυσης,” Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, 2015.
[27] Π. Ζησιμοπούλου, “Αποσύνθεση Bézier για παρεμβολή T-spline σε διδιάστατα προβλήματα πεδίου,” Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, 2022.
[28] T. Lyche and K. Mørken, “Making the Oslo Algorithm More Efficient,” *SIAM J. Numer. Anal.*, vol. 23, no. 3, pp. 663–675, Jun. 1986, doi: 10.1137/0723042.
[29] M. J. Borden, M. A. Scott, J. A. Evans, and T. J. R. Hughes, “Isogeometric finite element data structures based on Bézier extraction of NURBS,” *Numerical Meth Engineering*, vol. 87, no. 1–5, pp. 15–47, Jul. 2011, doi: 10.1002/nme.2968.
[30] T. Yu, B. Chen, S. Natarajan, and T. Q. Bui, “A locally refined adaptive isogeometric analysis for steady-state heat conduction problems,” *Engineering Analysis with Boundary Elements*, vol. 117, pp. 119–131, Aug. 2020, doi: 10.1016/j.enganabound.2020.05.005.
[31] T. Dokken, T. Lyche, and K. F. Pettersen, “Polynomial splines over locally refined box-partitions,” *Computer Aided Geometric Design*, vol. 30, no. 3, pp. 331–356, Mar. 2013, doi: 10.1016/j.cagd.2012.12.005.

Appendix A

A.1 Example of FEM formulation using Galerkin method

In this subsection an elasticity problem is formulated using Galerkin method. The problem is based the exercise 10.1 (pg.462) from [18], even though the formulation will be more generic than the exercise. Let an elastic beam with Young's Modulus E, surface A and length L, density ρ rotated with constant angular velocity ω about a joint at position $x = 0$ and with the other edge $(x = L)$ being free. In fig. A.1 the beam is loaded with constant stress σ_L. The purpose of this problem is to find the axial displacement u of free edge.

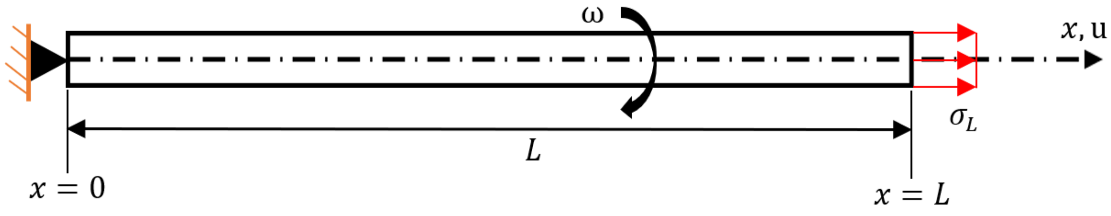


Figure A.1: The articulated beam rotating about $x = 0$ with constant angular velocity ω. Source: [18]

The differential equation describing the problem is:

$$D(\mathrm{u}(x)) = E\frac{\partial^2 \mathrm{u}}{\partial x^2} + \rho\omega^2 x = 0, \quad x \in (0, L) \tag{A.1}$$

And boundary conditions are:

$$\mathbf{B}(\mathrm{u}) = \begin{cases} \mathrm{u} = 0, \ \ x = 0 \\ E\dfrac{\partial \mathrm{u}}{\partial x} - \sigma_L = 0, \ \ x = L \end{cases} \tag{A.2}$$

In this problem we have both Dirichlet and Neumann BCs. Although the problem has exact solution[6] we will formulate it using Finite Element method. Let suppose that our mesh consists of only one element with nodes at $x = 0$ and $x = L$ (positions 1 and 2 respectively). Using notation from chapter 2.1 the distribution of axial displacement u (the unknown state variable of problem) that satisfy BCs can be approximated as:

$$(2.1) \Rightarrow \mathrm{u} \approx \tilde{\mathrm{u}} = a_1 N_1 + a_2 N_2 \tag{A.3}$$

where, $N_i = N_i(x)$ are suitable shape functions (i.e., Taylor mononyms: $N_1 = x$, $N_2 = x^2$) and a_i are the unknown nodal values at position 1 and 2. From eq. (2.6) for one element the weak form of the system using Galerkin method is:

$$(2.6) \overset{1-D}{\Longrightarrow} \int_\Omega N_i D(\tilde{\mathrm{u}}) \mathrm{d}\Omega + \int_\Gamma N_i B(\tilde{\mathrm{u}}) \mathrm{d}\Gamma \quad j = 1 \text{ to } 2 \tag{A.4}$$

The problem domain Ω is the beam with length L and area A, so it consists of elementary volumes $\mathrm{d}\Omega \equiv dV = A\mathrm{d}x$. The boundary Γ respectively consists of the edges of beam and particularly $x = L$ where the BC exists. So, the eq. (A.4) is written as:

[6]Exact solution: $\mathrm{u} = \frac{\rho\omega^2}{6E}(3L^2 x - x^3) + \frac{\sigma_L x}{E}$

$$(\mathrm{A.4}) \Rightarrow \int_0^L N_i D(\tilde{u}) A\mathrm{d}x + A[N_i B(\tilde{u})]_{x=L} = 0 \xRightarrow{(\mathrm{A.1})+(\mathrm{A.2})}$$

$$\Rightarrow \int_0^L N_i \left(E\frac{\partial^2\tilde{u}}{\partial x^2} + \rho\omega^2 x\right) A\mathrm{d}x + A\left[N_i\left(E\frac{\partial\tilde{u}}{\partial x} - \sigma_L\right)\right]_{x=L} = 0 \Rightarrow$$

$$\Rightarrow \int_0^L N_i \left(E\frac{\partial^2\tilde{u}}{\partial x^2} + \rho\omega^2 x\right) A\mathrm{d}x - A\left[N_i\left(E\frac{\partial\tilde{u}}{\partial x} - \sigma_L\right)\right]_{x=L} = 0 \qquad (\mathrm{A.5})$$

The substitution of "+" with "–" symbol means that forces and stresses execute positive work during a displacement. So, eq. $(\mathrm{A.5})$ continues as:

$$(\mathrm{A.5}) \Rightarrow \underbrace{\int_0^L N_i EA\frac{\partial^2\tilde{u}}{\partial x^2}\mathrm{d}x}_{\left[EAN_i\frac{\partial\tilde{u}}{\partial x}\right]_0^L - \int_0^L EA\frac{\partial N_i}{\partial x}\frac{\partial\tilde{u}}{\partial x}\mathrm{d}x} + \int_0^L N_i\rho\omega^2 xA\mathrm{d}x - A\left[N_i E\frac{\partial\tilde{u}}{\partial x}\right]_{x=L} + A[N_i\sigma_L]_{x=L} = 0 \Leftrightarrow$$

$$\Leftrightarrow -\left[EAN_i\frac{\partial\tilde{u}}{\partial x}\right]_{x=0} + \int_0^L N_i\rho\omega^2 xA - EA\frac{\partial N_i}{\partial x}\frac{\partial\tilde{u}}{\partial x}\mathrm{d}x + A[N_i\sigma_L]_{x=L} = 0 \xRightarrow{(\mathrm{A.3})\Rightarrow\frac{\partial\tilde{u}}{\partial x}=\frac{\partial N_1}{\partial x}a_1+\frac{\partial N_2}{\partial x}a_2}$$

$$\Leftrightarrow -EA\left[N_i\left(\frac{\partial N_1}{\partial x}a_1 + \frac{\partial N_2}{\partial x}a_2\right)\right]_{x=0} + A\int_0^L N_i\rho\omega^2 x - E\frac{\partial N_i}{\partial x}\left(\frac{\partial N_1}{\partial x}a_1 + \frac{\partial N_2}{\partial x}a_2\right)\mathrm{d}x + A[N_i\sigma_L]_{x=L} = 0 \Leftrightarrow$$

$$\Leftrightarrow -EA\left[\left[N_i\frac{\partial N_1}{\partial x}\right]_{x=0}a_1 + \left[N_i\frac{\partial N_2}{\partial x}\right]_{x=0}a_2\right] + A\int_0^L N_i\rho\omega^2 x\mathrm{d}x - A\int_0^L E\frac{\partial N_i}{\partial x}\frac{\partial N_1}{\partial x}a_1\mathrm{d}x$$
$$- A\int_0^L E\frac{\partial N_i}{\partial x}\frac{\partial N_2}{\partial x}a_2\mathrm{d}x + A[N_i\sigma_L]_{x=L} = 0 \Leftrightarrow$$

$$\Leftrightarrow \left[EA\left[\left[N_i\frac{\partial N_1}{\partial x}\right]_{x=0} + \int_0^L E\frac{\partial N_i}{\partial x}\frac{\partial N_1}{\partial x}\mathrm{d}x\right]\right]a_1 + \left[EA\left[\left[N_i\frac{\partial N_2}{\partial x}\right]_{x=0} + \int_0^L E\frac{\partial N_i}{\partial x}\frac{\partial N_2}{\partial x}\mathrm{d}x\right]\right]a_2$$
$$= A\int_0^L N_i\rho\omega^2 x\mathrm{d}x + A[N_i\sigma_L]_{x=L} \xRightarrow{i=1,2}$$

$$\Rightarrow \underbrace{\begin{bmatrix} EA\left[\left[N_1\frac{\partial N_1}{\partial x}\right]_{x=0} + \int_0^L E\frac{\partial N_1}{\partial x}\frac{\partial N_1}{\partial x}\mathrm{d}x\right] & EA\left[\left[N_1\frac{\partial N_2}{\partial x}\right]_{x=0} + \int_0^L E\frac{\partial N_1}{\partial x}\frac{\partial N_2}{\partial x}\mathrm{d}x\right] \\ EA\left[\left[N_2\frac{\partial N_1}{\partial x}\right]_{x=0} + \int_0^L E\frac{\partial N_2}{\partial x}\frac{\partial N_1}{\partial x}\mathrm{d}x\right] & EA\left[\left[N_2\frac{\partial N_2}{\partial x}\right]_{x=0} + \int_0^L E\frac{\partial N_2}{\partial x}\frac{\partial N_2}{\partial x}\mathrm{d}x\right] \end{bmatrix}}_{\mathbf{K}_{\mathrm{Global}}} \underbrace{\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}}_{\mathbf{a}}$$

$$= \underbrace{\begin{bmatrix} A\int_0^L N_1\rho\omega^2 x\mathrm{d}x + A[N_1\sigma_L]_{x=L} \\ A\int_0^L N_2\rho\omega^2 x\mathrm{d}x + A[N_2\sigma_L]_{x=L} \end{bmatrix}}_{\mathbf{f}} \overset{(2.6)}{\Longleftrightarrow}$$

$$\Leftrightarrow \mathbf{K}_{\mathrm{Global}}\mathbf{a} = \mathbf{f} \qquad (\mathrm{A.6})$$

With eq. (A.6) by choosing proper N_1,N_2 we can derive a system of linear equations about a_1, a_2 in order to have an approximate solution. If we choose from original example, $N_1 = x, N_2 = x^2$ then the nodal values will be:

$$a_1 = \frac{\sigma_L}{E} + \frac{7\rho\omega^2 L}{12E} \quad a_2 = -\frac{\rho\omega^2 L}{4E} \tag{A.7}$$

