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:
Computational Mechanics	June 14, 2011	277 (112 + 165)

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:

Thanh Ngan Nguyen

SUMMARY:
Data transmission between finite element analysis (FEA) and computer-aided design (CAD) is a huge bottle-neck today. Therefore, isogeometric analysis has been introduced with aim to merge these fields. While FEA utilizes Lagrange polynomials to approximate both the geometry and the solution field, isogeometric analysis employs non-uniform rational B-splines (NURBS) from CAD technology to this objective. Isogeometric analysis will therefore have the advantage in no geometric error in the sense that the model is exact.

T-splines are a recently introduced generalization of NURBS which allow local refinement, handling complex geometry in a subtle way with fewer degrees of freedom. Increasing the order of the elements in isogeometric analysis is easy and gives higher continuous basis functions than FEA, while also maintaining few degrees of freedom.

In conventional isogeometric analysis the basis functions are not confined to one single element, but span a global domain, complicating implementation. The Bézier extraction operator decomposes a set of NURBS or T-spline basis functions to linear combinations of Bernstein polynomials. These polynomials bear a close resemblance to the Lagrange polynomials as they allow for generation of C^0 continuous Bézier elements. A local data structure for isogeometric analysis close to traditional FEA is provided.

Codes are developed to illustrate conventional isogeometric data structures as well as structures based on Bézier extraction of NURBS. Modifications are made to the latter to be able to run analysis of T-splines modelled in the CAD system Rhino, and numerical studies are performed. Generally, NURBS elements display the same convergence rate as Lagrange elements of equal order, but higher accuracy. The reasons are a smooth solution field and exact geometrical representation.

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

Thanh Ngan Nguyen

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

Isogeometrisk elementanalyse basert på Bézier ekstraksjon av NURBS og 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 can be implemented into a standard finite element code for 2D elasticity 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 opposed to conventional isogeometric analysis, and also how this tool enables analysis based on Bézier extraction of 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

Data transmission between finite element analysis (FEA) and computer-aided design (CAD) is a huge bottle-neck today. Therefore, isogeometric analysis has been introduced with aim to merge these fields. While FEA utilizes Lagrange polynomials to approximate both the geometry and the solution field, isogeometric analysis employs non-uniform rational B-splines (NURBS) from CAD technology to this objective. Isogeometric analysis will therefore have the advantage in no geometric error in the sense that the model is exact.

T-splines are a recently introduced generalization of NURBS which allow local refinement, handling complex geometry in a subtle way with fewer degrees of freedom. Increasing the order of the elements in isogeometric analysis is easy and gives higher continuous basis functions than FEA, while also maintaining few degrees of freedom.

In conventional isogeometric analysis the basis functions are not confined to one single element, but span a global domain, complicating implementation. The Bézier extraction operator decomposes a set of NURBS or T-spline basis functions to linear combinations of Bernstein polynomials. These polynomials bear a close resemblance to the Lagrange polynomials as they allow for generation of C^0 continuous Bézier elements. A local data structure for isogeometric analysis close to traditional FEA is provided.

Codes are developed to illustrate conventional isogeometric data structures as well as structures based on Bézier extraction of NURBS. Modifications are made to the latter to be able to run analysis of T-splines modelled in the CAD system Rhino, and numerical studies are performed. Generally, NURBS elements display the same convergence rate as Lagrange elements of equal order, but higher accuracy. The reasons are a smooth solution field and exact geometrical representation.

Preface

This master thesis in Computational Mechanics is prepared in the 10th semester at Department of Structural Engineering, Norwegian University of Science and Technology (NTNU), spring 2011. The master thesis is the result of the work completed over a 20 week period from January to June.

The thesis is written as an introduction to isogeometric analysis as opposed to finite element analysis. Many papers on this subject have been published in the last few years, emphasizing on algorithms and proving the advantage of isogeometric analysis in a wide spectrum of applications. This thesis rather focuses on presenting the theory in a simple way with illustrative examples and necessary formulas, and comparing this to the finite element method.

Readers with background from finite element analysis applied to structural mechanics and who wants an introduction to isogeometric analysis may therefore find this thesis interesting.

The work is a continuation of the work performed in the specialization project at Department of Structural Engineering, autumn 2010. The project consisted of literature study and implementation of the computational procedures in both finite element analysis and isogeometric analysis. This master thesis looks further into the data structures for isogeometric analysis based on Bézier extraction of NURBS. Literature study on T-splines and applications of T-meshes have also been explored.

I would especially thank my advisors Kjell Magne Mathisen and Kjetil André Johannessen for preparing the assignment and for valuable guidance and good inspiration throughout the work. Also, thanks to Ole Jørgen Fredheim for lots of interesting discussions in the programming work.

Thanh Ngan Nguyen

Trondheim, June 14, 2011

Contents

List of Figures

List of Tables

Notation

Symbols frequently used in the thesis are listed. Less frequently used symbols are defined where they are used. Variables used in the isogeometric analysis MATLAB code are described in Appendix B.2.

Mathematical symbols

^{T}	Matrix transpose
$^{-1}$	Matrix inverse
$\|\ \|$	Length of vector
$\|\|\ \|\|$	Norm
$'$	Differentiation
,	Partial differentiation

Latin symbols

$\mathbf{B}$	Strain-displacement matrix, or vector of Bernstein polynomials
B_i	i^{th} Bernstein polynomial
$B_{i,p}$	i^{th} Bernstein polynomial, degree p
$\mathbf{C}$	B-spline or NURBS curve, or Bézier extraction operator
$\mathbf{D}$	Global displacement vector
$\mathbf{d}$	Element displacement vector
d	Number of physical dimensions
d_p	Number of parametric dimensions
$\mathbf{E}$	Constitutive matrix of elastic stiffnesses
E	Modulus of elasticity
e	Element number
$\mathbf{J}$	Jacobian matrix
J	Jacobian, $J = \det(\mathbf{J})$
$\mathbf{K}$	Global stiffness matrix
$\mathbf{k}$	Element stiffness matrix
m	Number of basis functions/control points in η direction, or number of new basis function/control points, or number of new knots, or number of elements in η dir.
$M_{j,q}$	j^{th} B-spline basis function in η direction, degree $q = p$
$\mathbf{N}$	Vector of shape functions, or vector of B-spline basis functions

n	Number of basis functions/control points in ξ direction, or number of old basis functions/control points, or number of elements in ξ direction
N_i	Shape function for node i, or B-spline basis function, or T-spline basis function
$N_{i,p}$	i^{th} B-spline basis function in ξ direction, degree p
$\mathbf{P}$	Control points
p	Polynomial order
$\mathbf{P}^b$	Bézier control points
$\mathbf{q}$	Distributed load vector
q	Value of distributed load
$\mathbf{R}$	Global load vector, or vector of NURBS basis functions
$\mathbf{r}_e$	Consistent nodal loads
R_i^p	i^{th} NURBS basis function, degree p
$\mathbf{S}$	B-spline or NURBS surface
$\mathbf{s}_i$	Anchor
U	Strain energy
$\mathbf{u}$	Displacement in x and y direction
u	Displacement in x direction
v	Displacement in y direction
$\mathbf{W}$	Diagonal matrix of NURBS weights
W	Weighting function, or weight of Gauss point
$\mathbf{w}$	Vector of NURBS weights
$\mathbf{W}^b$	Diagonal matrix of Bézier weights
W^b	Bézier weighting function
$\mathbf{w}^b$	Vector of Bézier weights
w_i	NURBS weight i
x, y	Cartesian coordinates

Greek symbols

α	Knot insertion variable
γ_{xy}	Shear strain
$\boldsymbol{\varepsilon}$	Vector of strains
ε_x	Normal strain in x direction
ε_y	Normal strain in y direction
$\mathcal{H}$	Knot vector in η direction
η_j	j^{th} knot
ν	Poisson's ratio
Ξ	Knot vector in ξ direction
ξ_i	i^{th} knot
ξ, η	Parametric coordinates, or reference element coordinates
$\hat{\xi}, \hat{\eta}$	Reference element coordinates
$\boldsymbol{\sigma}$	Vector of stresses, or matrix of stresses
σ_e	von Mises stress
σ_x	Normal stress in x direction

σ_y	Normal stress in y direction
τ_{xy}	Shear stress
$\mathbf{\Phi}$	Vector of tractions

Abbreviations

BC	Boundary condition(s)
CAD	Computer-aided design
CAE	Computer-aided engineering
DOF	Degree(s) of freedom
FE	Finite element
FEA	Finite element analysis
FEM	Finite element method
IEN	Internal entry number ("element nodes")
NURBS	Non-uniform rational B-splines

Chapter 1

Introduction

The purpose of this master thesis is to compare the finite element method (FEM) to a new method, so-called isogeometric analysis. This introduction presents some background information as a motivation for the work, before the scope and outline of the thesis are given.

1.1 Motivation

FEM, or finite element analysis (FEA), was developed in the 1950s to 1960s and is today the prevailing method for numerical solution of differential equations. Differential equations can describe a physical problem which spans a certain region. This region is according to FEM parted into finite elements, for which the solution can be approximated. The physical problem is typically modelled in, or imported as a complete model into, a FEA software, and thereafter an analysis which solves the physical problem is performed as specified by the user. The process is often referred to as computer-aided engineering (CAE). For solid and structural mechanics, solving the physical problem generally involves finding the displacements and reaction forces, and also stresses and strains within the material.

The model used for analysis is often a designer's perception of the physical problem. Whether this problem is a structural detail of a building, a dam construction or technical equipment, computer-aided design (CAD) is a commonly spent tool for modelling. Most CAD systems are based on spline basis functions, and these are often non-uniform rational B-splines (NURBS). When this model is transferred to a FEM formulation, the geometry must in most cases be approximated by piecewise lower order Lagrange polynomials. The process is both time-consuming and unfortunate in the sense that the exact geometrical representation is lost. Therefore Hughes et al. [11] addressed this problem in its first collected form in 2005, to unification of CAD and FEA. A comprehensive work on this subject was published in 2009 by Cottrell et al. [6]. They suggest an analytical framework which employs the same isoparametric concept as FEA utilizes. The same set of basis functions used to model the geometry is also used for the solution space. However, while FEA makes use of Lagrange polynomials for this purpose, isogeometric analysis employs NURBS to this objective. By doing so, the exact geometry is taken into account for the numerical analysis, hence the name isogeometric analysis.

