Department of Structural Engineering
Faculty of Engineering Science and Technology
NTNU- Norwegian University of Science and Technology

ACCESSIBILITY
Open

MASTER THESIS 2011

SUBJECT AREA:	DATE:	NO. OF PAGES:
Structural Engineering	June 7th, 2011	50 + appendices

TITLE:

Isogeometric Finite Element Analysis based on Bézier Extraction of NURBS and T-splines

Isogeometrisk elementanalyse basert på Bézier ekstraksjon av NURBS og T-splines

BY:

Ole Jørgen Fredheim

SUMMARY:
This thesis will give a theoretical overview of B-splines, as well as NURBS and T-splines which are based on B-splines, and also the concept of Bezier decomposition of these spline functions. Bezier decomposition will decompose the splines into Bernstein polynomials which are defined over the domain of one quadrature element. This theoretical background will then be used to implement a Matlab isogeometric finite element analysis program. Two different choices for implementation are explored, a isogeometric finite element solver built from scratch for use of NURBS, and the use of Bezier extraction to implement isogeometric analysis with NURBS and T-splines in an already existing finite element solver. The main focus will be on use of Bezier extraction, which will significantly ease the implementation. Numerical studies are performed with problems of linear elasticity and heat conduction, to study the convergence of an isogeometric analysis. The accuracy of isogeometric analysis will prove to be better than for a traditional FEA for the analyzed problems.

RESPONSIBLE TEACHER: Kjell Magne Mathisen

SUPERVISOR(S): Kjell Magne Mathisen and Kjetil André Johannessen

CARRIED OUT AT: Department of Structural Engineering, NTNU

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY NTNU
FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY
DEPARTMENT OF STRUCTURAL ENGINEERING

TKT4915 Beregningsmekanikk, masteroppgave

Masteroppgave 2011

for

Ole Jørgen Fredheim

Isogeometrisk elementanalyse basert på Bézier ekstraksjon av NURBS og T-splines

Isogeometric Finite Element Analysis based on Bézier Extraction of NURBS and T-splines

Isogeometric analysis was introduced by Hughes *et al*. (2005) as a generalization of standard finite element analysis. In isogeometric analysis the solution space for dependent variables is represented in terms of the same functions which represent the geometry. The geometric representation is typically smooth, whereas the solution space for standard finite element analysis is continuous but not smooth. Adopting the isogeometric concept has shown computational advantages over standard finite element analysis in terms of accuracy in many application areas, including solid and structural mechanics. Most CAD systems use spline basis functions and often Non-Uniform Rational B-Splines (NURBS) of different polynomial order to represent geometry. In order to overcome the tensor product restrictions inherent in NURBS, T-spline basis has been used to generate analysis-suitable geometrical models of arbitrary topology.

The purpose of this master thesis is to study and demonstrate how isogeometric finite element analysis based on Bézier extraction of NURBS and T-splines can be implemented into a standard finite element code for 2D elasticity and heat conduction problems. The report should provide a review of the isogeometric concept in general and emphasize on the construction of isogeometric Bézier elements. In particular describe how the Bézier extraction operator can provide an element structure for isogeometric analysis similar to standard finite element analysis as oppose to conventional isogeometric analysis, and also how this tool enables analysis based on Bézier extraction of NURBS and T-splines. The study should emphasize theory and computational formulation of isogeometric analysis as well as demonstrate how isogeometric analysis compares to standard finite element analysis when solving problems in solid and structural mechanics.

The master thesis should be organized as a research report. It is emphasized that clarity and structure together with precise references are central requirements in writing a scientific report.

Advisors: Kjell Magne Mathisen and Kjetil André Johannessen

The master thesis should be handed in at the Department of Structural Engineering within June 14, 2011.

NTNU, January 17, 2011
Kjell Magne Mathisen
Principal Advisor

Abstract

This thesis will give a theoretical overview of B-splines, as well as NURBS and T-splines which are based on B-splines, and also the concept of Bézier decomposition of these spline functions. Bézier decomposition will decompose the splines into Bernstein polynomials which are defined over the domain of one quadrature element. This theoretical background will then be used to implement a Matlab isogeometric finite element analysis program. Two different choices for implementation are explored, a isogeometric finite element solver built from scratch for use of NURBS, and the use of Bézier extraction to implement isogeometric analysis with NURBS and T-splines in an already existing finite element solver. The main focus will be on use of Bézier extraction, which will significantly ease the implementation. Numerical studies are performed with problems of linear elasticity and heat conduction, to study the convergence of an isogeometric analysis. The accuracy of isogeometric analysis will prove to be better than for a traditional FEA for the analysed problems.

Preface

The following thesis is written as a completion of my masters degree in civil engineering at Department of Structural Engineering, NTNU, in the spring of 2011.

I would like to acknowledge the help from a number of people. First of all, my principal advisor Kjell Magne Mathiesen, whose help has been of great importance. His feedback and guidance throughout the work with this thesis and my project work last fall has been crucial. Also Kjetil André Johannessen deserves a great thank-you as he has been very helpful for the understanding of splines and isogeometric analysis. Trond Kvamsdal and Knut Morten Okstad has also been helpful. I would like to thank fellow student Thanh Ngan Nguyen for help with the implementation in Matlab, troubleshooting the program code, and for comparison of results. I will also thank Merete Berg, for reading and commenting my entire thesis.

Ole Jørgen Fredheim, Trondheim, June 7^{th} 2011.

Contents

Chapter 1

Introduction

1.1 Computer Aided Engineering

Use of Computer Aided Engineering (CAE) is almost unavoidable for engineers today. The complexity of the analysed problems makes hand calculations very impractical, and often impossible. Thus numerical solutions to the analysed problems must be obtained by use of computers.

Many different numerical solution methods exist, and the finite element method (FEM) is one frequently used method. This method discretize the analysed problem into smaller elements, where the governing equations are solved. The name "Finite Element Method" was coined by Ray W. Clough in 1960 [7], although "[...] it had essentially no impact on the civil engineering profession, mainly because the method could be applied effectively only by means of an automatic digital computer, and these were not readily available to typical structural engineers."[8] Since then, the computational power has increased tremendously, and finite element analysis can be performed everywhere.

Many different types of elements exist, but isoparametric elements has been predominant in finite element analysis since it's development in the 1960's [15]. The concept of isoparametric elements is that the mathematical functions used for the interpolation of the unknown variables are also used to describe the geometry of the element. Frequently utilized functions in a traditional FEM are the Lagrange and Hermite polynomials, which has the desired mathematical properties for a finite element analysis (FEA). However, these polynomials are not able to represent all geometries such as conical sections, and this will lead to a geometric error in the model.

1.2 Computer Aided Design

Computer Aided Design (CAD) or Computer Aided Geometric Design (CAGD) is also a necessary tool for an engineer. The design process often start by first creating a CAD model of the construction, and the subsequent analysis is done with the information from this model.

The CAD systems used today has its origins in the work of two French automotive engineers, Pierre Bézier of Renault and Paul de Faget de Casteljau of Citroen [13]. Bézier utilized the Bernstein polynomials to generate curves

and surfaces. In 1972, B-splines was established by Riesenfield [20], and NURBS followed in 1975 by Versprille [28].

Even though there has been continuous development in CAD, and other geometric representations has been developed, such as subdivision surfaces, NURBS has retained its dominant position for engineering design.

1.3 Isogeometric Analysis

Because CAE and CAD was developed independently, they are not compatible. The geometry is described differently, as the purpose of CAE and CAD has been different. This leads to an extensive amount of overlapping work done, as most of the time a structure is first modelled as a CAD model, and then remodelled in order to perform a finite element analysis. Isogeometric analysis is a way to integrate these two systems, and allow for the CAD-models to be used in the FEA.

Isogeometric analysis was introduced by Hughes *et al.* [14, 13]. The concept of isogeometric analysis is to use the same basis for the analysis as is being used in description of the geometry. This as opposed to the traditional finite element method, where the basis for the analysis is what is used to describe the geometry. But still, isogeometric elements are also isoparametric elements, since the basis for geometry and analysis is the same. Since the NURBS that are being used in the CAD software are able to provide an exact representation of the geometry, these functions are used as the basis for the analysis. This choice of basis is necessary to use the imported geometry from CAD in the analysis.

Isogeometric analysis is being extensively studied, and is being applied to many different fields of analysis. For structural analysis purposes, isogeometric analysis has been applied to, for instance, shells [2, 3, 16], fluid-structure interaction [25] and finite deformation of elastoplastic solids[19].

1.4 State of the Art

T-splines is a recent development in CAGD which allows for local refinement and fewer control points [24, 23], and these properties are also interesting for isogeometric analysis purposes [12, 1]. An isogeometric analysis demand extensive changes to the finite element framework, and Bézier extraction has been proposed as a method to ease the implementation for both NURBS [5] and T-splines [22]. T-splines has been proven linearly independent for a certain class of T-spline geometry [6, 21], but in the general case this is not true, and a linear dependence between basis functions result in T-splines being useless for analysis purposes. Locally refined B-splines, in short LR B-spline, is a proposed solution to this issue [17]

1.5 This thesis

This thesis will begin with an thorough introduction to splines, by first introducing B-splines and NURBS in Chapter 2, and then expanding this to T-splines in Chapter 3. This theoretical understanding of splines will be necessary when introducing Bézier extraction of NURBS and T-splines in Chapter 4, which will

be used as the basis for isogeometric analysis. In Chapter 5 it is discussed how to construct an isogeometric finite element solver for NURBS, and also how to implement NURBS and T-splines in an existing finite element code by the use of Bézier extraction operators. Numerical studies of two linear elastic, static problems will be presented in Chapter 6, as well as a thermo-mechanical problem. In the appendices some key Matlab algorithms are given. Also, a report co-authored for the "MekIT'11 Sixth National Conference on Computational Mechanics", and the corresponding presentation given at the conference, is presented.

Throughout the thesis the discussion will be restricted to two spatial dimensions, and only surfaces will be discussed. The theory presented here will be enough for an expansion to three spatial dimensions and solids, but this will not handled here, as the concepts are better understood in 2d. For the numerical examples only in-plane forces are considered, and all examples are in a state of either plane stress or plane strain. The basic theory of FEM will be omitted, but this can be found in any textbook, for instance Cook *et.al.* [9].

Some of the key concepts will be explained with illustrative examples, in order to provide some additional clarity to the theory discussed.

Chapter 2

B-splines

B-splines, or basis-splines, is the basis for both NURBS and T-splines, and it is necessary to understand B-splines in order to understand NURBS and T-splines which will be discussed later. B-splines will in this chapter be introduced as a basis for analysis, and the framework needed to manipulate these functions will be established, before expanding the knowledge to NURBS, and in the next chapter, T-splines.

2.1 Knot vector

A *knot vector* is a set of increasing parameter space coordinates. *Parameter space* is an imaginary space where the basis functions are defined, and is partitioned into knot spans between the knots. The knot vector is written as $\Xi = [\xi_1, \xi_2, \ldots, \xi_{n+p+1}]$, where ξ_i is the i^{th} knot, i is the knot index, $i = 1, 2, \ldots, n + p + 1$, p is the polynomial order, and n is the number of basis functions used to create the B-spline curve. The knot vector is the basis needed to generate B-splines.

In the knot vector several knot values may be equal, and the *multiplicity* is then the number of repeated knots. An increased multiplicity of knots will be reflected in the basis functions, as this multiplicity will cause a decrease of the continuity at the corresponding knots. If the multiplicity at the ends is equal to *p+1*, the knot vector is said to be open. A basis spline constructed from an open knot vector is discontinuous at the end, and a curve created from such a basis spline will be interpolatory at the end control points.

2.2 B-splines

B-splines are piecewise polynomial functions, and are defined by the *Cox-de Boor recursion formula* [10, 11]:

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{2.1}$$

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi \in [\xi_i, \xi_{i+1}) \\ 0 & \text{else} \end{cases} \tag{2.2}$$

In addition to (2.1) and (2.2), the restriction is imposed that if a denominator is equal to zero, then the respective term in the equation is defined to be zero.

A B-spline function of polynomial order p=0 or p=1 will be the same as the respective Lagrange shape functions used in classical FEM. Therefore an analysis with linear elements, such as the 4-noded quadrilateral, will be identical to an analysis with a classical FEM For any order p, the B-spline basis will form a partition of unity, as the Lagrangian functions do.

2.2.1 Example: Quadratic B-spline basis functions

To better understand the nature of B-splines an example is necessary. It is here shown how to build the quadratic basis functions from the open knot vector $\Xi = [0, 0, 0, 1, 2, 2, 3, 3, 3]$. First construct the piecewise constant (p=0) basis functions, then use these to create the linear (p=1) basis functions, and lastly the quadratic (p=2) basis functions can be created. From (2.2) we have that

$$N_{1,0}(\xi) = \begin{cases} 1 & \text{if } \xi \in [\xi_1, \xi_2) \\ 0 & \text{else} \end{cases} \tag{2.3}$$

From the knot vector Ξ we see that $\xi_1 = \xi_2 = 0$ thus $N_{1,0} \equiv 0$. (2.2) is applied to all indices and we obtain the piecewise constant basis functions:

$$\begin{aligned}
N_{1,0}(\xi) &= 0 \\
N_{2,0}(\xi) &= 0 \\
N_{3,0}(\xi) &= \begin{cases} 1 & \text{if } \xi \in [0, 1) \\ 0 & \text{else} \end{cases} \\
N_{4,0}(\xi) &= \begin{cases} 1 & \text{if } \xi \in [1, 2) \\ 0 & \text{else} \end{cases} \\
N_{5,0}(\xi) &= 0 \\
N_{6,0}(\xi) &= \begin{cases} 1 & \text{if } \xi \in [2, 3) \\ 0 & \text{else} \end{cases} \\
N_{7,0}(\xi) &= 0 \\
N_{8,0}(\xi) &= 0
\end{aligned} \tag{2.4}$$

Now use (2.1) to build the linear basis functions $N_{i,1}$ from the piecewise constant ones. For i=1 we get

$$N_{1,1}(\xi) = \frac{\xi - 0}{0 - 0} N_{1,0}(\xi) + \frac{0 - \xi}{0 - 0} N_{2,0}(\xi) \tag{2.5}$$

Since the fraction is defined to be zero in cases where the denominator equals zero, $N_{1,1}(\xi) = 0$. For i=2 we get

$$N_{2,1}(\xi) = \frac{\xi - 0}{0 - 0} N_{2,0}(\xi) + \frac{1 - \xi}{1 - 0} N_{3,0}(\xi) \tag{2.6}$$

Recalling that $N_{3,0} = \begin{cases} 1 & \text{if } \xi \in [0, 1) \\ 0 & \text{else} \end{cases}$, (2.6) reads

$$N_{2,1}(\xi) = \begin{cases} 1 - \xi & \text{if } \xi \in [0, 1) \\ 0 & \text{else} \end{cases} \tag{2.7}$$

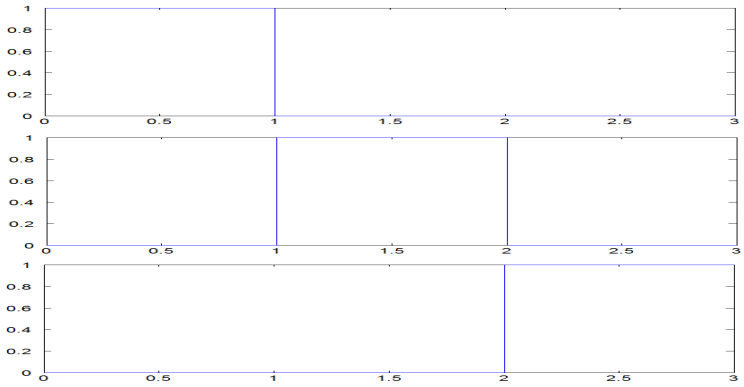
The other linear functions are obtained from the same procedure

$$
\begin{aligned}
N_{1,1}(\xi) &= 0 \\
N_{2,1}(\xi) &= \begin{cases} 1-\xi & \text{if } \xi \in [0,1) \\ 0 & \text{else} \end{cases} \\
N_{3,1}(\xi) &= \begin{cases} \xi & \text{if } \xi \in [0,1) \\ 2-\xi & \text{if } \xi \in [1,2) \\ 0 & \text{else} \end{cases} \\
N_{4,1}(\xi) &= \begin{cases} \xi-1 & \text{if } \xi \in [1,2) \\ 0 & \text{else} \end{cases} \\
N_{5,1}(\xi) &= \begin{cases} 3-\xi & \text{if } \xi \in [2,3) \\ 0 & \text{else} \end{cases} \\
N_{6,1}(\xi) &= \begin{cases} \xi-2 & \text{if } \xi \in [2,3) \\ 0 & \text{else} \end{cases} \\
N_{7,1}(\xi) &= 0
\end{aligned} \tag{2.8}
$$

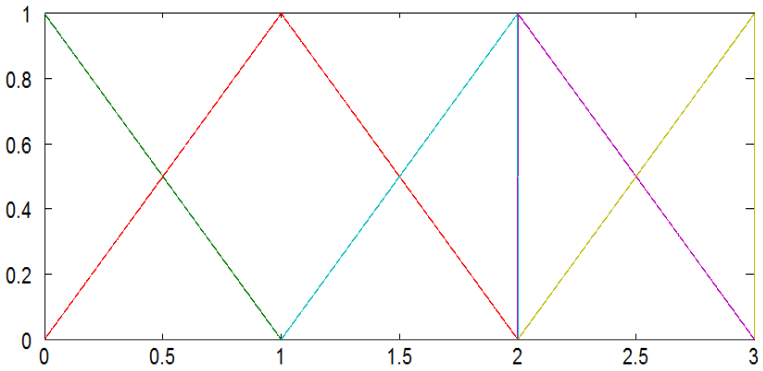
The example is completed by building the quadratic basis functions, in the same fashion as for the linear basis functions

$$
\begin{aligned}
N_{1,2}(\xi) &= \frac{\xi-0}{0-0}N_{1,1}(\xi) + \frac{1-\xi}{1-0}N_{2,1}(\xi) \\
&= \begin{cases} (1-\xi)^2 & \text{if } \xi \in [0,1) \\ 0 & \text{else} \end{cases} \\
N_{2,2}(\xi) &= \begin{cases} 3\xi - 2\xi^2 & \text{if } \xi \in [0,1) \\ (2-\xi)(1-\xi) & \text{if } \xi \in [1,2) \\ 0 & \text{else} \end{cases} \\
N_{3,2}(\xi) &= \begin{cases} \frac{1}{2}\xi^2 & \text{if } \xi \in [0,1) \\ 4\xi - \frac{3}{2}\xi^2 - 2 & \text{if } \xi \in [1,2) \\ 0 & \text{else} \end{cases} \\
N_{4,2}(\xi) &= \begin{cases} (\xi-1)^2 & \text{if } \xi \in [1,2) \\ (3-\xi)^2 & \text{if } \xi \in [2,3) \\ 0 & \text{else} \end{cases} \\
N_{5,2}(\xi) &= \begin{cases} 10\xi - 2\xi^2 - 12 & \text{if } \xi \in [2,3) \\ 0 & \text{else} \end{cases} \\
N_{6,2}(\xi) &= \begin{cases} (\xi-2)^2 & \text{if } \xi \in [2,3) \\ 0 & \text{else} \end{cases}
\end{aligned} \tag{2.9, 2.10}
$$

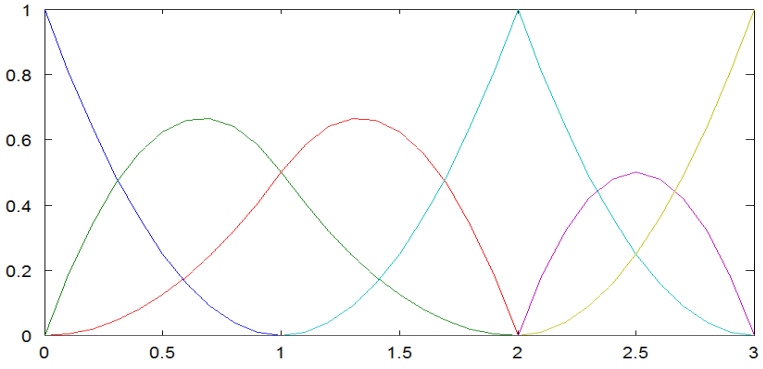
The piecewise constant, linear and quadratic basis functions are plotted in Figure 2.1.



(a) The nonzero 0^{th} order basis functions



(b) Linear basis functions



(c) Quadratic basis functions

Figure 2.1: B-spline basis functions of degree 0 to 2 for $\Xi = [0, 0, 0, 1, 2, 2, 3, 3, 3]$

2.3 Derivatives of B-spline basis functions

The derivatives of the B-splines are defined by a recursive function, as the B-splines also are. For a given polynomial order, p, and knot vector, Ξ, the derivative of the i^{th} basic function is given by:

$$\frac{d}{d\xi}N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}N_{i+1,p-1}(\xi) \tag{2.11}$$

This equation can be generalized to higher order derivatives by differentiating each side, which leads to the following equation

$$\frac{d^k}{d^k\xi}N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}\left(\frac{d^{k-1}}{d^{k-1}\xi}N_{i,p-1}(\xi)\right) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}\left(\frac{d^{k-1}}{d^{k-1}\xi}N_{i+1,p-1}(\xi)\right) \tag{2.12}$$

2.4 Anchors

An open knot vector with *n+p+1* knots will give n basis functions, thus it is not a one-to-one correspondence between knots and basis functions. In order to locate the basis functions in the parametric space, the *anchor* t_i is defined as the parametric coordinate to the center of support for basis function $N_{i,p}$.

$$t_i = \begin{cases} \xi_{i+(p+1)/2} & \text{if } p \text{ is odd,} \\ \frac{1}{2}(\xi_{i+(p/2)} + \xi_{i+(p/2)+1}) & \text{if } p \text{ is even} \end{cases} \tag{2.13}$$

For an odd polynomial degree the anchor will be at the center of a knot, while in the case of even polynomial degree the anchor will be positioned in the center of a knot span. In Figure 2.2 anchors of even and odd degree is shown along with the corresponding B-spline basis functions. B-splines can easily be described without the notion of anchors, but they will be helpful when T-splines are discussed.

2.5 B-spline curves

B-Spline curves are created similarily as in classical FEA, by a linear combination of B-spline basis functions. What separates B-spline curves from curves in FEA is that instead of interpolating a set of nodal points, the B-splines are related to a set of *control points*. These control points are the equivalent to the nodes, but the curve will generally not pass through the control points. For a given set of n p^{th} order basis functions, $N_{i,p}(\xi)$, i=1,2...,n, and a corresponding set of control points $\mathbf{B_i} \in \mathbb{R}^d$,i=1,2,...,n, the piecewise-polynomial B-spline curve is given by

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)\mathbf{B_i} \tag{2.14}$$

Since there is a one-to-one correspondence between the control points and the basis functions, there is also a one-to-one correspondence between the control points and the anchors. The continuity of the B-spline curve is the same as for the basis functions of which it is constructed, so that the curve is C^{p-1}-continuous everywhere except at knots, where the curve is C^{p-m}-continuous.

Thus any B-spline curve constructed from an open knot vector will always be C^{-1}-continuous at the ends, since the multiplicity at the ends are p+1.

2.5.1 Example: B-Spline curve from quadratic basis functions

In this example the quadratic basis functions, $N_{i,2}(\xi)$, i=1,2...,6, which were created in the previous example, are being used to generate a 2-dimensional B-spline curve. In order to do this, the corresponding control points, $\mathbf{B_i} \in \mathbb{R}^2$, i=1,2,...,6, is needed. If the control points is chosen to be (0,0),(0,1),(1,2),(1,0),(0.5,-1), and (0,-1), the resulting curve will be the one shown in Figure 2.3.

It is clear from the figure how the multiplicity of the knot vector is influencing the curve. At control point number 1 and 6 the curve is discontinuous, due to the multiplicity being p+1, and at control point number 4 the curve is C^0-continuous since the multiplicity equals p at this control point. These 3 control points are also the only control points that are interpolating the curve, which can easily be predicted by looking at the basis functions in Figure 2.1c where the value of the corresponding basis function is 1, and all others equal 0.

2.6 B-spline surfaces

The expansion from B-Spline curves to B-Spline surfaces is fairly straight forward. To generate a surface, a net of control points $\{\mathbf{B_{i,j}}\}$, i=1,2,...,n, j=1,2,...,m, polynomial orders p and q, and knot vectors $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, and $\mathcal{H} = \{\eta_1, \eta_2, \ldots, \eta_{m+q+1}\}$ is needed. A tensor product B-spline surface is then defined by

$$\mathbf{S}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \mathbf{B_{i,j}} \tag{2.15}$$

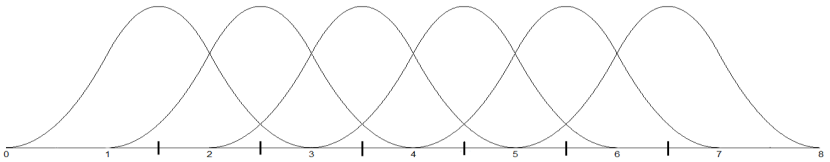
where $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$ are univariate B-spline basis functions of order p and q, corresponding to knot vectors Ξ and H, respectively. An example of a B-spline surface is given in Figure 2.4.