A.2 Example of isoparametric FEM formulation using Lagrange Shape Functions

In this subsection we will present an example of isoparametric element formulation of plane (2-D) stress problem using Q4 elements which are derived from Lagrange shape functions. The example is taken from [3]. However, let first consider some preliminaries on mechanics. Considering that we work on linear elasticity region the stress-strain relationship is given by [17]:

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \mathbf{D}(\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_0) + \boldsymbol{\sigma}_0 \Leftrightarrow$$

$$\Leftrightarrow \boldsymbol{\sigma} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1}{2}(1-\nu) \end{bmatrix} (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_0) + \boldsymbol{\sigma}_0 \tag{A.8}$$

where, $\boldsymbol{\sigma}, \boldsymbol{\varepsilon}$ are the vectors of plane stresses and strains respectively, $\mathbf{D}$ is the elasticity matrix, E is the Young's modulus and ν is the Poisson's ratio for the specific material. The subscript "0" defines the initial stress or strains but in our case, we consider them zero. In every elasticity problem the primary variable is the displacement and it is inserted in problem by strain. The strain-displacement relationship is:

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \boldsymbol{\partial}\mathbf{u} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{bmatrix} \tag{A.9}$$

Now we would need to formulate Finite element method using Galerkin method (eq. (2.6)). For this purpose, we would firstly need to discretize the arbitrary domain Ω and then after choosing suitable shape functions we then apply the process described on section 2.1. However, the differential equations are needed to apply Galerkin. The differential equations of elasticity (Navier equations) are complex. Thus, we use an alternative and equivalent method to formulate eq. (2.6), the principle of virtual work. According to this method if we apply a virtual force to the system then the external work $\mathbf{W}_{\text{EXT}}$ has to be equal to the internal $\mathbf{W}_{\text{INT}}$. So, for one element the alternate notation of eq. (2.6) is:

$$\underbrace{\int_{\Omega} (\delta\varepsilon)^{\mathrm{T}} \boldsymbol{\sigma} \mathrm{d}\Omega}_{\mathbf{W}_{\text{INT}}} = \underbrace{\int_{\Omega} (\delta\mathbf{u})^{\mathrm{T}} \mathbf{F} \mathrm{d}\Omega + \int_{\Gamma} (\delta\mathbf{u})^{\mathrm{T}} \boldsymbol{\Phi} \mathrm{d}\Gamma}_{\mathbf{W}_{\text{EXT}}} \tag{A.10}$$

where δ denotes the virtual strain or displacement, $\mathbf{F}$ are the virtual external forces in domain (volume) Ω and $\boldsymbol{\Phi}$ are the surface tractions on the boundary Γ. For example, we create a mesh with m elements and n nodes. The vector of nodal parameters will be noted as: $\mathbf{a} = [u_1\, u_2 \dots u_n\, v_1\, v_2 \dots v_n]^{\mathrm{T}}$. Using the previous notation from before the approximation of displacement $\mathbf{u}$ according to eq. (2.1) will be written as:

$$(2.1) \Rightarrow \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} \cong \mathbf{Na} = \begin{bmatrix} \sum_{i=1}^{n} N_i u_i \\ \sum_{i=1}^{n} N_i v_i \end{bmatrix} = \begin{bmatrix} N_1 & N_2 & \dots & N_n & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & N_1 & N_2 & \dots & N_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \\ v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \qquad (\mathrm{A}.11)$$

where $N_i, i = 1,2,\dots,n$ are the element's shape functions. According to the approximation above from eq. $(\mathrm{A}.9)$, strains will be written as:

$$(\mathrm{A}.9) \xRightarrow{(\mathrm{A}.11)} \boldsymbol{\varepsilon} = \partial(\mathbf{Na}) \xLeftrightarrow{\mathrm{a}\to\mathrm{const.}}$$

$$\Leftrightarrow \boldsymbol{\varepsilon} = \partial(\mathbf{N})\mathbf{a} \xLeftrightarrow{\boldsymbol{\partial}\mathbf{N}\equiv\mathbf{B}}$$

$$\Leftrightarrow \boldsymbol{\varepsilon} = \mathbf{Ba} \qquad (\mathrm{A}.12)$$

where **B** is called strain-displacement matrix.

Now we need to formulate the eq. $(\mathrm{A}.10)$ to a similar manner as eq. (2.6) is order to derive a system of equations that can be solved. From eq. $(\mathrm{A}.11)$ and eq. $(\mathrm{A}.12)$ the virtual strains and displacements can be written as:

$$\delta\mathbf{u}^{\mathrm{T}} = (\delta\mathbf{a})^{\mathrm{T}}\mathbf{N}^{\mathrm{T}} \quad \text{and} \quad \delta\boldsymbol{\varepsilon}^{\mathrm{T}} = (\delta\mathbf{a})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}} \qquad (\mathrm{A}.13)$$

Then eq. $(\mathrm{A}.10)$ will be expressed as:

$$(\mathrm{A}.10) \xRightarrow{(\mathrm{A}.13)} \int_{\Omega} (\delta\mathbf{a})^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\boldsymbol{\sigma}\mathrm{d}\Omega = \int_{\Omega} (\delta\mathbf{a})^{\mathrm{T}}\mathbf{N}^{\mathrm{T}}\mathbf{F}\mathrm{d}\Omega + \int_{\Gamma} (\delta\mathbf{a})^{\mathrm{T}}\mathbf{N}^{\mathrm{T}}\boldsymbol{\Phi}\mathrm{d}\Gamma \xRightarrow{(\mathrm{A}.8)\to\boldsymbol{\sigma}=\mathbf{D}\boldsymbol{\varepsilon}=\mathbf{DBa}}$$

$$\Rightarrow \delta\mathbf{a}^{\mathrm{T}}\left(\int_{\Omega} \mathbf{B}^{\mathrm{T}}\mathbf{DBa}\mathrm{d}\Omega - \int_{\Omega} \mathbf{N}^{\mathrm{T}}\mathbf{F}\mathrm{d}\Omega - \int_{\Gamma} \mathbf{N}^{\mathrm{T}}\boldsymbol{\Phi}\mathrm{d}\Gamma\right) = 0 \xRightarrow[\mathbf{a} \neq \mathbf{a}(x,y)]{\delta\mathbf{a}^{\mathrm{T}}:\mathbf{arbitrary}}$$

$$\Rightarrow \int_{\Omega} \mathbf{B}^{\mathrm{T}}\mathbf{DBa}\mathrm{d}\Omega - \int_{\Omega} \mathbf{N}^{\mathrm{T}}\mathbf{F}\mathrm{d}\Omega - \int_{\Gamma} \mathbf{N}^{\mathrm{T}}\boldsymbol{\Phi}\mathrm{d}\Gamma = 0 \Leftrightarrow$$

$$\Leftrightarrow \mathbf{Ka} = \mathbf{f} \qquad (\mathrm{A}.14)$$

with $\mathbf{K} = \int_{\Omega} \mathbf{B}^{\mathrm{T}}\mathbf{DBa}\mathrm{d}\Omega$ and $\mathbf{f} = \int_{\Omega} \mathbf{N}^{\mathrm{T}}\mathbf{F}\mathrm{d}\Omega + \int_{\Gamma} \mathbf{N}^{\mathrm{T}}\boldsymbol{\Phi}\mathrm{d}\Gamma$. After deriving the FE equation, we need to choose the proper shape functions, in other words the element type. As element we chose an isoparametric element, the Bilinear Quadrilateral (Q4) element (see fig. A.2). From fig. A.2 this element consists of 4 nodes on vertices of the element with natural coordinates $\xi, \eta \in [-1, 1]$. For this element we have four shape functions that are matched to four nodes and they are derived from $1st$ -order Lagrange polynomials as:

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta)$$

$$N_2 = \frac{1}{4}(1+\xi)(1-\eta)$$

$$N_3 = \frac{1}{4}(1-\xi)(1+\eta)$$

$$N_4 = \frac{1}{4}(1+\xi)(1+\eta)$$

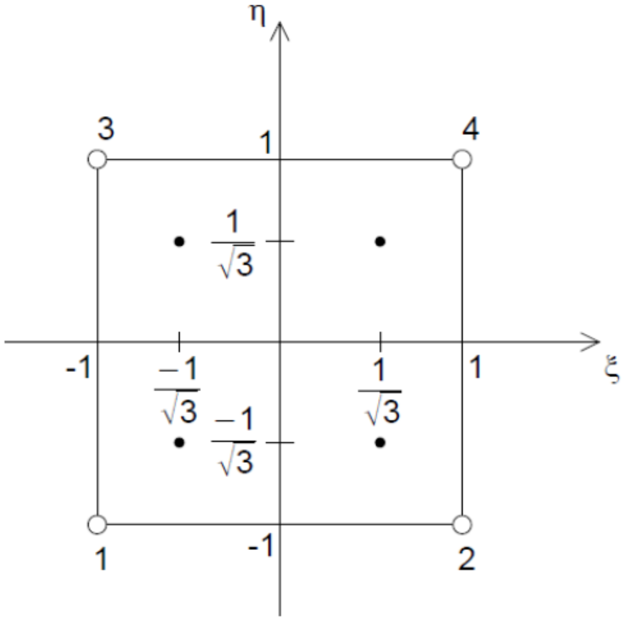


Figure A.2: Bilinear Quadrilateral Element in ξ, η-plane. The white dots are the 4 nodes of element and the black dots are the Gauss points for the accurate numerical integration of this element. Source: [3]

In isoparametric formulation a mapping between global and parent elements exists. This mapping is defined by Jacobian matrix. The Jacobian matrix $\mathbf{J}$ is defined as:

$$\begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \mathbf{J} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \tag{A.16}$$

In isoparametric elements the same shape functions that are used for displacement are also used for the geometry in Cartesian space. According to chapter 2.2 the x, y coordinates are written for Q4 elements as:

$$x = \sum_{i=1}^{4} N_i x_i \quad \text{and} \quad y = \sum_{i=1}^{4} N_i y_i \tag{A.17}$$

Substituting eq. $(A.17)$ to Jacobian matrix then:

$$\mathbf{J}=\begin{bmatrix}\sum_{i=1}^{4}\frac{\partial N_i}{\partial \xi}x_i & \sum_{i=1}^{4}\frac{\partial N_i}{\partial \xi}y_i\\ \sum_{i=1}^{4}\frac{\partial N_i}{\partial \eta}x_i & \sum_{i=1}^{4}\frac{\partial N_i}{\partial \eta}y_i\end{bmatrix}=\frac{1}{4}\begin{bmatrix}(\eta-1) & (1-\eta) & -(1+\eta) & (1+\eta)\\ (\xi-1) & (1-\xi) & -(1+\xi) & (1+\xi)\end{bmatrix}\begin{bmatrix}x_1 & y_1\\ x_2 & y_2\\ x_3 & y_3\\ x_4 & y_4\end{bmatrix}=\begin{bmatrix}\boldsymbol{J_{11}} & \boldsymbol{J_{12}}\\ \boldsymbol{J_{21}} & \boldsymbol{J_{22}}\end{bmatrix}\quad (\mathrm{A}.18)$$