NURBS, being the basic foundation for isogeometric analysis, were developed in the 1970s and is the current industry standard for computational geometry. In 2003, Sederberg et al. [24, 23] introduced T-splines as a generalization of NURBS technology. Although NURBS is suggested as a direct step from CAD to FEA, the process is not that streamlined. NURBS models are often made of several patches and contain gaps which are invisible in modelling perspective, but inhibit an analysis to be performed. T-splines offer a solution to this by their local refinement

property, which allow for a single watertight model to be created. This simplification of CAD models was the motivation of Sederberg et al. [24, 23] for T-spline development. However, T-splines were shortly after also investigated as a basis for isogeometric analysis in Dörfel et al. [9], Bazilevs et al. [1] with promising results.

To ease the integration of NURBS and T-splines in an existing finite element (FE) context, Borden et al. [3], Scott et al. [21] developed FE data structures based on Bézier extraction of NURBS and T-splines. The Bézier extraction operator decomposes the NURBS or T-spline based elements to C^0 continuous Bézier elements which bear a close resemblance to the Lagrange elements. The global smoothness of NURBS and T-splines is localized to an element level similar to FEA, making isogeometric analysis compatible with existing FE codes while still utilizing the excellent properties of the spline basis functions as a basis for modelling and analysis. Isogeometric data structures based on Bézier extraction of T-splines are therefore one of the most promising steps towards integration of CAD and FEA.

In Norway, research in isogeometric analysis is performed under the Integrated Computer Aided Design and Analysis (ICADA) project. The project is a collaboration between Department of Applied Mathematics, SINTEF ICT; Department of Mathematical Sciences and Department of Structural Engineering, Norwegian University of Science and Technology (NTNU), together with industrial partners. The main research activities abroad come from experts in the field of computational geometry and mechanics at several universities in the United States of America: The University of Texas at Austin (isogeometric analysis' origin), Brigham Young University, University of California, Berkeley and University of California, San Diego among others.

1.2 Outline

The project work leading up to this master thesis compared FEA and isogeometric analysis by developing traditional FE solver and B-spline based FE solver in MATLAB [17]. The comparison work is continued in this master thesis by implementing isogeometric analysis based on Bézier extraction of NURBS. Also, isogeometric analysis based on Bézier extraction of T-splines have been explored using the program based on Bézier extraction of NURBS.

The work has been carried out with emphasis on the theoretical formulation as the foundation for implementation. Literature study has been performed on the subject, reflecting the structure of this thesis.

In Chapter 2, the mathematical basis of FEM is briefly presented. Thereafter the description of the computational formulations is given. The chapter forms a foundation for comparing isogeometric analysis to FEA.

The purpose of Chapters 3 and 4 is to review the theory of isogeometric analysis while continuously having in mind the differences compared to FEA.

Chapter 3 starts with a thorough account of B-splines and NURBS. The review of T-splines is more a review of recent work of importance rather than a theoretical presentation, since applications and not data structures were explored in this field.

The concept of Bézier extraction is reviewed in Chapter 4. First the construction of Bézier elements and the Bézier extraction operator for NURBS are described. Then the Bézier extraction operator for T-splines as opposed to the extraction operator for NURBS is discussed.

Chapter 5 discusses the computational procedures for isogeometric analysis, first the conventional structures, thereafter the formulations based on Bézier extraction.

Numerical studies to verify isogeometric analysis are presented in Chapter 6. Investigations of T-meshes from the CAD system Rhino with T-splines plug-in are also described.

Finally, Chapter 7 presents concluding remarks, and Chapter 8 discusses further work.

The thesis is restricted to linear elastic problems in state of plane stress or plane strain, modelled with quadrilateral elements for both FEA and isogeometric analysis.

Chapter 2

Finite Element Analysis

This chapter reviews shortly basic formulas for the finite element method applied to solid and structural mechanics, including typical data structures. The Lagrange Q4 and Q9 elements are chosen for closer inspection since they are suitable for illustrating the differences between FEA and isogeometric analysis.

All formulas in this chapter are taken from Cook et al. [5]. The notation of Cook et al. [5] is employed, but without the brackets indicating vectors and matrices.

2.1 Basic Formulations for Plane Conditions

A two-dimensional problem is considered, either plane stress or plane strain.

2.1.1 Stress, Strain and Displacement Relations

Stress-strain and strain-displacement relations are fundamental for solid and structural mechanics. Assuming no initial stress or strains, stresses $\boldsymbol{\sigma}$ are related to strains $\boldsymbol{\varepsilon}$ as follow,

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \mathbf{E}\boldsymbol{\varepsilon} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1}{2}(1-\nu) \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} \tag{2.1}$$

where $\mathbf{E}$ is the constitutive matrix of plane stress, E is the elastic modulus and ν is the Poisson's ratio. If the problem is in a state of plane strain, $\mathbf{E}$ is replaced by

$$\mathbf{E} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix} \tag{2.2}$$

The strain-displacement relations read

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \boldsymbol{\partial}\mathbf{u} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_{,x} \\ u_{,y} \\ v_{,x} \\ v_{,y} \end{bmatrix} \tag{2.3}$$

where u and v are global displacements in x and y directions, respectively, and the notation $u_{,x} = {}^{\partial u}/_{\partial x}$ has been used.

2.1.2 Principle of Virtual Work

To obtain the element matrices the principle of virtual work, or the principle of virtual displacements, is employed,

$$\int_V (\delta\boldsymbol{\varepsilon})^T \boldsymbol{\sigma} dV = \int_V (\delta\mathbf{u})^T \mathbf{F} dV + \int_S (\delta\mathbf{u}) \boldsymbol{\Phi} dS \tag{2.4}$$

Here δ denotes virtual strains and displacements. The internal strain energy is equal to the external work done by the body forces $\mathbf{F}$ in the volume V and the surface tractions $\boldsymbol{\Phi}$ on the surface S.

The principle of virtual work is applicable for solid and structural mechanics only. A more general formulation involving the weak form of the differential equations (i.e. weighted residual methods like the Galerkin method) is also possible, but is not considered here.

The nature of a finite element solution summons compatibility between displacements $\mathbf{u}$ and strains $\boldsymbol{\varepsilon}$, and equilibrium between forces $\mathbf{F}$ and stresses $\boldsymbol{\sigma}$. These requirements must be satisfied at all nodes.

2.1.3 Discretization and Interpolation of Displacements

In this thesis the n noded finite element displacement vector is defined as

$$\mathbf{d} = \begin{bmatrix} u_1 & u_2 & \dots & u_n & v_1 & v_2 & \dots & v_n \end{bmatrix}^T \tag{2.5}$$

The displacements $\mathbf{u}$ are then interpolated by

$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{N}\mathbf{d} = \begin{bmatrix} N_1 & N_2 & \dots & N_n & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & N_1 & N_2 & \dots & N_n \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \\ v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n N_i u_i \\ \sum_{i=1}^n N_i v_i \end{bmatrix} \tag{2.6}$$

where $\mathbf{N}$ are the shape functions for the element. Strains are according to Eq. (2.3) then given by

$$\boldsymbol{\varepsilon} = (\boldsymbol{\partial}\mathbf{N})\mathbf{d} = \mathbf{B}\mathbf{d} \tag{2.7}$$

where $\mathbf{B}$ is the strain-displacement matrix.

2.1.4 The Element Analysis

From Eqs. (2.6) and (2.7) the virtual strains and displacements may be expressed as

$$\delta\mathbf{u}^T = (\delta\mathbf{d})^T \mathbf{N}^T \qquad \text{and} \qquad \delta\boldsymbol{\varepsilon}^T = (\delta\mathbf{d})^T \mathbf{B}^T \tag{2.8}$$

Inserting Eqs. (2.1) and (2.8) into the principle of virtual work, Eq. (2.4), gives

$$(\delta\mathbf{d})(\int_V \mathbf{B}^T\mathbf{E}\mathbf{B}dV\mathbf{d} - \int_V \mathbf{N}^T\mathbf{F}dV - \int_S \mathbf{N}^T\mathbf{\Phi}dS) = 0 \tag{2.9}$$

For an arbitrary $\delta\mathbf{d}$, Eq. (2.9) yields

$$\mathbf{k}\mathbf{d} = \mathbf{r}_e \tag{2.10}$$

where $\mathbf{k}$ is the element stiffness matrix,

$$\mathbf{k} = \int_V \mathbf{B}^T\mathbf{E}\mathbf{B}dV \tag{2.11}$$

and the external load is

$$\mathbf{r}_e = \int_V \mathbf{N}^T\mathbf{F}dV + \int_S \mathbf{N}^T\mathbf{\Phi}dS \tag{2.12}$$

These are the element matrices.

2.1.5 Isoparametric Bilinear Quadrilateral (Q4) Element

Figure 2.1 shows the bilinear quadrilateral (Q4) element using isoparametric representation. The solution space is defined by natural coordinates ξ, η, while the physical space is given by Cartesian coordinates x, y.

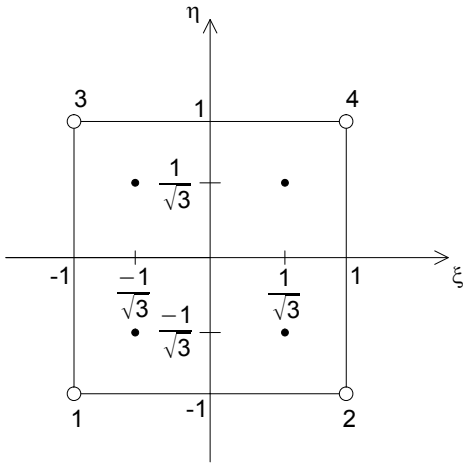


Figure 2.1: Bilinear quadrilateral (Q4) element

For this element, the shape functions are

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta) \tag{2.13a}$$

$$N_2 = \frac{1}{4}(1+\xi)(1-\eta) \tag{2.13b}$$

$$N_3 = \frac{1}{4}(1-\xi)(1+\eta) \tag{2.13c}$$

$$N_4 = \frac{1}{4}(1+\xi)(1+\eta) \tag{2.13d}$$

These shape functions are known as linear Lagrange polynomials.