2.7 Knot insertion

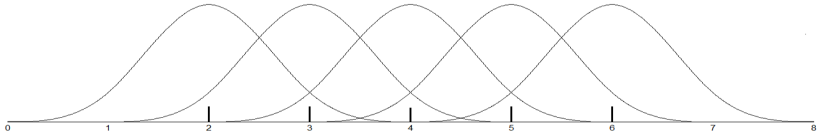
Knots may be inserted in the knot vector without altering the B-spline curve if the new control points $\bar{\mathbf{P}}$ are placed according to a specific procedure, known as knot insertion [4]. Knot insertion is what allows for refinement of element meshes and Bézier decomposition.

$$\bar{P}_A = \begin{cases} P_1 & A = 1 \\ \alpha_A P_A + (1 - \alpha_A) P_A & 1 < A < m \\ P_n & A = m \end{cases} \tag{2.16}$$

$$\alpha_A = \begin{cases} 1 & 1 < A < k - p \\ \frac{\bar{\xi} - \xi_A}{\xi_{A+p} - \xi_A} & k - p + 1 < A < k \\ 0 & A \geq k + 1 \end{cases} \tag{2.17}$$



(a) Anchors for even degree, p=2



(b) Anchors for odd degree, p=3

Figure 2.2: Anchors for some B-spline basis functions

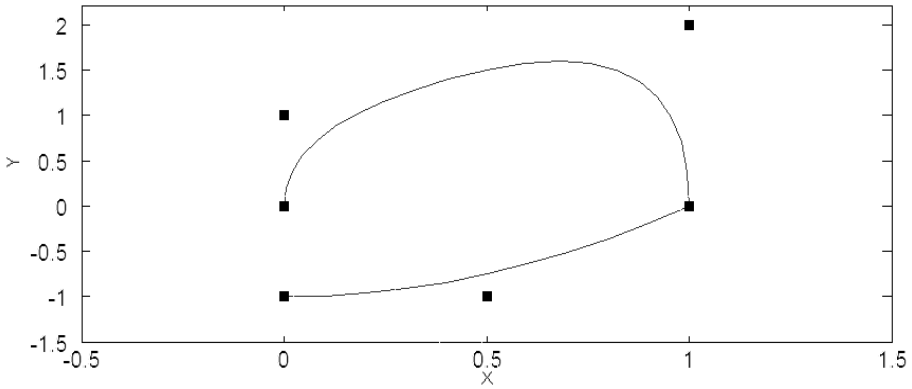


Figure 2.3: B-Spline curve constructed from quadratic basis functions, and knot vector $\Xi = [0, 0, 0, 1, 2, 2, 3, 3, 3]$. The control points are marked as the dots

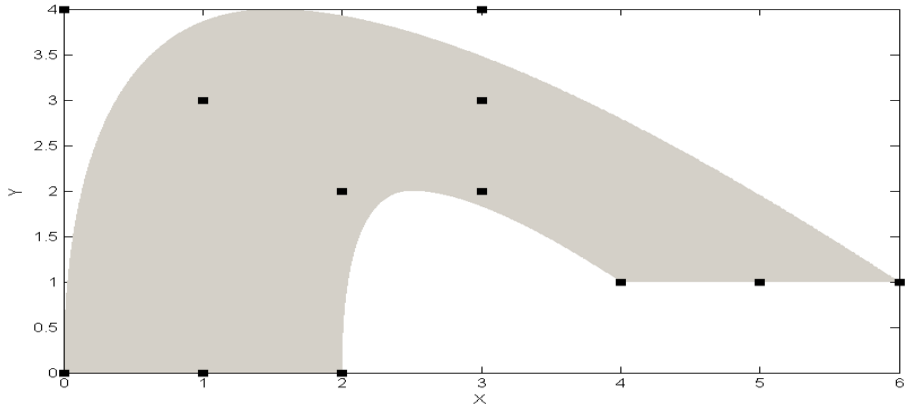


Figure 2.4: B-Spline surface constructed from quadratic basis functions, and knot vectors $\Xi = [0, 0, 0, 0.5, 1, 1, 1]$ and $H = [0, 0, 0, 1, 1, 1]$. The control points are marked with dots

2.8 Non-Uniform Rational B-splines

Non-Uniform Rational B-splines (NURBS) can be understood by investigating them from both a geometric and a algebraic perspective. From the geometric perspective, a NURBS entity in $\mathbb{R}^d$ is the result of a projective transformation of a B-Spline entity in $\mathbb{R}^{d+1}$. For instance, a 3-dimensional B-Spline curve projected onto the plane z=1 will create in a 2-dimensional NURBS curve, as seen in Figure 2.5

$$(\mathbf{B}_i)_j = (\mathbf{B}_i^w)_j / w_i, \ j = 1, \ldots, d \tag{2.18}$$

$$w_i = (\mathbf{B}_i^w)_{d+1} \tag{2.19}$$

where w_i is the *weight* of the control point. This transformation will be applied to all points on the curve, by dividing by the *weighting function* defined as

$$W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) w_i \tag{2.20}$$

Now the curve can be written in terms of the B-spline curve as

$$(\mathbf{C}(\xi))_j = \frac{(\mathbf{C}^w(\xi))_j}{W(\xi)} \tag{2.21}$$

In order to being able to manipulate the NURBS basis functions it is necessary to understand the algebraic perspective. The univariate rational basis function $R_{i,p}(\xi)$ is given by

$$R_{i,p}(\xi) = \frac{N_{i,p}(\xi) w_i}{W(\xi)} = \frac{N_{i,p}(\xi) w_i}{\sum_{\tilde{i}=1}^{n} N_{\tilde{i},p}(\xi) w_{\tilde{i}}} \tag{2.22}$$

This equation is used to express the equation of a curve, equivalently to how a B-spline curve is defined:

$$C(\xi) = \sum_{i=1}^{n} \mathbf{P}_i R_{i,p}(\xi) \tag{2.23}$$

This form is exactly the same as (2.21), but it is now written in a form which is easier to manipulate, and it is this form that will be used for analysis purposes. The bivariate tensor product basis functions for a surface is defined as

$$R_{i,j}^{p,q}(\xi, \eta) = \frac{N_{i,p}(\xi) M_{j,q}(\eta) w_{i,j}}{\sum_{\hat{i}=1}^{n} \sum_{\hat{j}=1}^{m} N_{\hat{i},p}(\xi) M_{\hat{j},q}(\eta) w_{\hat{i},\hat{j}}} = \frac{N_A(\xi, \eta) w_A}{W(\xi, \eta)} \tag{2.24}$$

$\mathbf{W}$ is defined as the diagonal matrix of weights

$$\mathbf{W} = w_i \delta_{ij} \tag{2.25}$$

and (2.22) and (2.24) are rewritten in matrix form as

$$\mathbf{R}(\xi) = \frac{1}{W(\xi)} \mathbf{W} \mathbf{N}(\xi) \tag{2.26}$$

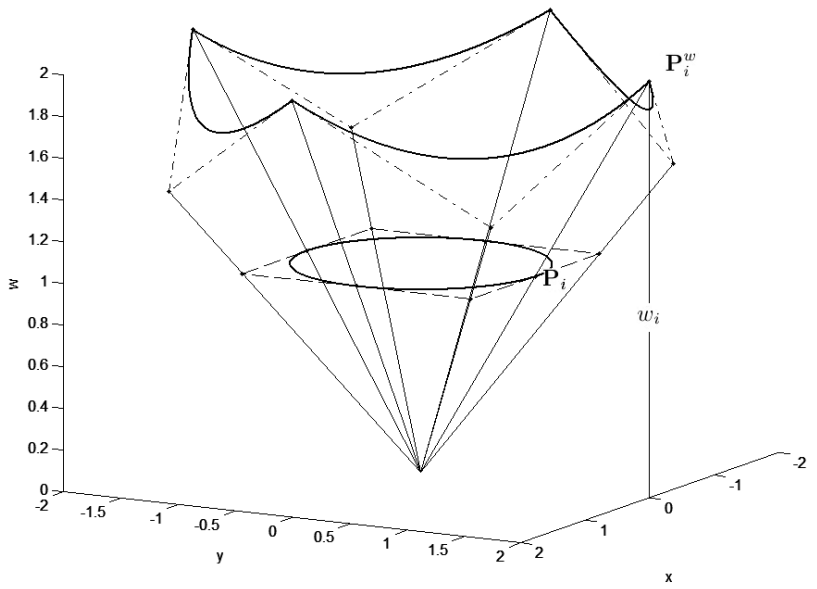


Figure 2.5: B-Spline curve projected onto the plane z=1 to create the NURBS representation of a circle

and

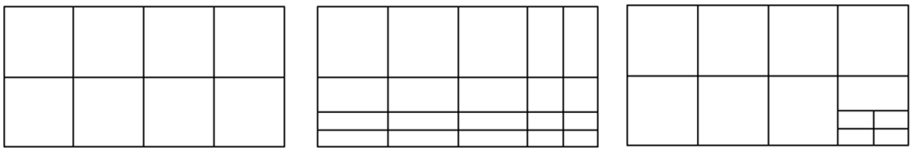
$$\mathbf{R}(\xi,\eta) = \frac{1}{W(\xi,\eta)}\mathbf{WN}(\xi,\eta) \tag{2.27}$$

Since all NURBS are a projective transformation of a B-spline, all algorithms that are to be applied to the NURBS must be applied to the B-spline, and this is done by first projecting the NURBS into d+1 dimensions, then modify the B-spline, and finally project back to d dimensions. This is done when inserting new knots, and also for Bézier decomposition, which is discussed in Chapter 4.

Chapter 3

T-splines

T-splines was first introduced by Sederberg *et.al.* 2003 [24] as a generalization of NURBS, and has recently also been introduced in an analysis setting [12, 1]. As opposed to B-splines and NURBS, T-splines are not restricted to a tensor product structure. That is, while NURBS control points must lie in a rectangular grid, T-splines may form an incomplete grid, with *T-junctions*. The tensor product structure of NURBS implicate that a local refinement is impossible, since complete rows and columns must be inserted in the control point grid, which will create unwanted degrees of freedom elsewhere in the analysed problem, see Figure 3.1. Since this is avoided with T-splines, they are superior to NURBS when it comes to local refinement. In this chapter introduce T-splines and its properties are introduced, after first introducing PB-splines.



(a) No refinement (b) Tensor product refinement with NURBS (c) Local refinement with T-splines

Figure 3.1: Tensor product refinement vs. local refinement

3.1 Point-based B-splines

As discussed earlier, a B-spline surface originates from the index space defined by Ξ and $\mathcal{H}$, which will create a rectangular grid. Point-based B-splines, or *PB-splines*, will remove this tensor product restriction. Unlike B-splines the PB-splines are not related to a grid but to independent points and knot vectors. For each PB-spline there will be a local knot vector in each parametric direction. As discussed in Chapter 2, B-Spline basis functions are only dependent on the global knot vector, and all B-spline functions can be calculated from this knot vector. By further inspection one will realise that each of the basis functions are only influenced by a certain set of knots. For instance, if the knot vector

$\Xi = [0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4]$ is given, there will be 7 cubic B-splines, and each B-spline will correspond to a *local knot vector* with length *p+2*. The basis function $N_{1,3}(\xi)$ will depend on the local knot vector $\Xi_1 = [0, 0, 0, 0, 1]$, $N_{2,3}(\xi)$ will depend on $\Xi_2 = [0, 0, 0, 1, 2]$, $N_{3,3}(\xi)$ will depend on $\Xi_2 = [0, 0, 1, 2, 3]$, and so on. With this in mind, it is obvious that a global knot vector is not needed in order to create a spline, and this is what allows the creation of PB-splines.

By removing the limitation to global knot vector, and only locking at individual knot vectors for each spline, PB-splines are created. PB-splines is a collection of *blending functions*, N_i, which are B-spline basis functions created from the local knot vectors Ξ_i and $\mathcal{H}_i$. The equation for a PB-spline is

$$\mathbf{P}(\xi, \eta) = \frac{\sum_{i=1}^{n} \mathbf{P}_i N_i(\xi, \eta)}{\sum_{i=1}^{n} N_i(\xi, \eta)}, \quad (\xi, \eta) \in \mathbf{D} \tag{3.1}$$

where $\mathbf{D}$ is the domain the PB-spline is defined in, where $\sum_{i=1}^{n} N_i(\xi, \eta) > 0$.

Each of the knot vectors Ξ_i and $\mathcal{H}_i$ are independent of each other, and may be comprised of the same knot values. Since the knot vectors Ξ_i and $\mathcal{H}_i$ are independent from Ξ_j and $\mathcal{H}_j$ if $i \neq j$, the blending functions N_i and N_j will no longer be linearly independent. This is why they are referred to as blending functions and not basis functions, as a basis infer linear independence. In a finite element setting, PB-splines are not suitable, as basis functions are needed, but this will be resolved by T-splines.

3.2 T-splines

T-splines will be discussed as an extension of B-splines, but all concepts also apply to NURBS, such that T-splines can be rational functions. T-splines are PB-splines with some restrictions imposed on the control points, which now are organised in the *T-mesh*. The T-mesh is the index space representation of the control point net. Often the term T-mesh is used to describe both the index space representation and the physical representation of the control point mesh. Here T-mesh will always refer to the index space, as the index space offer a more intuitive understanding. The T-mesh is essentially a rectangular grid which do not need to have complete rows and columns, thereby allowing *T-junctions*. Each of the lines in the T-mesh is referred to as a ξ-edge with constant ξ-value, or a η-edge with constant η-value. A T-junction can then be described as vertex shared by two ξ-edges and one η-edge, or vice versa. Each edge is labelled with a knot interval, constrained by the following rules: [24]

- **Rule 1**: The sum of knot intervals on opposing edges of any face must be equal.
- **Rule 2**: If a T-junction on one edge of a face can be connected to a T-junction on an opposing edge of the face (thereby splitting the face into two faces) without violating Rule 1, that edge must be included in the T-mesh.

It is from the T-mesh the local knot vectors, Ξ_i and $\mathcal{H}_i$, are extracted for each basis function N_i. For T-splines of odd degree each of the vertices in the T-mesh is an anchor for a control point, and for even degree the anchors will be at the center of every cell. Figure 3.2 shows an example of a T-mesh.

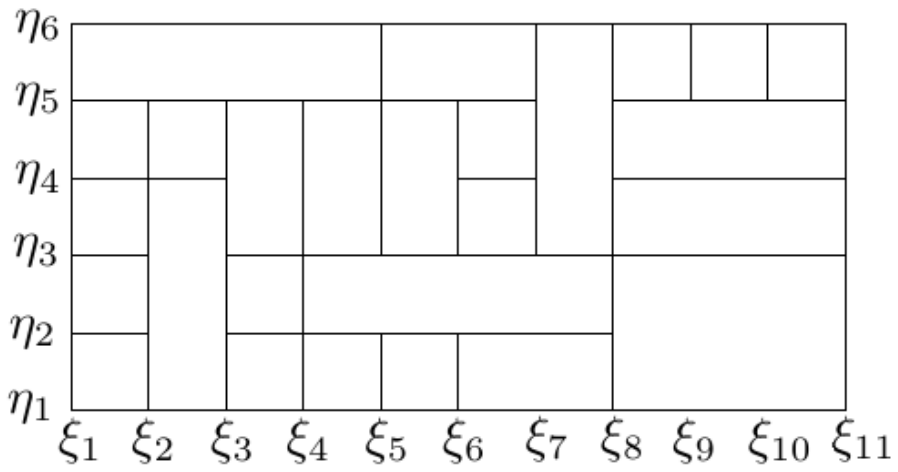


Figure 3.2: Example T-mesh

3.3 Local knot vectors

The local knot vectors for each anchor will be extracted from the T-mesh. From here on local knot vector will only be referred to as knot vectors, as all knot vectors are local in a T-spline setting. Since the location of the anchors differ between odd and even T-splines, the cases of odd or even order must be treated differently. The case of odd degree will be discussed first. The first value added to the knot vector is the coordinate value of the anchor itself, this is placed at the center of the knot vector. For the anchor P_1 in Figure 3.3a, this will give us $\Xi_1 = \{.,.,\xi_8,.,.\}$ and $\mathcal{H}_1 = \{.,.,\eta_5,.,.\}$. The remaining values are found by traversing along the edges $\xi = \xi_8$ and $\eta = \eta_5$ until (p+1)/2 intersecting edges are crossed, and filling in the corresponding knot values in the knot vector. For Ξ_1, first march right to intersect ξ_9 and ξ_{10}, and add these to the knot vector, $\Xi_1 = \{.,.,\xi_8,\xi_9,\xi_{10}\}$. Then march left, to intersect ξ_7 and ξ_6, to obtain the complete knot vector $\Xi_1 = \{\xi_6,\xi_7,\xi_8,\xi_9,\xi_{10}\}$. For $\mathcal{H}_1$, first march upwards and intersect the line at η_6. This is at the edge of the mesh with no more intersecting lines to cross. In this case the last intersected knot value is added until the knot vector is filled up, thus resulting in $\mathcal{H}_1 = \{.,.,\eta_5,\eta_6,\eta_6\}$. Then march downwards, intersecting η_4 and η_3, and the knot vectors for anchor P_1 becomes

$$\Xi_1 = \{\xi_6,\xi_7,\xi_8,\xi_9,\xi_{10}\} \tag{3.2}$$

$$\mathcal{H}_1 = \{\eta_3,\eta_4,\eta_5,\eta_6,\eta_6\} \tag{3.3}$$

For the anchor P_2 shown in Figure 3.3b this procedure will result in the knot vectors

$$\Xi_2 = \{\xi_5,\xi_6,\xi_8,\xi_{11},\xi_{11}\} \tag{3.4}$$

$$\mathcal{H}_2 = \{\eta_1,\eta_1,\eta_2,\eta_3,\eta_4\} \tag{3.5}$$

In the case of even order the knot vector extraction is only slightly different. Since the anchor is in the middle of a cell, the coordinates of the anchor is

omitted from the knot vector. The elements of the knot vector are found by adding the first (p/2)+1 lines encountered in each direction, as done for the odd case. For the anchors shown in Figure 3.4a and 3.4b, the following knot vectors are obtained:

$$\Xi_1 = \{\xi_3, \xi_4, \xi_8, \xi_{11}\} \tag{3.6}$$

$$\mathcal{H}_1 = \{\eta_1, \eta_2, \eta_3, \eta_4\} \tag{3.7}$$

and

$$\Xi_2 = \{\xi_1, \xi_1, \xi_2, \xi_{31}\} \tag{3.8}$$

$$\mathcal{H}_2 = \{\eta_3, \eta_4, \eta_5, \eta_6\} \tag{3.9}$$

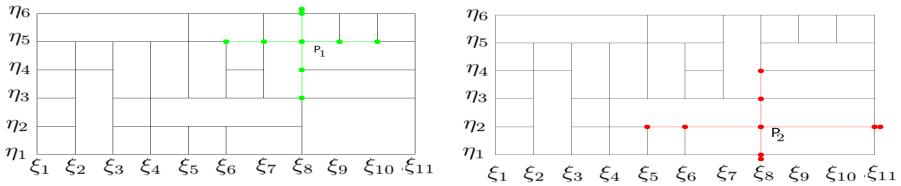
The traversing from the T-junctions will cause *T-junction extension* which are lines of reduced continuity. T-junction extensions will be discussed closer in section 3.6.

3.4 Generating a T-spline surface

Consider the T-mesh shown in Figure 3.5a. If a T-spline of quadratic order is modelled from this, there will be 16 anchors t_i, corresponding to control point $\mathbf{P}_i$ and basis function $N_{i,2}$. The anchors are shown in the same figure. For each anchor, the knot vectors Ξ_i and $\mathcal{H}_i$ shown in Table 3.1 are extracted. With the knot vectors for each anchor the basis functions corresponding to each set of knot vectors can be computed, as shown in Figure 3.5b. Note the change in continuity of the basis functions across the line $\xi = 0.5$. The added zero valued knot interval in the T-mesh results in a C^0-continuity of the basis functions for the anchors affected by this knot, while the basis functions has no continuity reductions in other areas of the T-mesh. This leads to an abrupt change of continuity in the resulting T-spline surface as shown in Figure 3.5c. The surface is C^∞ along y=-1, since the basis functions $N_{i,2}, \quad i = 7, \ldots, 16$ which has reduced continuity equals zero along this edge. Along along y=2 it is C^0-continuous at x=0.

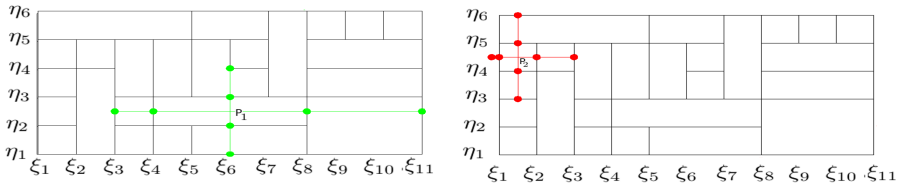
3.5 Element mesh

In a finite element analysis the geometry needs to be meshed into finite elements, where the numerical quadrature is being performed, and the T-mesh cannot be directly used for this purpose. An *element* is defined as the domain over which the T-splines are C^∞-continuous. A new mesh will need to be defined, called the *extended* or *elemental* T-mesh. This mesh includes all lines of reduced continuity, and the lines defining the T-mesh. Every element will then be bounded by the lines in the extended T-mesh. This mesh will only contain rectangular elements, even if there are L-shaped areas in the T-mesh. The lines of reduced continuity originates from the T-junction extensions. The extended T-mesh for cubic T-splines formed from the T-mesh in Figure 3.2 is shown in Figure 3.6.



(a) Anchor P_1 (b) Anchor P_2

Figure 3.3: Knot vector extraction for anchors of odd degree, p=3

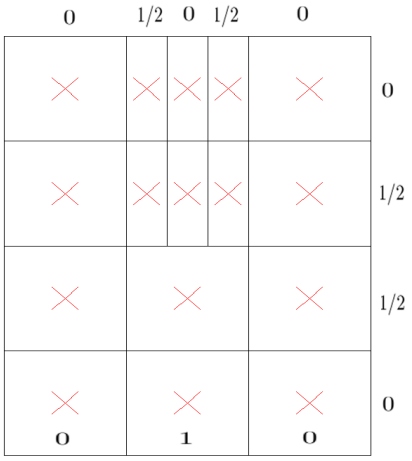


(a) Anchor P_1 (b) Anchor P_2

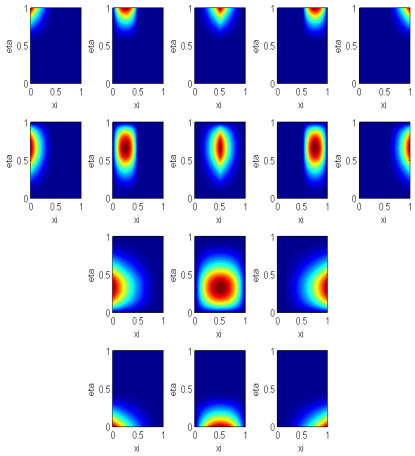
Figure 3.4: Knot vector extraction for even degree, p=2

i	Ξ_i	$\mathcal{H}_i$
1	{0, 0, 0, 1}	{0, 0, 0, 0.5}
2	{0, 0, 1, 1}	{0, 0, 0, 0.5}
3	{0, 1, 1, 1}	{0, 0, 0, 0.5}
4	{0, 0, 0, 1}	{0, 0, 0.5, 1}
5	{0, 0, 1, 1}	{0, 0, 0.5, 1}
6	{0, 1, 1, 1}	{0, 0, 0.5, 1}
7	{0, 0, 0, 0.5}	{0, 0.5, 1, 1}
8	{0, 0, 0.5, 0.5}	{0, 0.5, 1, 1}
9	{0, 0.5, 0.5, 1}	{0, 0.5, 1, 1}
10	{0.5, 0.5, 1, 1}	{0, 0.5, 1, 1}
11	{0.5, 1, 1, 1}	{0, 0.5, 1, 1}
12	{0, 0, 0, 0.5}	{0.5, 1, 1, 1}
13	{0, 0, 0.5, 0.5}	{0.5, 1, 1, 1}
14	{0, 0.5, 0.5, 1}	{0.5, 1, 1, 1}
15	{0.5, 0.5, 1, 1}	{0.5, 1, 1, 1}
16	{0.5, 1, 1, 1}	{0.5, 1, 1, 1}

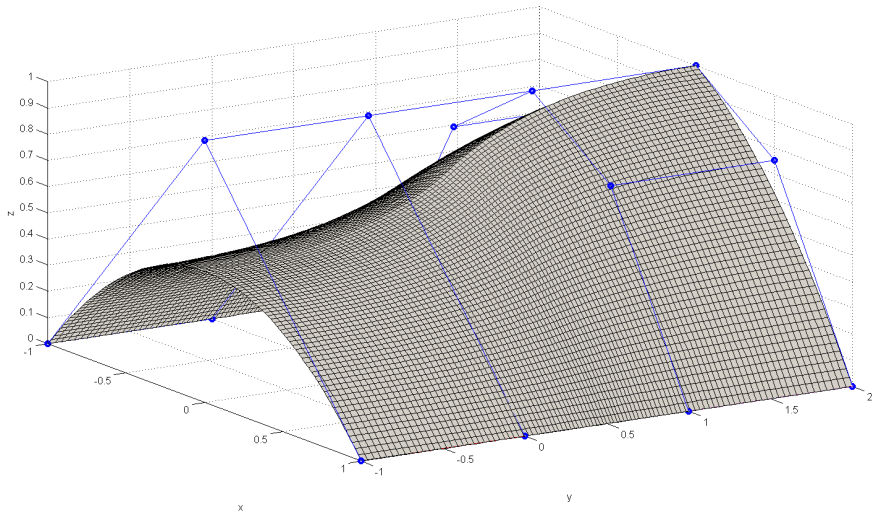
Table 3.1: Knot vectors for the anchors



(a) T-mesh with anchors marked as red crosses, and labelled with knot interval values



(b) Basis functions corresponding to each anchor and local knot vector



(c) Resulting T-spline surface with control points

Figure 3.5: Example of generation of a T-spline surface

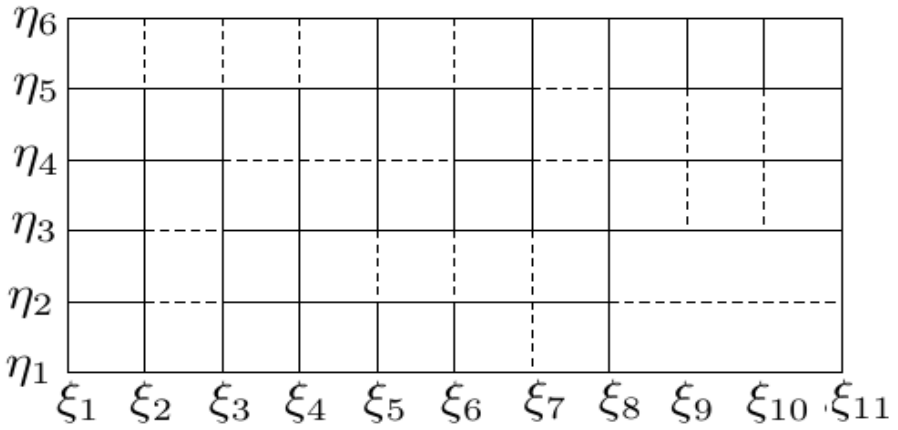
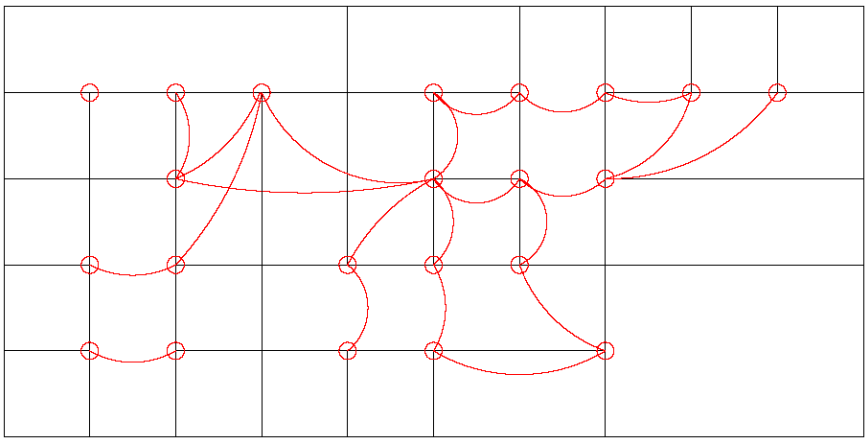


Figure 3.6: Extended T-mesh with lines of reduced continuity defining the quadrature elements, for p=3

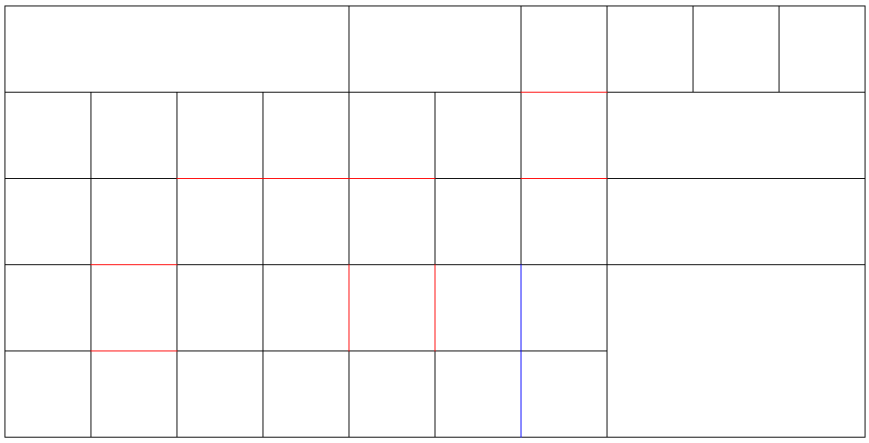
3.6 Suitability for analysis

In order to use T-splines as a basis for FEM, it is necessary that they are linearly independent. It has recently been proven that some T-splines are linearly dependent [6], and the term *analysis suitable T-splines* was introduced by Li *et.al.* [18]. Analysis suitable T-splines form a subset of T-splines which are always linearly independent. In section 3.3, T-junction extensions was mentioned. T-junction extensions are formed by marching from the T-junctions in the direction of a missing edge, until $\frac{p+1}{2}$ perpendicular edges are crossed. This will create a *face extension*. If an edge is attached to the T-junction in the opposite direction, a *edge extension* is formed by marching in the opposite direction of the face extension until a vertex is encountered. Scott *et.al.* [21] states; "An analysis suitable T-spline is one whose extended T-mesh is analysis-suitable. An analysis suitable extended T-mesh is one where no T-junction extensions intersect". If an endpoint of two T-junction extensions intersect this will also count as an intersection. Intersecting T-junction extensions can be visualized by the extension graph $E(T_{ext})$, where T_{ext} is the extended T-mesh. In this graph all T-junctions will be plotted, and if two extensions intersect there will be drawn an edge between the T-junctions. If the extension graph includes no such edges, there is no intersecting T-junction extensions, and the T-spline will be analysis suitable.

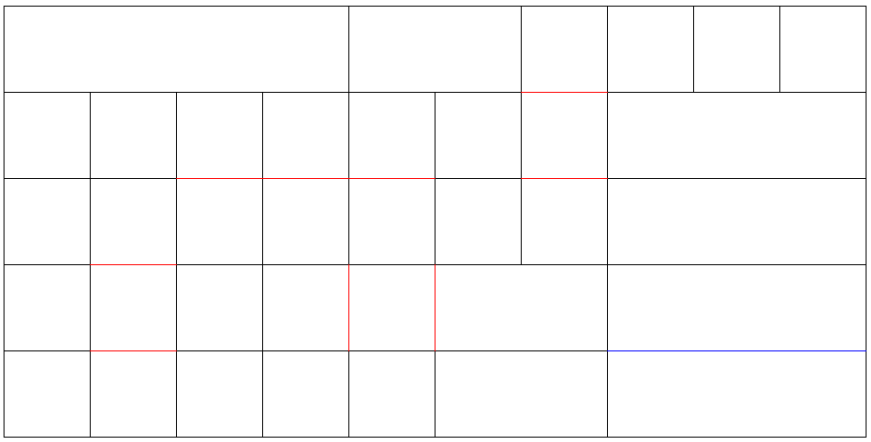
In Figure 3.7a the extension graph for the T-mesh shown in Figure 3.2 (p.17) is given. All T-junctions are marked with a circle, and a red arc is drawn between all T-junctions which will have intersecting T-junction extensions. As the graph shows, several T-junctions extensions are intersecting, and this T-mesh is not analysis suitable. However, the T-mesh can be made analysis suitable by adding new edges to eliminate some of the T-junctions, such that no T-junction extensions will intersect. Figure 3.7b and 3.7c are showing how edges can be added to accomplish this. All the red edges will need to be added, and the blue edge from one of the figures will also need to be added.