The benefit of isoparametric analysis is to write eq. $(A.14)$ about natural coordinates ξ,η where calculations are easier, because for integrals Gauss quadrature can be used. The reason that Gauss Quadrature is used because the fact that $\xi,\eta\in[-1,1]$. So, the derivatives of natural coordinates in eq. $(\mathrm{A}.16)$ need to be expressed about Cartesian coordinates using the inverse Jacobian matrix $\boldsymbol{\Gamma}$. The inverse Jacobian is defined as:

$$\boldsymbol{\Gamma}\equiv\mathbf{J}^{-1}=\frac{1}{\det(\mathbf{J})}\begin{bmatrix}J_{22} & -J_{12}\\ -J_{21} & J_{11}\end{bmatrix}\qquad (\mathrm{A}.19)$$

So, using eq. $(A.16)$ and eq. $(A.19)$ for displacement derivatives we have:

$$\begin{bmatrix}\frac{\partial u}{\partial x}\\ \frac{\partial u}{\partial y}\\ \frac{\partial v}{\partial x}\\ \frac{\partial v}{\partial y}\end{bmatrix}=\begin{bmatrix}\boldsymbol{\Gamma} & 0\\ 0 & \boldsymbol{\Gamma}\end{bmatrix}\begin{bmatrix}\frac{\partial u}{\partial \xi}\\ \frac{\partial u}{\partial \eta}\\ \frac{\partial v}{\partial \xi}\\ \frac{\partial v}{\partial \eta}\end{bmatrix}\qquad (\mathrm{A}.20)$$

If we differentiate about natural coordinates from eq. $(\mathrm{A}.11)$ we have:

$$(\mathrm{A}.11)\Rightarrow\begin{bmatrix}\frac{\partial u}{\partial \xi}\\ \frac{\partial u}{\partial \eta}\\ \frac{\partial v}{\partial \xi}\\ \frac{\partial v}{\partial \eta}\end{bmatrix}=\begin{bmatrix}\frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \xi} & 0 & 0 & 0 & 0\\ \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \xi} & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_4}{\partial \eta}\\ 0 & 0 & 0 & 0 & \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_4}{\partial \eta}\end{bmatrix}\mathbf{a}\qquad (\mathrm{A}.21)$$

So, by combining the eq. $(\mathrm{A}.9)$, eq. $(\mathrm{A}.20)$, eq. $(\mathrm{A}.21)$ the strain-displacement matrix becomes:

$$\mathbf{B}=\begin{bmatrix}\frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y}\\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x}\end{bmatrix}\qquad (\mathrm{A}.22)$$

Now we have all the needed parts in order to write the eq. $(\mathrm{A}.14)$ in terms of natural coordinates, as it is demanded by isoparametric formulation. Having plane stress problem, the elementary volume $\mathrm{d}\Omega = t\,\mathrm{d}x\,\mathrm{d}y$ where t is the constant thickness in z-direction. If we apply "Substitution rule" to the integrals of stiffness matrix changing the Cartesian to natural coordinates and Gaussian quadrature for numerical integration we have:

$$(\mathrm{A.14}) \Rightarrow t\int_{-1}^{1}\int_{-1}^{1}\mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\,\det(\mathbf{J})\,\mathrm{d}\xi\mathrm{d}\eta \approx t\sum_{i}\sum_{j}\mathbf{B}^{\mathrm{T}}(\xi_i,\eta_j)\mathbf{D}\mathbf{B}(\xi_i,\eta_j)\,\det(\mathbf{J})\,w_i w_j \qquad (\mathrm{A.23})$$

where, ξ_i, η_j are the coordinates of Gauss points and w_i, w_j are the corresponding weights. In previous figure it is shown that the chosen values for 4 Gauss points for the integration of Q4 element, $(\xi_i, \eta_j) = \left(\pm\frac{1}{\sqrt{3}}, \pm\frac{1}{\sqrt{3}}\right)$ with $w_i, w_j = 1$ for all points. The same logic can be applied to **f** where the shape functions are written about natural coordinates. Then according to flowchart of fig. 2.1 the unknown values of **a** are computed along with useful quantities (e.g., stresses, strains etc.).

Appendix B

In this appendix we will present an example of computing basis functions using eq. (3.3) The knot vector in which basis functions will be computed is the $\Xi = \left\{ \underbrace{0}_{\xi_1}, \underbrace{0}_{\xi_2}, \underbrace{0}_{\xi_3}, \underbrace{0.2}_{\xi_4}, \underbrace{0.4}_{\xi_5}, \underbrace{0.6}_{\xi_6}, \underbrace{0.8}_{\xi_7}, \underbrace{0.8}_{\xi_8}, \underbrace{1}_{\xi_9}, \underbrace{1}_{\xi_{10}}, \underbrace{1}_{\xi_{11}} \right\}$. The order of this B-spline according to this open knot vector is $p + 1 = 3$; thus, the degree is $p = 2$. The number of basis functions is needed to be computed are $n + p + 1 = 11 \Leftrightarrow n = 8$. Then $N_1^2(\xi), N_2^2(\xi), N_3^2(\xi), N_4^2(\xi), N_5^2(\xi), N_6^2(\xi), N_7^2(\xi), N_8^2(\xi)$ should be computed. In order to compute these basis functions, we need firstly to compute the corresponded basis functions of reduced degree. Let us begin with basis functions of zero degree $(p = 0)$.

For computing them we will define each basis function in the corresponding interval $[\xi_i, \xi_{i+1})$. The length of Ξ is 11, therefore the number of zero-degree basis functions is 10, because 10 intervals exist. The first interval is $[\xi_1, \xi_2) = [0,0)$. Because $\xi_1 = \xi_2 \Rightarrow N_1^0(\xi) \equiv N_1^0 = 0, \forall \xi$. The same is also for N_2^0 because $[\xi_2, \xi_3) = [0,0)$. For the third basis functions the interval is $[\xi_3, \xi_4] = [0,0.2) \Rightarrow \xi_3 \neq \xi_4$. So according to eq. (3.3) $N_3^0 = \begin{cases} 1, & \xi \in [\xi_3, \xi_4) \\ 0, & \text{otherwise} \end{cases}$. The corresponding procedure goes for other zero-degree functions. The results are:

$$N_1^0 = N_2^0 = N_7^0 = N_9^0 = N_{10}^0 = 0$$

and

$$N_3^0 = \begin{cases} 1, & \xi \in [0,0.2) \\ 0, & \text{otherwise} \end{cases}$$

$$N_4^0 = \begin{cases} 1, & \xi \in [0.2,0.4) \\ 0, & \text{otherwise} \end{cases}$$

$$N_5^0 = \begin{cases} 1, & \xi \in [0.4,0.6) \\ 0, & \text{otherwise} \end{cases}$$

$$N_6^0 = \begin{cases} 1, & \xi \in [0.6,0.8) \\ 0, & \text{otherwise} \end{cases}$$

$$N_8^0 = \begin{cases} 1, & \xi \in [0.8,1) \\ 0, & \text{otherwise} \end{cases}$$

Now, that N_i^0 have been defined the next degree basis functions must be computed. As the degree of basis functions is increasing, the number of calculated basis functions is decreased by 1. As a result, the number of N_i^1 is 9. The N_i^1 will be computed using the recursive form of eq. (3.3). For example, we could start by calculating N_1^1. According to eq. (3.3) the function will be:

$$N_1^1 = \frac{\xi - \xi_1}{\xi_2 - \xi_1} N_1^0 + \frac{\xi_2 - \xi}{\xi_3 - \xi_2} N_2^0 \tag{B.24}$$

The terms $\frac{\xi - \xi_1}{\xi_2 - \xi_1}, \frac{\xi_2 - \xi}{\xi_3 - \xi_2}$ are not defined because ($\xi_1 = \xi_2$ and $\xi_3 = \xi_2$). However, the terms $N_1^0 = N_2^0 = 0$ from before; thus from eq. $(A.24)$ the $N_1^1 = 0$. Proceeding to N_2^1 we have:

$$N_2^1 = \frac{\xi - \xi_2}{\xi_3 - \xi_2} N_2^0 + \frac{\xi_4 - \xi}{\xi_4 - \xi_3} N_3^0 \tag{B.25}$$

The $\frac{\xi-\xi_2}{\xi_3-\xi_2}$ is not defined again, but $\frac{\xi_4-\xi}{\xi_4-\xi_3}=\frac{0.2-\xi}{0.4-0.2}=\frac{\frac{1}{5}-\xi}{\frac{1}{5}}=1-5\xi$. This coefficient is multiplied by N_3^0 which is 1 on interval $[0,0.2)$ and 0 elsewhere. So, from $(\mathrm{B}.25)$ we will have:

$$N_2^1=(1-5\xi)N_3^0=\begin{cases}1-5\xi, & \xi\in[0,0.2)\\ 0, & \text{otherwise}\end{cases}$$

The same procedure goes for other basis functions. So, the results will be:

$$N_1^1=N_9^1=0$$

$$N_2^1=1-5\xi,\xi\in[0,0.2)$$

$$N_3^1=\begin{cases}5\xi, & \xi\in[0,0.2)\\ 2-5\xi, & \xi\in[0.2,0.4)\end{cases}$$

$$N_4^1=\begin{cases}5\xi-1, & \xi\in[0.2,0.4)\\ 3-5\xi, & \xi\in[0.4,0.6)\end{cases}$$

$$N_5^1=\begin{cases}5\xi-2, & \xi\in[0.4,0.6)\\ 4-5\xi, & \xi\in[0.6,0.8)\end{cases}$$

$$N_6^1=5\xi-3,\ \ \xi\in[0.6,0.8)$$

$$N_7^1=5-5\xi,\ \ \xi\in[0.8,1)$$

$$N_8^1=5\xi-4,\ \ \xi\in[0.8,1)$$

Now, N_i^2 must be computed with same procedure as above. Finally:

$$N_1^2=\frac{\xi-\xi_1}{\xi_3-\xi_1}N_1^1+\frac{\xi_4-\xi}{\xi_4-\xi_2}N_2^1=\cdots=(1-5\xi)^2,\qquad \xi\in[0,0.2)$$

$$N_2^2=\begin{cases}5\xi(1-5\xi)+\left(1-\frac{5}{2}\xi\right)5\xi, & \xi\in[0,0.2)\\ \left(1-\frac{5}{2}\xi\right)(2-5\xi), & \xi\in[0.2,0.4)\end{cases}$$

$$N_3^2=\begin{cases}\frac{25}{2}\xi^2, & \xi\in[0,0.2)\\ \frac{5}{2}\xi(2-5\xi)+\left(\frac{3-5\xi}{2}\right)(5\xi-1), & \xi\in[0.2,0.4)\\ \left(\frac{(3-5\xi)^2}{2}\right), & \xi\in[0.4,0.6)\end{cases}$$

$$N_4^2=\begin{cases}\frac{(5\xi-1)^2}{2}, & \xi\in[0.2,0.4)\\ \frac{(5\xi-1)}{2}(3-5\xi)+\frac{(4-5\xi)}{2}(5\xi-2), & \xi\in[0.4,0.6)\\ \frac{(4-5\xi)^2}{2}, & \xi\in[0.6,0.8)\end{cases}$$

$$N_5^2 = \begin{cases} \dfrac{(5\xi-2)^2}{2}, & \xi \in [0.4,0.6) \\ \dfrac{(5\xi-2)}{2}(4-5\xi) + \dfrac{4-5\xi}{2}(5\xi-3), & \xi \in [0.6,0.8) \end{cases}$$

$$N_6^2 = \begin{cases} \dfrac{(5\xi-3)^2}{2}, & \xi \in [0.6,0.8) \\ \dfrac{(5-5\xi)^2}{2}, & \xi \in [0.6,0.8) \end{cases}$$

$$N_7^2 = \frac{(5\xi-4)}{2}(5-5\xi) + \frac{(5-5\xi)}{2}(5\xi-4), \qquad \xi \in [0.8,1)$$

$$N_8^2 = \frac{(5\xi-4)^2}{2}, \qquad \xi \in [0.8,1)$$

Below the plot of N_3^2 is following. As we can see each polynomial constructs a specific part of curve.