The relation between natural derivatives and Cartesian derivatives is

$$\begin{bmatrix} \frac{\partial}{\partial \xi} \\ \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \mathbf{J} \begin{bmatrix} \frac{\partial}{\partial x} \\ \\ \frac{\partial}{\partial y} \end{bmatrix} = \begin{bmatrix} x_{,\xi} & y_{,\xi} \\ \\ x_{,\eta} & y_{,\eta} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \\ \frac{\partial}{\partial y} \end{bmatrix} \tag{2.14}$$

where the Jacobian matrix $\mathbf{J}$ gives the mapping between the physical space and the reference element. Knowing that the physical coordinates are interpolated in the same manner as displacements,

$$x = \sum_{i=1}^{4} N_i x_i \qquad \text{and} \qquad y = \sum_{i=1}^{4} N_i y_i \tag{2.15}$$

The Jacobian matrix becomes

$$\begin{aligned} \mathbf{J} &= \begin{bmatrix} \sum_{i=1}^{4} N_{i,\xi} x_i & \sum_{i=1}^{4} N_{i,\xi} y_i \\ \\ \sum_{i=1}^{4} N_{i,\eta} x_i & \sum_{i=1}^{4} N_{i,\eta} y_i \end{bmatrix} \\ &= \tfrac{1}{4} \begin{bmatrix} -(1-\eta) & (1-\eta) & -(1+\eta) & (1+\eta) \\ -(1-\xi) & -(1+\xi) & (1-\xi) & (1+\xi) \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \end{aligned} \tag{2.16}$$

The inverse Jacobian is

$$\mathbf{\Gamma} = \frac{1}{\det(\mathbf{J})} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} = \frac{1}{J} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \tag{2.17}$$

where $J = \det(\mathbf{J})$ is the Jacobian. Using Eqs. (2.14) and (2.17),

$$\begin{bmatrix} u_{,x} \\ u_{,y} \\ v_{,x} \\ v_{,y} \end{bmatrix} = \begin{bmatrix} \mathbf{\Gamma} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Gamma} \end{bmatrix} \begin{bmatrix} u_{,\xi} \\ u_{,\eta} \\ v_{,\xi} \\ v_{,\eta} \end{bmatrix} \tag{2.18}$$

Further the natural derivatives of the displacements are interpolated similarly to Eq. (2.6),

$$\begin{bmatrix} u_{,\xi} \\ u_{,\eta} \\ v_{,\xi} \\ v_{,\eta} \end{bmatrix} = \begin{bmatrix} N_{1,\xi} & N_{2,\xi} & N_{3,\xi} & N_{4,\xi} & 0 & 0 & 0 & 0 \\ N_{1,\xi} & N_{2,\xi} & N_{3,\xi} & N_{4,\xi} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & N_{1,\eta} & N_{2,\eta} & N_{3,\eta} & N_{4,\eta} \\ 0 & 0 & 0 & 0 & N_{1,\eta} & N_{2,\eta} & N_{3,\eta} & N_{4,\eta} \end{bmatrix} \mathbf{d} \tag{2.19}$$

Combining Eqs. (2.3), (2.18) and (2.19), the $\mathbf{B}$ matrix becomes

$$\mathbf{B} = \begin{bmatrix} N_{1,x} & N_{2,x} & N_{3,x} & N_{4,x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & N_{1,y} & N_{2,y} & N_{3,y} & N_{4,y} \\ N_{1,y} & N_{2,y} & N_{3,y} & N_{4,y} & N_{1,x} & N_{2,x} & N_{3,x} & N_{4,x} \end{bmatrix} \tag{2.20}$$

Assuming constant thickness t, the element stiffness matrix can now be numerically integrated,

$$\mathbf{k} = t \int_{-1}^{1} \int_{-1}^{1} \mathbf{B}^T \mathbf{E} \mathbf{B} J d\xi d\eta \approx t \sum_i \sum_j \mathbf{B}^T(\xi_i, \eta_j) \mathbf{E} \mathbf{B}(\xi_i, \eta_j) J W_i W_j \tag{2.21}$$

where ξ_i, η_j are the coordinates of Gauss points and W_i, W_j are the corresponding weights. In this thesis, full integration is chosen for all analyses. Full integration is defined as a quadrature rule of sufficient accuracy to exactly integrate all stiffness coefficient k_{ij} of an undistorted element [5]. For full integration of the Q4 element, $(\xi_i, \eta_j) = (\pm 1/\sqrt{3}, \pm 1/\sqrt{3})$ (see Figure 2.1) and $W_i = W_j = 1.0$ for all four points.

2.1.6 Isoparametric Quadratic Quadrilateral (Q9) Element

Figure 2.2 shows the quadratic quadrilateral (Q9) element using isoparametric representation.

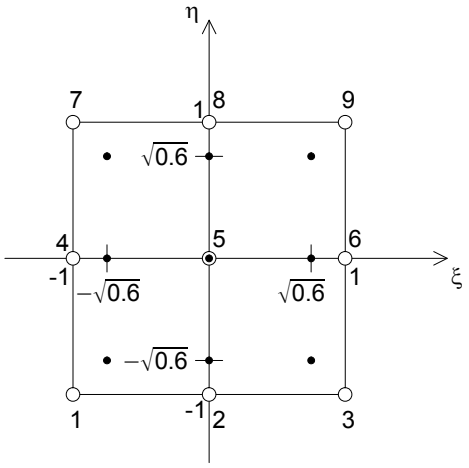


Figure 2.2: Quadratic quadrilateral (Q9) element

For this element, the shape functions are

$$N_1 = \frac{1}{4}\xi\eta(1-\xi)(1-\eta) \tag{2.22a}$$

$$N_2 = -\frac{1}{2}\eta(1-\xi^2)(1-\eta) \tag{2.22b}$$

$$N_3 = -\frac{1}{4}\xi\eta(1+\xi)(1-\eta) \tag{2.22c}$$

$$N_4 = -\frac{1}{2}\xi(1-\xi)(1-\eta^2) \tag{2.22d}$$

$$N_5 = (1-\xi^2)(1-\eta^2) \tag{2.22e}$$

$$N_6 = \frac{1}{2}\xi(1+\xi)(1-\eta^2) \tag{2.22f}$$

$$N_7 = -\frac{1}{4}\xi\eta(1-\xi)(1+\eta) \tag{2.22g}$$

$$N_8 = \frac{1}{2}\eta(1-\xi^2)(1+\eta) \tag{2.22h}$$

$$N_9 = \frac{1}{4}\xi\eta(1+\xi)(1+\eta) \tag{2.22i}$$

These shape functions are the quadratic Lagrange polynomials. The Jacobian matrix $\mathbf{J}$ reads

$$\mathbf{J} = \begin{bmatrix} -\frac{1}{4}\eta(1-2\xi)(1-\eta) & -\frac{1}{4}\xi(1-\xi)(1-2\eta) \\ -\frac{1}{4}\xi\eta(1-\eta) & -\frac{1}{2}(1-\xi^2)(1-2\eta) \\ -\frac{1}{4}\eta(1+2\xi)(1-\eta) & -\frac{1}{4}\xi(1+\xi)(1-2\eta) \\ \frac{1}{2}(1-2\xi)(1-\eta^2) & \xi\eta(1-\xi) \\ -2\xi(1-\eta^2) & -2\eta(1-\xi^2) \\ \frac{1}{2}(1+2\xi)(1-\eta^2) & -\xi\eta(1+\xi) \\ -\frac{1}{4}\eta(1-2\xi)(1+\eta) & -\frac{1}{4}\xi(1-\xi)(1+2\eta) \\ -\xi\eta(1+\eta) & \frac{1}{2}(1-\xi^2)(1+2\eta) \\ \frac{1}{4}\eta(1+2\xi)(1+\eta) & \frac{1}{4}\xi(1+\xi)(1+2\eta) \end{bmatrix}^T \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \\ x_5 & y_5 \\ x_6 & y_6 \\ x_7 & y_7 \\ x_8 & y_8 \\ x_9 & y_9 \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{2.23}$$

The $\mathbf{B}$ matrix is found similar to the $\mathbf{B}$ matrix for the Q4 element,

$$\mathbf{B} = \begin{bmatrix} N_{1,x} & N_{2,x} & \dots & N_{9,x} & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & N_{1,y} & N_{2,y} & \dots & N_{9,y} \\ N_{1,y} & N_{2,y} & \dots & N_{9,y} & N_{1,x} & N_{2,x} & \dots & N_{9,x} \end{bmatrix} \tag{2.24}$$

The element stiffness matrix for the Q9 element is obtained likewise the Q4 element, Eq. (2.21). For full integration of the Q9 element, 3rd order Gauss quadrature with points 0 and $\pm\sqrt{0.6}$ (see Figure 2.2) is used with corresponding weights $8/9$ and $5/9$.

2.1.7 The System Analysis

To perform the system analysis, contributions from element stiffnesses and nodal loads are assembled into the global system according to element topology. The global stiffness matrix reads

$$\mathbf{K} = \sum_{i=1}^{N_{els}} \mathbf{k}_i \tag{2.25}$$

where N_{els} is the total number of elements. The global load vector when assuming no direct point loads is,

$$\mathbf{R} = \sum_{i=1}^{N_{els}} \mathbf{r}_{e_i} \tag{2.26}$$

The global system matrices thus become

$$\mathbf{KD} = \mathbf{R} \tag{2.27}$$

where $\mathbf{D}$ are the nodal displacements for the global system, first all components of u, then v. The strains for each element and each Gauss point are found by employing Eq. (2.7) and the stresses by Eq. (2.1). The von Mises stress for plane conditions after von Mises yield criterion is

$$\sigma_{\mathrm{e}} = \sqrt{\sigma_x^2 - \sigma_x\sigma_y + \sigma_y^2 + 3\tau_{xy}^2} \tag{2.28}$$

The strain energy of the system is found by either one of Eqs. (2.29), (2.30) or (2.31),

$$U_1 = \mathbf{D}^T\mathbf{R} \tag{2.29}$$

$$U_2 = \sum_{i=1}^{N_{els}} \mathbf{d}^T\mathbf{kd} \tag{2.30}$$

$$U_3 = \sum_{i=1}^{N_{els}} \int_V \boldsymbol{\varepsilon}^T\mathbf{E}\boldsymbol{\varepsilon} dV \tag{2.31}$$

Note that the strain energy is twice the total strain energy as defined in Cook et al. [5].

2.2 Computational Procedures using Q4 and Q9 Elements

An important part of this thesis is to review the computational formulations for isogeometric analysis as opposed to finite element analysis. This section presents some basic data structures for the finite element method using the bilinear quadrilateral and the quadratic quadrilateral elements. The theory which has been reviewed in Section 2.1 is the foundation for the computational formulations.

Typically, a FEA program contains the following steps:

1. Preprocessing
2. Solving
3. Postprocessing

Figure 2.3 shows a flow chart of the processes in a typical FEA code.