(a) The extension graph for the T-mesh



(b) T-mesh made analysis suitable



(c) T-mesh made analysis suitable

Figure 3.7: a) Extension graph, showing that the T-mesh is not analysis suitable b) and c) The T-mesh can be made analysis suitable by adding the red edges and either of the blue edges

Chapter 4

Bézier Extraction

As discussed in previous chapters, B-splines, and thus also NURBS and T-splines, are spanning more than one element, and they are defined over the entire domain of the structure. This in contrast to Lagrangian shape functions which are common in the finite element method, where shape functions are defined locally to each element. This introduces new problems in the implementation of an isogeometric analysis. The framework will have to localize which of the basis functions will have support in the domain of an element. Also, the Gaussian integration points will have to be transformed to parametric coordinates in order to calculate the values of the shape functions. Both these problems will be solved by the concept of Bézier decomposition. This is done by calculating the B-spline basis in terms of an other basis, which is defined only over the element domain. By choosing Bernstein polynomials as that basis, this is possible. This chapter will introduce Bernstein polynomials and the Bézier extraction operator.

4.1 Bernstein polynomials and Bézier curves

A set of Bernstein polynomial basis functions are defined as $\mathbf{B}(\xi) = \{B_{a,p}(\xi)\}_{a=1}^{p+1}$, which corresponds to the set of vector valued control points $\mathbf{P} = \{\mathbf{P}_a\}_{a=1}^{p+1}$ where each $\mathbf{P}_a \in \mathbb{R}^d$, where d is the number of spatial dimensions and $\mathbf{P}$ is a matrix of dimension $n \times d$. The Bernstein polynomials can be defined recursively as [5]

$$B_{a,p}(\xi) = (1-\xi)B_{a,p-1}(\xi) + \xi B_{a-1,p-1}(\xi) \quad \xi \in [0,1] \tag{4.1}$$

where

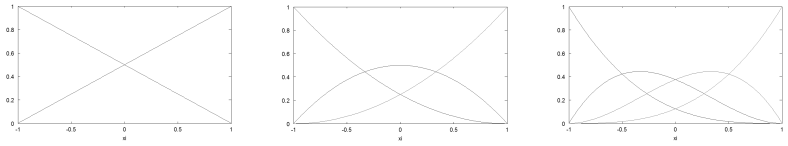
$$B_{1,0}(\xi) \equiv 1 \tag{4.2}$$

and

$$B_{a,p}(\xi) \equiv 0 \text{ if } a < 1 \text{ or } a > p+1 \tag{4.3}$$

The linear, quadratic and cubic Bernstein polynomials are shown in Figure 4.1. These polynomials are identical to B-spline basis functions across one knot span if the multiplicity of the knots at both ends of the knot span is equal to the polynomial order. A Bézier curve of degree p is a linear combination of $p+1$ Bernstein polynomial basis functions and can be written as

$$C(\xi) = \sum_{a=1}^{p+1} \mathbf{P}_a B_{a,p}(\xi) = \mathbf{P}^T \mathbf{B}(\xi) \tag{4.4}$$



(a) p=1 (b) p=2 (c) p=3

Figure 4.1: Bernstein polynomials of order 1,2 and 3

The Bernstein polynomials are defined over the interval [0,1], while in finite element method these functions are used in quadrature over the interval [-1,1], thus it is reasonable to redefine the basis functions so that they span this interval. By doing so the basis functions reads

$$B_{a,p} = \frac{1}{2}(1-\xi)B_{a,p-q}(\xi) + \frac{1}{2}(1+\xi)B_{a-1,p-1}(\xi) \tag{4.5}$$

$$\frac{\partial B_{a,p}}{\partial \xi} = \frac{1}{2}p\big(B_{a-1,p-1}(\xi) - B_{a,p-1}(\xi)\big) \tag{4.6}$$

4.2 Bézier decomposition

Given a B-spline curve $T(\xi)$ of order p, and a knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, additional knots may be inserted at the internal knots by the use of knot insertion, (2.16) and (2.17), until the multiplicity of each knot equals p. By doing so, the B-spline basis functions will be C^0-continuous between each element, and within each element they will be identical to the Bernstein polynomials of order p. This series of knot insertions is called Bézier decomposition. In Figure 4.2 the series of knot insertions needed for a Bézier decomposition of the NURBS representation of a quarter of a circle is shown.

Assume a knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ and a set of control points $\mathbf{P} = \{\mathbf{P}_A\}_{A=1}^{n}$, which defines a B-Spline curve. Let $\{\bar{\xi}_1, \bar{\xi}_2, \ldots, \bar{\xi}_m\}$ be the set of knots that are required to produce the Bézier decomposition of the B-Spline. Then for each new knot, $\bar{\xi}_j, j = 1, 2, \ldots, m$, define $\alpha_A^j, A = 1, 2, \ldots, n+j$, to be the A^{th} alpha as defined in (2.17). Now, defining $C^j \in \mathbb{R}^{(n+j-1)\times(n+j)}$ to be

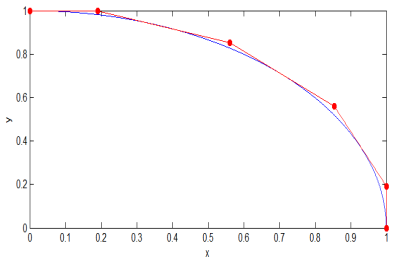
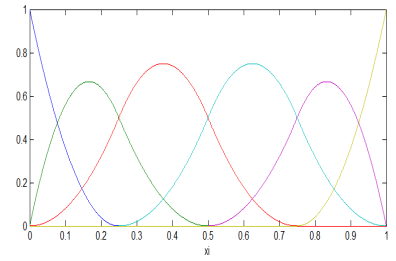
$$C^j = \begin{bmatrix} \alpha_1 & 1-\alpha_2 & 0 & \ldots & & & 0 \\ 0 & \alpha_2 & 1-\alpha_3 & 0 & \ldots & & 0 \\ 0 & 0 & \alpha_3 & 1-\alpha_4 & 0 & \ldots & 0 \\ \vdots & & & & \ddots & & \\ 0 & \ldots & & & 0 & \alpha_{n+j-1} & 1-\alpha_{n+j} \end{bmatrix} \tag{4.7}$$

and letting $\bar{\mathbf{P}}^{\mathbf{1}} = \mathbf{P}$, one can rewrite (2.17) in matrix form to represent the sequence of knot insertions needed as

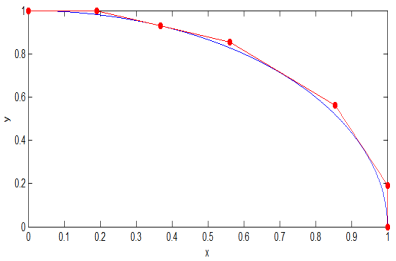
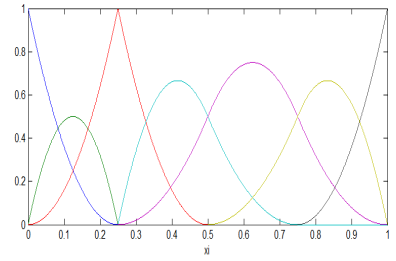
$$\bar{\mathbf{P}}^{j+1} = (\mathbf{C}^j)^T \mathbf{P}^j \tag{4.8}$$

The control points for the Bézier elements, $\mathbf{P}^b$, are given as the final set of control points, $\mathbf{P}^b = \bar{\mathbf{P}}^{m+1}$. Defining $\mathbf{C}^T = (\mathbf{C}^m)^T(\mathbf{C}^{m-1})^T \ldots (\mathbf{C}^1)^T$ will yield

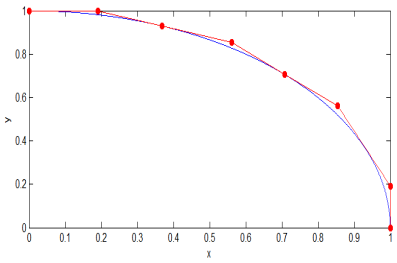
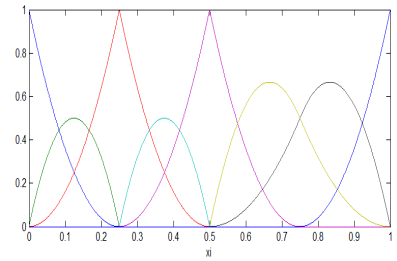
$$\mathbf{P}^b = \mathbf{C}^T\mathbf{P} \tag{4.9}$$

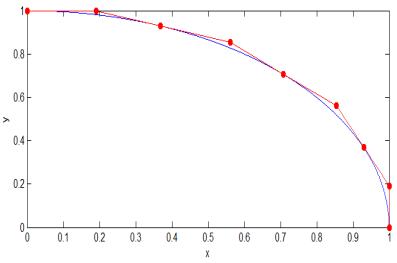
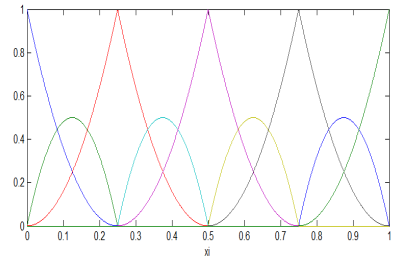
(a) Original NURBS curve

(b) Insert knot: 0.25

(c) Insert knot: 0.50

(d) Insert knot: 0.75

Figure 4.2: Control points and rational basis functions created by knot insertion in order to create the Bézier decomposition of a NURBS curve, with original knot vector [0 0 0 0.25 0.5 0.75 1 1 1]

Since the Bézier decomposition of a curve does not cause any parametric or geometric change to a curve, it can be written

$$T(\xi) = \mathbf{P}^T\mathbf{N}(\xi) = (\mathbf{P}^b)^T\mathbf{B}(\xi) = (\mathbf{C}^T\mathbf{P})^T\mathbf{B}(\xi) = \mathbf{P}^T\mathbf{C}\mathbf{B}(\xi) \tag{4.10}$$

The control points $\mathbf{P}$ are arbitrary, thus it is shown that

$$\mathbf{N}(\xi) = \mathbf{C}\mathbf{B}(\xi) \tag{4.11}$$

where $\mathbf{C}$ is the linear Bézier extraction operator. The Bézier extraction operator is constructed with only information from the knot vector, and it does not depend on the control points of the B-spline curve or the basis functions.

4.3 Bézier extraction of NURBS

As discussed in Chapter 2, NURBS are constructed from the B-spline basis functions, which allows to apply the extraction operator to NURBS. If (4.11) is substituted into (2.26) (p.12) , then (2.23) can be written as

$$T(\xi) = \frac{1}{W(\xi)}\mathbf{P}^T\mathbf{W}\mathbf{N}(\xi) = \frac{1}{W(\xi)}\mathbf{P}^T\mathbf{W}\mathbf{C}\mathbf{B}(\xi) = \frac{1}{W(\xi)}(\mathbf{C}^T\mathbf{W}\mathbf{P})^T\mathbf{B}(\xi) \tag{4.12}$$

Now rewrite the weight function, W(ξ), in terms of the Bernstein basis as

$$\begin{aligned} W(\xi) &= \sum_{i=1}^{n} w_i N_i(\xi) = \mathbf{w}^T\mathbf{N}(\xi) \\ &= \mathbf{w}^T\mathbf{C}\mathbf{B}(\xi) = (\mathbf{C}^T\mathbf{w})^T\mathbf{B}(\xi) \\ &= (\mathbf{w}^b)^T\mathbf{B}(\xi) = W^b(\xi) \end{aligned} \tag{4.13}$$

where $\mathbf{w}^b = \mathbf{C}^T\mathbf{w}$ are the Bézier weights. As with knot insertion, Bézier decomposition of control points are done directly to the B-Spline curve which defines the NURBS curve. Geometrically this is done by projecting the NURBS control points into d+1 dimensions, then the Bézier extraction operator is applied to the B-Spline contol points, and then the curve is projected back into d dimensions to obtain the Bézier control points, $\mathbf{P}^b$. Define $\mathbf{W}^b$ to be the diagonal matrix consisting of Bézier weights, equivalent to (2.25) (p. 12) .

$$\mathbf{W}^b = w_i^b\delta_{ij} \tag{4.14}$$

Now the decomposition of the NURBS control points, $\mathbf{P}$, can be calculated as

$$\mathbf{P}^b = (\mathbf{W}^b)^{-1}\mathbf{C}^T\mathbf{W}\mathbf{P} \tag{4.15}$$

Multiply by $\mathbf{W}^b$ to get

$$\mathbf{W}^b\mathbf{P}^b = \mathbf{C}^T\mathbf{W}\mathbf{P} \tag{4.16}$$

and then substitute into (4.12) to obtain the equation for a NURBS curve in terms of C^0 Bézier elements

$$T(\xi) = \frac{1}{W^b(\xi)}(\mathbf{W}^b\mathbf{P}^b)^T\mathbf{B}(\xi) = \sum_{i=1}^{n+m}\frac{\mathbf{P}_i^b w_i^b B_i(\xi)}{W^b(\xi)}$$

For a surface the bivariate extraction operator is needed. This is defined for an element as

$$\mathbf{C}_A^e = \mathbf{C}_\eta^i \otimes \mathbf{C}_\xi^j \tag{4.17}$$

where $\otimes$ is the tensor product defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{11}\mathbf{B} & A_{12}\mathbf{B} & \dots \\ A_{21}\mathbf{B} & A_{22}\mathbf{B} & \\ \vdots & & \ddots \end{bmatrix} \tag{4.18}$$

4.3.1 Example of Bézier decomposition

In order to increase the understanding of the Bézier decomposition, an example is useful. This example will take a closer look at a circular beam that is to be analysed. In the analysis quadratic NURBS basis functions are used, and the element mesh is consisting of 6 elements in the radial direction, and 3 elements in the tangential direction. Thus, the parametric space will be defined by the open knot vectors

$$\Xi = \{0, 0, 0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}, 1, 1, 1\} \tag{4.19}$$

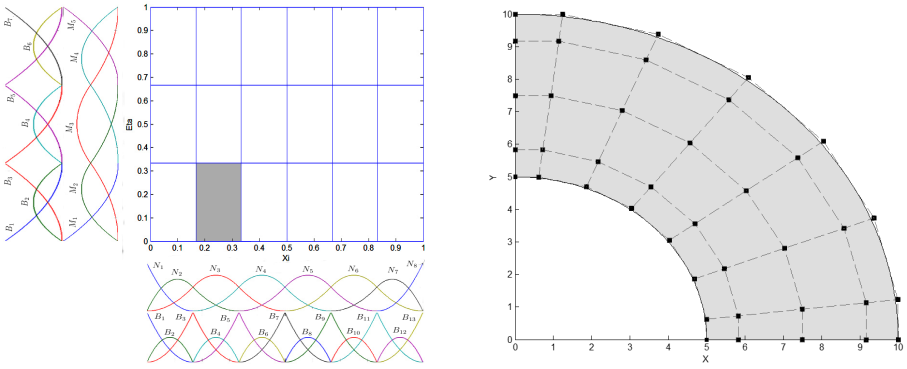
and

$$H = \{0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1\} \tag{4.20}$$

In the parametric directions ξ and η, the univariate B-spline basis functions are $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$, respectively. The basis functions are plotted in Figure 4.3a, and in Figure 4.3b the beam with control points is shown. Recalling (4.11), and dropping subscripts p and q, the basis functions can be in terms of the Bézier extraction operator and Bernstein polynomials as

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \\ N_7 \\ N_8 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{Bmatrix} \tag{4.21}$$

$$\begin{Bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.0 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \end{Bmatrix} \tag{4.22}$$



(a) Parametric space with univariate basis functions and Bernstein polynomials plotted along the axes

(b) Circular beam with control point net

Figure 4.3: Parametric space and control points

In general, the global extraction operator does not need to be calculated, the local extraction operators will be calculated for each element with an algorithm given in Appendix A. Here the global operator is calculated for the sake of clarity. With the information in figure 4.3a, and (4.21) and (4.22), the extraction operators for each element can be localized. For the shaded element one gets

$$\begin{Bmatrix} N_1^2 \\ N_2^2 \\ N_3^2 \end{Bmatrix} = \begin{Bmatrix} N_2 \\ N_3 \\ N_4 \end{Bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_3 \\ B_4 \\ B_5 \end{Bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1^2 \\ B_2^2 \\ B_3^2 \end{Bmatrix} \tag{4.23}$$

$$\begin{Bmatrix} M_1^1 \\ M_2^1 \\ M_3^1 \end{Bmatrix} = \begin{Bmatrix} M_1 \\ M_2 \\ M_3 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1^1 \\ B_2^1 \\ B_3^1 \end{Bmatrix} \tag{4.24}$$

where the superscript denotes element number in each parametric direction. The bivariate extraction operator for the shaded element, element 2, then become

$$\begin{aligned} \mathbf{C}^2 &= \mathbf{C}_\eta^1 \otimes \mathbf{C}_\xi^2 \\ &= \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \otimes \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \end{aligned} \tag{4.25}$$

With the extraction operators at hand, the control points are computed for the Bézier elements with (4.15), and the physical element mesh is calculated with (4.12). This is shown in Figure 4.4

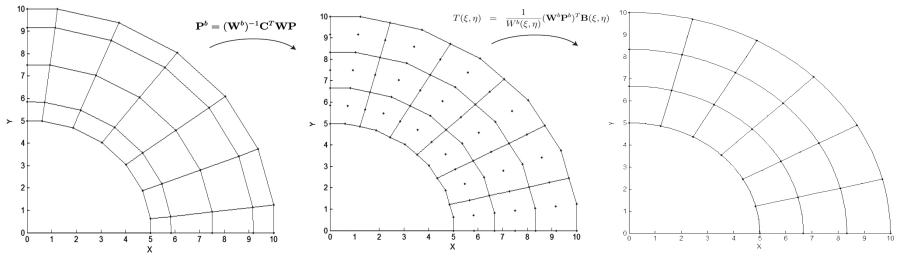


Figure 4.4: From control points to Bézier control points to Bézier physical mesh

4.4 Bézier extraction of T-splines

Bézier extraction of T-splines be treated slightly different form the case with NURBS, but the concept is the same. In the case with NURBS, the extraction operator was calculated for the entire elements. In the case of T-splines, each T-spline will be extracted independently, and add a single row to the extraction operator. For T-splines the local knot vectors are in general knot open, and an extended knot vector will handle this. The *extended knot vector* is created from the local knot vector by adding additional knots to the ends of the knot vector to increase the multiplicity to *p+1*, such that one are dealing with an open knot vector. For example, if the local knot vector is $\Xi = \{3, 4, 5, 6, 7\}$, add $\{3, 3, 3\}$ and $\{7, 7, 7\}$ to obtain the extended knot vector $\bar{\Xi} = \{3, 3, 3, 3, 4, 5, 6, 7, 7, 7, 7\}$. Now define n_t as the number of additional knots added in front of the knot vector, and the T-spline basis function will be numbered n_t+1, if numbered left to right. Figure 4.5 shows a T-spline basis function plotted with the additional basis functions from the extended knot vector. Figure 4.6 shows the Bernstein polynomials the curve will be decomposed to.

After obtaining the extended knot vector, the Bézier extraction is done as with NURBS, except only the one necessary row will be calculated.

$$N_A(\xi_A)|_e = N_a^e(\tilde{\xi}) = \mathbf{e}_a^T\mathbf{N}^e(\tilde{\xi}) = \mathbf{e}_a^T\mathbf{C}^e\mathbf{B}(\tilde{\xi}) = (\mathbf{c}_a^e)^T\mathbf{B}(\tilde{\xi}) \tag{4.26}$$

where $\mathbf{e}_a$ is a unit vector equal to 1 in entry a and zero elsewhere. The vector $\mathbf{c}_a^e$ extracts basis function A for element e. The Bézier extraction operators for each element in the knot vector is shown in Table 4.1. The rows which is needed for the decomposition of N_7 is highlighted in bold font. Figure 4.7 shows how the Bernstein polynomials scaled according to the extraction operator will sum up to N_7.

For a surface bivariate extraction is needed, which is done as a product of the basis functions in each parametric direction. The bivariate basis function in terms of Bézier extractor and Bernstein polynomials is defined by the following formula:

$$N_A(\boldsymbol{\xi}_A)|_e = N_a^e(\tilde{\boldsymbol{\xi}}) = N_a^{e,1}(\tilde{\xi^1})N_a^{e,2}(\tilde{\xi^2}) = \left[(\mathbf{c}_a^{e,1})^T\mathbf{B}^1(\tilde{\xi^1})\right]\left[(\mathbf{c}_a^{e,2})^T\mathbf{B}^2(\tilde{\xi^2})\right] \tag{4.27}$$

where superscript 1 or 2 denotes the parametric direction. An algorithm for calculation of Bézier extraction operators for T-splines is given by Scott it et.al. 2011 [22].

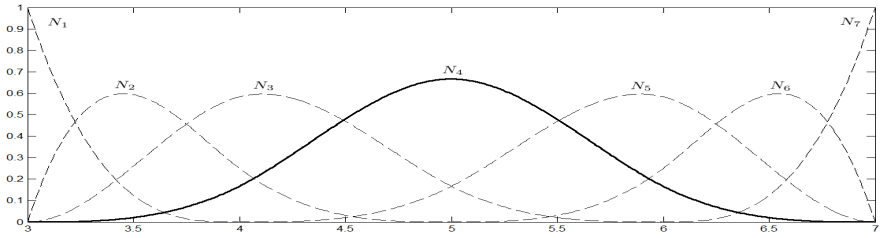


Figure 4.5: The T-spline function N_7 from $\Xi = \{3, 4, 5, 6, 7\}$ is plotted in solid line. The additional basis functions are plotted in dashed lines

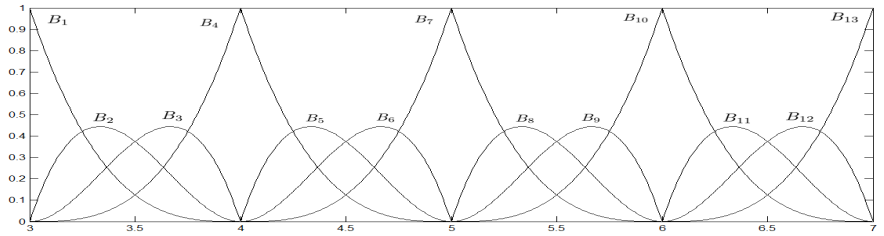


Figure 4.6: The Bernstein polynomials for Bézier elements after knot insertion

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0.5 & 0.25 \\ 0 & 0 & 0.5 & 0.5833 \\ 0 & 0 & 0 & 0.1667 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{Bmatrix} \qquad \begin{Bmatrix} N_2 \\ N_3 \\ N_4 \\ N_5 \end{Bmatrix} = \begin{bmatrix} 0.25 & 0 & 0 & 0 \\ 0.5833 & 0.6667 & 0.3333 & 0.1667 \\ 0.1667 & 0.3333 & 0.6667 & 0.6667 \\ 0 & 0 & 0 & 0.1667 \end{bmatrix} \begin{Bmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \end{Bmatrix}$$

$$\begin{Bmatrix} N_3 \\ N_4 \\ N_5 \\ N_6 \end{Bmatrix} = \begin{bmatrix} 0.1667 & 0 & 0 & 0 \\ 0.6667 & 0.6667 & 0.3333 & 0.1667 \\ 0.1667 & 0.3333 & 0.6667 & 0.5833 \\ 0 & 0 & 0 & 0.25 \end{bmatrix} \begin{Bmatrix} B_7 \\ B_8 \\ B_9 \\ B_{10} \end{Bmatrix} \qquad \begin{Bmatrix} N_4 \\ N_5 \\ N_6 \\ N_7 \end{Bmatrix} = \begin{bmatrix} 0.1667 & 0 & 0 & 0 \\ 0.5833 & 0.5 & 0 & 0 \\ 0.25 & 0.5 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{Bmatrix}$$

Table 4.1: Bézier extraction operators. Only the highlighted rows are needed to compute the T-spline N_7

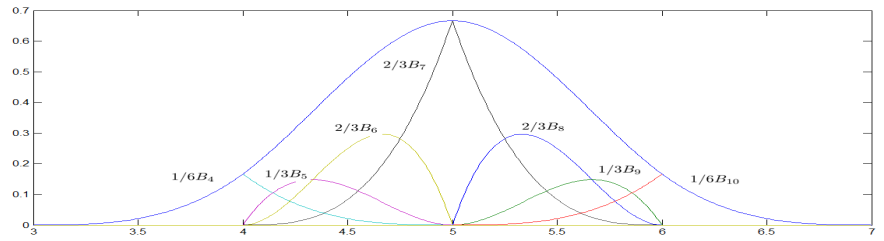


Figure 4.7: The T-spline function N_7 from $\Xi = \{3, 4, 5, 6, 7\}$ decomposed to Bernstein basis. The influencing Bernstein basis functions, which sums up to N_7, are plotted according to the Bézier extractor for each Bézier element

Chapter 5

Isogeometric Finite Element Analysis

The concept of isogeometric analysis is to use the NURBS or T-spline functions which defines the geometry of the analysed problem as shape functions in the finite element analysis. This in contrast to the traditional isoparametric concept, where the shape functions used in the finite element analysis are used to describe the geometry. The isoparametric concept will often lead to an inaccurate geometry, as the Lagrangian shape functions often used, are not able to exactly represent for instance conical sections. An isogeometric analysis program can be implemented in two different ways. The first is to write an entire new program code, to account for the basis functions which are spanning more than one quadrature element, which cannot be handled by a traditional finite element solver. The second is to use the concept of Bézier extraction to confine the basis to each quadrature element, and thus only the shape function routine will need to be changed. Both these options will be discussed, so that the advantage of using Bézier extraction can be appreciated.

5.1 Creating a isogeometric finite element solver

The main structure of an isogeometric analysis program will follow the structure of classical FEA, where the same steps needed to be performed. The stiffness matrix $\mathbf{K}$ needs to be established by a loop over all the elements to construct the element stiffness matrices $\mathbf{k}_e$, and also the load vector $\mathbf{P}_0$ from the element load vectors $\mathbf{p}_{0e}$

This section will focus on a NURBS based isogeometric analysis, as this is is easier to fully understand, even if most of the concepts discussed can be extended to also include T-splines. Both NURBS and T-spline based analysis will handled in section 5.2.

As discussed in section 3.5, each element is defined as the domain bounded by lines of reduced continuity. In the case of NURBS this will give us elements for each of the knot spans in the parametric space. Each element is labelled with *NURBS coordinates* i and j, $p+1 \leq i \leq n$ and $q+1 \leq j \leq m$, which refers to a knot in the ξ- and η-direction, respectively. The NURBS coordinates of

each element will give the element number e, defined as

$$e = (j - q - 1)(n - p) + (i - p). \tag{5.1}$$

and the domain of each element in the parametric space is then:

$$\hat{\Omega}^e = [\xi_i, \xi_{i+1}] \times [\eta_j, \eta_{j+1}] \tag{5.2}$$

Each of the NURBS basis functions are defined over the entire domain of the parametric space, even if they only have support in a small region of the domain, as shown in Figure 5.1 where the shape function corresponding to control point 19 in a 5x5 element mesh is shown. It is possible for each element to calculate the value of all NURBS basis functions and then extract the non-zero values when evaluating the stiffness matrix and load vector. But this is a very inefficient way to handle this problem. A much more elegant work around is to use the information given in the knot vectors to determine which basis function will have support in the elements prior to calculating them, and thus only need to calculate the non-zero values. This is accomplished with the *ElementNode-matrix* (IEN-matrix). The IEN-matrix is a matrix that relates the global element number e and local basis function a to the global control point A, such that A=IEN(a,e). In Table 5.1 the IEN matrix for the 5x5 elements mesh used in Figure 5.1 is shown. All the entries in the IEN matrix such that 19=IEN(a,e) is in bold font, to show which elements shape function 19 will have support in. An algorithm to calculate this matrix is given in [13].

For each element, the evaluation of the element stiffness matrix and load vector is done by performing numerical integration, Gaussian quadrature, in the domain of a *parent element*, $\tilde{\Omega}^e = [-1, 1] \times [-1, 1]$.

$$\mathbf{k}_e = \sum_{i=1}^{n} \mathbf{B}_i^T \mathbf{D} \mathbf{B}_i t_h w_i |\mathbf{J}_i| \tag{5.3}$$

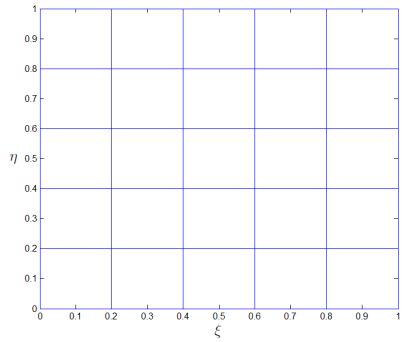
where n is the number of quadrature points needed to perform the integration, t_h is the thickness, w is the weight of the quadrature point, and subscript i denotes the i^{th} quadrature point.