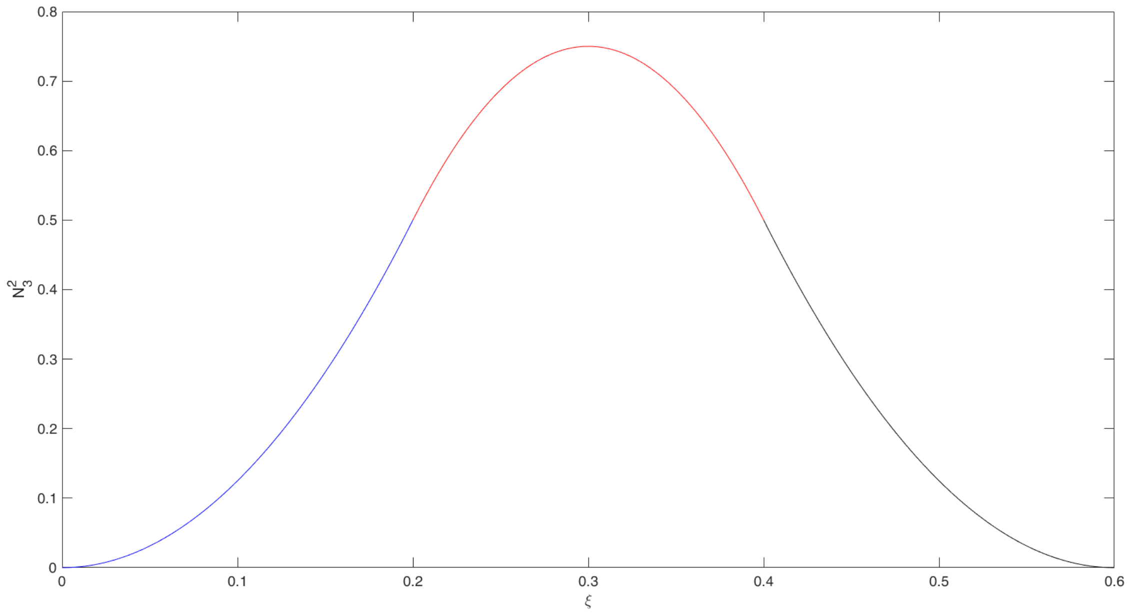


Figure B.1: The plot of polynomial N_3^2.decomposed to its different parts.

Appendix C: Manual of I3GA

In this Appendix we will present the manual of the current version of I3GA.

C.1 Central menu and Loading Geometries

The program runs by executing the following process.

1. Run from Command Window the program as: g=i3ga
2. Call the menu with the commnad: g.menu()

The menu of I3GA is the following:

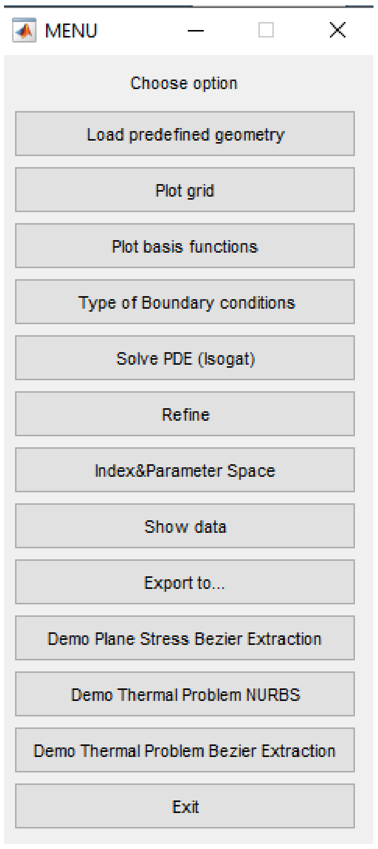


Figure C.1: The central menu of I3GA.

As we can see there is a variety of choices. The first thing that should be done in order to start the analysis process is to use a geometry. Loading the geometry is done in tab "Load predefined geometry". If we select it, we go to the following submenu:

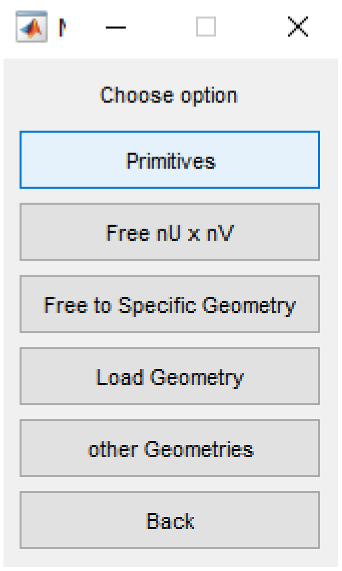


Figure C.2: Submenu “Load Predefined geometry”.

Except to tab “Back” that returns to the initial menu, the other 5 tabs create the geometry.

“Primitives” Tab

The “Primitives” tab contains some initial shapes which can be manipulated and processed freely. More specifically we have 9 shapes:

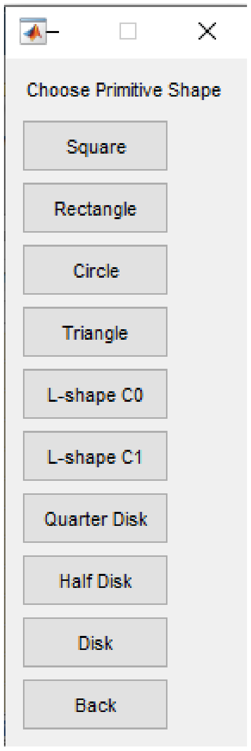
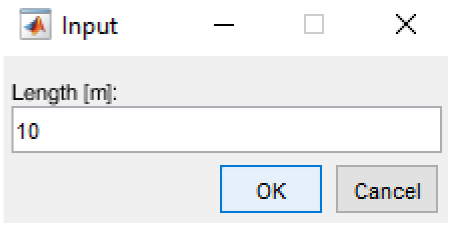


Figure C.3: Primitive shapes

Except for the triangle by choosing a geometry we can perform an analysis to it or to an edited version of it. After choosing the shape and pass the needed geometry parameters (i.e., lengths, radius, etc.) we can see the geometry by clicking in central menu the "Plot grid" tab.

Example:

Let suppose that we have chosen the "Square" in Primitives menu. Then it is asked one parameter, the length of the square's edge.


Input

Length [m]:

10

Figure C.4: Bar in which we put the length of square' s edge.

Let suppose that the length is 10 m. Then we go back to the initial menu and we click the "Plot grid" button. The result is following below:

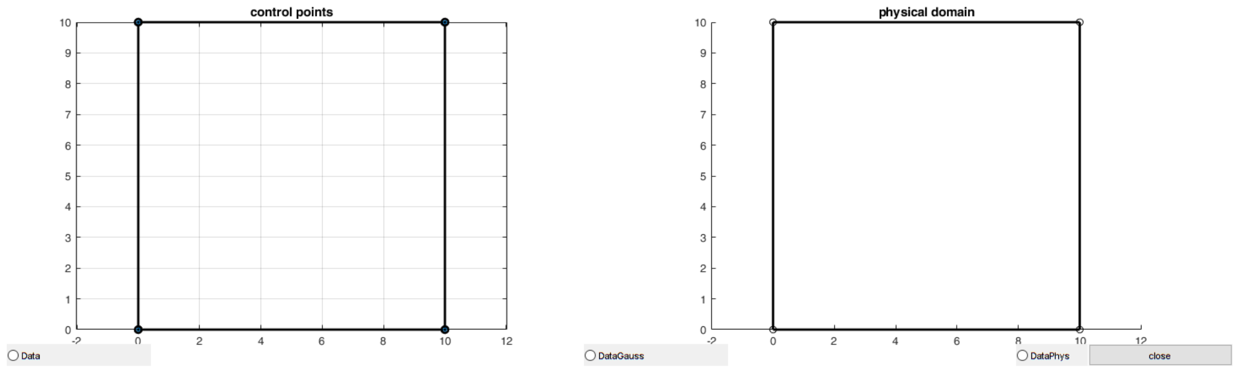
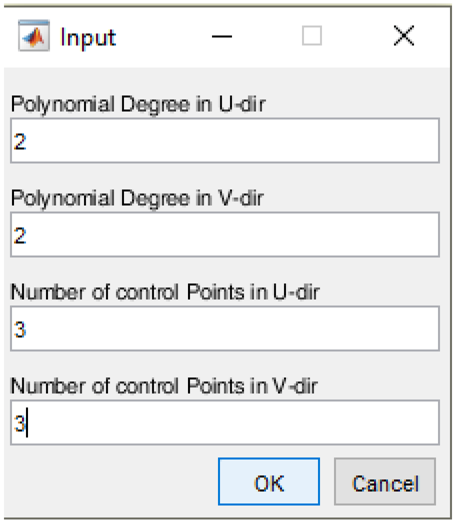


Figure C.5: Result of "Plot Grid" tab.

On the left we have the control net, on the right is the physical space. Later we will see how can create free-hand objects from "Plot grid" tab.

Free nU x nV

This tab permits the construction of a grid in which the user chooses the number of control points per direction. As we can see in figure C.6 the user inserts the degree of NURBS Basis functions the number of control points per parametric direction (u $\equiv \xi$ and $v \equiv \eta$ parametric direction). Next the user inserts the max and min values of desired knot vectors per direction (fig. C.7). In the end as it is shown in fig. C.8, a window is showed up in which we insert the control points whenever we want with left click. At the last point we push right mouse click. Next by clicking again "Plot grid" tab in the central menu we can see the result. In the example that is following we create a quadratic NURBS curve by 3x3 grid.


Input

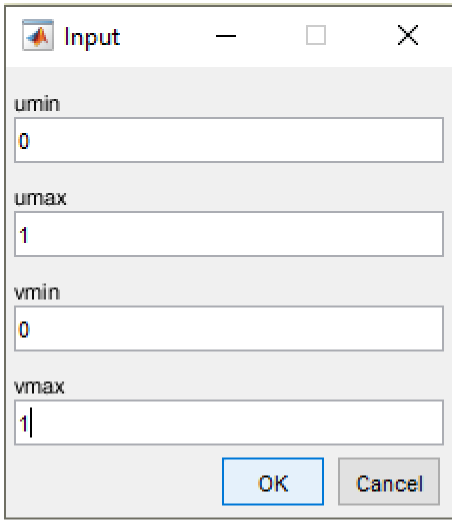
Polynomial Degree in U-dir: 2

Polynomial Degree in V-dir: 2

Number of control Points in U-dir: 3

Number of control Points in V-dir: 3

Figure C.6: 1st Window in tab “Free nU x nV”.