2.2.1 Preprocessing

In the preprocessing step, the FE program needs to read input about material properties, geometry, node coordinates and element topology. Assuming homogeneous and linear-elastic material, the constitutive matrix $\mathbf{E}$ is constant and hence this input can be given before forming the element stiffness matrix $\mathbf{k}$.

Figure 2.4 shows how a typical FE program organizes elements and global nodes for Q4 and Q9 elements, respectively. Large bold numbers are elements, while small medium numbers are nodes.

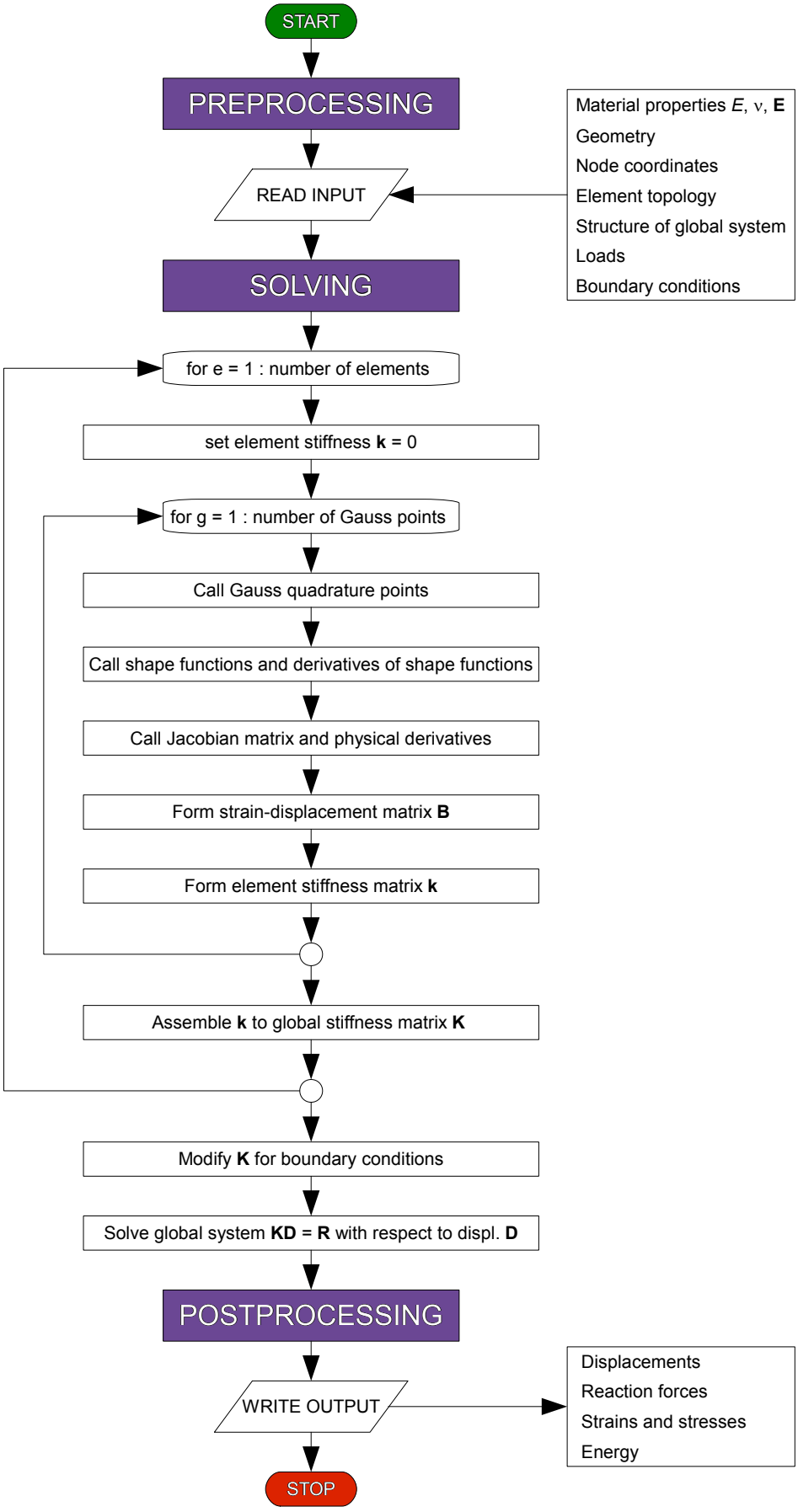
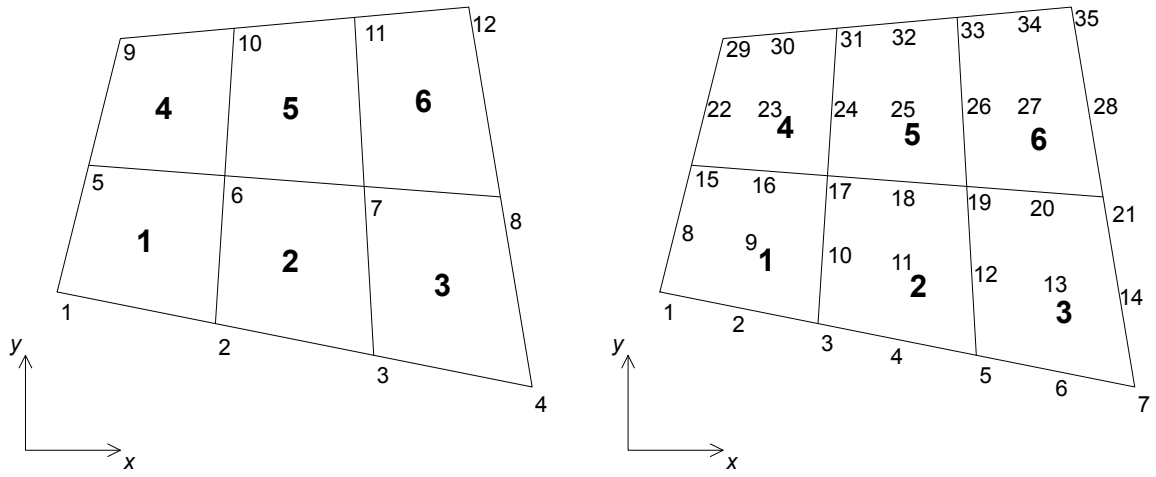


Figure 2.3: Flow chart for a typical FEA program



(a) Q4 elements

(b) Q9 elements

Figure 2.4: Elements and global nodes

The geometry is defined by storing the physical coordinates x, y of each node in a matrix where rows represents node number,

$$\texttt{NodeCoord}_{\texttt{Q4}} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{12} & y_{12} \end{bmatrix} \quad \text{and} \quad \texttt{NodeCoord}_{\texttt{Q9}} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{35} & y_{35} \end{bmatrix} \tag{2.32}$$

The element topology is also known as the IEN ("element nodes") array, and is a matrix connecting the nodes to the elements. The IEN arrays for this geometry are

$$\texttt{IEN}_{\texttt{Q4}} = \begin{bmatrix} 1 & 2 & 5 & 6 \\ 2 & 3 & 6 & 7 \\ 3 & 4 & 7 & 8 \\ 5 & 6 & 9 & 10 \\ 6 & 7 & 10 & 11 \\ 7 & 8 & 11 & 12 \end{bmatrix} \quad \text{and} \quad \texttt{IEN}_{\texttt{Q9}} = \begin{bmatrix} 1 & 2 & 3 & 8 & 9 & 10 & 15 & 16 & 17 \\ 3 & 4 & 5 & 10 & 11 & 12 & 17 & 18 & 19 \\ 5 & 6 & 7 & 12 & 13 & 14 & 19 & 20 & 21 \\ 15 & 16 & 17 & 22 & 23 & 24 & 29 & 30 & 31 \\ 17 & 18 & 19 & 24 & 25 & 26 & 31 & 32 & 33 \\ 19 & 20 & 21 & 26 & 27 & 28 & 33 & 34 & 35 \end{bmatrix} \tag{2.33}$$

where rows represent the elements and columns represent the nodes that support the element. Organizing all displacements u before displacements v, the global node numbers are directly degrees of freedom (DOF) in x direction, while DOF in y direction are the global node numbers plus the total number of nodes. The element topology may be the same using polar axes. The physical node coordinates for the element can be found by extracting these values using the row number of the IEN array. For example, for element number 4 in the Q4 mesh,

$$\texttt{NodeCoord}(\texttt{IEN\_e}) = \begin{bmatrix} x_5 & y_5 \\ x_6 & y_6 \\ x_9 & y_9 \\ x_{10} & y_{10} \end{bmatrix} \quad \text{where} \quad \texttt{IEN\_e} = \begin{bmatrix} 5 & 6 & 9 & 10 \end{bmatrix} \tag{2.34}$$

In the preprocessing step, the structure of the global system is defined as matrices of zeros.

Loads are either given as direct point loads or consistent nodal loads for a distributed load. In general, the consistent nodal loads can be found by numerically integrating the shape functions,

$$\mathbf{r}_e = \int_s \mathbf{N}^T \mathbf{N} \mathbf{q} ds \approx \sum_i \mathbf{N}^T(\xi_i)\mathbf{N}(\xi_i)\mathbf{q}(\xi_i) J W_i \tag{2.35}$$

where s is the boundary for which the distributed load $\mathbf{q}$ is to be applied. Here, J is the Jacobian for the boundary s, i.e. $J = \left| \begin{array}{c} \partial x/\partial s \\ \partial y/\partial s \end{array} \right|$. This expression should be used for Q4 elements, since the shape functions of the Q4 element cannot represent nonlinear distributed loads. However, for rectangles and parallelograms, the Jacobian is constant $J = A/4$, where A is the area of the physical element.

For a distributed load with constant value q, the consistent nodal loads for the Q4 element will be

$$\mathbf{r}_e = \begin{bmatrix} \frac{1}{2}qa \\ \\ \frac{1}{2}qa \end{bmatrix} \tag{2.36}$$

where a is the distance between two adjacent nodes. The shape functions of the Q9 element can represent a parabolic load. In this case the exact integral may be evaluated, and the consistent nodal loads become

$$\mathbf{r}_e = \frac{a}{15} \begin{bmatrix} 4 & 2 & -1 \\ 2 & 16 & 2 \\ -1 & 2 & 4 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{2.37}$$

where q_i is the value of the distributed load at node i. For $q_1 = q_2 = q_3 = q$, this reduces to

$$\mathbf{r}_e = \begin{bmatrix} \frac{1}{3}qa \\ \\ \frac{4}{3}qa \\ \\ \frac{1}{3}qa \end{bmatrix} \tag{2.38}$$

Last, the preprocessing step contains information about boundary conditions (BC), given as prescribed DOF. The prescribed DOF may be either suppressed ($u = 0$, $v = 0$), or prescribed a value ($u =$ a constant number, $v =$ a constant number). For example, to make the left boundary of the geometry in Figure 2.4a fixed, the BC is expressed as a column vector of DOF numbers,

$$\texttt{prDof} = \begin{bmatrix} 1 & 5 & 9 & 13 & 17 & 21 \end{bmatrix}^T \tag{2.39}$$

If in addition the right boundary is to be prescribed with the vertical displacement $v = 10$, this value must be applied to the displacement vector,

$$\mathbf{D}(16) = \mathbf{D}(20) = \mathbf{D}(24) = 10$$

and also included with the other BC, i.e.

$$\texttt{prDof} = \begin{bmatrix} 1 & 5 & 9 & 13 & 16 & 17 & 20 & 21 & 24 \end{bmatrix}^T \tag{2.40}$$

In the solving step, the program makes use of this boundary condition vector.