In classical finite element analysis the sampling of the shape functions, and its derivatives, are done directly at the gauss points for numerical quadrature, which is possible since the shape functions are defined only on the domain $\tilde{\Omega}^e$. In isogeometric analysis this is not true, thus the gauss points will have to be transformed to parametric coordinates, in order to obtain the value in these points. These points are calculated from the gauss points $\tilde{\xi}$, $\tilde{\eta}$, and the knot vectors Ξ, $\mathcal{H}$.

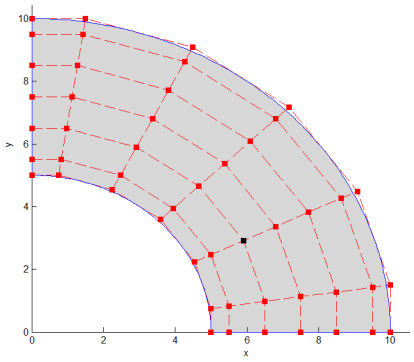
$$\xi = \xi_i + (\tilde{\xi} + 1)\frac{(\xi_{i+1} - \xi_i)}{2} \tag{5.4}$$

$$\eta = \eta_j + (\tilde{\eta} + 1)\frac{(\eta_{j+1} - \eta_j)}{2} \tag{5.5}$$

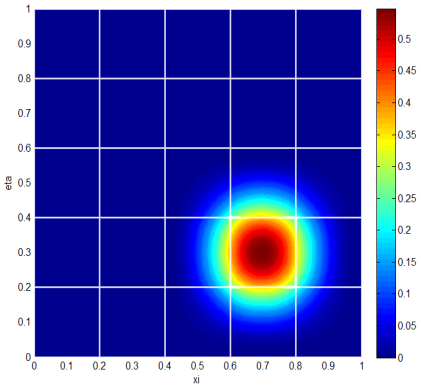
Due to this transformation of the quadrature points, it is not sufficient to only reprogram the shape function, but also all of the elements in the element library will need some recoding. The transformed Gauss points will need to be calculated prior to calling the shape function algorithm. It would be possible to also



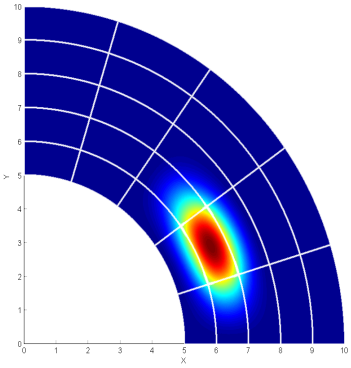
(a) 5×5 element mesh plotted in the parametric space



(b) The physical domain of a the circular beam, with 49 control points. Control point 19 marked with black dot



(c) Shape function plotted in the parametric space



(d) Shape function plotted on the physical domain

Figure 5.1: One of the main complications in isogeometric analysis. The shape functions are spanning more than one element. The shape function corresponding to control point number 19 in a 5×5 element mesh is shown for a circular beam with 49 control points and 25 9-noded quadratic quadrilateral elements

a \ e	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	1	2	3	4	5	8	9	10	11	12	15	16	17	18	**19**	22	23	24	25	26	29	30	31	32	33
2	2	3	4	5	6	9	10	11	12	13	16	17	18	**19**	20	23	24	25	26	27	30	31	32	33	34
3	3	4	5	6	7	10	11	12	13	14	17	18	**19**	20	21	24	25	26	27	28	31	32	33	34	35
4	8	9	10	11	12	15	16	17	18	**19**	22	23	24	25	26	29	30	31	32	33	36	37	38	39	40
5	9	10	11	12	13	16	17	18	**19**	20	23	24	25	26	27	30	31	32	33	34	37	38	39	40	41
6	10	11	12	13	14	17	18	**19**	20	21	24	25	26	27	28	31	32	33	34	35	38	39	40	41	42
7	15	16	17	18	**19**	22	23	24	25	26	29	30	31	32	33	36	37	38	39	40	43	44	45	46	47
8	16	17	18	**19**	20	23	24	25	26	27	30	31	32	33	34	37	38	39	40	41	44	45	46	47	48
9	17	18	**19**	20	21	24	25	26	27	28	31	32	33	34	35	38	39	40	41	42	45	46	47	48	49

Table 5.1: IEN matrix for a 5x5 mesh of Q9 elements. A=IEN(a,e). Shape function 19 from Figure 5.1 has support in elements 3, 4, 5, 8, 9, 10, 13, 14, 15

calculate these within the shape function routine, but this would lead to calculating them in every loop of the shape function, instead of every loop through elements. This is therefore a computational waste that should be avoided.

Evaluation of B-splines are performed for each Gauss points, within the shape function algorithm. The B-spline basis functions can be calculated with (2.1) (p. 5). Use of this recursive formula will be computationally expensive, as many of the lower order basis functions will be calculated several times. This will then lead to more computational effort needed to construct the stiffness matrix. Although more efficient algorithms using dynamic programming exist, this is beyond the scope of this thesis.

5.2 Implementation of Bézier Extractor in FEM

Use of the Bézier extraction operator will help to avoid some of the complicating factors mentioned in the previous section. As discussed in Chapter 4, the Bézier extraction operator will allow us to represent a NURBS or T-spline basis function in terms of Bernstein polynomials.

Since the shape functions now will be defined separately for each elements, and only on the domain of each element, the implementation is simplified. The NURBS coordinates will no longer be needed, and the Gauss points will not need the transformation to parametric coordinates. In the construction of the stiffness and load matrices, each Bézier element will be similar to the Lagrange elements. The shape function algorithm will be called to calculate the shape functions and derivatives. The IEN-array will still be needed, and will replace the normal connectivity array. The shape function from Figure 5.1d is shown again in Figure 5.2, but now it is calculated with the Bézier extraction operator, and is only defined in the domain of one element.

In order to take advantage of the Bézier extraction operators these will need to be pre-calculated prior to the assembly of the stiffness matrix, and the multivariate extraction operators needs to be stored in the memory. As the size of the analysed problem increases, the memory requirements for storing the operators can become extensive. If only the univariate extraction operators are pre-calculated and then the tensor product multivariate operators are evaluated when needed, or if each extraction operator are calculated when needed in the shape function routine, the memory requirements can be severely reduces. But doing so will also demand more changes to the finite element code, also to other routines than the shape function algorithm. Doing so will also increase the total number of calculations needed in the complete analysis. Thus it is not recommended to do so, unless the size of the problems which can be analysed is restricted by available memory only.

Once the extraction operators are calculated one can calculate the Bézier control points and respective Bézier weights. The Bézier weights must be calculated and used in the shape function algorithm, while the Bézier control points are not strictly necessary, and one may choose whether or not these should be calculated. Using $\mathbf{P}^b$ instead of $\mathbf{P}$ in analysis will be less computational efficient, since $\mathbf{P}$ would then have to be calculated from $\mathbf{P}^b$ whenever needed. However, for visualization $\mathbf{P}^b$ may be useful, for instance if one want to visualize the Bézier element mesh.

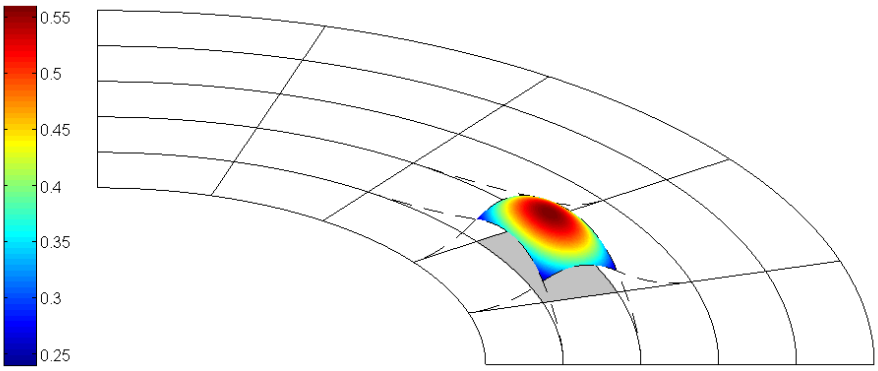


Figure 5.2: The shape function from Figure 5.1d, defined only over the domain of the shaded element, by use of Bézier extraction

It is important to understand that the Bézier extraction will not introduce additional degrees of freedom. The Bézier control points are merely virtual control points, used to ease the implementation. They have no physical meaning, other than to define the physical domain of the Bézier element, and the Bézier elements will still relate to the NURBS control points. Also, the physical domain of the Bézier elements is identical to the physical domain of the NURBS or T-spline elements. This should be apparent from the fact that Bézier extraction do not cause any changes to the basis functions. If this was not true, a FEA based on Bézier extraction would not be possible.

5.3 Shape function algorithm

The shape function algorithm is an integral part of the analysis software, and with the use of Bézier extraction this is the only part that will need to be changed. A thorough description of this algorithm is given here. The purpose of the shape function algorithm is to calculate the value of the basis function and its derivatives at the quadrature points, as well as calculating the Jacobian determinant. I.e. all necessary variables in (5.3).

Recall (2.26) (p. 12) which in terms of the Bézier extraction operator and Bernstein polynomials gives us

$$\mathbf{R}(\boldsymbol{\xi}) = \mathbf{W}\frac{\mathbf{N}(\boldsymbol{\xi})}{W(\boldsymbol{\xi})} = \mathbf{W}\mathbf{C}\frac{\mathbf{B}(\boldsymbol{\xi})}{W^b(\boldsymbol{\xi})} \tag{5.6}$$

where $\mathbf{C}$ is the bivariate extraction operator. This equation can be written for each element as:

$$\mathbf{R}^\mathbf{e}(\boldsymbol{\xi}) = \mathbf{W}^\mathbf{e}\mathbf{C}^\mathbf{e}\frac{\mathbf{B}^\mathbf{e}(\boldsymbol{\xi})}{W^b(\boldsymbol{\xi})} \tag{5.7}$$

Differentiating this equation with respect to parametric coordinates, ξ_i, yields

$$\begin{aligned}\frac{\partial \mathbf{R}^\mathbf{e}(\boldsymbol{\xi})}{\partial \xi_i} &= \mathbf{W}^\mathbf{e}\mathbf{C}^\mathbf{e}\frac{\partial}{\partial \xi_i}\left(\frac{\mathbf{B}^\mathbf{e}(\boldsymbol{\xi})}{W^b(\boldsymbol{\xi})}\right) \\ &= \mathbf{W}^\mathbf{e}\mathbf{C}^\mathbf{e}\left(\frac{1}{W^b(\boldsymbol{\xi})}\frac{\partial \mathbf{B}^\mathbf{e}(\boldsymbol{\xi})}{\partial \xi_i} - \frac{\partial W^b(\boldsymbol{\xi})}{\partial \xi_i}\frac{\mathbf{B}^\mathbf{e}(\boldsymbol{\xi})}{(W^b(\boldsymbol{\xi}))^2}\right)\end{aligned} \tag{5.8}$$

Equations (5.7) and (5.8) require the values of the Bernstein basis, the weight function, and the derivatives of the basis and weight function with respect to $\boldsymbol{\xi}$. The Bernstein basis and derivatives will be calculated in a separate algorithm which will be called from the shape function algorithm by use of (4.5) and (4.6) (p. 24). The weight function is calculated by (4.13) (p. 26) and its derivatives are calculated as

$$\frac{\partial W^b(\xi)}{\partial \xi_i} = \sum_{a=1}^{A} \frac{\partial B_a(\xi)}{\partial \xi_i} w_a^b \tag{5.9}$$

The calculation of the stiffness matrix requires the derivatives of $\mathbf{R}$ with respect to physical coordinates (x_1, x_2). These derivatives are obtained by use of the chain rule, and the resulting equation is

$$\frac{\partial \mathbf{R^e}(\boldsymbol{\xi})}{\partial x_i} = \sum_{j=1}^{2} \frac{\partial \mathbf{R^e}(\boldsymbol{\xi})}{\partial \xi_j} \frac{\partial \xi_j}{\partial x_i} \tag{5.10}$$

The calculation of $\partial\boldsymbol{\xi}/\partial\mathbf{x}$ is done by first computing the Jacobian $\partial\mathbf{x}/\partial\boldsymbol{\xi}$, and then computing the inverse.

$$\frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} = (\mathbf{P}^e)^T \frac{\partial \mathbf{R^e}}{\partial \boldsymbol{\xi}} \tag{5.11}$$

$$= ((\mathbf{W}^e)^{-1}(\mathbf{C}^e)^{-T}\mathbf{W}^{b,e}\mathbf{P}^{b,e})^T \frac{\partial \mathbf{R^e}}{\partial \boldsymbol{\xi}} \tag{5.12}$$

(5.11) and (5.12) are equivalent, the difference is whether $\mathbf{P}$ or $\mathbf{P}^b$ is used in the calculation. With the results form (5.7), (5.10) and (5.11) or (5.12) the necessary variables needed are obtained, and the shape function algorithm is finished for this quadrature point. The shape function for Matlab is given in Appendix A. When the extraction operator is utilized, both T-splines and NURBS are handled by the same shape function algorithm. Since the shape function is written in terms of $\mathbf{C}$ and Bernstein polynomials, the only difference is that for T-splines there might be varying number of shape functions with support in each element. This is handled by the shape function algorithm.

5.4 Finite Element Analysis with CAD software

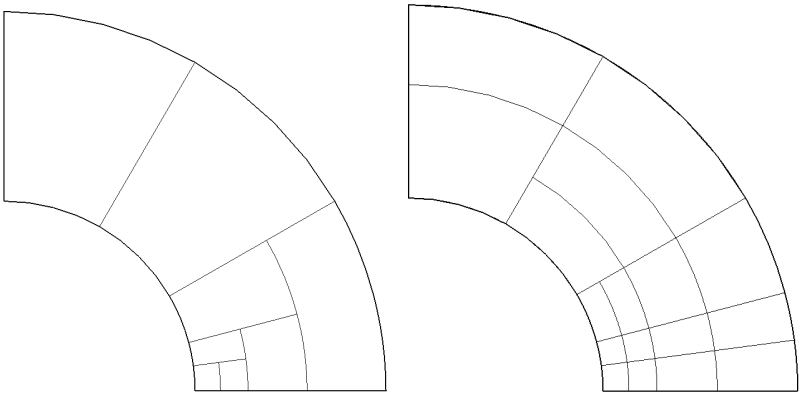
CAD software usually use NURBS as the basis for geometric modelling. However, some CAD software has the possibility to add T-spline compatibility by use of plug ins. Such software includes Autodesk Maya and Rhinoceros. In this thesis, "T-Splines for Rhino" has been used to explore the possibilities of using such software together with a FEA.

With this software the user has the possibility to export both control points and Bézier extraction operators for each of the elements in the extended T-mesh. As earlier discussed, this is the only information needed to model the geometry for the finite element analysis. The exported Bézier extractors will not be in a format which is possible to import directly into the FE framework, but contains the necessary data to create the C-matrices and the IEN-array. In order to use the exported data it will need to be rearranged into the needed matrices, but this can easily be done with some sort of parsing script. With this information it is now possible to perform an analysis after applying boundary conditions

and loading.When C and IEN has been established, the analyst will not need to know any T-spline theory in order to perform the analysis. The extraction operator will deal with that. A parsing script for Matlab is given in Appendix A.

Figure 5.3 shows both the T-mesh and the extended T-mesh for a geometry imported from Rhino for analysis. Table 5.2 shows the IEN-matrix that is generated from the imported data. As the IEN-matrix clearly shows, there is not a fixed number of shape functions that will have support in each element. This is due to the T-junctions and associated knot vectors that defines the support of each basis function in the parametric space, and this is not the same for each knot vector.

When it comes to refinement of the element mesh, this method is not very suitable. Of course, it is possible to some extent to refine the T-mesh in Rhino, but this is no optimal solution. If a finer mesh than the one imported from Rhino is wanted, a refinement of the mesh is needed to be done by a refinement algorithm. This algorithm will not be further mentioned here, as this algorithm is beyond the scope of this thesis.



(a) T-mesh (b) Extended T-mesh

Figure 5.3: T-meshes imported from T-splines for Rhino

a \ e	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	9	9	9	9	9	9	9	9	1	1	9	5	5	5	9	9	9
2	10	10	10	10	10	10	10	10	2	2	10	6	6	6	10	10	10
3	11	11	11	14	14	14	14	14	3	3	14	7	7	7	11	13	13
4	12	12	12	15	18	15	15	19	4	4	19	8	8	8	12	14	14
5	14	14	14	16	22	16	16	23	5	5	20	9	9	9	13	15	17
6	15	15	15	18	25	19	19	25	6	6	23	10	10	10	14	17	18
7	16	16	16	22	26	25	20	26	7	7	24	11	11	11	15	18	21
8	18	25	25	25	27	26	25	27	8	8	25	12	12	12	17	21	22
9	22	27	28	26	28	27	28	28	9	9	28	13	14	14	18	22	25
10	25	28	29	27	30	28	29	29	10	10	29	14	15	15	21	25	26
11	27	29	30	28	31	29	30	30	11	11	30	15	16	16	22	26	27
12	28	30	33	30	32	30	32	31	12	12	31	17	18	25	25	27	28
13	30	33	34	32	33	32	33	32	13	14	32	18	25	28	27	28	31
14	33	34	35	33	34	33	34	33	14	15	33	25	28	29	28	32	32
15	34	36	36	34	37	34	35	34	15	16	34	28	30	30	33	33	33
16	37	37	37	37	38	36	36	36	25	25	35	33	33	33	34	34	34
17				39	39	37	37	37			36						
18						39	39	38			37						
19								39			38						
20											39						

Table 5.2: IEN matrix for the T-mesh imported from Rhino. The number of shape functions with support in each element is not fixed, but always (p+1)(q+1) or higher

Chapter 6

Numerical Examples

Isogeometric analysis will in this chapter be used to solve some example problems, to investigate the convergence rate and accuracy of the obtained solution and also to show some areas of application of isogeometric analysis. The problems involve linear elasticity and transient heat conductivity. For two of the elasticity problems a convergence study is performed in order to examine the convergence properties.

6.1 Curved beam with end shear

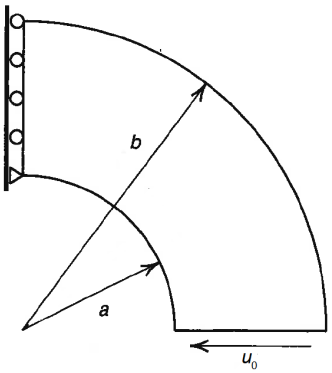
The first problem analysed consist of a cantilevered beam shaped as a quarter of a circle, with the outline as shown in Figure 6.1a. The material is linear elastic and in a state of plane stress, with elastic modulus E=10000 and a Poisson ratio $\nu = 0.25$. The inner radius a is 5 units and the outer radius b is 10 units. The beam is analysed with the free end subjected to a prescribed displacement in the negative x-direction, $u_0 = 0.01$, and the resulting strain energy is calculated. The analytical solution for the strain energy of the system is given by Timoshenko and Goodier [27] , and the results from the isogeometric analysis is compared with the results obtained by Zienkiewicz and Taylor [29] from a traditional finite element analysis. The error in the strain energy is calculated as

$$\|e\|_E^2 = U - U^h \tag{6.1}$$

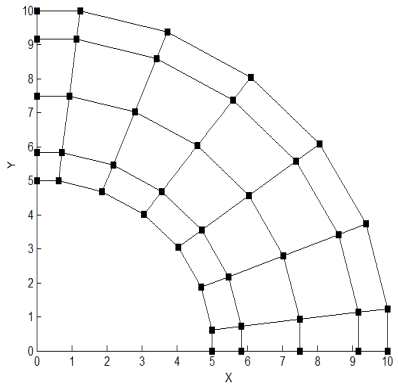
The beam is analysed with 9- and 16-noded Bézier quadrilaterals, with coarsest meshes consisting of 3x6 and 2x4 elements, respectively. The coarsest meshes are shown in Figure 6.1c and 6.1e. The results obtained from the analysis is given in Table 6.1 and 6.2, and the resulting error is plotted logarithmically versus number of degrees of freedom in Figure 6.2.

In Figure 6.3 the stresses from an analysis performed with 48x24 quadratic elements is shown, plotted on the deformed shape of the beam. The displacements in both x- and y-direction are magnified to a scale of 50 times the displacements, in order to be shown properly.

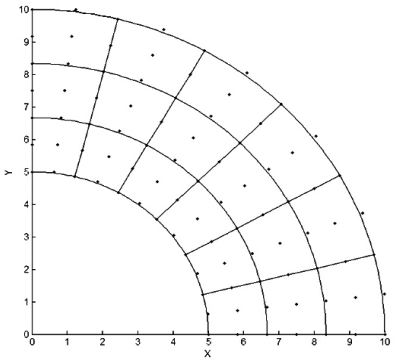
As seen in the convergence plot, the NURBS based finite element analysis is performing better than the traditional finite element analysis. The convergence rates are as expected the same as for a traditional FEA, but the accuracy is better.



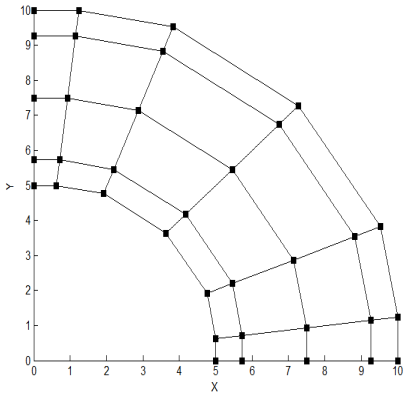
(a) Geometry and boundary conditions for the curved beam



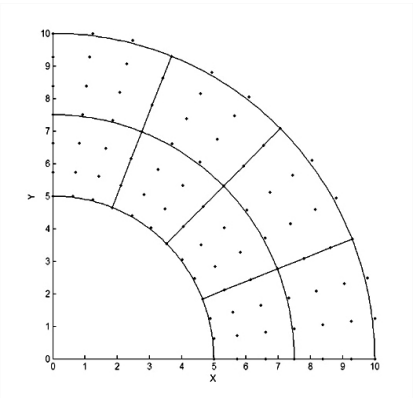
(b) Coarsest mesh: Control points for Q9



(c) Coarsest mesh: Bézier mesh with Bézier control points for Q9



(d) Coarsest mesh: Control points for Q16



(e) Coarsest mesh: Bézier mesh with Bézier control points for Q16

Figure 6.1: Circular beam with end shear

NURBS-Q9			NURBS-Q16		
DOFs	Elmts	Energy	DOFS	Elmts	Energy
80	18	0.029708322024974	70	8	0.029653101738195
224	72	0.029653401864295	154	32	0.029649732629062
728	288	0.029649900409357	418	128	0.029649669346247
2600	1152	0.029649682879498	1330	512	0.029649668455942
9800	4608	0.029649669343369	4690	2048	0.029649668442595
Exact		0.029649668442380			0.029649668442380

Table 6.1: Strain energy for circular beam with NURBS elements

	Lagrange Q4		Lagrange Q9		Lagrange Q16	
DOFs	Elmts	Energy	Elmts	Energy	Elmts	Energy
182	72	0.03042038175071	18	0.02970101373401	8	0.02965327376971
650	288	0.02984351371323	72	0.02965318188484	32	0.02964975296446
2450	1152	0.02969820784232	288	0.02964989418870	128	0.02964966996157
9506	4608	0.02966180825828	1152	0.02964968266120	512	0.02964966846707
37442	18432	0.02965270370808	4608	0.02964966933301	2048	0.02964966844276
Exact		0.029649668442380		0.029649668442380		0.029649668442380

Table 6.2: Strain energy for circular beam with Lagrangian elements

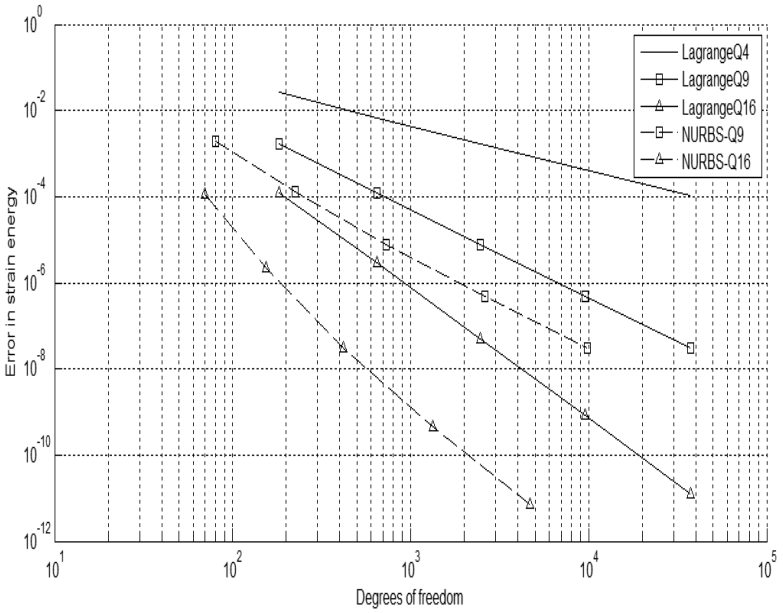
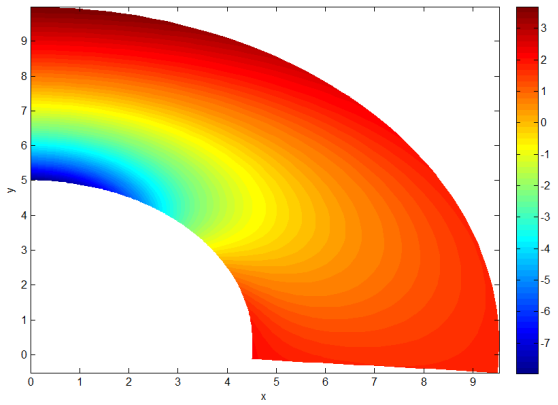
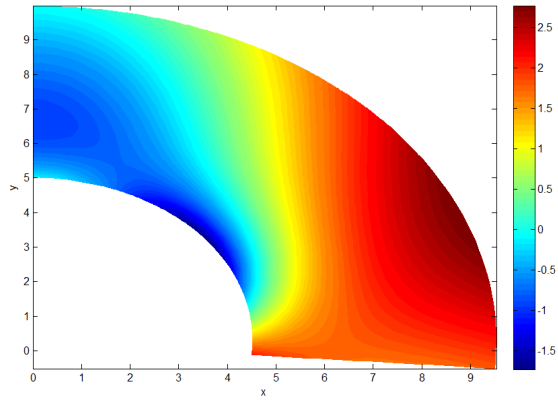


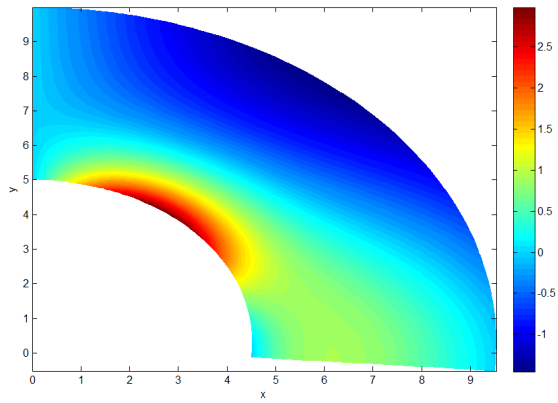
Figure 6.2: Error in strain energy vs. degrees of freedom for the circular beam



(a) σ_x



(b) σ_y



(c) τ_{xy}

Figure 6.3: Stresses in the curved beam analysed. The stresses are plotted onto the deformed shape of the beam, with displacements exaggerated 50-fold. No stress-smoothing is necessary as the stresses are continuous over element boundaries

6.2 Infinite plate with a circular hole under far-field uniaxial tension

The infinite plate with a circular hole is a frequently used benchmark problem for finite element solvers. The problem consist of a plate which is infinitely large in the x- and y-direction, and the thickness is small. The plate has a hole with radius equal 1 in the center. The plate is linear elastic, and in a state of plane strain. The elasticity modulus is 1000, and the Poisson ratio is 0.3. The plate is loaded with stress in x-direction, $\sigma_x = 1$, far from the hole.