Input

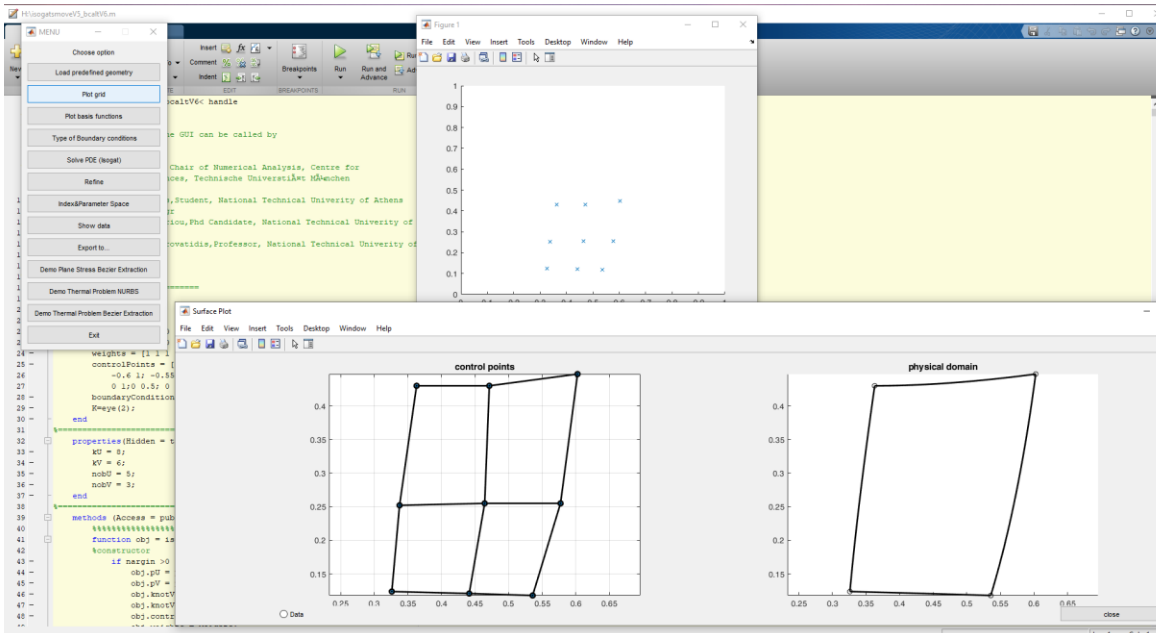
umin: 0

umax: 1

vmin: 0

vmax: 1

Figure C.7: 2nd Window in tab “Free nU x nV”

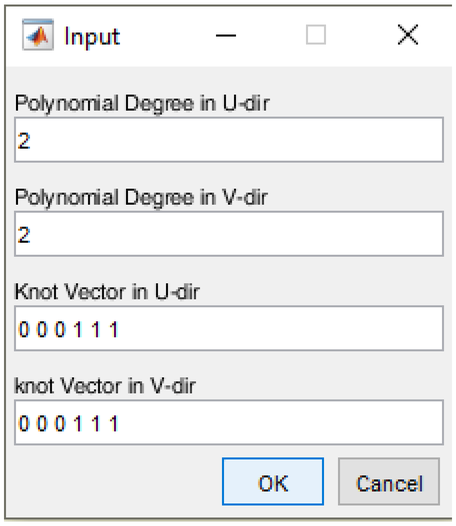


```
caltV6< handle
e GUI can be called by
Chair of Numerical Analysis, Centre for
ces, Technische UniverstiÃ¤t MÃ¼nchen
,Student, National Technical Univerity of Athens
r
iou,Phd Candidate, National Technical Univerity of
ovatidis,Professor, National Technical Univerity of
=======
        weights = [1 1 1
        controlPoints = [
            -0.6 1; -0.55
            0 1;0 0.5; 0
        boundaryCondition
        K=eye(2);
    end
%========================
    properties(Hidden = t
        kU = 8;
        kV = 6;
        nobU = 5;
        nobV = 3;
    end
%========================
    methods (Access = pub
        %%%%%%%%%%%%%%%%%%
        function obj = is
        %constructor
            if nargin >0
                obj.pU =
                obj.pV =
                obj.knotV
                obj.knotV
                obj.contr
```

Figure C.8: 3rd window in tab “Free nU x nV” along with the result.

Specific geometry

The “Specific” geometry tab was created for the construction of a specific geometry by providing the degree and knot vector of a grid. The user can manipulate the geometry by using the capabilities of Plot grid. Initially after the user clicks on this tab, he/she is asked to insert the degrees and the knot vectors of NURBS surface (see fig. 2.9). Next two windows are showed up. The first windows shows the number of control points that need to be inserted in every direction. The other is the window for placing the control points on the net. The result is showed again by using “Plot grid” in fig. C.11.



Input

Polynomial Degree in U-dir
2

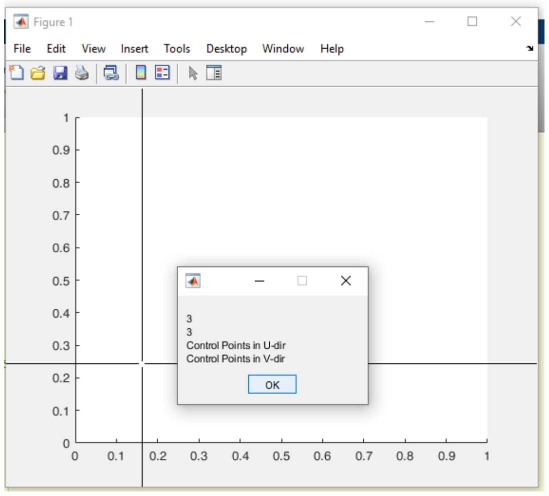
Polynomial Degree in V-dir
2

Knot Vector in U-dir
0 0 0 1 1 1

knot Vector in V-dir
0 0 0 1 1 1

OK Cancel

Figure C.9: 1st window on “Specific geometry”.



3
3
Control Points in U-dir
Control Points in V-dir

Figure C.10: 2nd Window on “Specific geometry”

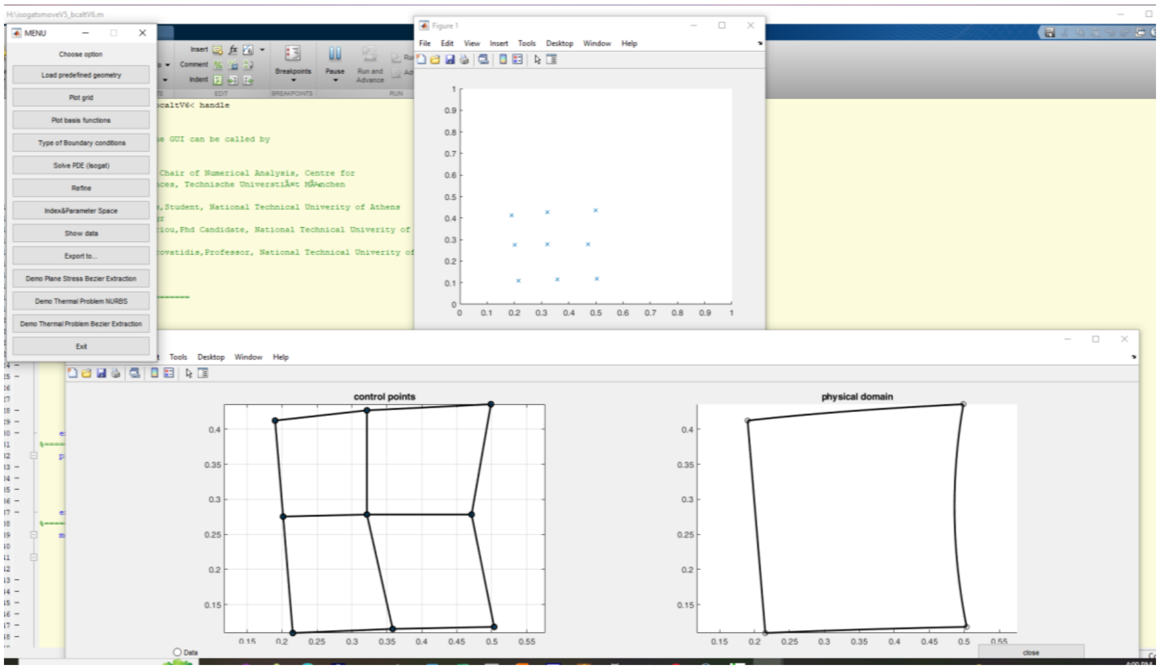


Figure C.11: Result of Specific geometry

Load Geometry

With this tab we load a saved geometry to edit it. The saved geometries are saved in .mat files and their format is a structure with the name obj. The obj structure contains the following properties:

```
obj.pU → Degree of basis function in ξ-direction.
obj.pV → Degree of basis function in η-direction.
obj.knotVectorU → Knot vector in ξ-direction.
obj.knotVectorV → Knot vector in η-direction.
obj.controlPoints → Control points.
obj.weights → Weights
obj.kU → Length of knot vector in ξ-direction.
obj.kV → Length of knot vector in η-direction.
```

```
                    obj.nobU → Number of basis functions in ξ-direction.
                    obj.nobV → Number of basis functions in η-direction.
```

Plot Basis Functions

After selecting the geometry, the user can see the basis functions for the NURBS surface he has chosen. He can do this in the central menu by selecting the tab “Plot basis functions”. The result for a square with edge length 10 m (4 control points → 4 basis functions) is shown in figure below.

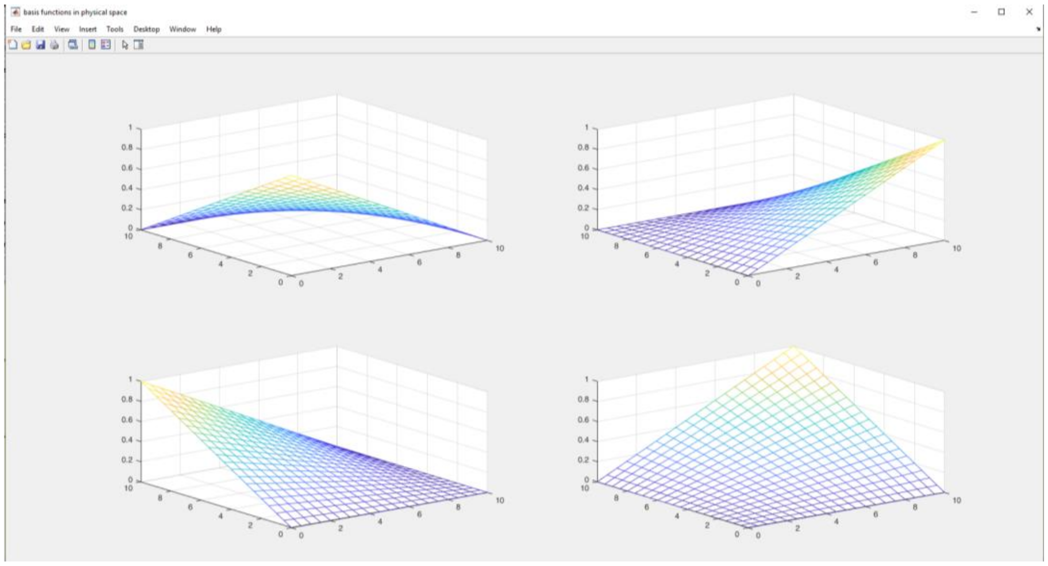

basis functions in physical space
File Edit View Insert Tools Desktop Window Help

Figure C.12: "Plot basis functions tab"

Index & Parameter space

The user can also observe the index and the parameter space if he chooses the tab “Index&Parameter” space from central menu. In fig. C.13 we observe the two spaces for a square with length 10 m and knot vectors {0 0 1 1} per direction.