2.2.2 Solving

The solving step involves forming the global stiffness matrix $\mathbf{K}$ and solving the global system, Eq. (2.27), with respect to $\mathbf{D}$. The element stiffness matrix $\mathbf{k}$ is formed by employing Eq. (2.21), i.e. numerically integrate the stiffness contributions at the Gauss points.

The element stiffness matrix $\mathbf{k}$ is formed at first by setting $\mathbf{k} = 0$ outside the loop through the Gauss points, see Figure 2.3. When the Gauss point loop is complete, the element stiffness matrix is also complete for the current element. The global stiffness matrix $\mathbf{K}$ is then formed by assembling the element stiffness matrices according to the element topology (place the stiffness contributions at the DOF numbers according to the IEN array), in the loop through the elements. If $\mathbf{k}$ is not needed separately, the assembling of $\mathbf{K}$ can also be performed inside the Gauss point loop, i.e. move the assembling of $\mathbf{k}$ to $\mathbf{K}$ to inside the Gauss loop, and delete the redundant process $\mathbf{k} = 0$. However, note that the element stiffness matrix may be needed for instance in the calculation of the strain energy.

After the global system matrices are created, $\mathbf{KD} = \mathbf{R}$ is usually modified by separating the free (active) DOF $\mathbf{D}_f$ and suppressed (prescribed) DOF $\mathbf{D}_s$ [2],

$$\begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{fs} \\ \mathbf{K}_{sf} & \mathbf{K}_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{D}_f \\ \mathbf{D}_s \end{bmatrix} = \begin{bmatrix} \mathbf{R}_f \\ \mathbf{R}_s \end{bmatrix} \tag{2.41}$$

where $\mathbf{D}_f = \mathbf{D}(\texttt{aDof})$, $\mathbf{D}_s = \mathbf{D}(\texttt{prDof})$ and `aDof` is a column vector containing all DOF numbers not in `prDof`. From the first line of equations, the following is obtained,

$$\mathbf{K}_{ff}\mathbf{D}_f = \mathbf{R}_f - \mathbf{K}_{fs}\mathbf{D}_s = \hat{\mathbf{R}}_f \tag{2.42}$$

where $\hat{\mathbf{R}}_f$ are the forces including possible forces due to prescribed displacement different from zero. $\mathbf{K}_{fs}$ is obtained by extracting the right contributions from $\mathbf{K}$, $\mathbf{K}_{fs} = \mathbf{K}(\texttt{aDof}, \texttt{prDof})$, and $\mathbf{R}_f = \mathbf{R}(\texttt{aDof})$. The system of equations is then solved with respect to active DOF by for example Gaussian elimination (built-in operator in MATLAB). The forces may be reclaimed by evaluating $\mathbf{R} = \mathbf{KD}$ after the system is solved.

2.2.3 Postprocessing

In the postprocessing step, the program writes outputs of interest and request. For structural mechanics, these quantities may for instance be displacements, reaction forces, strains, stresses and strain energy.

Displacements u and v may be extracted from $\mathbf{D}$ for nodes of interest, or plotted as a displacement field. Reaction forces are related to suppressed DOF, and may therefore be extracted from $\mathbf{R}$.

Strains and stresses may be calculated by rebuilding the $\mathbf{B}$ matrix in the same manner as when forming the element stiffness matrix $\mathbf{k}$. This means that all the processes under the solving step are repeated, except for forming $\mathbf{k}$. Strains are then given by Eq. (2.7) and stresses by Eq. (2.1). Since the $\mathbf{B}$ matrix is evaluated at the Gauss points, strains and stresses evaluated will also be at the Gauss point. This can be organized as one matrix for each strain component ε_x, ε_y and γ_{xy}, or stress component σ_x, σ_y and τ_{xy}, on the form

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_1^1 & \sigma_2^1 & \dots & \sigma_g^1 \\ \sigma_1^2 & \sigma_2^2 & \dots & \sigma_g^2 \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_1^e & \sigma_2^e & \dots & \sigma_g^e \end{bmatrix} \tag{2.43}$$

where e is the number of elements and g is the number of Gauss points per element. If desired, stresses at nodes can be found by extrapolating the stresses from the Gauss points using the shape functions.

The strain energy may be calculated in three different ways, by Eqs. (2.29), (2.30) or (2.31), or by all formulas for comparison of numerical accuracy. Energy found by employing Eq. (2.31) also requires rebuilding of the $\mathbf{B}$ matrix and in addition, energy calculated from Eq. (2.30) requires evaluation of the $\mathbf{k}$ matrix.

Chapter 3

B-Splines, NURBS and T-Splines

3.1 Overview

In FEA, the Lagrange polynomials are the basis for numerical analysis. In isogeometric analysis non-uniform rational B-splines (NURBS), or the more general T-splines, are used instead. Both FEA and isogeometric analysis employ the isoparametric concept, which means that the same basis is used for geometry and analysis. The difference is that Lagrange polynomials are used to approximate both the unknown solution and the known geometry, while NURBS and T-splines can represent the exact geometry and also approximate the solution field.

The quality of being able to exactly represent the geometry is one of the main reasons why NURBS are widely used in computer-aided design (CAD) in present time. The newly introduced T-splines are the general form of NURBS and allow local refinement, rendering the possibility to generate analysis-suitable models of arbitrary topology.

The benefit of isogeometric analysis being able to model the geometry exactly is clearly quite attractive, but the method of isogeometric analysis also provides advantages concerning accuracy compared to computational cost. Since FEA is a method for numerical solution of differential equations, being able to incorporate NURBS or T-splines into the same concept is very beneficial.

One obstacle however, is that Lagrange polynomials have gained ground in FEA for decades. Another is that NURBS and T-splines are not as straightforward as Lagrange polynomials. Therefore, an introduction to B-splines, NURBS and T-splines is given subsequently.

All formulas in Sections 3.2 and 3.3 are taken from Cottrell et al. [6], Piegl and Tiller [19], while the theory and formulas presented in Section 3.4 are extracted from Bazilevs et al. [1], Scott et al. [21], Li et al. [15], Scott et al. [20].

3.2 B-Splines

Unlike FEA, the B-spline parameter space (i.e. the space which the basis spans) is local to *patches* and not elements. In FEA the parameter space is the reference element which is mapped into each single element in the physical space. In isogeometric analysis, the parameter space consists of several elements, and the mapping to the physical representation involves all these elements rather than one single. For simpler problems, one patch is enough to be able to represent the geometry and properties for analysis. However, if the problem is complex, it may be necessary to use several patches with different properties to form the mesh.

The parameter space is partitioned into elements by a *knot vector* in each direction, which is a non-decreasing set of coordinates in one dimension. These elements are also known as *knot spans*, because they span between knot values.

This thesis is restricted to single patch problems, indicating that only one knot vector defines the basis functions in each direction and that only one parameter space represents the physical problem.

3.2.1 Knot Vectors

The knot vector is written

$$\Xi = \{\xi_1, \xi_2, ..., \xi_i, ..., \xi_{n+p+1}\} \tag{3.1}$$

i.e. the length of the knot vector $|\Xi| = n + p + 1$. Here ξ_i denotes the i^{th} knot, i is the knot index, n is the number of basis functions and p is the polynomial order.

The knot vector may be *uniform*, meaning that the knots are equally spaced in the parameter space, or *non-uniform*, meaning that knot values may be repeated. A knot vector is *open* if its first and last value appear $(p + 1)$ times. An open knot vector forms basis functions which are interpolatory at the ends of the parameter space, and are therefore used henceforth in the development of the basis functions for isogeometric analysis. The property of being interpolatory at the boundary of the parameter space is important for boundary conditions and when considering multiple patch problems, because patches are joined at the ends of the parameter space.

As mentioned, the knot vector stipulates the element mesh. An element exists between two knots which have different values.

3.2.2 Basis Functions

When a knot vector is chosen, the basis functions are defined according to the Cox-de Boor recursion formula [8, 7], for $p = 0$,

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

and for $p = 1, 2, 3, \ldots$,

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \tag{3.3}$$

which means that the basis functions are on parametric form in contrast to FEA, where the Lagrange polynomials are explicit functions.

Note that each set of basis functions of polynomial order p are dependent on the previous set of basis functions, order $(p - 1)$.

These basis functions,

- like shape functions in FEA, constitute the partition of unity, i.e. $\sum_{i=1}^{n} N_{i,p}(\xi) = 1$.
- like shape functions in FEA, are linearly independent, i.e. $\sum_{i=1}^{n} a_i N_{i,p}(\xi) = 0 \iff a_i = 0$, $i = 1, 2, \ldots, n$.
- like shape functions in FEA, have a compact support.
- unlike shape functions in FEA, are non-negative over the entire domain.

- unlike shape functions in FEA, have $(p-1)$ continuous derivatives across the knots (element boundaries), if the knot vector is uniform.

In addition, for non-uniform knot vectors, the basis functions of order p are C^{p-m_i} continuous across knot ξ_i, where m_i is the number of the value ξ_i repeated. Note that the continuity property of B-splines makes the basis functions non-interpolatory across element boundaries in contrast to shape functions.