Due to symmetry only a quarter of the plate needs to be analysed. Normally a quadratic part of the upper right quadrant is analysed, with the exact solution to the stresses along the edges applied as a traction. This approach will introduce additional singularity in the sharp corner which is created by this representation. For an isogeometric analysis this will be avoided by choosing a different parametrisation. The plate will be analysed as a quarter of an annulus, which can be exactly represented by quadratic NURBS, and therefore no further error is introduced here. The exact stresses is then applied to the outer radius of the quarter annulus as a traction boundary condition. The geometry of the infinite plate and the geometry of the analysed part is shown in Figure 6.4. The problem was analysed with NURBS and Lagrangian 9- and 16-noded quadrilaterals.

The analytical solution to stresses at any given point (r,θ) in the plate is given by [30] as

$$\begin{aligned}
\sigma_x &= 1 - \frac{a^2}{r^2}\left(\frac{3}{2}\cos 2\theta + \cos 4\theta\right) + \frac{3}{2}\frac{a^4}{r^4}\cos 4\theta \\
\sigma_y &= -\frac{a^2}{r^2}\left(\frac{1}{2}\cos 2\theta - \cos 4\theta\right) - \frac{3}{2}\frac{a^4}{r^4}\cos 4\theta \\
\tau_{xy} &= -\frac{a^2}{r^2}\left(\frac{1}{2}\sin 2\theta + \sin 4\theta\right) + \frac{3}{2}\frac{a^4}{r^4}\sin 4\theta
\end{aligned} \tag{6.2}$$

The exact strain energy of the analysed part can be calculated as

$$E = \int_V \boldsymbol{\sigma}^T \boldsymbol{\varepsilon} dV = \int_0^{\pi/2}\int_1^4 \boldsymbol{\sigma}^T \mathbf{D}^{-1}\boldsymbol{\sigma} r dr d\theta = 0.01197664128 \tag{6.3}$$

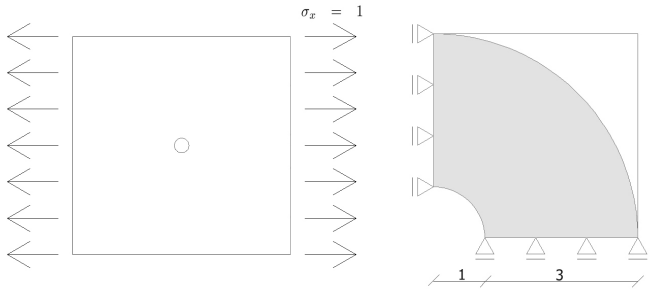
where

$$\boldsymbol{\sigma} = \begin{bmatrix}\sigma_x \\ \sigma_y \\ \tau_{xy}\end{bmatrix}, \quad \mathbf{D} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)}\end{bmatrix} \tag{6.4}$$

At the loaded edge of the plate the exact stresses is applied to the system as a traction

$$\mathbf{t} = \hat{\mathbf{n}}\begin{bmatrix}\sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y\end{bmatrix} \tag{6.5}$$

where $\hat{\mathbf{n}}$ is the unit outward normal. The plate is analysed with 9- and 16-noded elements, and the resulting strain energy is given in Tables 6.3 and 6.4, and a convergence plot is shown in Figure 6.5. The convergence rates are as expected for quadratic and cubic elements, and also for this problem the NURBS solution achieves better accuracy than the Lagrange elements, with the same convergence rates.



(a) The problem

(b) Part of the problem that is analyzed

Figure 6.4: Infinite plate with a circular hole under far-field uniaxial tension

NURBS-Q9			NURBS-Q16		
DOFs	Elmts	Energy	DOFS	Elmts	Energy
144	40	0.011953123289377	154	32	0.011973812023053
576	220	0.011975309108001	418	128	0.011976608009310
2304	1012	0.011976577690435	1330	512	0.011976640728685
9216	4324	0.011976637976321	4690	2048	0.011976641280095
36864	17860	0.011976641099244	17554	8192	0.011976641287725
Exact		0.011976641287842			0.011976641287842

Table 6.3: Strain energy for infinite plate with NURBS elements

Lagrange-Q9			Lagrange-Q16		
DOFs	Elmts	Energy	DOFS	Elmts	Energy
306	32	0.011958472345477	650	32	0.011975035686685
122	128	0.011974281506711	2450	128	0.011976572964402
4290	512	0.011976443358185	9506	512	0.011976639679251
16770	2048	0.0119766277358221	37424	2048	0.011976641260059
Exact		0.011976641287842			0.011976641287842

Table 6.4: Strain energy for infinite plate with Lagrange elements

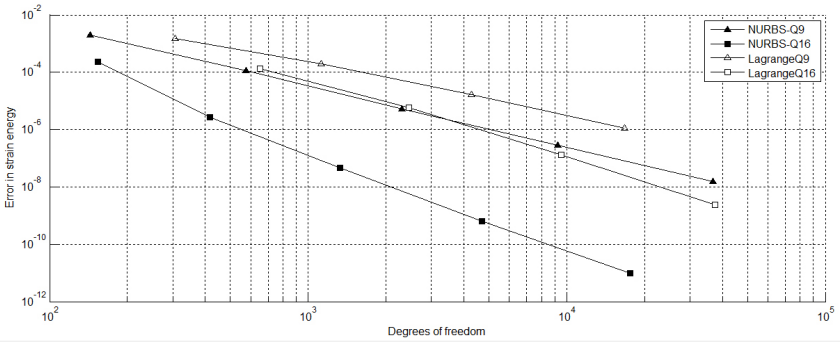


Figure 6.5: Error in strain vs. degrees of freedom for the infinite plate

6.3 Thermo-mechanical analysis: Beam with temperature gradient

Changes is temperature in a material will lead to thermal stresses and displacements. In this section, both a transient and a steady-state thermal analysis of a beam is performed. In this problem, a prescribed temperature is applied to each horizontal edge at t=0. The temperatures in the structure from the steady-state solution will then be applied to a static linear elastic analysis of the same beam, to obtain the displacements due to the thermal gradient.

Heat conduction for an isotropic material is governed by [9]

$$\frac{\partial}{\partial x}kT,x + \frac{\partial}{\partial y}kT,y + Q - c\rho\dot{T} \tag{6.6}$$

where k is the thermal conductivity, T is the temperature, Q is thermal flux, c is thermal capacity and ρ is the density. After integrating to obtain a weak form, and using a Galerkin formulation of FEM [26], the finite element formulation of heat conduction is given by

$$\int_{\Omega_e} \mathbf{N}^T \rho c \mathbf{N} d\Omega \dot{\mathbf{T}} + \int_{\Omega_e} \frac{\partial \mathbf{N}}{\partial \mathbf{x}}^T \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix} \frac{\partial \mathbf{N}}{\partial \mathbf{x}} d\Omega \mathbf{T} = \int_{\Omega_e} \mathbf{N}^T Q d\Omega - \int_{\Gamma_e q} \mathbf{N}^T \bar{q}_n d\Gamma \tag{6.7}$$

which in matrix form is

$$\mathbf{M}\dot{\mathbf{T}} + \mathbf{K}\mathbf{T} = \mathbf{F} \tag{6.8}$$

where $\mathbf{M}$ is the heat capacity matrix, and $\mathbf{K}$ is the conductivity matrix, similar to the mass- and stiffness matrices in a linear elastic problems. A central difference approach is used to calculate the value of $\dot{T}$ as

$$\dot{T}_{n+1} = \frac{T_{n+1} - T_n}{\Delta t} \tag{6.9}$$

and (6.8) is written as

$$\left(\frac{\mathbf{M}}{\Delta t} + \mathbf{K}\right) \mathbf{T}_{n+1} = \mathbf{F}_{n+1} + \frac{\mathbf{M}}{\Delta t}\mathbf{T}_n \tag{6.10}$$

A simply supported beam subjected to a thermal loading will now be analysed. The beam has length 100 units, and heigth 5 units. The material is isotropic, linear elastic with E=210000, ν=0.3, $\alpha = 1.1x10^{-5}$, k=1, ρ=1, and c=1. At t=0 a prescribed temperature of +25 is applied to the lower edge, and -25 at the upper edge of the beam. The initial temperature in the beam, at $t < 0$, is 0. The vertical edges are completely insulated. Thus, the boundary conditions are:

$$T(x,y,t<0) = 0 \quad T(x,+2.5,t\geq 0) = -25, \quad T(x,-2.5,t\geq 0) = 25,$$
$$q(0,y,t) = q(100,y,t) = 0 \tag{6.11}$$

$$u(0,0) = v(0,0) = v(100,0) = 0 \tag{6.12}$$

The beam with boundary conditions is shown in Figure 6.6

The transient problem is solved by a implicit method, which is numerically stable, without a critical time step that needs to be calculated. This in contrast

to an explicit method. This is at the cost of having to solve the system of equations at each time step. For a finite element code in Matlab, the main computational effort lies in establishing the stiffness matrix. The actual solving of the system of equations is done quickly. Hence a implicit solution method is a good choice. The transient problem is solved in the time interval $t \in (0s, 1s]$, with a time step $\Delta t = 0.01s$. The resulting temperature field is shown in Figure 6.7, for four different values of time, t.

For the case of steady-state at $t \simeq \infty$, the exact solution to the temperature in the beam is $T = -25y$. The thermal expansion of the material due to this imposed temperature field will lead to a constant curvature $\kappa = 10\alpha$ over the length of the beam. This curvature will give a deflection of the beam, which at the center line of the beam can be calculated from

$$\kappa = \frac{\partial^2 y}{\partial x^2} \tag{6.13}$$

where y is the deflection at position x. Integrating and imposing boundary conditions yield

$$y = \frac{1}{2}\kappa x^2 - 50\kappa x = 0 \tag{6.14}$$

which at center of the beam, x=50, will result in a maximum deflection of -0.1375. The solution to the temperature field is linear, and the deflection at center is a quadratic polynomial. For a quadratic element both these solution is within the trial space of the finite element solution, and the analysis is expected to obtain the exact solution of the deflection at midpoint with a single Q9-quadrilateral. This problem can then be used as a patch test to identify errors in the element code, at least to some extent.

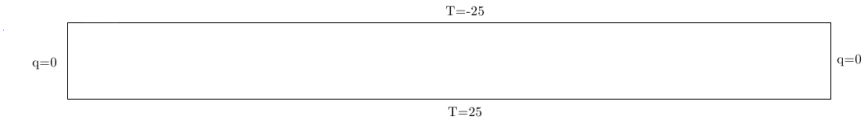
For the static elasticity problem, any change in temperature from the initial state will result in a thermal strain due to thermal expansion:

$$\boldsymbol{\varepsilon}_o = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \tau_{xy} \end{Bmatrix} = \begin{Bmatrix} \alpha \\ \alpha \\ 0 \end{Bmatrix} \Delta T, \quad \Delta T = \mathbf{N}\boldsymbol{\Delta}\mathbf{T} \tag{6.15}$$

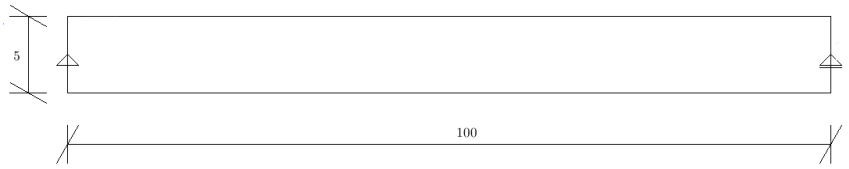
where $\boldsymbol{\Delta}\mathbf{T}$ is the change of temperature at the control points from the thermal analysis. The temperature strains are then applied to the structure in a linear elastic analysis as a consistent load vector

$$\mathbf{p}_0 = \int_{\Omega_e} \mathbf{B}^T \mathbf{D} \boldsymbol{\varepsilon}_o d\Omega = \int_{\Omega_e} \mathbf{B}^T \mathbf{D} \begin{Bmatrix} \alpha \\ \alpha \\ 0 \end{Bmatrix} \mathbf{N}\boldsymbol{\Delta}\mathbf{T} d\Omega \tag{6.16}$$

As expected, a downwards deflection v=0.1375 at the mid-node is obtained with a single Q9 element.

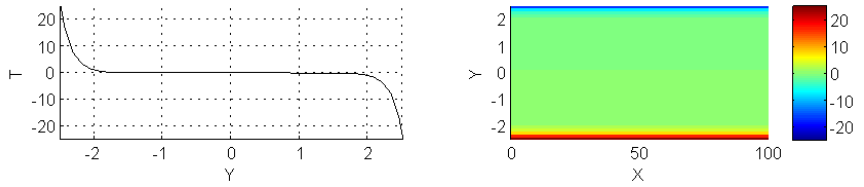


(a) Thermal boundary conditions

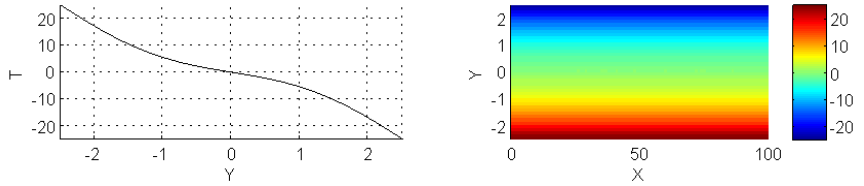


(b) Mechanical boundary conditions

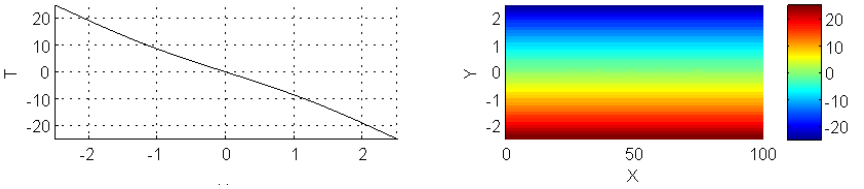
Figure 6.6: Boundary conditions for the beam that is analysed



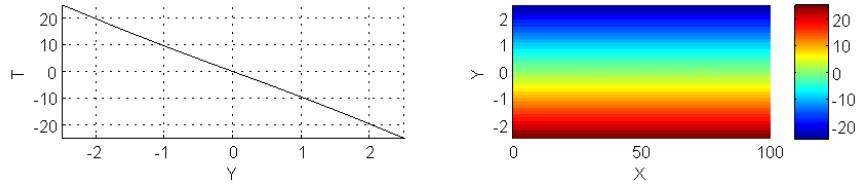
(a) t=0.01s



(b) t=0.31s



(c) t=0.61s



(d) t=0.91s $\simeq$ steady-state solution

Figure 6.7: Temperature distribution in a rectangular beam, with a temperature of ± 25 applied to the horizontal edges at t=0. The left figures are showing the y-T axis, and the right figures are x-y axis

Chapter 7

Discussion

7.1 Concluding remarks

Use of the Bézier extraction operator in order to implement isogeometric analysis in an existing finite element framework seems to be a reasonable choice. By using this method, one can easily implement isogeometric analysis (IGA) in existing software, without having to make much changes to the code. Since the Bézier extraction approach to isogeometric analysis is easing the implementation this might prove to be a essential step forward in the integration of FEM and CAD. Implementation of IGA into existing software will be important for the spread of IGA.

Also, use of isogeometric elements may not always be the best choice for all analysis purposes, and the choice of using Lagrangian elements may be better in some situations. For this purpose the user should be able to choose whether to use isogeometric or traditional elements for the analysis. Bézier extraction will allow for this, as the finite element framework can handle both isogeometric Bézier elements and Lagrange elements.

One problematic issue with Bézier extraction is the storage of the Bézier extraction operator, which can be very demanding when the number of elements increases. The simplest choice for storing the operators, at least in Matlab, is the use of a 3 dimensional matrix. But a 3-d matrix can not be stored as a sparse matrix, which leads to huge demand of memory. For the analysed problems in Chapter 6 the extraction operator demanded more memory than the stiffness matrix, which could be stored sparsely, and this restricted the number of degrees of freedom that could be used.

When it comes to the numerical solutions obtained by an isogeometric analysis, NURBS and T-spline based isogeometric analysis seems to be better choice than traditional finite element analysis. As seen in the numerical examples performed in this thesis, the accuracy is superior to Lagrangian elements which gives us less error while the convergence rates stays unchanged. Another advantage is the that the stresses calculated from an IGA with $p \geq 2$ will be continuous over element boundaries due to the C^{p-1}-continuity.

When T-splines are used, the ability to perform a local refinement is a significant advantage compared to a NURBS based isogeometric analysis. This local refinement may lead to a massive reduction of needed degrees of freedom close

to regions where smaller elements are needed, for example close to singularities in the solution. But T-splines also has flaws when it comes to linear dependence. Since T-splines in general are not linearly independent, in contrast to NURBS, the T-spline geometry from CAD software might not be suitable for analysis.

The ability to directly import geometries from CAD software is a huge advantage, and one of the main reasons for performing a isogeometric analysis instead of a traditional FEA. Since the geometry is already defined, the analyst will not need to remodel the geometry for the finite element analysis, which might be very time consuming. Also, the error in geometry description, which often is unavoidable with isoparametric elements, is now completely removed. This is of course only true if the CAD model is considered to be the exact geometry of the analysed structure. Geometric deviations from the model will also be maintained in the analysis.

7.2 Future work

In this thesis the isogeometric analysis has been implemented in a Matlab finite element framework, which is fine for academical purposes. The concept of Bézier decomposition can easily be studied in this programming environment, and for the numerical studies performed the computational cost and time consumption is not relevant. However, for engineering purposes in a commercial situation this is not acceptable, and the computational efficiency needs to be improved.

In this thesis Bézier extraction of T-splines has only been used with imported extraction operators from CAD software. While importing T-splines from CAD software is useful, refinement should be performed by analysis software and not design software. Therefore the Bézier extraction operator will also need to be extracted by the analysis software, and this algorithm needs to be implemented.

As mentioned briefly in the introduction, LR B-splines is another approach to local refinement of splines, and research is being done on this topic. In parallel to this work, Bézier decomposition of LR B-splines should also be researched. Since this class of of splines are also based on B-splines, Bézier extraction should be possible similarly to NURBS and T-splines.

Appendix A

Algorithms

The two most conceptually important algorithms in an isogeometric analysis based on Bézier extraction is the algorithm to create the Bézier extraction operator and the shape function algorithms. Here these two are presented.

In Algorithm 1 the algorithm for the construction of the univariate extraction operators for NURBS is presented. This algorithm requires only the knot vector and the polynomial order.

In Algorithm 2 the shape function algorithm is presented. This algorithm will work for both NURBS and T-splines with extraction operator. The algorithm can handle a varying number of shape functions with support in each elements due to T-splines. This algorithm requires an additional algorithm, bernstein_basis, which calculates the values of the Bernstein polynomials and derivatives. This is given in Algorithm 3. Both algorithms 1 and 2 are written for Matlab, and are slightly modified versions of the algorithms given in [5].

A parsing script that will import the Bézier extraction operators and IEN matrix from Rhino to a readable format for a Matlab finite element program is given in Algorithm 5. This script require a text file where all information other than numbers are removed prior to the parsing.

Input: Knot vector and polynomial order
Output: Univariate element extraction operators, and number of elements nb

```
1: function [C  nb] = bezier_extraction(knot,p)
2: m=length(knot)-p-1;
3: a=p+1;
4: b=a+1;
5: nb=1;
6: C(:,:,1) = eye(p+1);
7: while b≤m do
8:    C(:,:,nb+1) = eye(p+1);
9:    i=b;
10:   while b ≤ m && knot(b+1) == knot(b); do
11:      b=b+1;
12:   end while
13:   multiplicity = b-i+1;
14:   if multiplicity < p then
15:      numerator=knot(b)-knot(a);
16:      for  j=p:-1:multiplicity+1  do
17:         alphas(j-multiplicity)=numerator/(knot(a+j)-knot(a));
18:      end for
19:      r=p-multiplicity;
20:      for j=1:r do
21:         save = r-j+1;
22:         s = multiplicity + j;
23:         for k=p+1:-1:s+1 do
24:            alpha=alphas(k-s);
25:            C(:,k,nb)=alpha*C(:,k,nb)+(1-alpha)*C(:,k-1,nb);
26:         end for
27:         if  b≤m then
28:            C(save:save+j,save,nb+1)=C(p-j+1:p+1,p+1,nb);
29:         end if
30:      end for
31:      nb=nb+1;
32:      if b ≤ m then
33:         a=b;
34:         b=b+1;
35:      end if
36:   else if multiplicity==p then
37:      if b ≤ m then
38:         nb=nb+1;
39:         a=b;
40:         b=b+1;
41:      end if
42:   end if
43: end while
```

Algorithm 1: Algorithm to create univariate Bézier extractors from knot vector

Input: Quadrature points GP=[ξ η], element number e, Bézier weights W_b, Element order p and q, bivariate element extraction operator C, IEN-matrix, Weights W, Control points P

Output: Shape function values $\mathbf{R}$, Shape function derivatives $\frac{\partial \mathbf{R}}{\partial \mathbf{x}}$, determinant of the Jacobian $|J|$

```
1: function [R dRdx detJ]=shape2dIGA(GP,e,Wb,p,q,C,IEN,W,P)
2: %% Initialize variables:
3: ncpt=(p+1)*(q+1);
4: B=zeros(ncpt,1);
5: dBdxi=zeros(ncpt,2);
6:
7: wb=0;
8: dwbdxi=zeros(2,1);
9:
10: R=zeros(nen,1);
11: dRdxi=zeros(nen,2);
12: dRdx=zeros(nen,2);
13:
14: %% Calculations:
15: %% Calculate bernstein shape function and its derivatives
16: [B   dBdxi]=bernstein_basis(p,q,GP(1),GP(2));
17:
18: %% Calculate weight function and its derivatives
19: for a=1:ncpt; do
20:    wb = wb + B(a)*Wb(a);
21:    dwbdxi(1) = dwbdxi(1) + dBdxi(a,1)*Wb(a);
22:    dwbdxi(2) = dwbdxi(2) + dBdxi(a,2)*Wb(a);
23: end for
24:
25: a=find(IEN(:,e));
26: C=C(a,:);
27: W=W(IEN(a,e));
28: %% Shapefunction and derivatives
29: R=diag(W)*C*B/wb;
30: dRdxi(:,1)=diag(W)*C*(dBdxi(:,1)/wb-dwbdxi(1)*B/(wb*wb));
31: dRdxi(:,2)=diag(W)*C*(dBdxi(:,2)/wb-dwbdxi(2)*B/(wb*wb));
32: %% Jacobian matrix
33: dxdxi=P'*dRdxi;
34:
35: dxidx=inv(dxdxi);
36: dRdx=dRdxi*dxidx;
37: detJ=det(dxdxi);
```

Algorithm 2: Shape function algorithm

```
function [B dBdxi]=bernstein_basis(p,q,xi,eta)
%% Initialization
ncpt=(p+1)*(q+1);
B=zeros(ncpt,1);
dBdxi=zeros(ncpt,2);
%% Calculation
for j=1:q+1 do
  for i=1:p+1 do
    B((p+1)*(j-1)+i)=bernstein(p,i,xi)*bernstein(q,j,eta);
    dBdxi((p+1)*(j-1)+i,1)=0.5*p*(bernstein(p-1,i-1,xi)-bernstein(p-
    1,i,xi))*bernstein(q,j,eta);
    dBdxi((p+1)*(j-1)+i,2)=bernstein(p,i,xi)*0.5*q*(bernstein(q-1,j-
    1,eta)-bernstein(q-1,j,eta));
  end for
end for
```

Algorithm 3: Bivariate Bernstein polynomials and derivatives

```
function B=bernstein(p,a,xi)
if p==0 && a==1 then
  B=1;
else if p==0 && a~=1 then
  B=0;
else
  if  a<1 || a>p+1 then
    B=0;
  else
    B1=bernstein(p-1,a,xi);
    B2=bernstein(p-1,a-1,xi);
    B=0.5*(1-xi)*B1+0.5*(1+xi)*B2;
  end if
end if
```

Algorithm 4: Univariate Bernstein polynomial

Input: Text file "Bezier.txt" with extraction operators from Rhino
Output: Bézier extraction operators C, and IEN matrix

```
clear all
id = fopen('Bezier.txt');
readin=0;
IEN=0;
while readin != -1 do
  readin=fgetl(id);
  if readin == -1 then
    break
  end if
  readin = str2num(readin);
  readin(1) = readin(1)+1;
  readin(2) = readin(2)+1;
  readin(3) = readin(3)+1;
  j=0;
  for i = 5:2:length(readin); do
    readin(i)=readin(i)+1;
    % Establish IEN Matrix
    if  readin(1)>size(IEN)(1) then
      IEN(readin(1),1)=0;
    end if
    if find(IEN(readin(1),:)==readin(i)) then

    else
      a=find(IEN(readin(1),:));
      IEN(readin(1),length(a)+1)=readin(i);
    end if
    % Establish C matrices
    c(4*(readin(2)-1)+readin(3),readin(i),readin(1))=readin(i-1);
  end for
end while
C=zeros(size(IEN)(2),16,size(IEN)(1));
for i=1:size(IEN)(1) do
  a=find(IEN(i,:));
  IEN(i,a)=sort(IEN(i,a));
  C(a,:,i)=c(:,IEN(i,a),i)';
end for
```

Algorithm 5: Parsing script for Rhino

Appendix B

Report for MekIT11

The following report, "Isogeometric Analysis based on Bézier Extraction of NURBS and T-Splines", was written with Thanh Ngan Nguyen, Kjetil André Johannessen and Kjell Magne Mathisen for the MekIT'11 Sixth National Conference on Computational Mechanics, held in Trondheim 23^{rd}-24^{th} May 2011.

Isogeometric finite element analysis based on Bézier extraction of NURBS and T-splines

Thanh Ngan Nguyen[*†], **Ole Jørgen Fredheim**[*†], **Kjetil André Johannessen**[‡] **and Kjell Magne Mathisen**[†]

[†)] Department of Structural Engineering,
Norwegian University of Science and Technology, N-7491 Trondheim, Norway
e-mail: thanhnga@stud.ntnu.no, olejorfr@stud.ntnu.no, kjell.mathisen@ntnu.no

[‡)] Department of Mathematical Sciences,
Norwegian University of Science and Technology, N-7491 Trondheim, Norway
e-mail: kjetijo@math.ntnu.no

Summary The presented study addresses use of Bézier extraction for NURBS and T-spline based isogeometric analysis. In isogeometric analysis the shape functions are not confined to one single element, but spans several elements, which complicates implementation. The Bézier extraction operator decomposes the NURBS or T-spline basis functions to Bernstein polynomials which allows generation of C^0-continuous Bézier elements, where all necessary changes in the finite element code are localized to the shape function routine. We will shortly review the theory of NURBS and T-splines and show how to compute the Bézier extraction operator. Also, numerical studies are performed to investigate performance of isogeometric analysis compared to traditional finite element analysis.

Introduction

Isogeometric analysis was introduced by Hughes *et al.* [5, 6]. The concept of isogeometric analysis is to use the same basis for the analysis as is being used in description of the geometry. This as opposed to the traditional finite element method (FEM), where the basis for the analysis is what is used to describe the geometry. Computer Aided Engineering (CAE) was introduced earlier than Computer Aided Design (CAD), and CAE and CAD have been developed independently. The idea of isogeometric analysis will help integrating these two concepts, and allow use of geometric models directly from CAD software in a finite element analysis (FEA).

In this paper we start with presenting the basic theory for B-splines, the non-rational part of NURBS, before the fundamentals of NURBS and T-splines are reviewed briefly. Then we describe the construction of isogeometric Bézier elements and the Bézier extraction operator for NURBS. A thorough example of the bivariate extraction operator is included. The Bézier extraction operator for T-splines as opposed to the extraction operator for NURBS is then discussed.

Numerical studies are performed using a finite element (FE) solver based on Bézier extraction of NURBS. The two examples consist of problems involving conical sections, where isogeometric analysis have the advantage in exact representation of the geometry. The first example is a cantilevered beam shaped as a quarter of a circle and the second is an infinite plate with circular hole subjected to far-field uniaxial tension. The latter problem is often modelled as a quarter of the geometry with the outer edge square shaped. Here, we have chosen to model the plate as a quarter of disk, which gives a geometry without singularties.

B-splines

Knot vector

A knot vector is a set of increasing parameter space coordinates. Parameter space is the space where the basis functions are defined, and is partitioned into knot spans between the knots. The

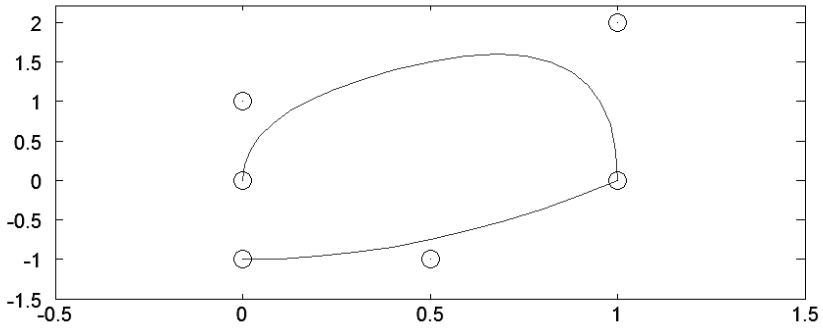


Figure 1: B-spline curve constructed from quadratic basis functions, and knot vector $\Xi = \{0, 0, 0, 1, 2, 2, 3, 3, 3\}$. The control points are marked as circles.

knot vector is written as $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, where ξ_i is the i^{th} knot, i is the knot index, $i = 1, 2, \ldots, n + p + 1$, p is the polynomial order, and n is the number of basis functions used to create the B-spline curve.