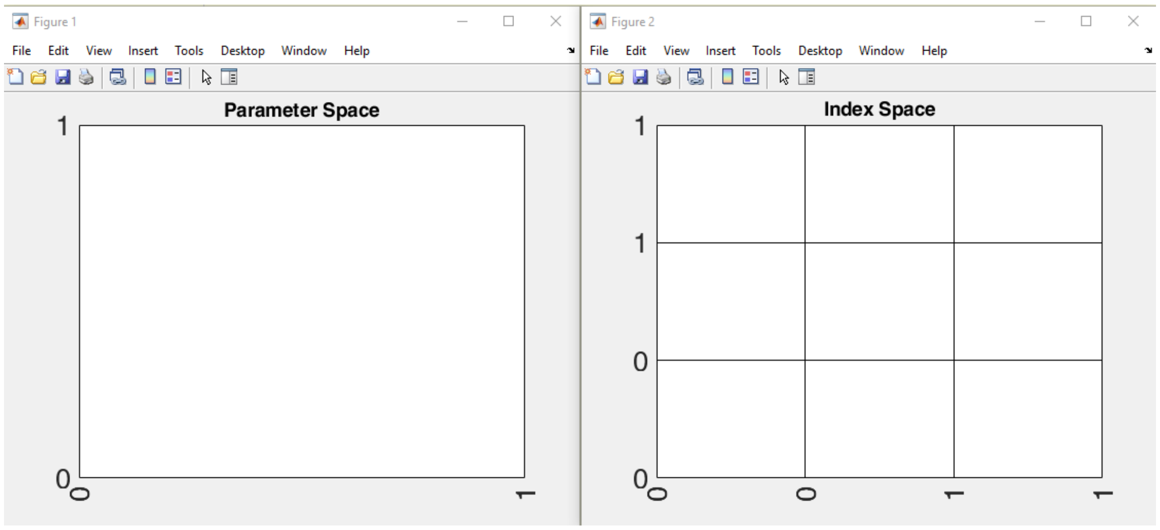


Figure 1
File Edit View Insert Tools Desktop Window Help
Parameter Space
1
0
0
1
Figure 2
File Edit View Insert Tools Desktop Window Help
Index Space
1
1
0
0
0
0
1
1

Figure C.13: Parameter and Index space

C.2 Refinement

If the user wants to refine geometry, he can do it through “Refine” tab in the central menu. This tab opens another menu that is shown in fig. C.13.

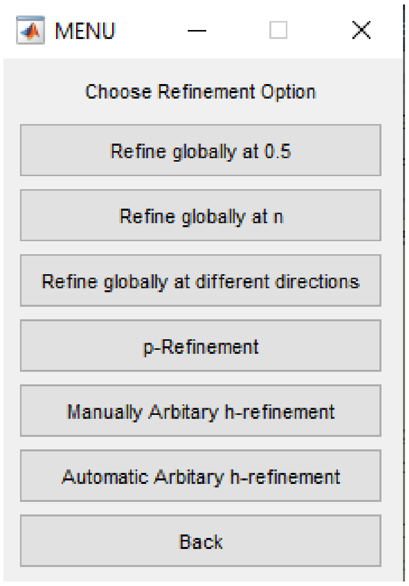

MENU
Choose Refinement Option
Refine globally at 0.5
Refine globally at n
Refine globally at different directions
p-Refinement
Manually Arbitary h-refinement
Automatic Arbitary h-refinement
Back

Figure C.14: “Refine” tab menu.

We will present the capabilities of each refinement option using as an example a 4-control point square.

Refine globally at 0.5 and at n

The first refinement option that the user could have used, is h-refienment or knot refinement. The first and the second tab in refinement menu, do knot insertion process on both directions (globally) by partitioning every non-empty knot span $[\xi_i, \xi_{i+1}]$ at its half or at value $n \in (0,1]$ proportional to the selected tab. Below two figures are following where he have two knot refinement options one at $n = 0.5$ and one at $n = 0.8$. The results are shown by pressing the “Plot grid” tab in central menu.

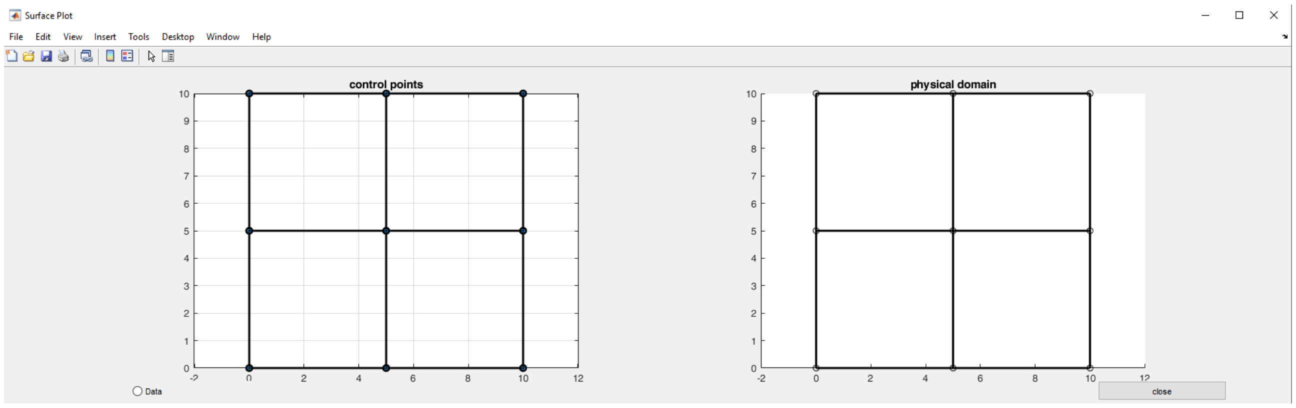


Figure C.15: Refine globally at 0.5.

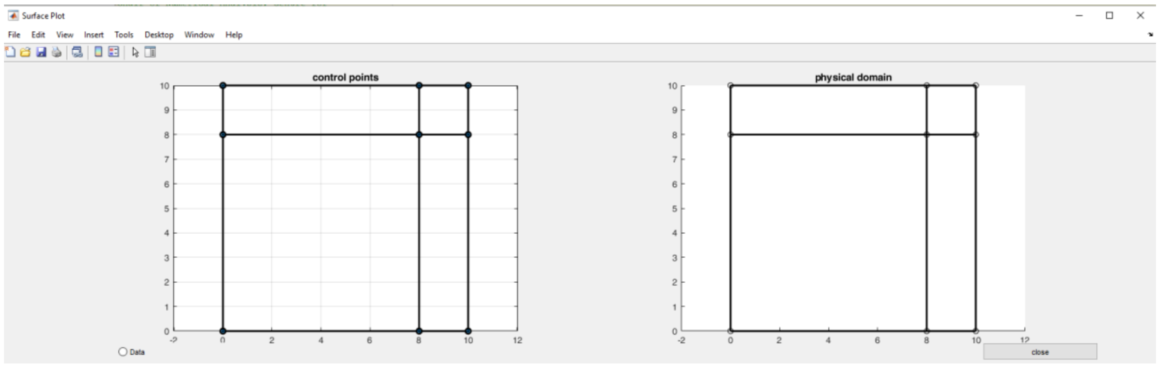
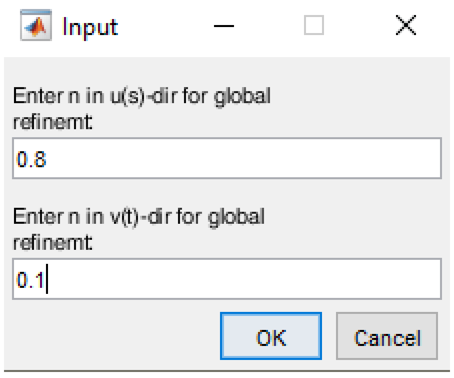


Figure C.16: Refine globally at $n = 0.8$.

Refine globally at different direction

This tab has the same concept with the others but the difference here is that the portioning of space in every direction happens at different n. Below it is following an example where $n_u = 0.8$ and $n_v = 0.1$.


Input

Enter n in u(s)-dir for global refinemt

0.8

Enter n in v(t)-dir for global refinemt

0.1

Figure C.17:Insertion bars for n_u, n_v.

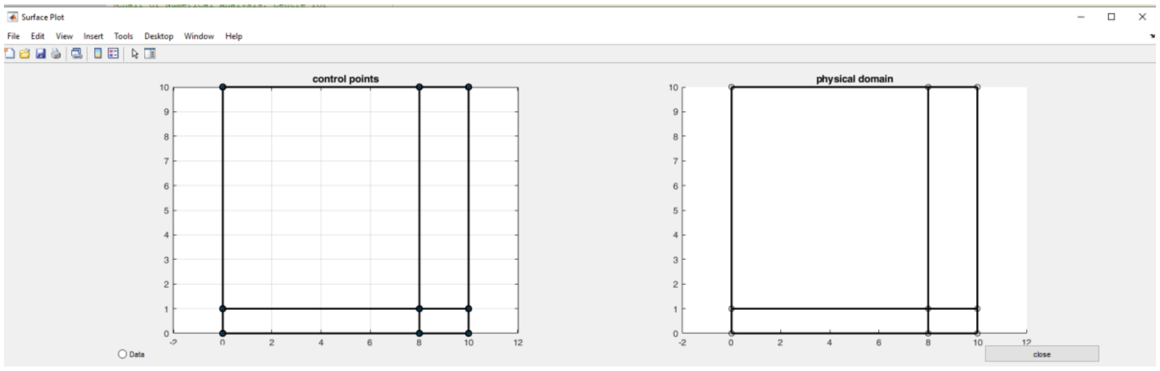
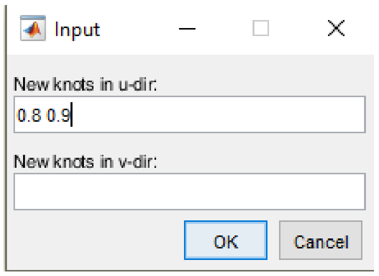


Figure C.18: Refine globally at different directions for $n_u = 0.8$ and $n_v = 0.1$

Manually arbitrary h-refinement

This specific tab is responsible for the knot insertion at desired knots. When we press it, two bars are showed up in where we can insert one or multiple knots in every direction. If the user does not want to insert knots in a direction, we just leave the bar empty. For example, here we have inserted the two knots at 0.8,0.9 only in ξ −direction.



New knots in u-dir:
0.8 0.9

New knots in v-dir:

Figure C.19: The two bars of the tab “Manually arbitrary h-refinement”.