The first derivatives of the basis functions are needed for the $\mathbf{B}$ matrix in the evaluation of the element stiffness matrix $\mathbf{k}$, Eq. (2.21). For a given polynomial order p and knot vector Ξ, the derivative of the i^{th} basis function is given by

$$\frac{d}{d\xi}N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}N_{i+1,p-1}(\xi) \tag{3.4}$$

3.2.3 Constructing Basis Functions from Knot Vector - An Example

To show how the basis functions are formed by Eqs. (3.2) and (3.3), consider an example which is a more thoroughgoing review of the same example in Cottrell et al. [6]. Given the knot vector $\Xi = \{\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6, \xi_7\} = \{0, 0, 0, 1, 2, 2, 2\}$, the polynomial order is 2, since its first and last values appear $p + 1 = 3$ times. The length of the knot vector is 7, hence the number of basis functions is 4; $N_{1,2}(\xi)$, $N_{2,2}(\xi)$, $N_{3,2}(\xi)$ and $N_{4,2}(\xi)$.

To obtain these four functions, start with the basis functions for $p = 0$. Because there is no such value $\xi \geq \xi_1 = 0$ and at the same time $\xi < \xi_2 = 0$,

$$N_{1,0}(\xi) \doteq 0 \tag{3.5a}$$

Following the same argument,

$$N_{2,0}(\xi) \doteq 0 \tag{3.5b}$$

Further,

$$N_{3,0}(\xi) = \begin{cases} 1 & 0 \leq \xi < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.5c}$$

$$N_{4,0}(\xi) = \begin{cases} 1 & 1 \leq \xi < 2 \\ 0 & \text{otherwise} \end{cases} \tag{3.5d}$$

Similar to the first basis function, because there is no such value $\xi \geq \xi_5 = 2$ and at the same time $\xi < \xi_6 = 2$,

$$N_{5,0}(\xi) \doteq 0 \tag{3.5e}$$

Following the same argument,

$$N_{6,0}(\xi) \doteq 0 \tag{3.5f}$$

Note that since $p = 0$, the number of basis functions $n = 7 - 1 - 0 = 6$.

After the basis functions for $p = 0$ are formed, the basis functions for $p = 1$ may be created. The number of linear basis functions is $n = 7 - 1 - 1 = 5$,

$$\begin{aligned} N_{1,1}(\xi) &= \tfrac{\xi-\xi_1}{\xi_{1+1}-\xi_1}N_{1,0}(\xi) + \tfrac{\xi_{1+1+1}-\xi}{\xi_{1+1+1}-\xi_{1+1}}N_{2,0}(\xi) = \tfrac{\xi-\xi_1}{\xi_2-\xi_1}N_{1,0}(\xi) + \tfrac{\xi_3-\xi}{\xi_3-\xi_2}N_{2,0}(\xi) \\ &= \tfrac{\xi-0}{0-0}0 + \tfrac{0-\xi}{0-0}0 \doteq 0 \end{aligned} \tag{3.6a}$$

$$N_{2,1}(\xi) = \frac{\xi-\xi_2}{\xi_3-\xi_2}N_{2,0}(\xi) + \frac{\xi_4-\xi}{\xi_4-\xi_3}N_{3,0}(\xi) = \frac{\xi-0}{0-0}0 + \frac{1-\xi}{1-0}N_{3,0}(\xi) = \begin{cases} 1-\xi & 0 \le \xi < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.6b}$$

$$\begin{aligned} N_{3,1}(\xi) &= \tfrac{\xi-0}{1-0}N_{3,0}(\xi) + \tfrac{2-\xi}{2-1}N_{4,0}(\xi) = \begin{cases} \xi & 0 \le \xi < 1 \\ 0 & \text{otherwise} \end{cases} + \begin{cases} 2-\xi & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} \xi & 0 \le \xi < 1 \\ 2-\xi & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.6c}$$

$$N_{4,1}(\xi) = \frac{\xi-1}{2-1}N_{4,0}(\xi) + \frac{2-\xi}{2-2}N_{5,0}(\xi) = \begin{cases} \xi-1 & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} \tag{3.6d}$$

$$N_{5,1}(\xi) = \frac{\xi-2}{2-2}N_{5,0}(\xi) + \frac{2-\xi}{2-2}N_{6,0}(\xi) = \begin{cases} \xi-1 & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} \tag{3.6e}$$

When the denominator is zero, the contribution is defined to be zero. Finally, the four basis functions for $p = 2$ are gained,

$$N_{1,2}(\xi) = \frac{\xi-0}{0-0}N_{1,1}(\xi) + \frac{1-\xi}{1-0}N_{2,1}(\xi) = \begin{cases} (1-\xi)^2 & 0 \le \xi < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.7a}$$

$$\begin{aligned} N_{2,2}(\xi) &= \tfrac{\xi-0}{1-0}N_{2,1}(\xi) + \tfrac{2-\xi}{2-0}N_{3,1}(\xi) = \begin{cases} \xi(\xi-1) & 0 \le \xi < 1 \\ 0 & \text{otherwise} \end{cases} + \begin{cases} \frac{1}{2}(2-\xi)\xi & 0 \le \xi < 1 \\ \frac{1}{2}(2-\xi)^2 & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} \xi(\xi-1) + \frac{1}{2}(2-\xi)\xi & 0 \le \xi < 1 \\ \frac{1}{2}(2-\xi)^2 & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.7b}$$

$$\begin{aligned} N_{3,2}(\xi) &= \tfrac{\xi-0}{2-0}N_{3,1}(\xi) + \tfrac{2-\xi}{2-1}N_{4,1}(\xi) = \begin{cases} \frac{1}{2}\xi^2 & 0 \le \xi < 1 \\ \frac{1}{2}\xi(2-\xi) & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} + \begin{cases} (2-\xi)(\xi-1) & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} \frac{1}{2}\xi^2 & 0 \le \xi < 1 \\ \frac{1}{2}\xi(2-\xi) + (2-\xi)(\xi-1) & 1 \le \xi < 2 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.7c}$$

$$N_{4,2}(\xi) = \frac{\xi - 1}{2 - 1} N_{4,1}(\xi) + \frac{2 - \xi}{2 - 2} N_{5,1}(\xi) = \begin{cases} (\xi - 1)^2 & 1 \leq \xi < 2 \\ 0 & \text{otherwise} \end{cases} \tag{3.7d}$$

The four basis functions are shown in Figure 3.1. Note that $N_{1,2}(\xi)$ and $N_{4,2}(\xi)$ coincide at $\xi = 1$. Also, over this knot the continuity is C^1. This may be seen as two basis functions different from zero over $\xi = 1$.

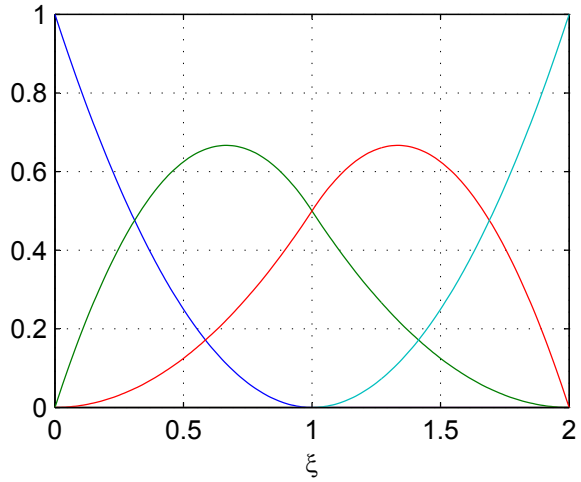


Figure 3.1: Basis functions from $\Xi = \{0, 0, 0, 1, 2, 2, 2\}$

3.2.4 Control Points and B-Spline Curves

To be able to construct a B-spline curve, a set of control points $\mathbf{P}_i$, $i = 1, 2, \ldots, n$, is needed. If the curve is to be drawn in the two-dimensional space, $\mathbf{P}_i \in \mathbb{R}^2$. The B-spline curve is then interpolated by

$$\mathbf{C}(\xi) = \begin{bmatrix} x \\ y \end{bmatrix} (\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) \mathbf{P}_i = \sum_{i=1}^{n} N_{i,p}(\xi) \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{3.8}$$

The control points serve as the degrees of freedom in isogeometric analysis. It may seem like these control points are analogous to nodal coordinates in FEA in the manner that they are the coefficient of the basis functions. However, the basis functions are *not* interpolatory at the control points. Unless straight lines are to be drawn, the alignment of control points will, in most cases, be outside the actual curve (except for the first and last point). Visually, the B-spline curve is pulled towards the controls points' alignment.

The way to draw curves with B-splines is important in the sense to how a physical problem can be modelled, especially in contrast to a model in FEA. Curved lines to be modelled for FEA are usually approximated with lower order Lagrange polynomials using isoparametric representation. In addition, Lagrange interpolation of discontinuous data leads to oscillations as the order increases [6].

In contrast to this, B-spline curves display a strong convex hull property, which means that the curve lies within the convex hull of its control points. The effect of each control point is diminished when the polynomial order increases, giving smoother curves. B-splines also possess the property of affine covariance, which means that transformations preserving parallel relationships in the physical space may be obtained by applying the transformation to the control points.

B-splines curves are revisited in Section 3.3.3.

3.2.5 B-Spline Surfaces

A B-spline surface is constructed by the basis functions in two directions, $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$, and a set of control points $\mathbf{P}_{i,j}$, $i = 1, 2, ..., n$, $j = 1, 2, ...m$. Similar to the first parametric direction ξ, $M_{j,q}(\eta)$ is also defined by Eqs. (3.2) and (3.3), but another knot vector $\mathcal{H} = \{\eta_1, \eta_2, ..., \eta_j, ..., \eta_{m+q+1}\}$ constitutes the foundation. Often the polynomial order is the same in both directions, i.e. $q = p$.

If the surface is to be drawn in the two-dimensional space, $\mathbf{P}_{i,j} \in \mathbb{R}^2$. The B-spline surface is then interpolated by

$$\mathbf{S}(\xi, \eta) = \begin{bmatrix} x \\ y \end{bmatrix} (\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \mathbf{P}_{i,j} = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(\xi) M_{j,q}(\eta) \begin{bmatrix} x_{i,j} \\ y_{i,j} \end{bmatrix} \tag{3.9}$$

A B-spline surface is the result of a tensor product. The *local support* of a basis function reads

$$N_{i,p}(\xi) M_{j,q}(\eta) = [\xi_i, \xi_{i+p+1}] \times [\eta_j, \eta_{j+q+1}] \tag{3.10}$$

meaning that the support of the bivariate function $N_{i,p}(\xi) M_{j,q}(\eta)$ extends over the area restricted by the knot values $[\xi_i, \xi_{i+p+1}] \times [\eta_j, \eta_{j+q+1}]$, where i,j is the knot index and p,q is the polynomial order in ξ and η direction, respectively.