Basis functions

B-splines are piecewise polynomial functions, and are defined by the following recursive formulas [3, 4]

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

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{2}$$

B-spline curves

B-spline curves (see figure 1) are created by a linear combination of B-spline basis functions. What separates B-spline curves from curves in FEA is that instead of interpolating a set of nodal points, the B-splines are related to a set of control points. These control points are the equivalent to the nodes, but the curve will generally not pass through the control points. For a given set of n p^{th} order basis functions, $N_{i,p}(\xi)$, $i = 1, 2 \ldots, n$, and a corresponding set of control points $\mathbf{B}_i \in \mathbb{R}^d$, $i = 1, 2, \ldots, n$, the piecewise-polynomial B-spline curve is given by

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) \mathbf{B}_i \tag{3}$$

B-spline surfaces

The expansion from B-spline curves to B-spline surfaces is straightforward. To generate a surface, we will need a net of control points $\{\mathbf{B}_{i,j}\}$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$, polynomial orders p and q, and knot vectors $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, and $\mathcal{H} = \{\eta_1, \eta_2, \ldots, \eta_{m+q+1}\}$. A tensor product B-spline surface is then defined by

$$\mathbf{S}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \mathbf{B}_{i,j} \tag{4}$$

where $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$ are univariate B-spline basis functions of order p and q, corresponding to knot vectors Ξ and $\mathcal{H}$, respectively.

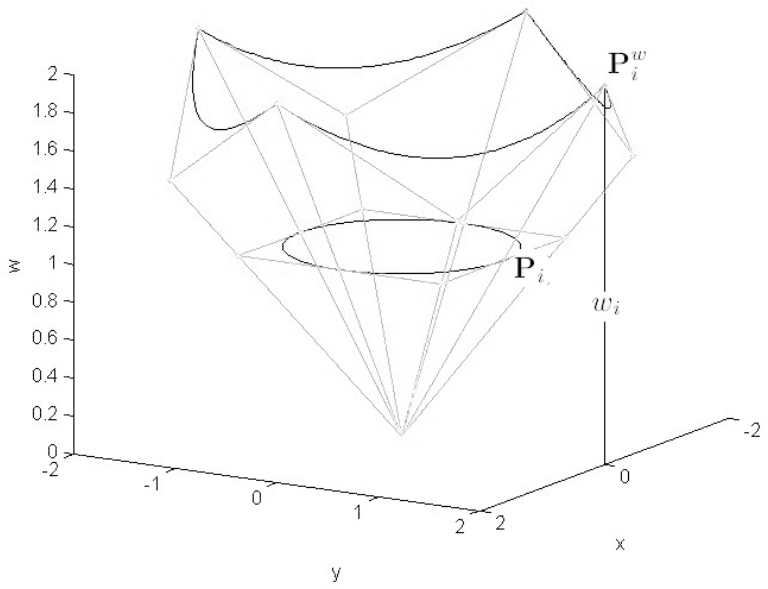


Figure 2: B-spline curve projected onto the plane $z = 1$ to create the NURBS representation of a circle.

Knot insertion

If a new knot, and corresponding control point, is added to the knot vector, the resulting B-spline curve will in general be different than the original curve. However, knots may be inserted in the knot vector without altering the B-spline curve if the control points are placed according to a specific knot insertion algorithm. Let $\Xi = \{\xi_1, \xi_2, ..., \xi_{n+p+1}\}$ be a given knot vector. Inserting a new knot $\bar{\xi} \in [\xi_k, \xi_{k+1}]$ with $k > p$ into the knot vector requires $n + 1$ new basis functions to be defined using equations 1 and 2. The $m = n + 1$ new control points, $\left\{\bar{\mathbf{P}}_i\right\}_{A=1}^{m}$, are formed from the original control points, $\{\mathbf{P}_i\}_{A=1}^{n}$, by

$$\bar{P}_A = \begin{cases} P_1 & A = 1 \\ \alpha_A P_A + (1 - \alpha_A) P_A & 1 < A < m \\ P_n & A = m \end{cases} \tag{5}$$

$$\alpha_A = \begin{cases} 1 & 1 < A < k - p \\ \frac{\bar{\xi} - \xi_A}{\xi_{A+p} - \xi_A} & k - p + 1 < A < k \\ 0 & A \geq k + 1 \end{cases} \tag{6}$$

Non-uniform rational B-splines

Non-uniform rational B-splines (NURBS) is an expansion from B-splines, which will remove some of the limitations of B-splines and allow us to exactly represent conical sections. Figure 2 shows an example of a B-spline curve projected onto the plane $z = 1$ to create the NURBS representation of a circle. A NURBS entity in $\mathbb{R}^d$ is the result of a projection of a B-spline entity in $\mathbb{R}^{d+1}$. The control points, $\mathbf{P}_i$, and weights, w_i, are calculated as

$$(\mathbf{P}_i)_j = (\mathbf{P}_i^w)_j / w_i, \; j = 1, \ldots, d \tag{7}$$

$$w_i = (\mathbf{P}_i^w)_{d+1} \tag{8}$$

The univariate rational basis functions $R_{i,p}(\xi)$ are defined as

$$R_{i,p}(\xi) = \frac{N_{i,p}(\xi)w_i}{W(\xi)} = \frac{N_{i,p}(\xi)w_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i},p}(\xi)w_{\hat{i}}} \tag{9}$$

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

$$W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi)w_i \tag{10}$$

and a NURBS curve is defined equivalently as its B-spline counterpart

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} R_{i,p}(\xi)\mathbf{P}_i \tag{11}$$

The bivariate basis functions for a surface is defined as

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_{i,p}(\xi)M_{j,q}(\eta)w_{i,j}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m} N_{\hat{i},p}(\xi)M_{\hat{j},q}(\eta)w_{\hat{i},\hat{j}}} = \frac{N_A(\xi,\eta)w_A}{W(\xi,\eta)} \tag{12}$$

We define $\mathbf{W}$ as the diagonal matrix of weights,

$$W_{ij} = w_i\delta_{ij} \tag{13}$$

and $\mathbf{N}(\xi)$ as a vector of basis function values, and rewrite equations (9) and (12) in matrix form

$$\mathbf{R}(\xi) = \frac{1}{W(\xi)}\mathbf{W}\mathbf{N}(\xi) \tag{14}$$

$$\mathbf{R}(\xi,\eta) = \frac{1}{W(\xi,\eta)}\mathbf{W}\mathbf{N}(\xi,\eta) \tag{15}$$

T-splines

Introduction

The theory and formulas presented is extracted from [1] and [7]. NURBS represent a restricted subset of T-splines since T-splines overcome the tensor product restriction associated with NURBS. This means that T-splines allow local refinement. While NURBS control points lie in a rectangular grid, rows and columns of T-spline control points may be incomplete as illustrated in figure 3, forming *T-junctions* in the *T-mesh*.

Local refinement has many benefits. For the same geometric representation, T-splines give less control points compared to NURBS, implying lower computational cost when performing analyses. Gaps which are non-avoidable in a NURBS model may be closed using T-spline merging, making it possible to create complex but still analysis-suitable models.

T-spline fundamentals

The origin of the T-mesh is the *index space*. Like a NURBS index space, each knot line represents a knot value, but the T-mesh knot lines may be incomplete. The top part of figure 4 illustrates a simple T-mesh. A valid T-mesh defines a T-spline basis function to each *anchor* and corresponding control point. If the polynomial order is even, the anchors are the mid-points of

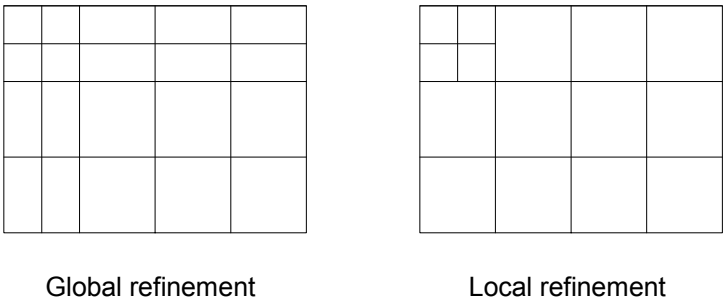


Figure 3: Global and local refinement.

the rectangles in the T-mesh. For odd polynomial degrees, the anchors coincide with the T-mesh vertices. The latter is most convenient in the review of T-spline fundamentals, and anchors of even polynomial degrees are therefore not considered.

To obtain a valid T-mesh, *local knot vectors* must be defined for each anchor. Consider the example in bottom left of figure 4 where the polynomial order $p = 3$. The local knot vector is found by marching horizontally and vertically from the anchor $\mathbf{s}_i$ until $\frac{(p+1)}{2}$ orthogonal edges or lines that terminates in a T-junction are encountered in each of the four directions from the anchor. If boundary edges are passed, the knot value is repeated until the places are filled up. The local knot vectors to $\mathbf{s}_1$ are therefore $\Xi_1 = \{\xi_1, \xi_1, \xi_2, \xi_3, \xi_4\}$ and $\mathcal{H}_1 = \{\eta_1, \eta_1, \eta_2, \eta_3, \eta_4\}$. Note that the length of the local knot vector is $p + 2$.

Often, the origin of the local knot vector in the index space is not of interest. A *local knot interval vector* is therefore defined as a sequence of knot intervals, $\Delta\Xi = \{\Delta\xi_1, \Delta\xi_2, ..., \Delta\xi_{p+1}\}$, such that $\Delta\Xi = \xi_{i+1} - \xi_i$. The *local basis function domain* is then defined as $\hat{\Omega}_A = [0, \Delta\xi_1 + \Delta\xi_2 + ... + \Delta\xi_{p+1}] \times [0, \Delta\eta_1 + \Delta\eta_2 + ... + \Delta\eta_{p+1}]$, $A = 1, 2, ...n$, where n is the number of control points. Over each local basis function domain, the T-spline basis functions in the parameter space are found similarly to NURBS basis functions, equation 4.

For a NURBS mesh, reduced continuity appears only at knot lines. In contrast to this, a T-mesh contains extra *lines of reduced continuity* which do not coincide with the knot lines. These lines are typically marked as dotted lines. The T-mesh including the lines of reduced continuity is referred to as the *extended T-mesh*, and it is over this mesh *T-spline elements* are defined. T-spline elements are rectangular regions over which the T-spline basis functions are smooth (C^∞ continuous). Thus, it is over these elements an analysis of numerical (Gaussian) quadrature can be performed. To find the lines of reduced continuity, at each T-junction extend the line until $\frac{(p+1)}{2}$ orthogonal edges are encountered. Note that this only prevails for odd polynomial degrees. The extended T-mesh for the example above appears as shown in the bottom right of figure 4. For a T-mesh to be analysis-suitable, all T-spline basis functions should also be linearly independent to each other. This requires that no lines of reduced continuity are intersecting.

Bézier Extraction Operator

Bernstein polynomials and Bézier curves

A set of Bernstein polynomial basis functions are defined as $\mathbf{B}(\xi) = \{B_{a,p}(\xi)\}_{a=1}^{p+1}$, which corresponds to the set of vector valued control points $\mathbf{P} = \{\mathbf{P}_a\}_{a=1}^{p+1}$ where each $\mathbf{P}_a \in \mathbb{R}^d$, where d is the number of spatial dimensions and $\mathbf{P}$ is a matrix of dimension $n \times d$. The Bernstein polynomials can be defined recursively as [2]

$$B_{a,p}(\xi) = (1 - \xi)B_{a,p-1}(\xi) + \xi B_{a-1,p-1}(\xi) \quad \xi \in [0, 1] \tag{16}$$

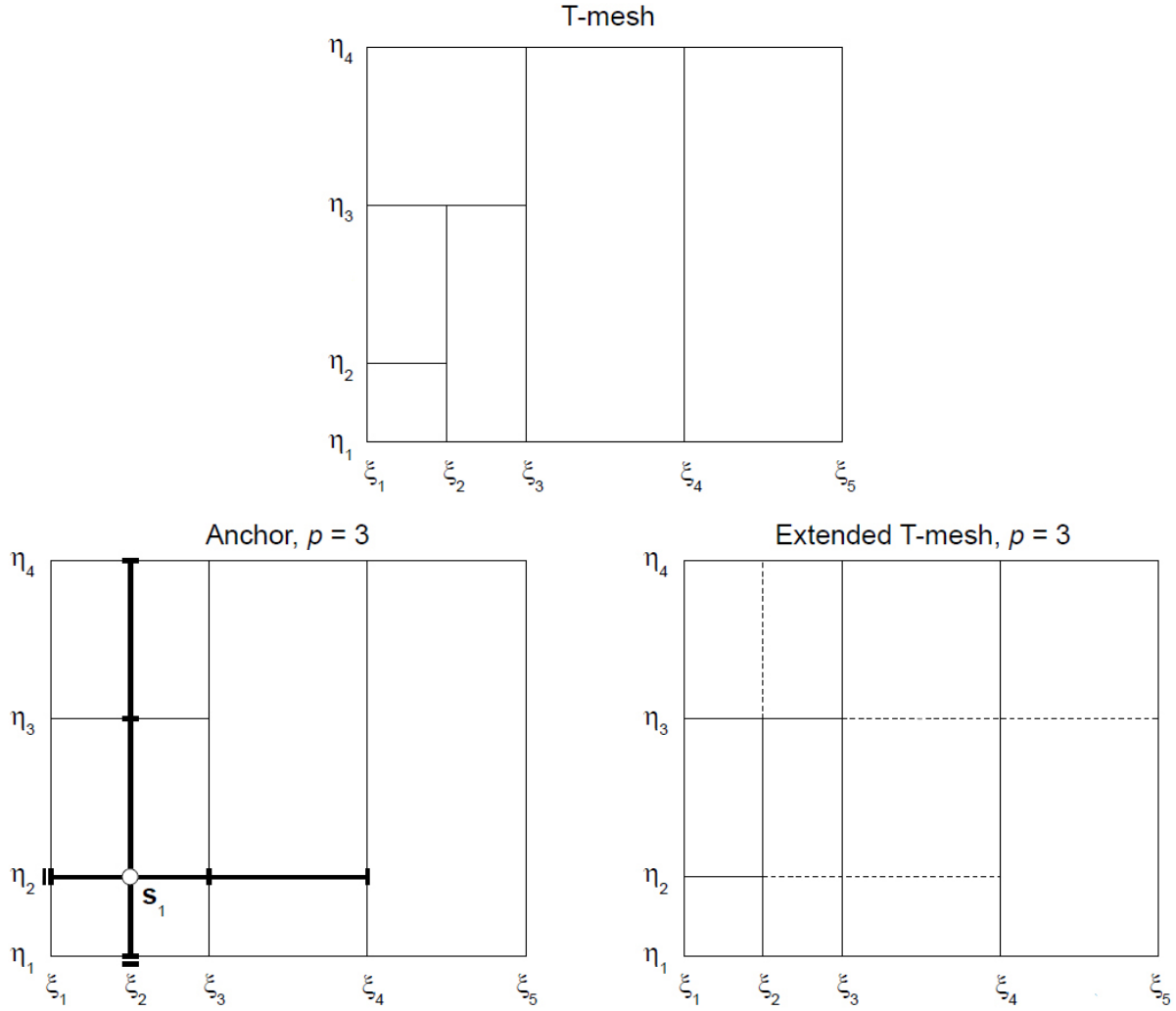


Figure 4: T-mesh, anchors of odd polynomial degrees ($p = q = 3$) and extended T-mesh, $p = 3$.

where

$$B_{1,0}(\xi) \equiv 1 \tag{17}$$

and

$$B_{a,p}(\xi) \equiv 0 \text{ if } a < 1 \text{ or } a > p + 1 \tag{18}$$

A Bézier curve of degree p is a linear combination of $p+1$ Bernstein polynomial basis functions and can be written as

$$C(\xi) = \sum_{a=1}^{p+1} \mathbf{P}_a B_{a,p}(\xi) = \mathbf{P}^T \mathbf{B}(\xi) \tag{19}$$

The Bernstein polynomials are defined over the interval [0,1], while in the FEM the Lagrange functions are used in quadrature over the interval [-1,1], thus it is reasonable to redefine the basis functions so that they span this interval. By doing so the basis functions read

$$B_{a,p} = \frac{1}{2}(1 - \xi)B_{a,p-q}(\xi) + \frac{1}{2}(1 + \xi)B_{a-1,p-1}(\xi) \tag{20}$$

and the derivatives

$$\frac{\partial B_{a,p}}{\partial \xi} = \frac{1}{2}p\big(B_{a-1,p-1}(\xi) - B_{a,p-1}(\xi)\big) \tag{21}$$

Bézier decomposition

Given a B-spline curve $T(\xi)$ of order p, and a knot vector $\Xi = [\xi_1, \xi_2, \ldots, \xi_{n+p+1}]$, additional knots may be inserted at the internal knots, by the use of equations (5) and (6), until the multiplicity of each knot equals p. By doing so, the B-spline basis functions will be C^0-continuous between elements, and within each element they will be identical to the Bernstein polynomials of order p. This series of knot insertions is called Bézier decomposition.

Assume that we are given a knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ and a set of control points $\mathbf{P} = \{\mathbf{P}_A\}_{A=1}^{n}$, that define a B-spline curve. Let $\{\bar{\xi}_1, \bar{\xi}_2, \ldots, \bar{\xi}_m\}$ be the set of knots that are required to produce the Bézier decomposition of the B-spline. Then for each new knot, $\bar{\xi}_j, j = 1, 2, \ldots, m$, we define $\alpha_A^j, A = 1, 2, \ldots, n + j$, to be the A^{th} alpha as defined in equation (6). Now, defining $C^j \in \mathbb{R}^{(n+j-1)\times(n+j)}$ to be

$$C^j = \begin{bmatrix} \alpha_1 & 1-\alpha_2 & 0 & \ldots & & & 0 \\ 0 & \alpha_2 & 1-\alpha_3 & 0 & \ldots & & 0 \\ 0 & 0 & \alpha_3 & 1-\alpha_4 & 0 & \ldots & 0 \\ \vdots & & & & \ddots & & \\ 0 & \ldots & & & 0 & \alpha_{n+j-1} & 1-\alpha_{n+j} \end{bmatrix} \tag{22}$$

and letting $\bar{\mathbf{P}}^1 = \mathbf{P}$, we can rewrite equation (6) in matrix form to represent the sequence of knot insertions needed as

$$\bar{\mathbf{P}}^{j+1} = (\mathbf{C}^j)^T \mathbf{P}^j \tag{23}$$

The control points for the Bézier elements, $\mathbf{P}^b$, are given as the final set of control points, $\mathbf{P}^b = \bar{\mathbf{P}}^{m+1}$. Defining $\mathbf{C}^T = (\mathbf{C}^m)^T (\mathbf{C}^{m-1})^T \ldots (\mathbf{C}^1)^T$ gives us

$$\mathbf{P}^b = \mathbf{C}^T \mathbf{P} \tag{24}$$

Since the Bézier decomposition of a curve does not cause any parametric or geometric change to a curve, we can write

$$T(\xi) = \mathbf{P}^T \mathbf{N}(\xi) = (\mathbf{P}^b)^T \mathbf{B}(\xi) = (\mathbf{C}^T \mathbf{P})^T \mathbf{B}(\xi) = \mathbf{P}^T \mathbf{C}\mathbf{B}(\xi) \tag{25}$$

The control points $\mathbf{P}$ are arbitrary, thus we have shown that

$$\mathbf{N}(\xi) = \mathbf{C}\mathbf{B}(\xi) \tag{26}$$

where $\mathbf{C}$ is the linear Bézier extraction operator. The Bézier extraction operator is constructed with only information from the knot vector, and it does not depend on the control points of the B-spline curve or the basis functions. NURBS are constructed from the B-spline basis functions, which allows us to apply the extraction operator to NURBS. Substituting equation (26) into (14),

$$T(\xi) = \frac{1}{W(\xi)} \mathbf{P}^T \mathbf{W} \mathbf{N}(\xi) = \frac{1}{W(\xi)} \mathbf{P}^T \mathbf{W} \mathbf{C}\mathbf{B}(\xi) = \frac{1}{W(\xi)} (\mathbf{C}^T \mathbf{W} \mathbf{P})^T \mathbf{B}(\xi) \tag{27}$$

We will also rewrite the weight function, $W(\xi)$, in terms of the Bernstein basis as

$$W(\xi) = \sum_{i=1}^{n} w_i N_i(\xi) = \mathbf{w}^T \mathbf{N}(\xi) = \mathbf{w}^T \mathbf{C}\mathbf{B}(\xi) = (\mathbf{C}^T \mathbf{w})^T \mathbf{B}(\xi) = (\mathbf{w}^b)^T \mathbf{B}(\xi) = W^b(\xi) \tag{28}$$

Where $\mathbf{w}^b = \mathbf{C}^T\mathbf{w}$ are the Bézier weights. As with knot insertion, Bézier decomposition of control points are done directly to the B-spline curve which defines the NURBS curve. Geometrically this is done by projecting the NURBS control points into $d+1$ dimensions, then the Bézier extraction operator is applied to the B-spline control points, and finally the curve is projected back into d dimensions to obtain the Bézier control points, $\mathbf{P}^b$. We define $\mathbf{W}^b$ to be the diagonal matrix consisting of Bézier weights, equivalent to (13),

$$W_{ij}^b = w_i^b \delta_{ij} \tag{29}$$

Now the Bézier decomposition of the NURBS control points, $\mathbf{P}^b$, can be calculated as

$$\mathbf{P}^b = (\mathbf{W}^b)^{-1}\mathbf{C}^T\mathbf{W}\mathbf{P} \tag{30}$$

We premultiply by $\mathbf{W}^b$ to get

$$\mathbf{W}^b\mathbf{P}^b = \mathbf{C}^T\mathbf{W}\mathbf{P} \tag{31}$$

and then substitute into equation (27) to obtain the equation for a NURBS curve in terms of C^0 Bézier elements,

$$T(\xi) = \frac{1}{W^b(\xi)}(\mathbf{W}^b\mathbf{P}^b)^T\mathbf{B}(\xi) = \sum_{i=1}^{n+m} \frac{\mathbf{P}_i^b w_i^b B_i(\xi)}{W^b(\xi)} \tag{32}$$

The bivariate extraction operator for an element is defined as

$$\mathbf{C}_A^e = \mathbf{C}_\eta^i \otimes \mathbf{C}_\xi^j \tag{33}$$

where $\otimes$ is the tensor product and is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{11}\mathbf{B} & A_{12}\mathbf{B} & \\ A_{21}\mathbf{B} & A_{22}\mathbf{B} & \\ \vdots & & \ddots \end{bmatrix} \tag{34}$$

Example of Bézier decomposition

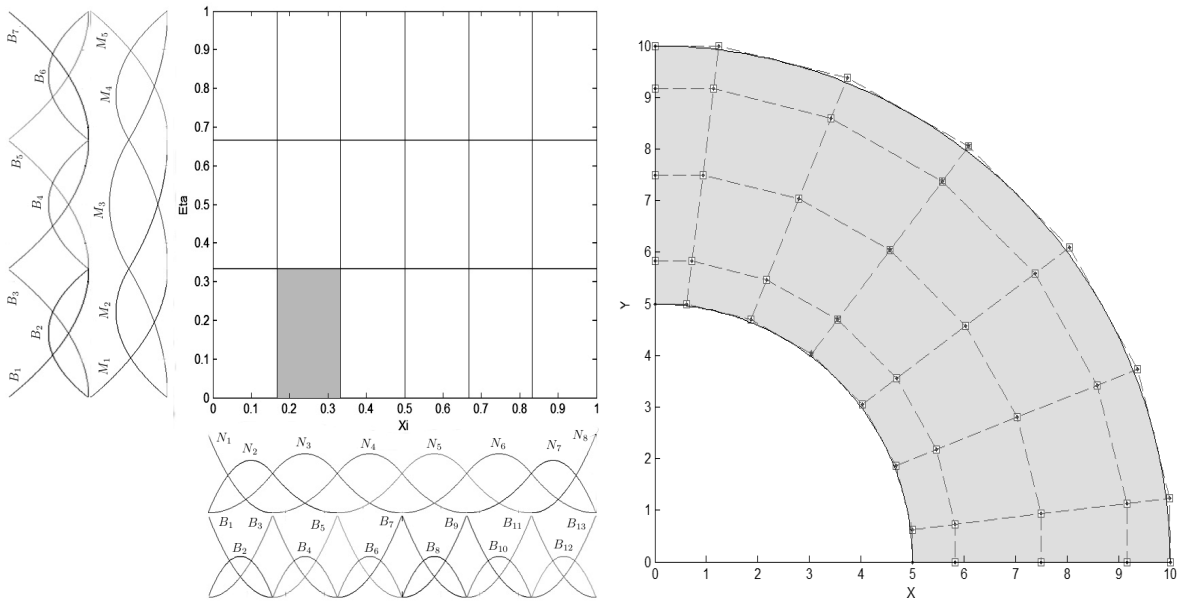
In order to increase our understanding of the Bézier decomposition we will take a closer look at a circular beam that is to be analysed. In the analysis we want to use quadratic NURBS basis functions, and we want an element mesh consisting of 6 elements in the tangential direction, and 3 elements in the radial direction. Thus, the parametric space will be defined by the open knot vectors

$$\Xi = \{0, 0, 0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}, 1, 1, 1\} \tag{35}$$

and

$$\mathcal{H} = \{0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1\} \tag{36}$$

In the parametric directions ξ and η we have the univariate B-spline basis functions $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$, respectively. The basis functions are plotted in figure 5(a).



(a) Parametric space with univariate basis functions and Bernstein polynomials plotted along the axes

(b) Circular beam with control point net

Figure 5: Parametric space and control points.

Recalling equation (26), and dropping subscripts p and q, we can write the basis functions in terms of the Bézier extraction operator and Bernstein polynomials as

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \\ N_7 \\ N_8 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{Bmatrix} \tag{37}$$

$$\begin{Bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 1.0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.0 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \end{Bmatrix} \tag{38}$$

With the information in figure 5(a) and equations (37) and (38), we can localize the element

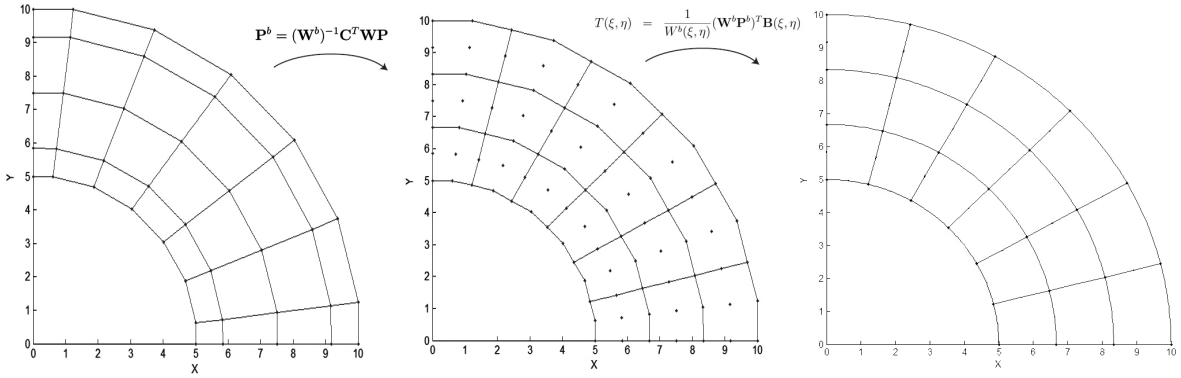


Figure 6: From control points to Bézier control points to Bézier physical mesh.

extraction operators. For the shaded element in figure 5(a) we get

$$\begin{Bmatrix} N_1^2 \\ N_2^2 \\ N_3^2 \end{Bmatrix} = \begin{Bmatrix} N_2 \\ N_3 \\ N_4 \end{Bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_3 \\ B_4 \\ B_5 \end{Bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1^2 \\ B_2^2 \\ B_3^2 \end{Bmatrix} \tag{39}$$

$$\begin{Bmatrix} M_1^1 \\ M_2^1 \\ M_3^1 \end{Bmatrix} = \begin{Bmatrix} M_1 \\ M_2 \\ M_3 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1^1 \\ B_2^1 \\ B_3^1 \end{Bmatrix} \tag{40}$$

where the superscript denotes element number in each parametric direction. In general the global extraction operator does not need to be calculated, since the local extraction operators will be calculated for each element. Here we have chosen to calculate it for the sake of clarity. The bivariate extraction operator for the shaded element in figure 5(a) then becomes

$$\mathbf{C}^2 = \mathbf{C}_\eta^1 \otimes \mathbf{C}_\xi^2 = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \otimes \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \tag{41}$$

With the extraction operators at hand we can compute the control points for the Bézier elements from equation (30).