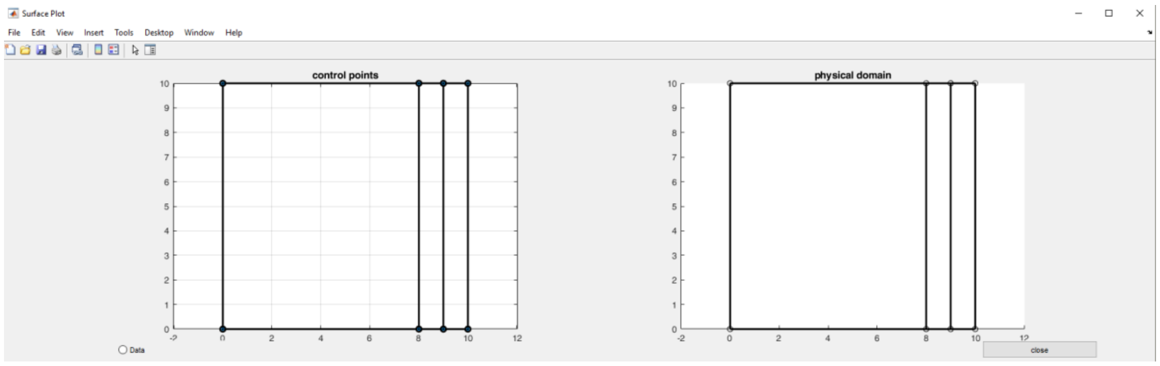
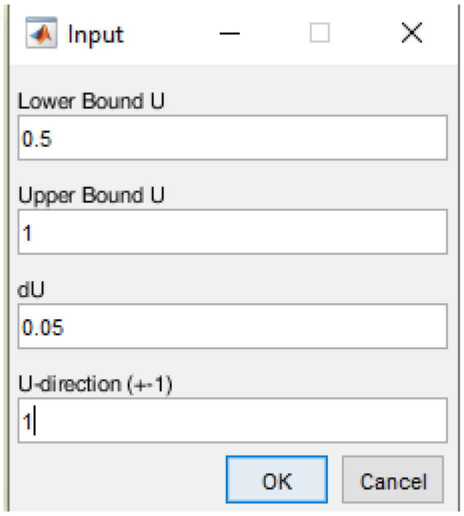


Figure C.20: Knot refinement in ξ-direction at 0.8,0.9.

Automatic arbitrary h-refinement

In this tab we can insert a series of knots in the desired direction. After we press this tab, then for evary direction a menu will be showed up.



Lower Bound U
0.5

Upper Bound U
1

dU
0.05

U-direction (+-1)
1

Figure C.21: Bars of sub-menu "Automatic arbitrary h-refinement"

At this menu we insert the limits of the interval that we want to do h-refinement, the knot step, and the direction where we want this knot series be deployed. From the smaller to larger value, we put $+1$ and for the inverse we put -1. Below we see an example where we have place knots in ξ-direction from 0.5 to 1 with knot step 0.05 and in η-direction from 0.8 to 1 with step 0.03.

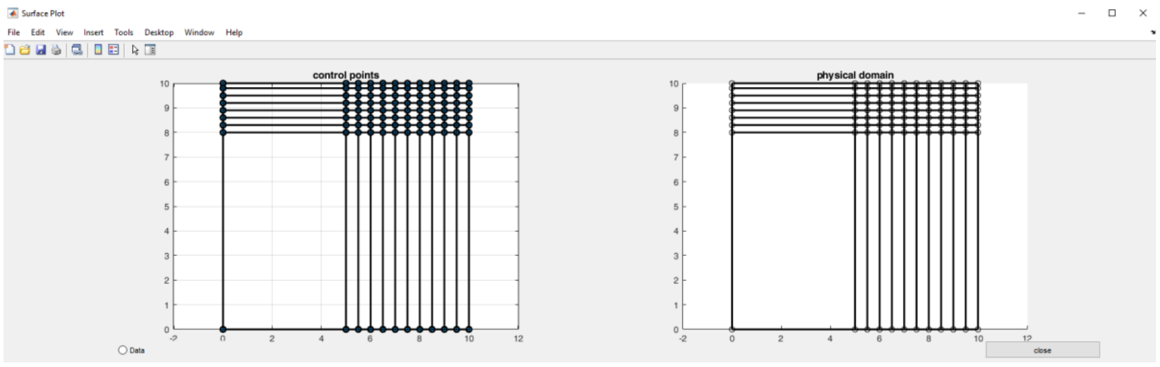
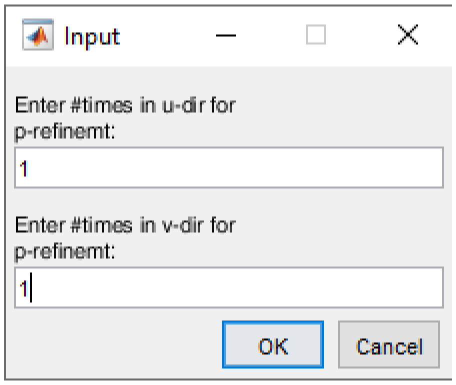


Figure C.22: Example for the case of “Automatic arbitrary h-refinement”.

P-refinement

With this last tab we can do p-refinement or degree elevation in the geometry. By pressing this tab, a window with two bars is showed up, which asks how many times we want to elevate the degree of the surface in each parametric direction. If the user does not desire to elevate a specific degree can leave the bar empty. Below we see an example where we have elevated the degree 1 time in both directions.



Input

Enter #times in u-dir for p-refinemt:

1

Enter #times in v-dir for p-refinemt:

1

Figure C.23: The two bars of tab “p-refinement”.

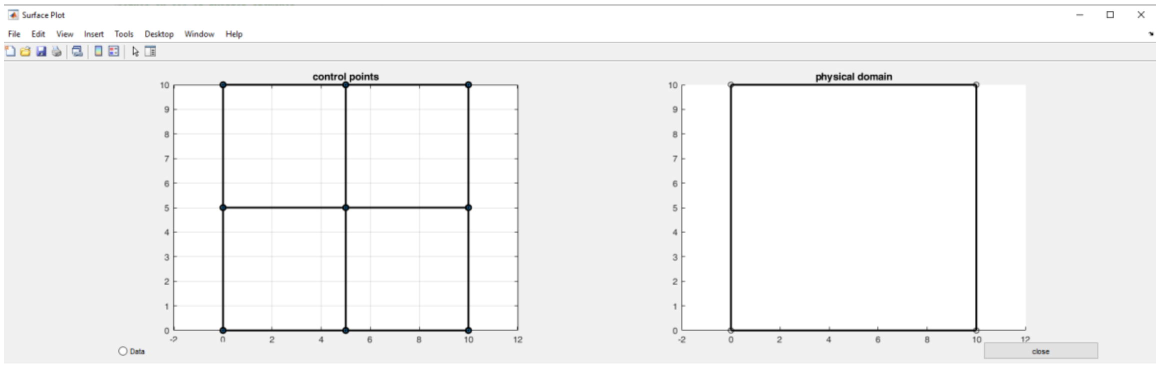


Figure C.24: Example of p-refinement

C.3 Plot grid

Except for displaying the surfaces this tab is in charge for the free manipulation of them. Specifically, the user can press on one control point with the mouse and by keeping the mouse pressed, move it to another position. The results of this procedure are showing up in figures C.25 and C.26.

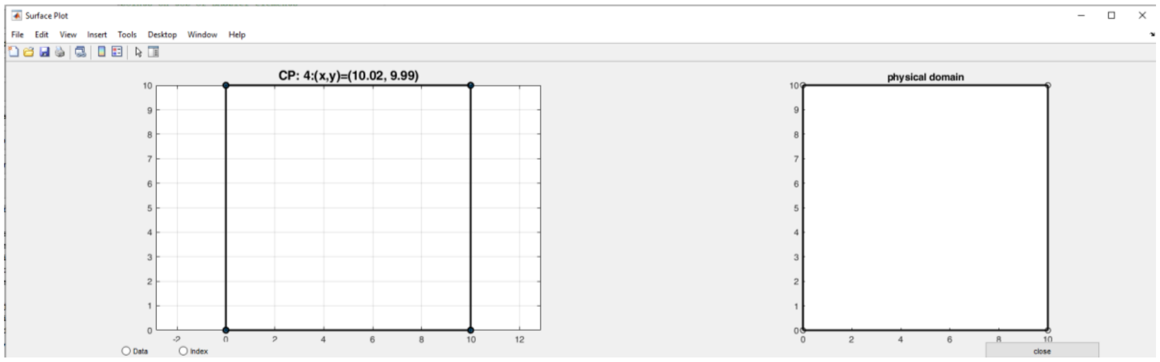

Surface Plot
File Edit View Insert Tools Desktop Window Help
CP: 4:(x,y)=(10.02, 9.99)
physical domain
Data
Index
close

Figure C.25:1st step of changing control point's position. On the title we see the numbering index and the coordinates of this control point.

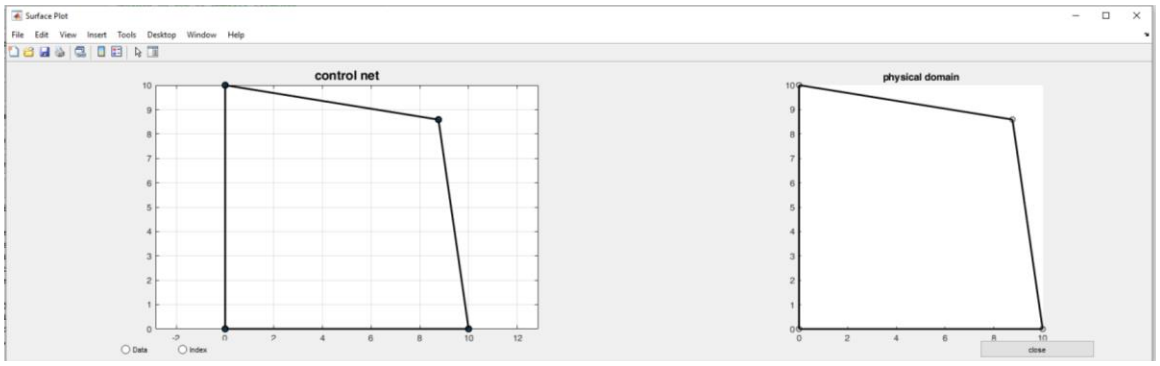
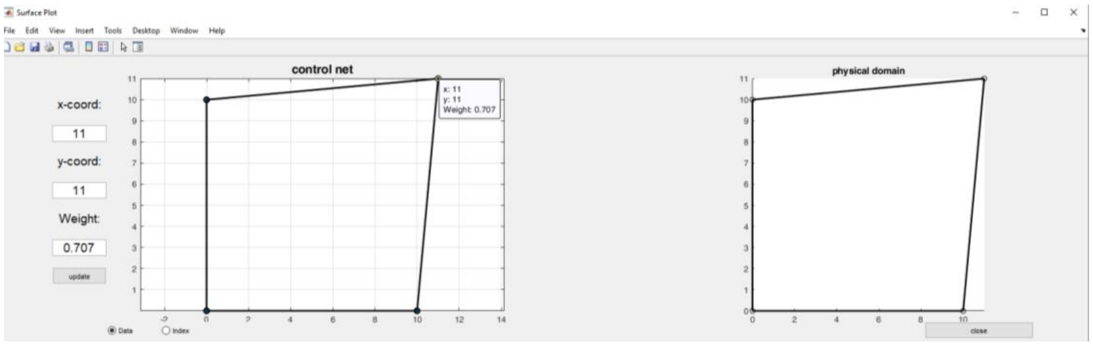


Figure C.26: 2nd step of changing the control point.