The support of basis functions and control points is best illustrated with an example. Consider a simple mesh with two elements, each with size 250 x 250, see Figure 3.2. This geometry can be modelled by the knot vectors $\Xi = \{0, 0, 0, 1, 2, 2, 2\}$ and $\mathcal{H} = \{0, 0, 0, 1, 1, 1\}$, i.e. polynomial order 2. The number of basis function in ξ direction is 4 and in η direction 3. Take the bivariate basis function $N_{1,2}(\xi) M_{2,2}(\eta)$, for example; this function has support over the domain $[\xi_1, \xi_4] \times [\eta_2, \eta_5]$ in *the index space*. Since the left element in *the parameter space* spans $[\xi_1, \xi_4] \times [\eta_1, \eta_6]$ and the right element $[\xi_4, \xi_7] \times [\eta_1, \eta_6]$, the given function supports the left element. Another function, $N_{3,2}(\xi) M_{1,2}(\eta)$, has support over $[\xi_3, \xi_6] \times [\eta_1, \eta_4]$, which makes the function supported in both elements.

3.2.6 Anchors

In isogeometric analysis, the number of basis functions is not determined by the number of knots like the number of Lagrange polynomials is determined by the number of nodes. This is because basis functions are not in a one-to-one correspondence to knots. Therefore, it is convenient to define an *anchor* $\mathbf{s}_i$ to each basis function, which identifies a location in the parameter space associated with the basis function,

$$\mathbf{s}_i = \begin{cases} \xi_{i+(p+1)/2} & \text{if } p \text{ is odd} \\ \\ \frac{1}{2}\left(\xi_{i+(p/2)} + \xi_{i+(p/2)+1}\right) & \text{if } p \text{ is even} \end{cases} \tag{3.11}$$

This means that for uniform knot vectors, the anchors placing will be *at the knots* for odd polynomial orders and *at the centre of the knot spans* for even polynomial orders. The number of anchors is equal to the number of control points, which makes it easier to relate the control points to a location in the index space.

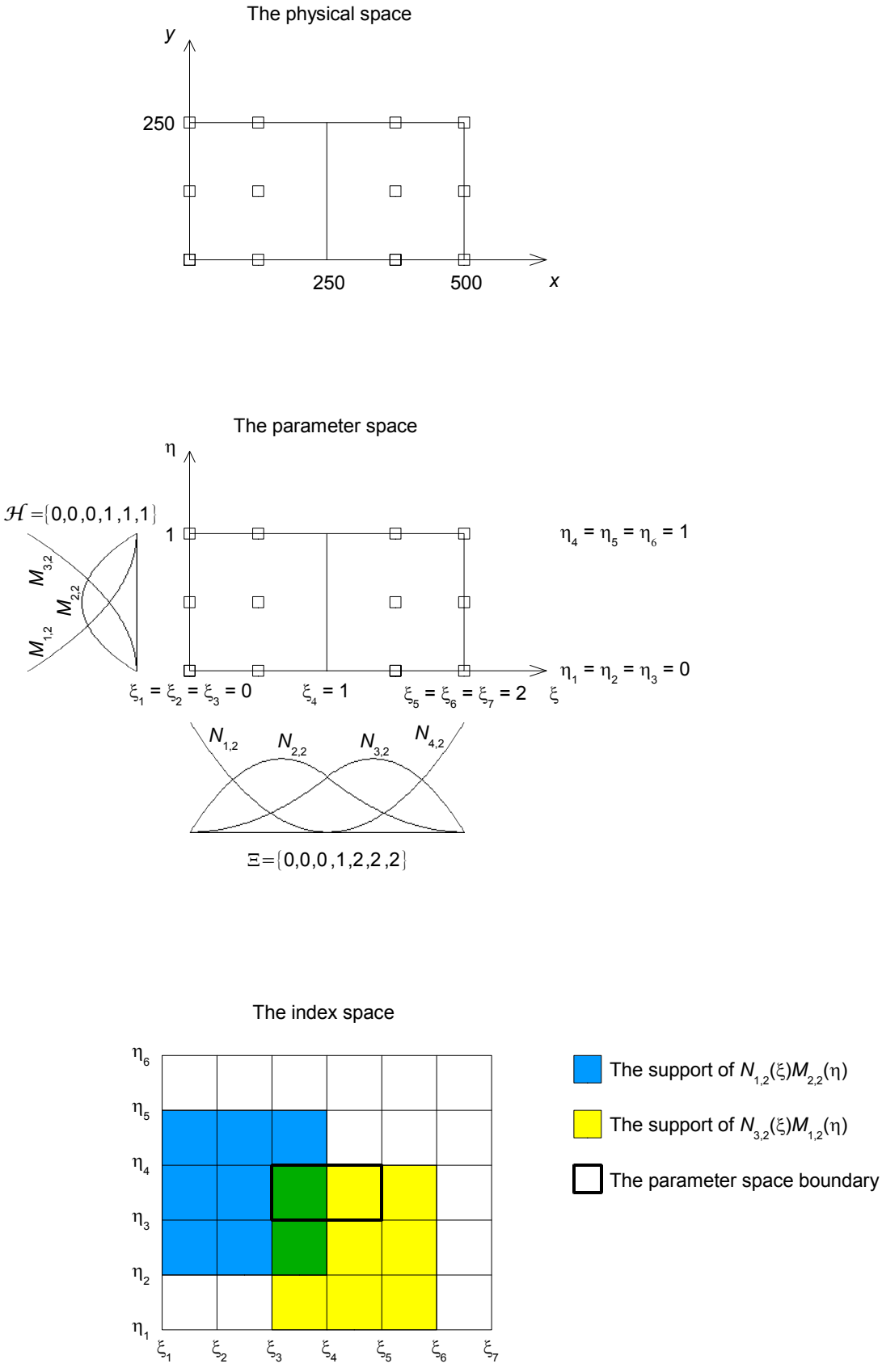


Figure 3.2: The viewing of B-splines and the support of basis functions

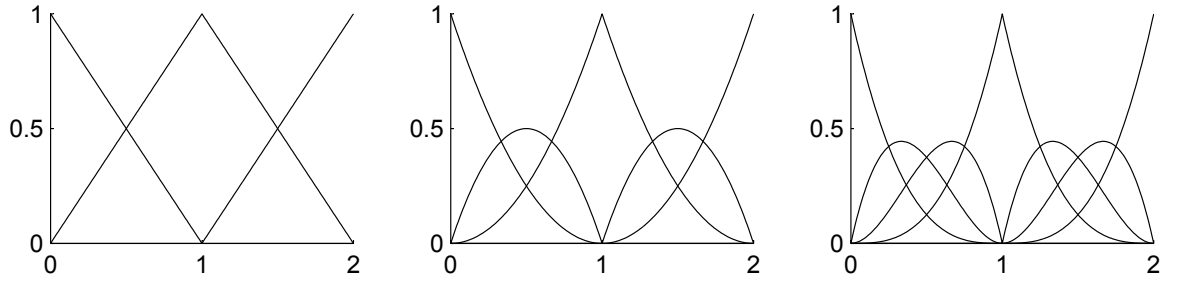
3.2.7 Refinement

When an analysis is to be performed, refinement of the mesh is usually desired for more accurate results. In isogeometric analysis, three types of refinement are operated:

- Order elevation
- Knot insertion
- k refinement

Order elevation can be thought of as p refinement in FEA. In FEA, increasing the number of nodes at the edges and within the element implies increasing the polynomial order of the element. When the Q4 elements of a FEA mesh are replaced by Q9 elements, the polynomial order is increased by one. In isogeometric analysis, order elevation involves increasing the multiplicity of each knot value by one. The geometry and the parametrization of the physical curve are not changed, however the number of basis functions and control points increases. The number of elements remains the same and the continuity across element boundaries remains the same since both inner and end knot values are increased by the same number, see Section 3.2.2.

The concept of order elevation with corresponding basis functions is illustrated in Figure 3.3. The starting knot vector $\Xi = \{0, 0, 1, 2, 2\}$ indicates a mesh of two elements, one spanning from $\xi = 0$ to $\xi = 1$ and the other from $\xi = 1$ to $\xi = 2$. Note that Figure 3.3a corresponds shape functions of linear elements of FEA (repeated once). The basis functions in Figure 3.3b have similarities to shape functions of quadratic elements of FEA, while the basis functions in Figure 3.3c are similar to shape functions of cubic elements of FEA (Q16). However, the basis functions are non-negative over the entire domain in contrast to the shape functions. Note also that the three set of basis functions are C^0 continuous over $\xi = 1$.



(a) $\Xi = \{0, 0, 1, 2, 2\}$ (b) $\Xi = \{0, 0, 0, 1, 1, 2, 2, 2\}$ (c) $\Xi = \{0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2\}$

Figure 3.3: Order elevation

Knot insertion has similarities to h refinement in FEA, which involves splitting the elements. In isogeometric analysis, like the name gives notice of, knot insertion involves adding new knot values between existing knots. This will simply render more elements, since the elements are bounded by knots of different values, as mentioned in Section 3.2.1. Like order elevation, neither the geometry nor the parametrization of the curve are changed during knot insertion, but the number of control points increases. A condition for this to be prevailing is that the new control points are chosen in a special manner.

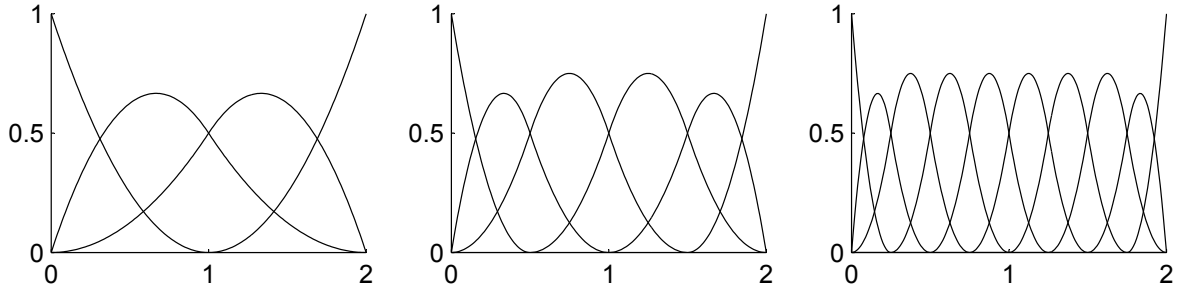
Let $\Xi = \{\xi_1, \xi_2, ..., \xi_{n+p+1}\}$ be a given knot vector. Inserting a new knot $\bar{\xi} \in [\xi_k, \xi_{k+1})$ with $k > p$ into the original knot vector generates one more basis function. However, all n old basis functions must be redefined using Eqs. (3.2) and (3.3), giving $m = n + 1$ new basis functions. The m new control points, $\left\{\bar{\mathbf{P}}_i\right\}_{i=1}^{m}$, are formed from the original control points, $\{\mathbf{P}_i\}_{i=1}^{n}$, by

$$\bar{\mathbf{P}}_i = \begin{cases} \mathbf{P}_1 & i = 1 \\ \alpha_i \mathbf{P}_i + (1 - \alpha_i)\mathbf{P}_{i-1} & 1 < i < m \\ \mathbf{P}_n & i = m \end{cases} \tag{3.12}$$

where

$$\alpha_i = \begin{cases} 1 & i \leq k - p \\ \frac{\bar{\xi} - \xi_i}{\xi_{i+p} - \xi_i} & k - p + 1 \leq i \leq k \\ 0 & i \geq k + 1 \end{cases} \tag{3.13}$$

The principle of knot insertion is illustrated in Figure 3.4, starting with the knot vector $\Xi = \{0, 0, 0, 1, 2, 2, 2\}$. In this example, the number of elements increases from two to eight.