Bézier extraction of T-splines

The theory presented is extracted from [7]. FE data structures for T-splines based on Bézier extraction is a generalization of data structures based on Bézier extraction of NURBS. Like Bézier extraction of NURBS, the idea is to extract the linear operator which maps the Bernstein polynomials on Bézier elements to the global T-spline basis.

For T-splines, no global tensor product domain exist however, a local domain can be defined for each basis function. Thus, the element extraction operators are not computed as a tensor product for each element as for NURBS. In contrast, the computation of the operators is performed function-by-function, resulting in a single row to each basis function in support of the T-spline element.

The second difference compared to NURBS is due to the local knot vectors of T-splines. Since the local knot vectors are in general not open, an extended knot vector is introduced by repeating the first and last knots until the multiplicity is equal to $p+1$. Figure 7 shows the univariate T-spline basis function N_3 (thick solid line) to the local knot vector $\Xi = \{0, 0, 1, 2, 3\}$.

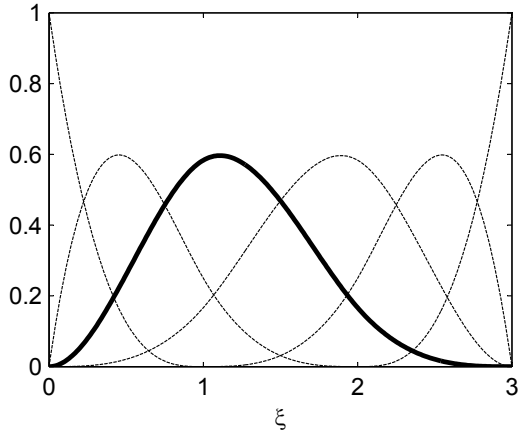


Figure 7: Basis function N_3 (thick solid line) to the local knot vector $\Xi = \{0, 0, 1, 2, 3\}$ and the additional basis functions (thin dotted lines) to the extended knot vector $\bar{\Xi} = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$.

The thin dotted lines are the additional basis functions when the extended knot vector $\bar{\Xi} = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$ is introduced. Conceptually, the extraction operators may now be computed similarly to NURBS to obtain the basis functions of the Bézier elements shown in figure 8. The extraction operators will therefore be equal to the operators in the case of NURBS,

$$\begin{bmatrix} N_1 \\ N_2 \\ \boldsymbol{N_3} \\ N_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ \mathbf{0} & \mathbf{0} & \mathbf{\frac{1}{2}} & \mathbf{\frac{7}{12}} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} \boldsymbol{B_1} \\ \boldsymbol{B_2} \\ \boldsymbol{B_3} \\ \boldsymbol{B_4} \end{bmatrix} \tag{42}$$

$$\begin{bmatrix} N_2 \\ \boldsymbol{N_3} \\ N_4 \\ N_5 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \mathbf{\frac{7}{12}} & \mathbf{\frac{2}{3}} & \mathbf{\frac{1}{3}} & \mathbf{\frac{1}{6}} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} \boldsymbol{B_4} \\ \boldsymbol{B_5} \\ \boldsymbol{B_6} \\ \boldsymbol{B_7} \end{bmatrix} \tag{43}$$

$$\begin{bmatrix} \boldsymbol{N_3} \\ N_4 \\ N_5 \\ N_6 \end{bmatrix} = \begin{bmatrix} \mathbf{\frac{1}{6}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \frac{7}{12} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{B_7} \\ \boldsymbol{B_8} \\ \boldsymbol{B_9} \\ \boldsymbol{B_{10}} \end{bmatrix} \tag{44}$$

Notice however that only the rows with bold typing in the extraction operators are necessary to map the Bernstein polynomials on Bézier elements in figure 8 to the global T-spline basis function N_3 in figure 7. Thus, an algorithm to find the Bézier extraction operators for T-splines does not compute the redundant rows.

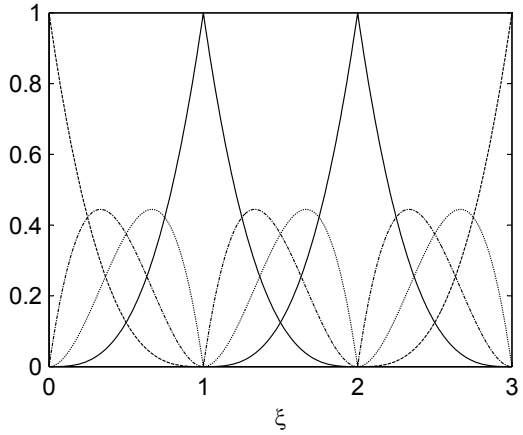


Figure 8: Basis functions of the Bézier elements after Bézier decomposition of $\bar{\Xi} = \{0,0,0,0,1,2,3,3,3,3\}$.

Implementation in a FE solver

To implement isogeometric analysis with Bézier extraction in a FE code the only necessary changes are confined to the shape function routine and the generation of the element extraction operators [7]. Figure 9 shows a flow chart for the shape function routine for NURBS using Bézier extraction. The routine is performed for each element.

Since the element extraction operators only need information given by the knot vectors, these can easily be pre-calculated and then called in before the shape function routine is performed. To calculate the Bézier weights, the NURBS weights are also needed. The Bézier basis functions and derivatives are calculated according to equations (20) and (21) in a separate routine, and are therefore also called into the shape function routine.

Isogeometric analysis based on Bézier extraction of T-splines is in this study performed by importing the extended T-mesh of a curved beam modelled in Rhinoceros with T-splines into the FE solver in MATLAB. The imported geometry is shown in figure 10. The input from Rhinoceros consists of 39 control points and corresponding weights, together with Bézier extraction operators for the 17 elements. A parsing script creates the IEN array for the extended T-mesh and modifies the data to be compatible with the programme based on Bézier extraction of NURBS. This illustrates that T-mesh analysis may be easily performed in a FE solver with a shape function routine adapted to NURBS based on Bézier extraction. The only input needed is the control points and the extraction operators.

Numerical studies

Circular beam subjected to end shear load

The problem consists of a cantilevered beam shaped as a quarter of a circle (see figure 11). The material is linear elastic and in a state of plane stress. The beam is analysed with the free end subjected to a prescribed displacement in the negative $x-$direction, and the resulting strain energy is calculated. The analytical solution for the strain energy of the system is given by Timoshenko and Goodier [8] , and the results from the isogeometric analysis is compared with the results obtained by Zienkiewicz and Taylor [9] with a traditional FEA. The error in the strain energy is defined as

$$\|e\|_E^2 = U - U^h \tag{45}$$

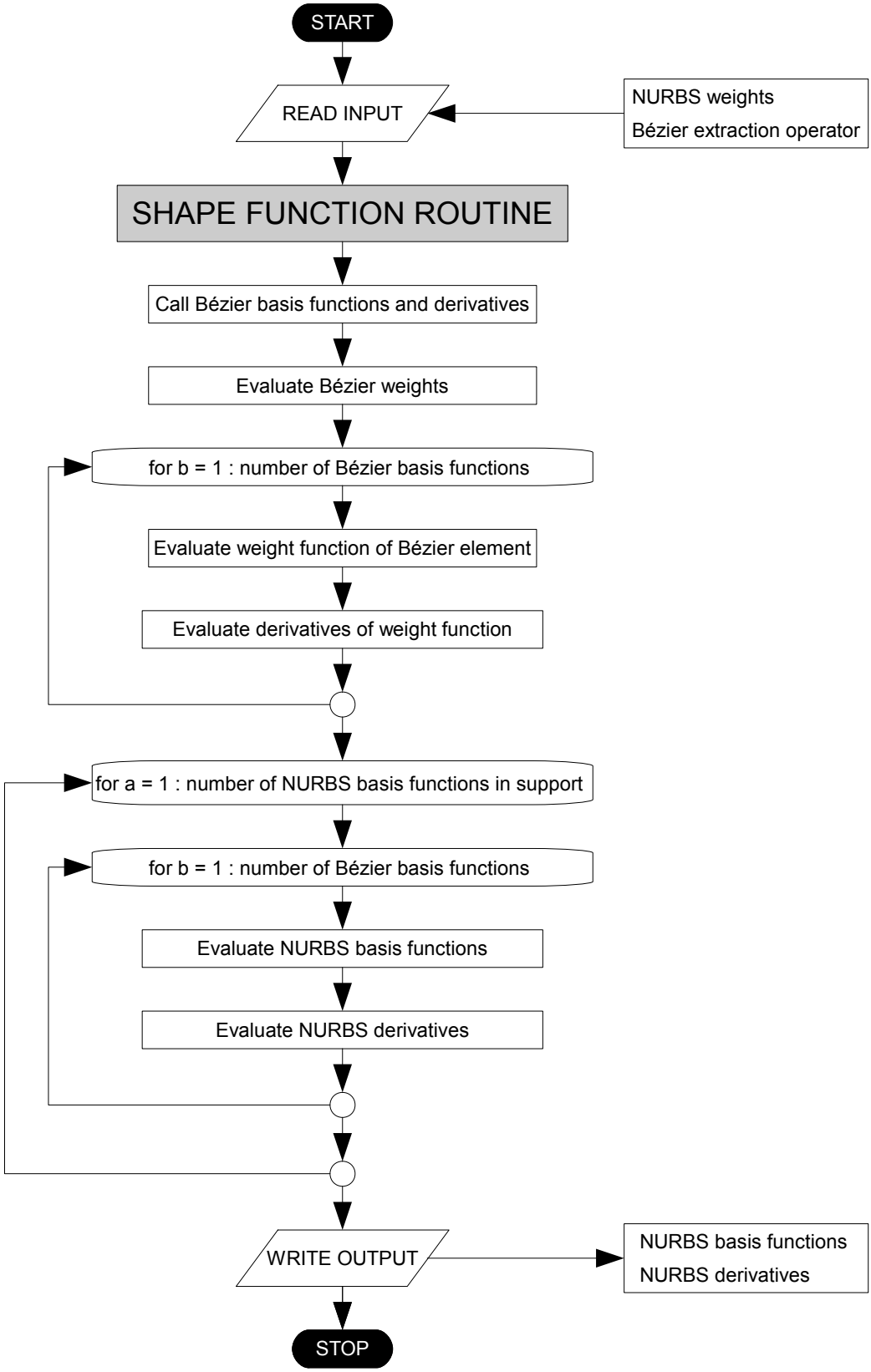


Figure 9: Flow chart of shape function routine for NURBS using Bernstein polynomials.

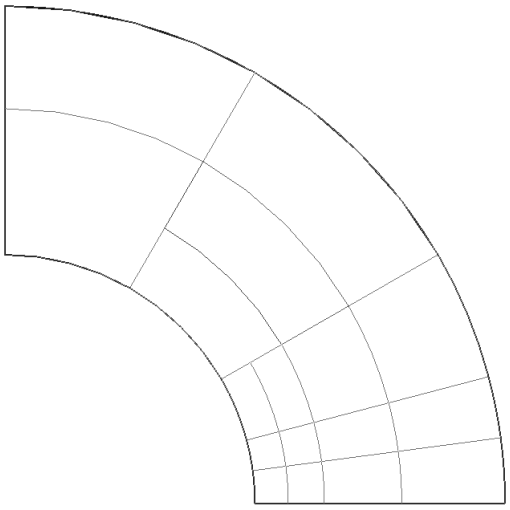


Figure 10: The extended T-mesh imported into the FE solver based on Bézier extraction of NURBS.

where U is the exact strain energy and U^h is the corresponding strain energy of the FE solution. The beam is analysed with $9-$ and $16-$noded Bézier quadrilaterals, with the coarsest meshes consisting of 3×6 and 2×4 elements, respectively, as shown in figure 12. The results in terms of energy are given in table 1 and 2, for NURBS and Lagrange elements, respectively. As seen in the convergence plots (see figure 13), the NURBS based FEA is performing better than the traditional FEA. The convergence rates are as expected the same as for the traditional FEA, but the accuracy is better.

Infinite plate with a circular hole under far-field uniaxial tension

The problem consist of a plate which is infinitely large in the $x-$ and $y-$direction, with a hole with radius equal 1 in the center of the plate (see figure 14). The plate is linear elastic, and in a state of plane strain. The elasticity modulus is 1000, and the Poisson ratio is 0.3. The plate is loaded with a uniform stress field in the $x-$direction, $\sigma_x = 1$ The analytical solution to stresses at an arbitrary point with coordinates (x, y) in the plate is given by [10]

$$\begin{aligned} \sigma_x &= 1 - \frac{a^2}{r^2}\left(\frac{3}{2}\cos 2\theta + \cos 4\theta\right) + \frac{3}{2}\frac{a^4}{r^4}\cos 4\theta \\ \sigma_y &= -\frac{a^2}{r^2}\left(\frac{1}{2}\cos 2\theta - \cos 4\theta\right) - \frac{3}{2}\frac{a^4}{r^4}\cos 4\theta \\ \tau_{xy} &= -\frac{a^2}{r^2}\left(\frac{1}{2}\sin 2\theta + \sin 4\theta\right) + \frac{3}{2}\frac{a^4}{r^4}\sin 4\theta \end{aligned} \tag{46}$$

The exact strain energy of the analysed part can be evaluated from the analytical solution to the stresses

$$U = \int_V \boldsymbol{\sigma}^T \boldsymbol{\varepsilon} dV = \int_0^{\pi/2} \int_1^4 \boldsymbol{\sigma}^T \mathbf{D}^{-1} \boldsymbol{\sigma} r dr d\theta = 0.01197664128784163 \tag{47}$$

where

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} \tag{48}$$

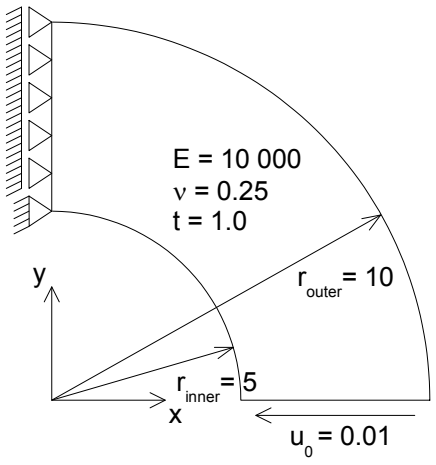
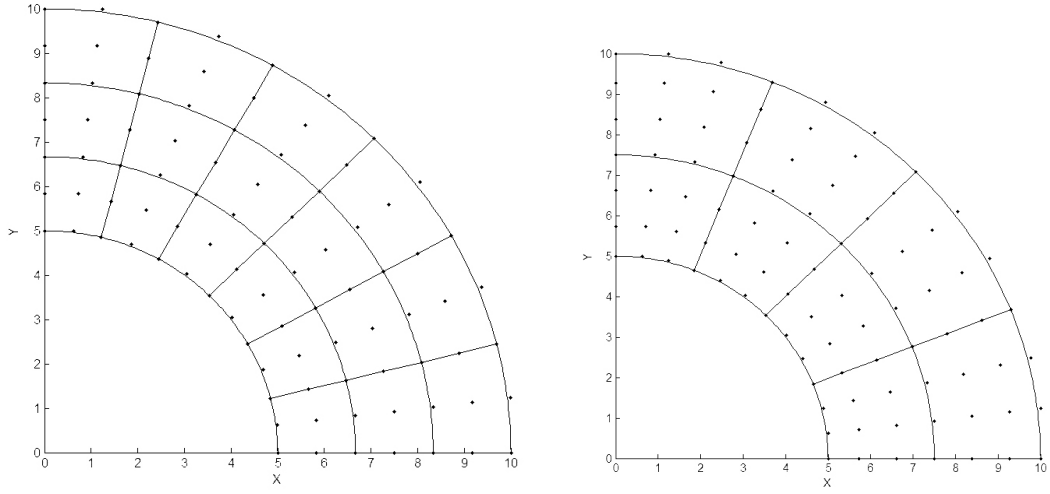


Figure 11: The geometry of the circular beam with end shear, with material properties, boundary conditions and prescribed displacements.



(a) 9-noded quadrilateral (b) 16-noded quadrilateral

Figure 12: Coarsest Bézier mesh with Bézier control points, for the circular beam.

NURBS-Q9			NURBS-Q16		
DOFs	Elmts	Strain energy	DOFS	Elmts	Strain energy
80	18	0.029708322024974	70	8	0.029653101738195
224	72	0.029653401864295	154	32	0.029649732629062
728	288	0.029649900409357	418	128	0.029649669346247
2600	1152	0.029649682879498	1330	512	0.029649668455942
9800	4608	0.029649669343369	4690	2048	0.029649668442595
Exact		0.029649668442380			0.029649668442380

Table 1: Strain energy for circular beam with NURBS elements.

	Lagrange Q4		Lagrange Q9		Lagrange Q16	
DOFs	Elmts	Strain energy	Elmts	Strain energy	Elmts	Strain energy
182	72	0.03042038175071	18	0.02970101373401	8	0.02965327376971
650	288	0.02984351371323	72	0.02965318188484	32	0.02964975296446
2450	1152	0.02969820784232	288	0.02964989418870	128	0.02964966996157
9506	4608	0.02966180825828	1152	0.02964968266120	512	0.02964966846707
37442	18432	0.02965270370808	4608	0.02964966933301	2048	0.02964966844276
Exact		0.029649668442380		0.029649668442380		0.029649668442380

Table 2: Strain energy for circular beam discretized with Lagrangian elements.

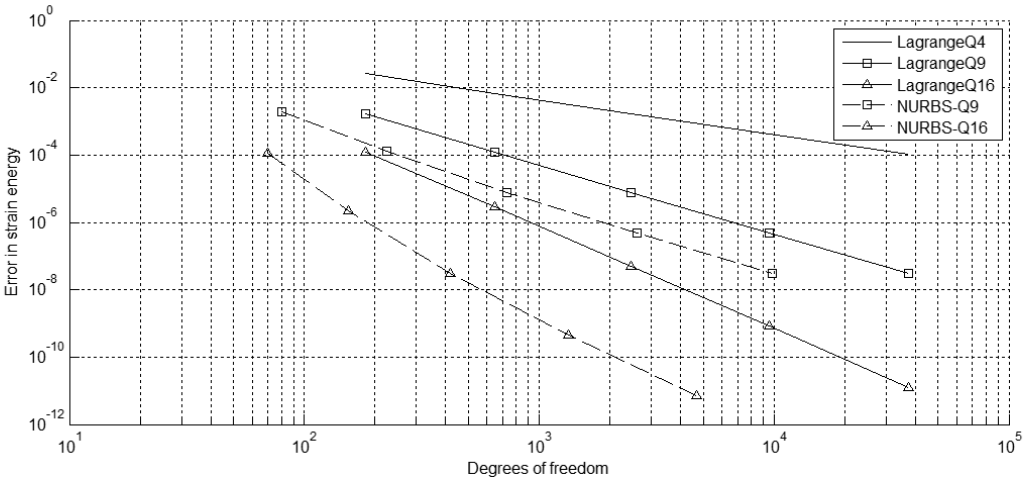
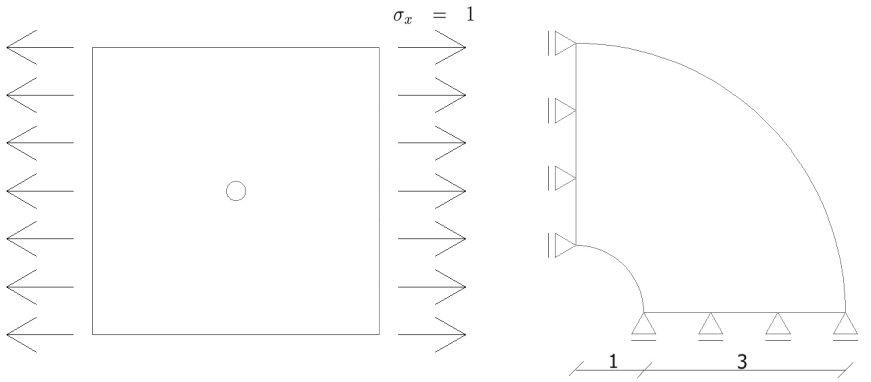


Figure 13: Error in strain energy vs. number of degrees of freedom for the circular beam.



(a) The problem

(b) Part of the problem that is analysed

Figure 14: Infinite plate with a circular hole under far-field uniaxial tension.

NURBS-Q9			NURBS-Q16		
DOFs	Elmts	Energy	DOFS	Elmts	Energy
144	40	0.011953123289377	154	32	0.011973812023053
576	220	0.011975309108001	418	128	0.011976608009310
2304	1012	0.011976577690435	1330	512	0.011976640728685
9216	4324	0.011976637976321	4690	2048	0.011976641280095
36864	17860	0.011976641099244	17554	8192	0.011976641287725
Exact		0.011976641287842			0.011976641287842

Table 3: Strain energy for infinite plate.

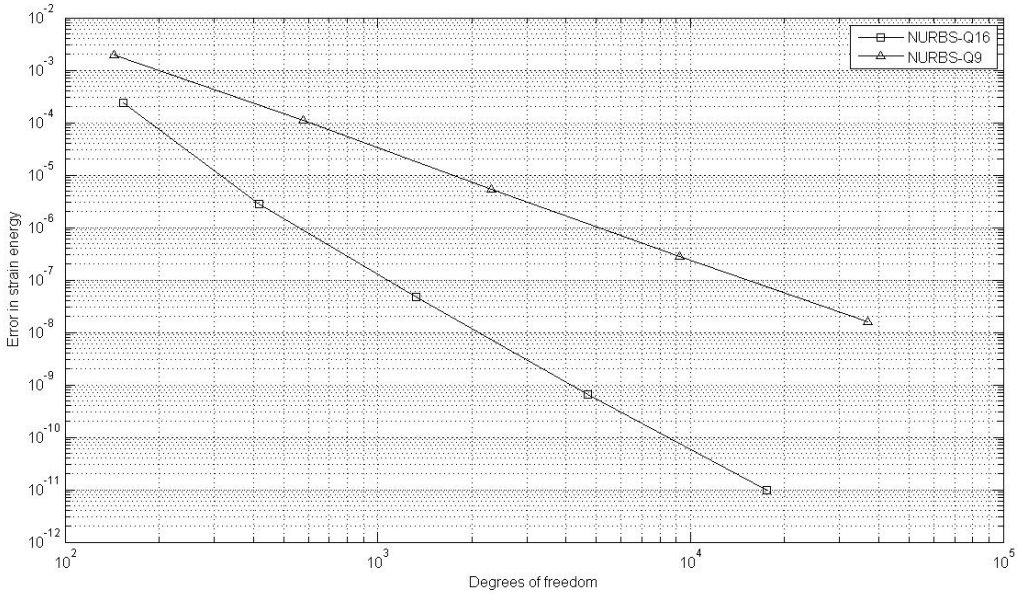


Figure 15: Error in strain energy vs. number of degrees of freedom for the infinite plate.

$$\mathbf{D} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix} \tag{49}$$

At the loaded edge of the plate the exact stresses is applied to the system as a traction field

$$\mathbf{t} = \hat{\mathbf{n}} \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \tag{50}$$

where $\hat{\mathbf{n}}$ is the unit outward normal. The plate is analysed with the $9-$ and the $16-$noded element, and the resulting strain energy is given in table 3. The convergence rates plotted in figure 15 are as expected for quadratic and cubic elements.

Conclusions

The Bézier extraction operator is significantly easing the implementation of isogeometric analysis in an existing FE code, since the only necessary changes can be done in the shape function routine. The rest of the code may be kept as it is. The eased implementation is at the cost of a

slight increase of computational effort in the computation of the stiffness matrix, compared to a FE code that is designed to do isogeometric analysis.

As shown in the example with the circular beam use of NURBS in analysis has an increased accuracy compared to a traditional FEA. The convergence rates are the same as expected for the order of elements, but for a given element mesh, isogeometric analysis will produce a smaller error compared to traditional FEA.

References

[1] Y.Bazilevs, V.Calo, J.Cottrell, J.Evans, T.Hughes, S.Lipton, M.Scott and T.Sederberg Isogeometric analysis using t-splines *Computer Methods in Applied Mechanics and Engineering*, **vol.199**(5-8), 229 – 263, 2010 Computational Geometry and Analysis.

[2] 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 *International Journal for Numerical Methods in Engineering*, 2010.

[3] M.Cox The numerical evaluation of B-splines *IMA Journal of Applied Mathematics*, **vol.10**(2), 134, 1972.

[4] C.de Boor On calculation with B-splines *J. Approx. Theory*, **vol.6**, 50–62, 1972.

[5] T.Hughes, J.Cottrell and Y.Bazilevs Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement *Computer methods in applied mechanics and engineering*, **vol.194**(39-41), 4135–4195, 2005.

[6] T. J.Hughes, J. A.Cottrell and Y.Bazilevs *Isogeometric Analysis Towards Unification of CAD and FEA* John Wiley & Sons, Ltd., West Sussex, 2009.

[7] M. A.Scott, M. J.Borden, C. V.Verhoosel, T. W.Sederberg and T. J. R.Hughes Isogeometric finite element data structures based on bézier extraction of t-splines *International Journal for Numerical Methods in Engineering*, 2011.

[8] S.Timoshenko and J.Goodier Theory of Elasticity, McGraw-Hill, New York, 1970 *J. Heydenreich, Rev. Roumaine Phys..*, **vol.1**, 1969–14.

[9] O. C.Zienkiewicz, R. L.Taylor and J. Z.Zhu *The Finite Element Method: Its Basis and Fundamentals* Elsevier Butterworth Heinemann, 2005.

[10] O. C.Zienkiewicz and J. Z.Zhu The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique *International Journal for Numerical Methods in Engineering*, **vol.33**(7), 1331–1364, 1992.

Appendix C

Presentation from MekIT11

This appendix contains the presentation of the report in Appendix B, given at the MekIT11 conference on May 23, 2011, with Thanh Ngan Nguyen.