Except for the free-hand positioning of control points the user can also move them on specific locations and change their weight. That can be done by choosing the “Data” button in “Plot grid” window. When we select it is showed up a cross with which we select the control point that we want to move. When we select a control point, a label shows up with data about the point (its coordinates and weight). Next, on the window left side we can insert the new data for the control point and then click “update”. The data will have been updated but in order to update the visual result we need to press again the “Data” button. Moreover, we can see the index numbering of control points with choice Index.



x-coord: 11
y-coord: 11
Weight: 0.707

x: 11
y: 11
Weight: 0.707

Figure C.27: “Data” operation.

Furthermore, the user can execute knot refinement in physical space by clicking the “DataPhys” button. When we hit this button a cross is showed up where we can click on the Physical space to insert a point. Then a menu is showed up with three choices. The three choices are if we want to insert a knot only in ξ-direction, only in η-direction and on both ξ, η-direction. In figure C.28 and C.29 we see an example.

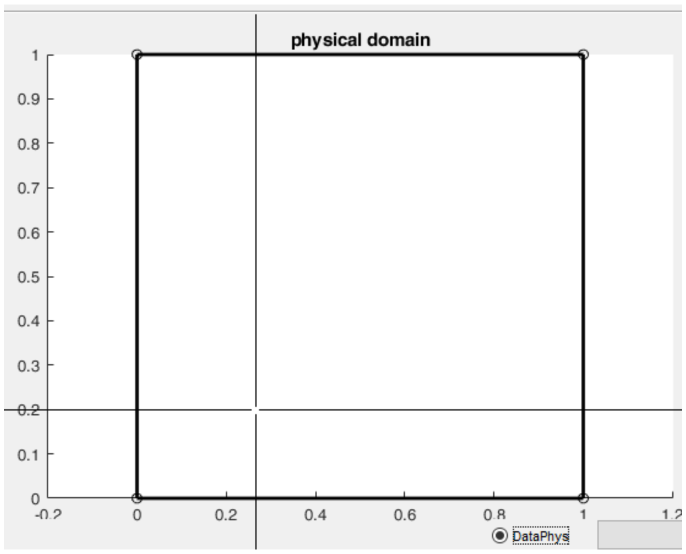


Figure C.28: Cross on "DataPhys".

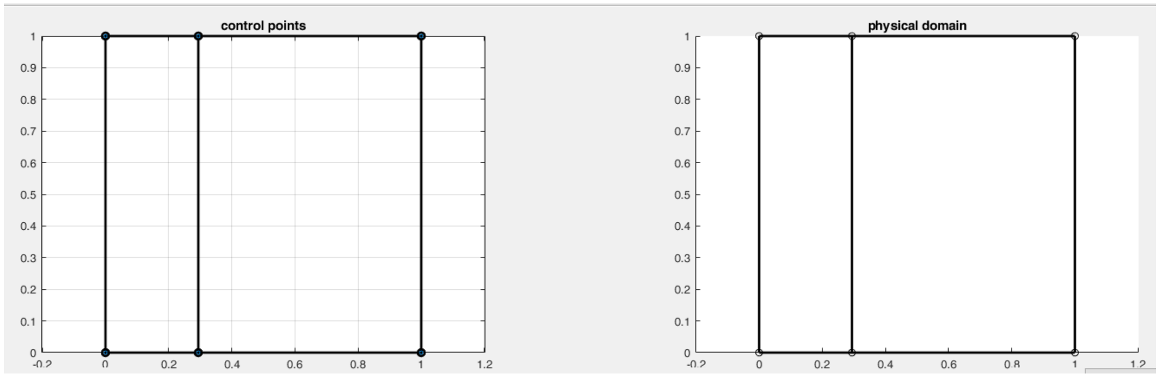
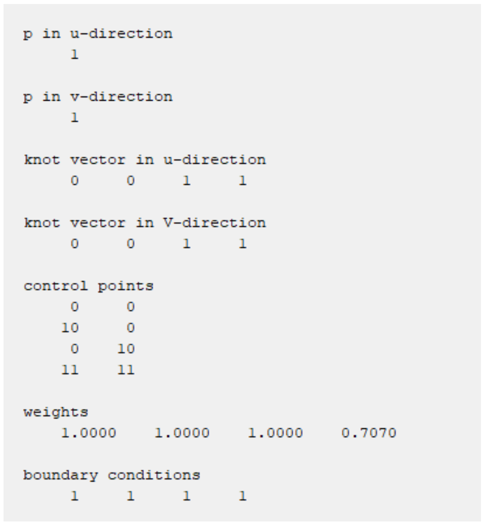


Figure C.29: Example of knot insertion with “DataPhys”.

Finally, with “DataGauss” button we can see the gauss points for the surface on parametric space.

Show Data

In this operation which is located in the initial menu, the user can see all the basic data of geometry in the command window. These data are the surface degrees, the knot vectors per parametric direction, the coordinates and weight of control points and the type of boundary conditions.



```
p in u-direction
     1

p in v-direction
     1

knot vector in u-direction
     0     0     1     1

knot vector in V-direction
     0     0     1     1

control points
     0     0
    10     0
     0    10
    11    11

weights
    1.0000    1.0000    1.0000    0.7070

boundary conditions
     1     1     1     1
```

Figure C.30: Data for a surface from “Show Data” button.

Export to…

With this tab that is accessible from the central menu the user can extract the NURBS geometry in three formats, .igs, .iga(IGAFEM) and .mat (This format saves the geometry to use it another time). After clicking on central sub-menu, a new menu is showed up with the 3-file format. If the user press a file format, then he can insert the name of the file without insert its format.

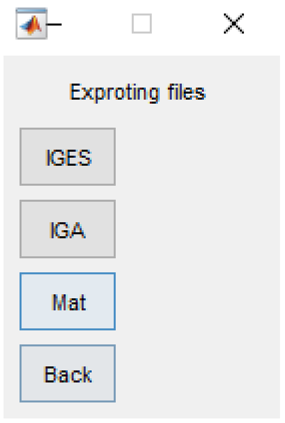


Figure C.31: Sub-menu of “Export-to”

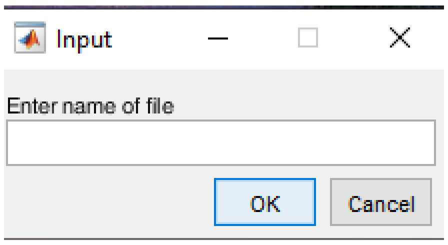


Figure C.32: Inserting the file name

C.4 Analysis

Type of Boundary conditions

In this tab, which is in the central menu, the user defines the type of boundary conditions in each side (up, down, right, left). The types are two Dirichlet or homogeneous Neumann. After pressing the tab, a sub-menu with the types of Boundary conditions is showed up and next another menu is showed up with the sides that the user must select. Except for the square the kind of each side ("up", "down", etc.) is not coincide with that on that is written on the menu, for example we may select a boundary condition in the "down" side but this corresponds visually to the left side.

Demo Thermal problem NURBS

In this demo problem we solve thermal problems with the use of IGA. After selecting the tab, a list with materials is showed up which is connected to the file "matData.xlsx". By choosing a material, next we can place the values of boundary conditions in the window that is following. Let that we examine the thermal conductivity in a square with Neumann conditions up and down, and Dirichlet left $T = 60$ and 20 right. After finishing this step, the analysis runs and the result are shown in a contour plot. The same steps we are following in "Demo Thermal Problem Bézier Extraction". In the "Demo Plane Stress Bézier Extraction" the only input from the user is the thickness of plate in this version of I3GA.

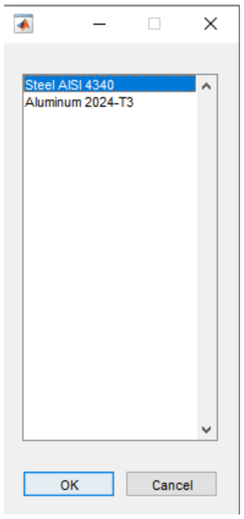
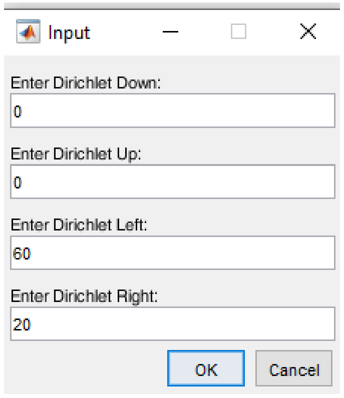


Figure C.33: Choosing material.



Enter Dirichlet Down:
0

Enter Dirichlet Up:
0

Enter Dirichlet Left:
60

Enter Dirichlet Right:
20

Figure C.34: Choosing boundary conditions.

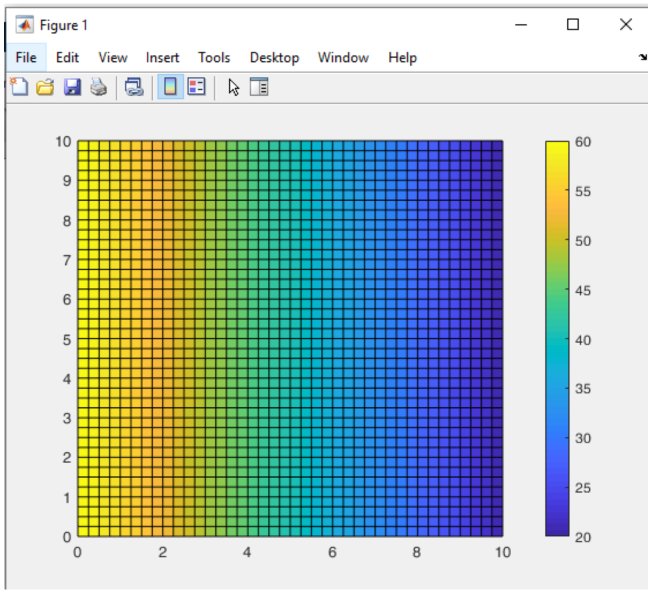
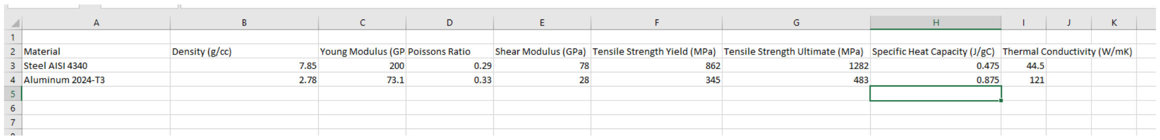


Figure C.35:Final result.

Material file "matData.xlsx"

In this file we can write the natural properties of the materials in format that can be written in MATLAB. The file is shown in the figure below:



	A	B	C	D	E	F	G	H	I	J	K
1											
2	Material	Density (g/cc)	Young Modulus (GP	Poissons Ratio	Shear Modulus (GPa)	Tensile Strength Yield (MPa)	Tensile Strength Ultimate (MPa)	Specific Heat Capacity (J/gC)	Thermal Conductivity (W/mK)		
3	Steel AISI 4340	7.85	200	0.29	78	862	1282	0.475	44.5		
4	Aluminum 2024-T3	2.78	73.1	0.33	28	345	483	0.875	121		
5											
6											
7											

Figure C.36: File "matData.xlsx".