Isogeometric finite element analysis based on Bézier extraction of NURBS and T-splines

Thanh Ngan Nguyen*†
Ole Jørgen Fredheim*†
Kjetil André Johannessen‡
Kjell Magne Mathisen†

† Department of Structural Engineering, NTNU
‡ Department of Mathematical Sciences, NTNU
Trondheim, Norway

May 23, 2011

Outline

- Motivation
- NURBS and T-splines
- Bernstein polynomials and Bézier curves
- Bézier extraction of NURBS and T-splines
- Implementation in a finite element solver
- Numerical examples
- Concluding remarks

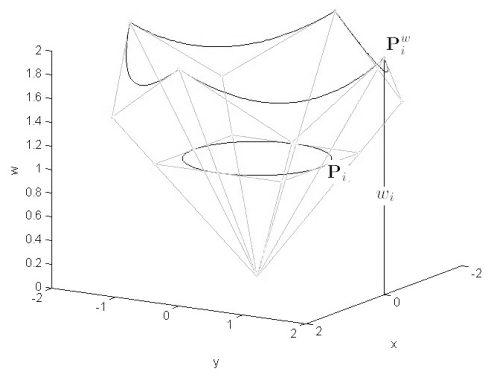
Motivation

- Isogeometric analysis: The basis for geometry is used for analysis
- Computer Aided Engineering (CAE) introduced before Computer Aided Design (CAD), CAD and CAE developed independently
- Isogeometric analysis: The shape functions span several elements which complicates implementation
- The Bézier extraction operator decomposes the NURBS or T-spline basis functions to be represented over C^0 continuous Bézier elements
- Bézier extraction confines the necessary changes in the finite element code to the shape function routine

NURBS

Non-Uniform Rational B-Splines

- Expansion from B-splines
- Projective transformation of a B-spline in $\mathbb{R}^{d+1}$
- $(\mathbf{P}_i)_j = (\mathbf{P}_i^w)_j / w_i,\ j = 1, \ldots, d$
- Weights $w_i = (\mathbf{P}_i^w)_{d+1}$



Rational basis functions defined as

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

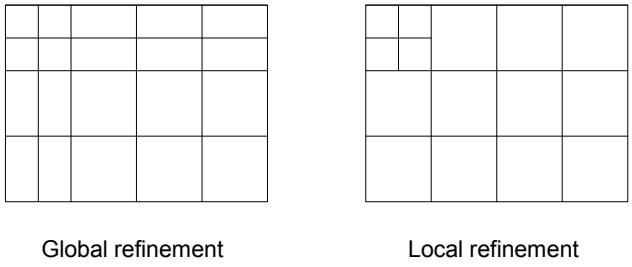
NURBS curve defined equivalently as B-spline curve

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} R_{i,p}(\xi) \mathbf{P}_i$$

T-spline fundamentals 1

T-spline fundamentals:

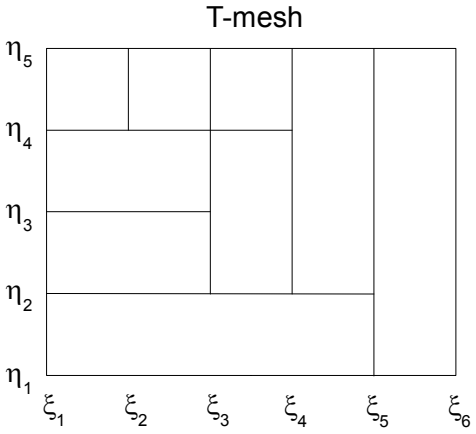
- No tensor product restriction as for NURBS
- Incomplete rows and columns of control points



Global refinement | Local refinement

Example: A simple T-mesh

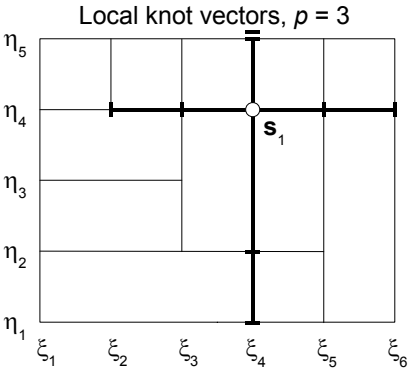
- Each knot line represents a knot value
- Incomplete knot lines terminates in *T-junctions*



T-spline fundamentals 2

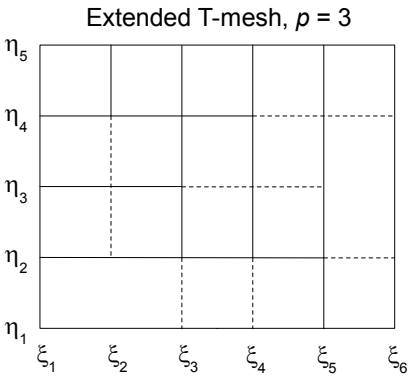
Local knot vectors:

- Local knot vectors define the T-spline basis function
- Found from the T-mesh
- The local knot vectors to $\mathbf{s}_1$ are $\Xi_1 = \{\xi_2, \xi_3, \xi_4, \xi_5, \xi_6\}$ and $\mathcal{H}_1 = \{\eta_1, \eta_2, \eta_4, \eta_5, \eta_5\}$



Extended T-mesh:

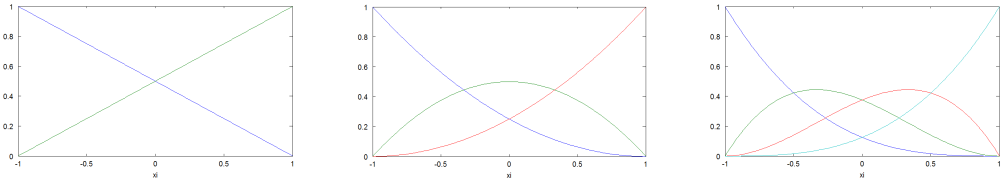
- *Lines of reduced continuity* define T-spline elements
- T-spline elements: regions over which T-spline basis functions are C^∞ continuous



Bernstein polynomials and Bézier curves

Bernstein polynomials

$$B_{a,p}(\xi) = (1-\xi)B_{a,p-1}(\xi) + \xi B_{a-1,p-1}(\xi) \quad \xi \in [0,1]$$



Identical to B-splines with multiplicity equal p at each knot

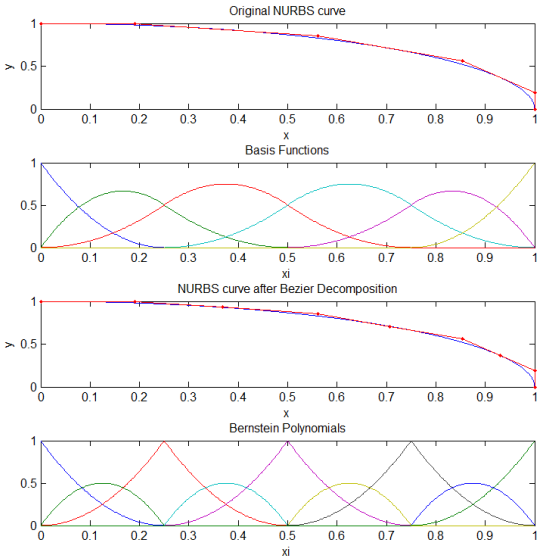
Bézier curves

A Bézier curve is a linear combination of Bernstein polynomials

$$C(\xi) = \sum_{a=1}^{p+1} \mathbf{P}_a B_{a,p}(\xi) = \mathbf{P}^T \mathbf{B}(\xi)$$

Bézier extraction

B-splines and NURBS can be written in terms of Bernstein polynomials and the Bézier extraction operator $\mathbf{C}$. $\mathbf{C}$ is generated by knot insertions until the multiplicity at each internal knot is equal to the polynomial order p.



B-splines

- $\mathbf{P}^b = \mathbf{C}^T \mathbf{P}$
- $\mathbf{N}(\xi) = \mathbf{C}\mathbf{B}(\xi)$

NURBS

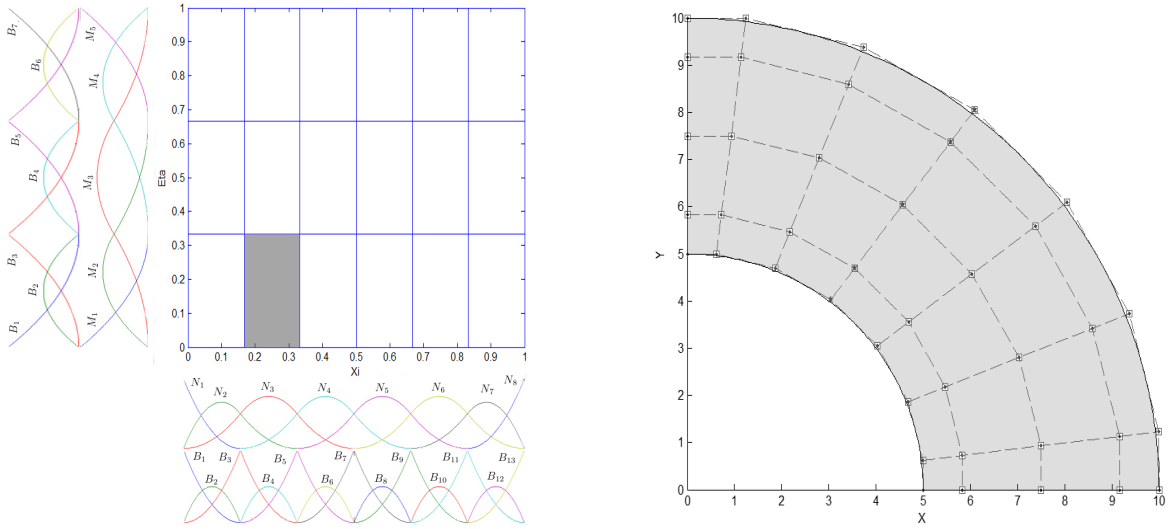
- $\mathbf{P}^b = (\mathbf{W}^b)^{-1}\mathbf{C}^T\mathbf{W}\mathbf{P}$
- $\mathbf{R}(\xi) = \mathbf{W}\mathbf{C}\frac{\mathbf{B}(\xi)}{W^b(\xi)}$

Example of Bézier decomposition 1

A circular beam

$$\Xi = \{0, 0, 0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}, 1, 1, 1\}$$

$$\mathcal{H} = \{0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1\}$$

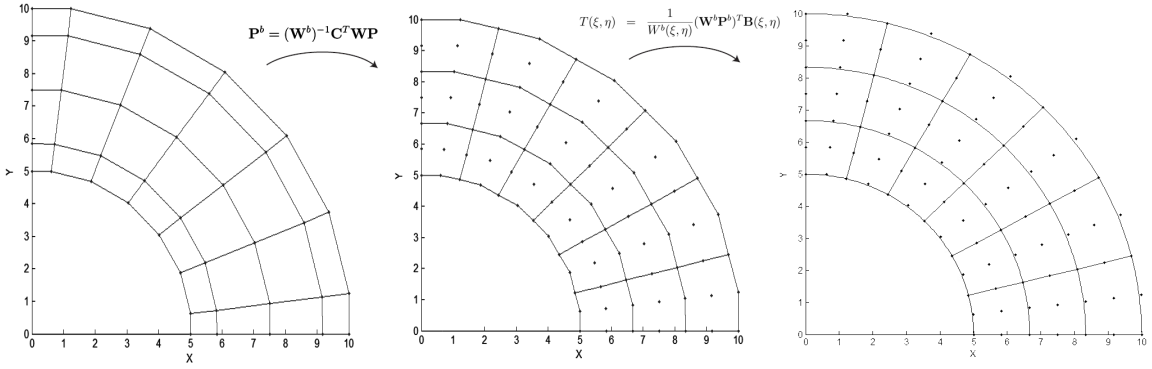


Example of Bézier decomposition 2

Basis functions can be written in terms of Bézier extraction operator and Bernstein polynomials, and for the shaded element we get

$$\begin{Bmatrix} N_1^2 \\ N_2^2 \\ N_3^2 \end{Bmatrix} = \begin{Bmatrix} N_2 \\ N_3 \\ N_4 \end{Bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_3 \\ B_4 \\ B_5 \end{Bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0.5 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1^2 \\ B_2^2 \\ B_3^2 \end{Bmatrix}$$

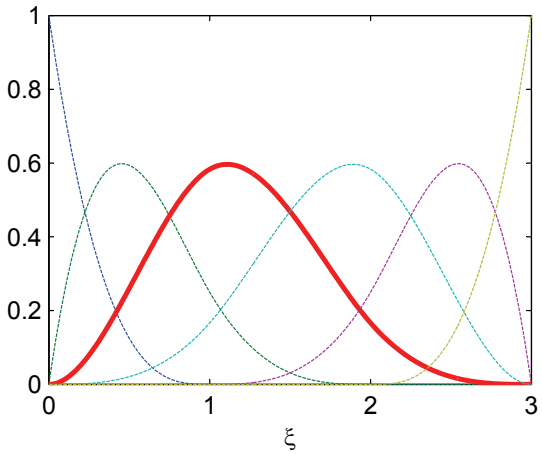
$$\begin{Bmatrix} M_1^1 \\ M_2^1 \\ M_3^1 \end{Bmatrix} = \begin{Bmatrix} M_1 \\ M_2 \\ M_3 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1^1 \\ B_2^1 \\ B_3^1 \end{Bmatrix}$$



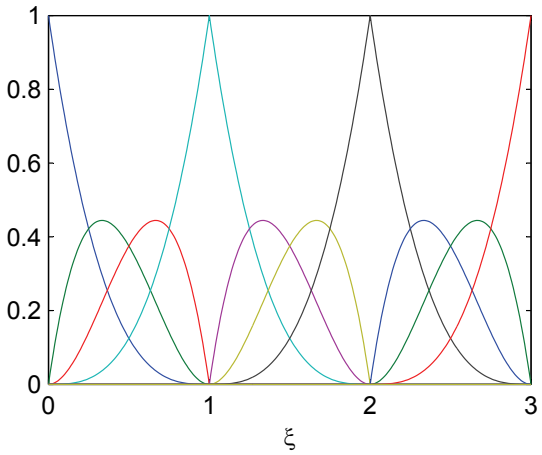
Bézier extraction of T-splines 1

Bézier extraction operator for T-splines:

- Same idea as Bézier extraction of NURBS
- Map T-spline basis function to Bernstein polynomials



Basis function N_3 (thick red line),
$\Xi = \{0, 0, 1, 2, 3\}$



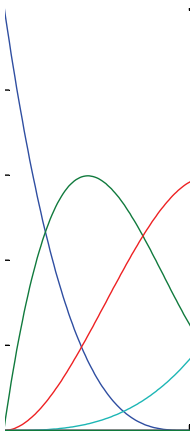
Bernstein polynomials

Bézier extraction of T-splines 2

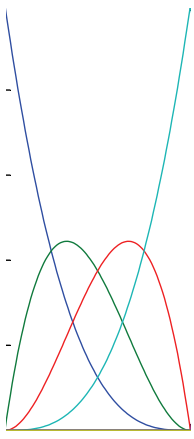
Differences compared to NURBS extraction operator:

- Local knot vectors vs. global knot vector $\Rightarrow$ introduce the extended knot vector $\bar{\Xi} = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$
- Local tensor product domains vs. global tensor product domain $\Rightarrow$ one row to each basis function in support
- The element extraction operator for the knot span [0,1) becomes

$$\begin{bmatrix} N_1 \\ N_2 \\ \mathbf{N_3} \\ N_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ \mathbf{0} & \mathbf{0} & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} \mathbf{B_1} \\ \mathbf{B_2} \\ \mathbf{B_3} \\ \mathbf{B_4} \end{bmatrix}$$



N_1, N_2, N_3, N_4

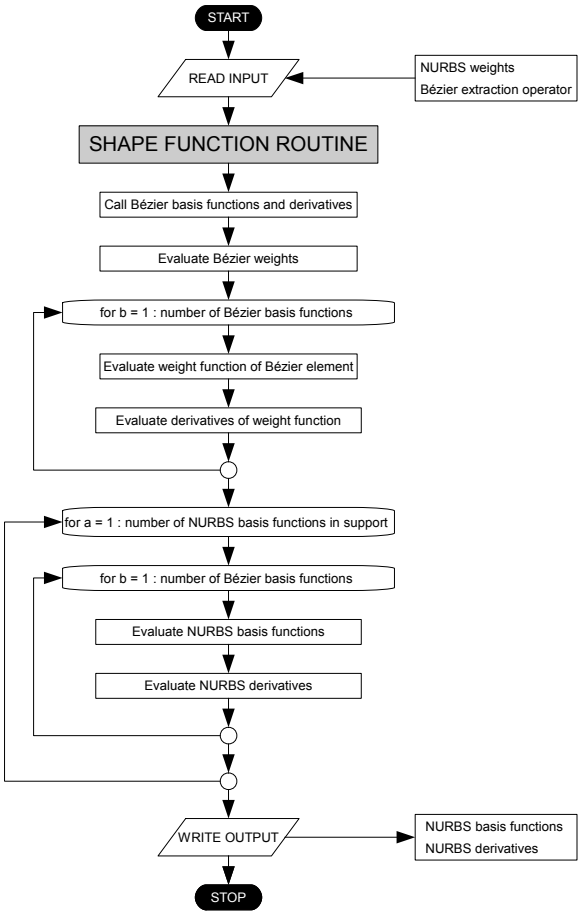


B_1, B_2, B_3, B_4

Implementation in a finite element solver 1

Isogeometric analysis based on Bézier extraction of NURBS:

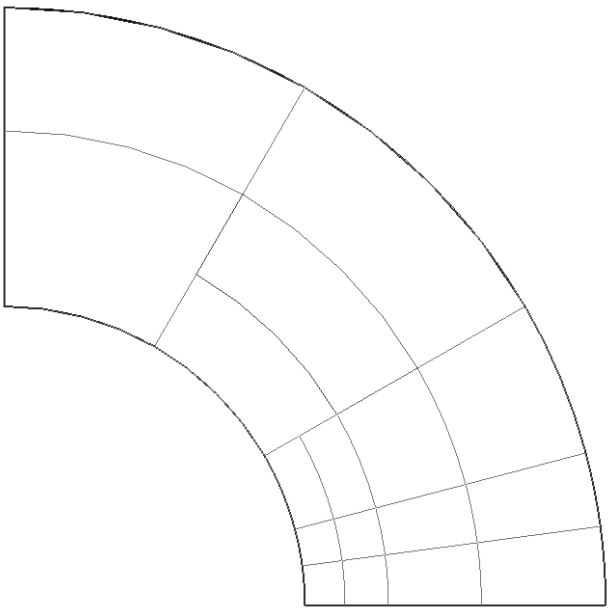
- Changes confined to shape function routine
- Extraction operators and Bézier basis functions are pre-calculated
- Output: NURBS basis functions and derivatives



Implementation in a finite element solver 2

Isogeometric analysis based on Bézier extraction of T-splines:

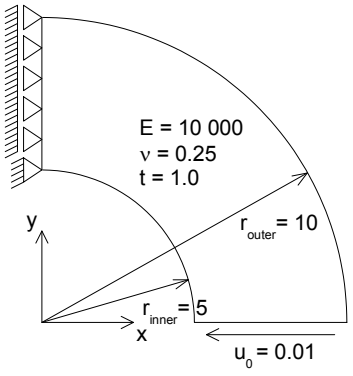
- Performed by importing a T-mesh from Rhinoceros into the FE solver
- Input: Control points and extraction operators
- A parsing script creates the IEN array for the extended T-mesh



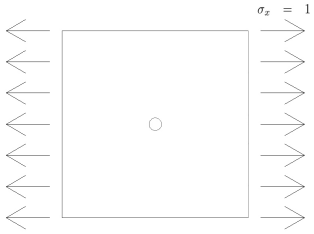
Numerical studies

Numerical studies has been performed with analysis of two linear elasticity problems

- Circular beam with end shear

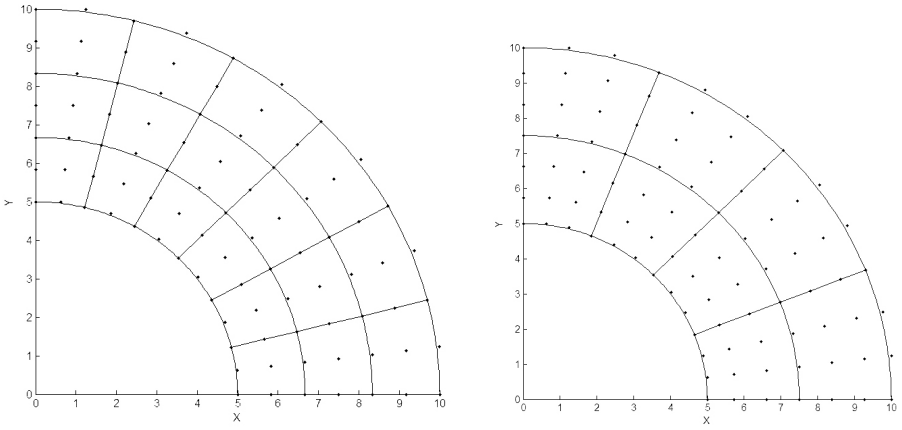


- Infinite plate with a circular hole under far-field uniaxial tension



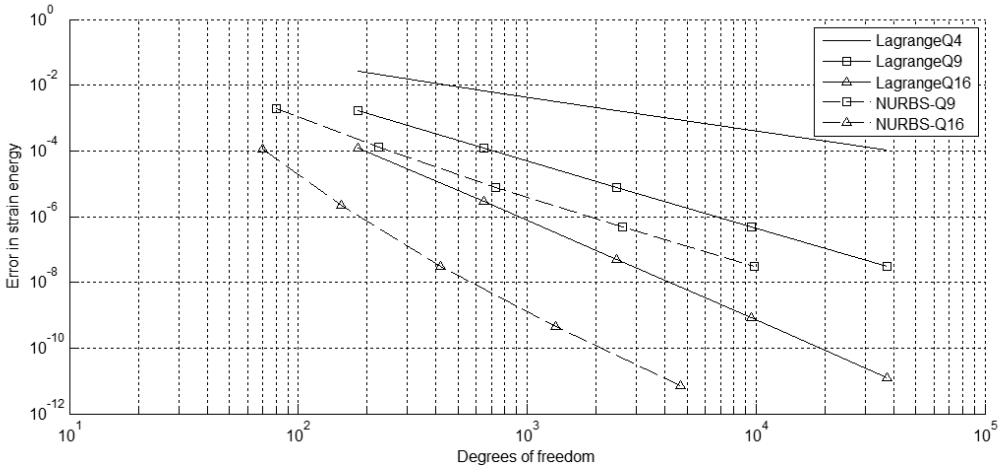
Circular beam with end shear 1

- The beam is in a state of plane stress
- Analysed with quadratic and cubic NURBS elements with coarsest meshes 6x3 and 4x2 elements, respectively



- Compared to solution obtained with Lagrange elements

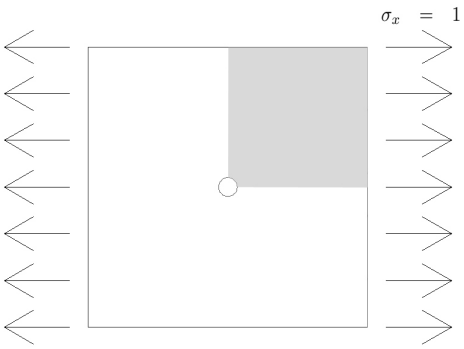
Circular beam with end shear 2

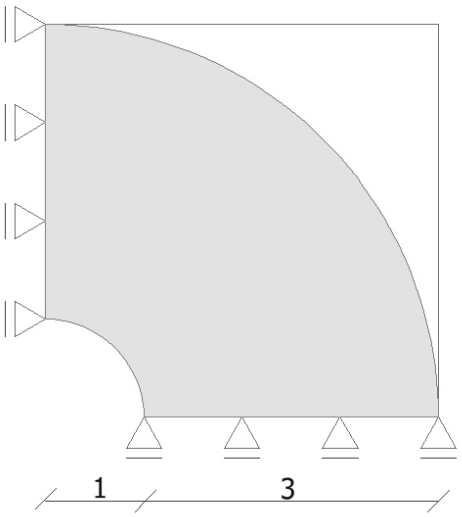


NURBS perform better than Lagrange

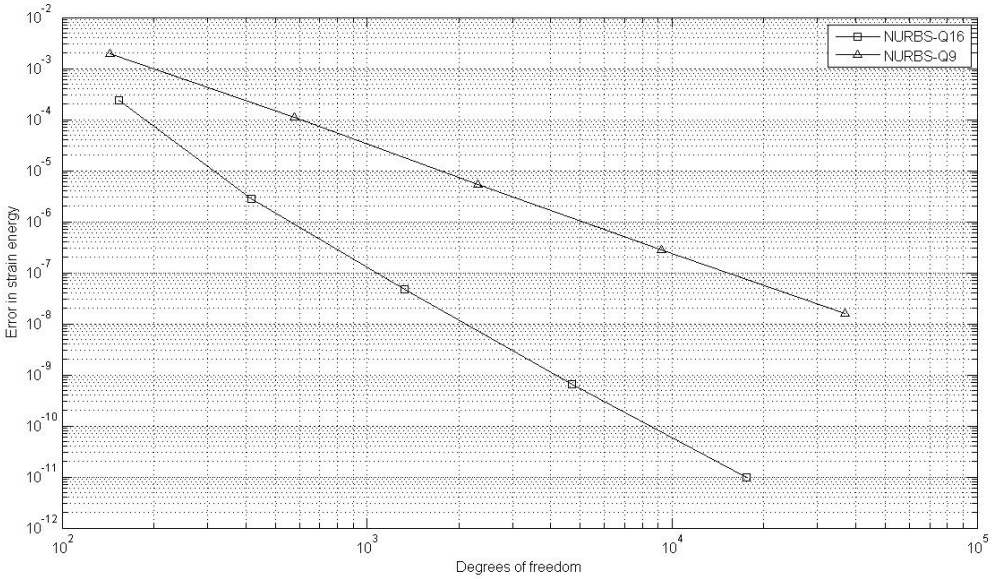
Infinite plate with a circular hole under far-field uniaxial tension 1

- A plain strain problem
- Only one quarter of plate needs to be analysed
- Analysed as a quarter of circle to avoid singularities in the corner, in contrast to when analysed as a quarter plate





Infinite plate with a circular hole under far-field uniaxial tension 1



Concluding remarks

- Bézier extraction is significantly easing implementation of isogeometric analysis in an existing FE framework
- NURBS avoid geometric error in discretization of the problem
- NURBS elements has higher accuracy than Lagrange elements
- NURBS elements has the same convergency rates as Lagrange elements

Bibliography

[1] Y. Bazilevs, V.M. Calo, J.A. Cottrell, J.A. Evans, T.J.R. Hughes, S. Lipton, M.A. Scott, and T.W. Sederberg. Isogeometric analysis using t-splines. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):229 – 263, 2010. Computational Geometry and Analysis.

[2] DJ Benson, Y. Bazilevs, MC Hsu, and TJR Hughes. Isogeometric shell analysis: the reissner-mindlin shell. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):276–289, 2010.

[3] DJ Benson, Y. Bazilevs, M.C. Hsu, and TJR Hughes. A large deformation, rotation-free, isogeometric shell. *Computer Methods in Applied Mechanics and Engineering*, 2010.

[4] W. Boehm. Inserting new knots into b-spline curves. *Computer-Aided Design*, 12(4):199–201, 1980.

[5] Michael J. Borden, Michael A. Scott, John A. Evans, and Thomas J. R. Hughes. Isogeometric finite element data structures based on bézier extraction of nurbs. *International Journal for Numerical Methods in Engineering*, 2010.

[6] A. Buffa, D. Cho, and G. Sangalli. Linear independence of the t-spline blending functions associated with some particular t-meshes. *Computer Methods in Applied Mechanics and Engineering*, 199(23-24):1437–1445, 2010.

[7] R.W. Clough. The finite element method in plane stress analysis. 1960.

[8] R.W. Clough. Original formulation of the finite element method. *Finite elements in analysis and design*, 7(2):89–101, 1990.

[9] Robert D Cook, David S Malkus, Michael E Plesha, and Robert J Witt. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, Inc., 2002.

[10] M.G. Cox. The numerical evaluation of B-splines. *IMA Journal of Applied Mathematics*, 10(2):134, 1972.

[11] C. de Boor. On calculation with B-splines. *J. Approx. Theory*, 6:50–62, 1972.

[12] M.R. Dörfel, B. Jüttler, and B. Simeon. Adaptive isogeometric analysis by local h-refinement with T-splines. *Computer methods in applied mechanics and engineering*, 199(5-8):264–275, 2010.

[13] Thomas J.R. Hughes, J. Austin Cottrell, and Yuri Bazilevs. *Isogeometric Analysis Towards Unification of CAD and FEA*. John Wiley & Sons, Ltd., West Sussex, 2009.

[14] TJR Hughes, JA Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39-41):4135–4195, 2005.

[15] B.M. Irons. Engineering applications of numerical integration in stiffness methods. *AIAA Journal*, 4:2035–2037, 1966.

[16] J. Kiendl, K.U. Bletzinger, J. Linhard, and R. Wuchner. Isogeometric shell analysis with kirchhoff-love elements. *Computer Methods in Applied Mechanics and Engineering*, 198(49-52):3902–3914, 2009.

[17] T. Kvamsdal, K.A. Johannesen, and T. Dokken. Adaptive refinement in isogeometric analysis using LR B-splines. In *MekIT'11 Sixth NationalConference on Compuational Mechanics*, pages 157–170, 2011.

[18] X. Li, J. Zheng, TW Sederberg, TJR Hughes, and MA Scott. On the linear independence of t-splines. *Computer Aided Geometric Design*, pages 10–40, 2010.

[19] K.M. Mathisen, K.M. Okstad, T. Kvamsdal, and S Raknes. Isogeometric analysis of finite deformation elastic and elastoplastic solid problems. In *MekIT'11 Sixth NationalConference on Compuational Mechanics*, pages 157–170, 2011.

[20] R.F. Riesenfeld. Applications of b-spline approximation to geometric problems of computer-aided design. 1973.

[21] M.A. Scott, X. Li, T.W. Sederberg, and T.J.R. Hughes. Local refinement of analysis-suitable t-splines. *Computer Methods in Applied Mechanics and Engineering*, 2011.

[22] Michael A. Scott, Michael J. Borden, Clemens V. Verhoosel, Thomas W. Sederberg, and Thomas J. R. Hughes. Isogeometric finite element data structures based on bézier extraction of t-splines. *International Journal for Numerical Methods in Engineering*, 2011.

[23] T.W. Sederberg, D.L. Cardon, G.T. Finnigan, N.S. North, J. Zheng, and T. Lyche. T-spline simplification and local refinement. *ACM Transactions on Graphics (TOG)*, 23(3):276–283, 2004.

[24] T.W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCs. In *ACM SIGGRAPH 2003 Papers*, pages 477–484. ACM, 2003.

[25] G. Skeie, S. Støle-Hentschel, V. Tharigopula, T. Driveklepp, T. Rusten, and Damhaug. A.C. Isogeometric analysis in linearized fluid-structure interaction in sloshing. In *MekIT'11 Sixth NationalConference on Compuational Mechanics*, pages 303–312, 2011.

[26] RL Taylor. Feap-a finite element analysis program: version 8.3 theory manual. *University of California at Berkeley*, 2011.

[27] SP Timoshenko and JN Goodier. Theory of Elasticity, McGraw-Hill, New York, 1970. *J. Heydenreich, Rev. Roumaine Phys..*, 1:1969–14.

[28] K.J. Versprille. Computer-aided design applications of the rational b-spline approximation form. 1975.

[29] O C Zienkiewicz, R L Taylor, and J Z Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier Butterworth Heinemann, 2005.

[30] O. C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, 33(7):1331–1364, 1992.