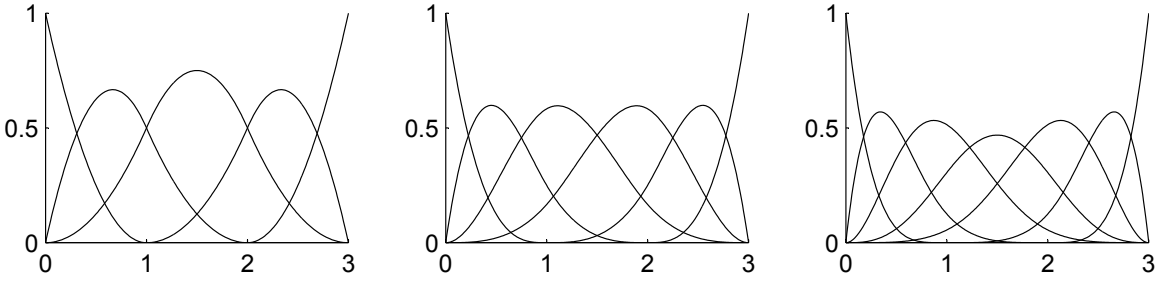
(a) $\Xi = \{0, 0, 0, 1, 2, 2, 2\}$ (b) $\Xi = \left\{0, 0, 0, \frac{1}{2}, 1, \frac{3}{2}, 2, 2, 2\right\}$ (c) $\Xi = \left\{0, 0, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, \frac{5}{4}, \frac{3}{2}, \frac{7}{4}, 2, 2, 2\right\}$

Figure 3.4: Knot insertion

k refinement has no analogous to refinements in FEA and is a combination of both order elevation and knot insertion. The idea is to increase both the order of the curve and also the continuity across knot values (element boundaries). This is done by increasing just the multiplicity of the first and last knot values. The continuity across knot values is then increased by the same number as the number of multiplicities in order elevation (C^{p-1} continuity).

This method can also be presented as starting with a knot vector with only first and last values, do order elevation as much as desired, and finally insert knots between first and last values. Hence combination of order elevation and knot insertion.

The concept of k refinement is shown in Figure 3.5, with $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$ as the chosen knot vector to start with. The number of elements (which is three) remains the same. Note the increased continuity across the element boundaries, at $\xi = 1$ and $\xi = 2$ in Figures 3.5b and 3.5c. This may be seen as increased number of basis functions with value different from zero over the boundaries. The last knot vector could also have been obtained by steps of order elevation and thereafter knot insertion, i.e. $\Xi = \{0, 0, 0, 3, 3, 3\} \rightarrow \Xi = \{0, 0, 0, 0, 3, 3, 3, 3\} \rightarrow \Xi = \{0, 0, 0, 0, 0, 3, 3, 3, 3, 3\} \rightarrow \Xi = \{0, 0, 0, 0, 0, 1, 2, 3, 3, 3, 3, 3\}$.



(a) $\Xi = \{0, 0, 0, 1, 2, 3, 3, 3\}$ (b) $\Xi = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$ (c) $\Xi = \{0, 0, 0, 0, 0, 1, 2, 3, 3, 3, 3, 3\}$

Figure 3.5: k refinement

To sum up, knot insertion is merely the technique to refine the element mesh in isogeometric analysis, while order elevation and k refinement are methods to increase the degree of the elements. These two latter methods of mesh refinement may be an unusual way of thinking in classical FEA, since p refinement in FEA is usually limited to lower order elements. However, increasing the order of the elements has a fairly widespread range of application in isogeometric analysis, because of the convenient use. The main advantage of k refinement compared to order elevation is that the method generates fewer number of basis functions (for the same starting knot vector). This gives lower computational costs, while nevertheless giving better properties concerning continuity.

Subsequently, in the development of computational formulations for isogeometric analysis, refinement of the element mesh is carried out by knot insertion, using either two (polynomial order 1), three (polynomial order 2), four (polynomial order 3) or five (polynomial order 4) repeated first and last knot values. This is also defined as three steps of k refinement. The basis functions will typically look like Figure 3.3a for polynomial order 1, Figure 3.4c for polynomial order 2, Figure 3.5b for polynomial order 3 and Figure 3.5c for polynomial order 4.

3.3 Non-Uniform Rational B-Splines (NURBS)

B-splines are non-rational, forming non-rational B-spline curves and surfaces. A rational curve or surface can represent conical sections in an exact manner. Non-uniform rational B-splines (NURBS) are therefore introduced by including weights to control points. The NURBS basis functions will differ from the B-spline basis functions, but knot vectors, the tensor product nature and refinement mechanisms are unchanged.

3.3.1 The Geometric Perspective

A NURBS entity in $\mathbb{R}^d$ is obtained by projecting a B-spline entity in $\mathbb{R}^{d+1}$, where d is the number of physical dimensions. Figure 3.6 illustrates how a semicircle $\mathbf{C}(\xi)$ in $\mathbb{R}^2$ is constructed by the projective transformation of a quadratic B-spline curve in $\mathbb{R}^3$, called the "projective curve" $\mathbf{C}^w(\xi)$. The control points $\mathbf{P}_i$ are given by

$$(\mathbf{P}_i)_j = \frac{(\mathbf{P}_i^w)_j}{w_i} \qquad j = 1, ..., d \tag{3.14}$$

where

$$w_i = (\mathbf{P}_i^w)_{d+1} \tag{3.15}$$

and $\mathbf{P}_i^w$ are the "projective control points". The weights w_i may geometrically be pictured as the height of the "projective control points".

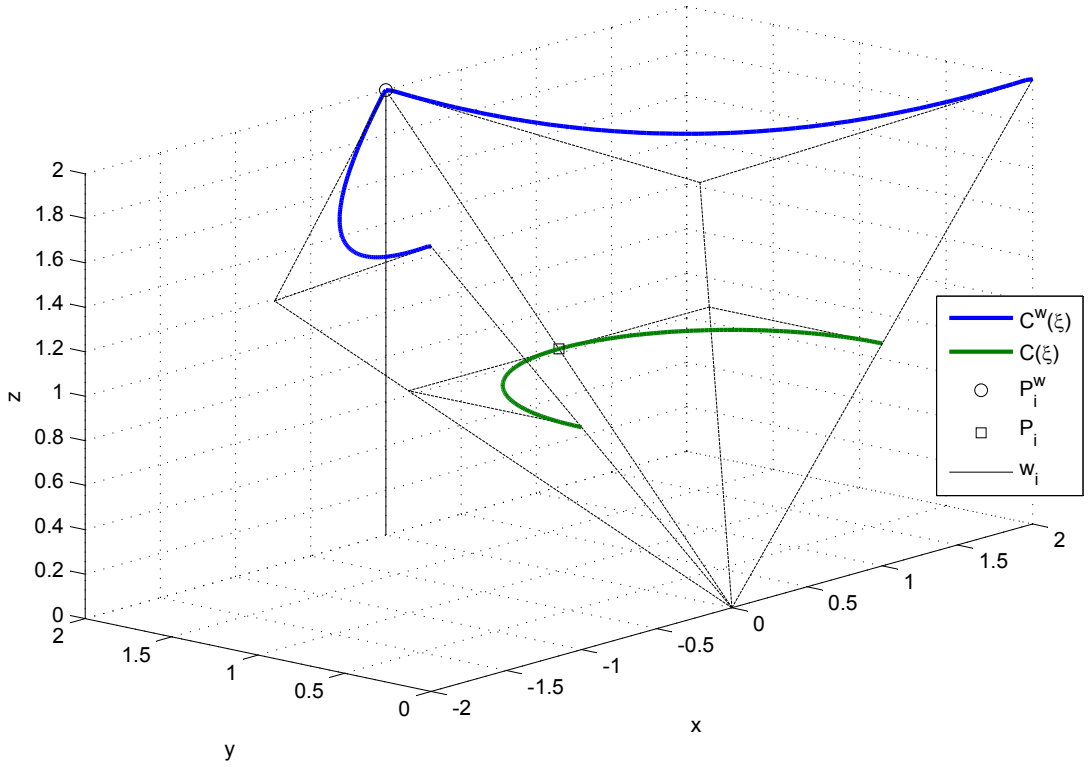


Figure 3.6: B-spline curve projected onto the plane $z = 1$ to create a NURBS semicircle

Geometrically, knot insertion to a NURBS entity is done by projecting the NURBS control points into $\mathbb{R}^{d+1}$, apply Eqs. (3.12) and (3.13) to the B-spline control points, and finally project back into $\mathbb{R}^d$ to obtain the new NURBS control points.

3.3.2 The Algebraic Perspective

The NURBS basis function for a NURBS curve is defined as

$$R_i^p(\xi) = \frac{N_{i,p}(\xi)w_i}{W(\xi)} = \frac{N_{i,p}(\xi)w_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i},p}(\xi)w_{\hat{i}}} \tag{3.16}$$

where $W(\xi) = \sum_{\hat{i}=1}^{n} N_{\hat{i},p}(\xi)w_{\hat{i}}$ is the weighting function and $N_{i,p}(\xi)$ is the previously introduced B-spline basis function.

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

$$\mathbf{W} = \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_n \end{bmatrix} \tag{3.17}$$

and let $\mathbf{N}(\xi)$ be a column vector of B-spline basis functions, Eq. (3.16) is rewritten in matrix form,

$$\mathbf{R}(\xi) = \frac{1}{W(\xi)}\mathbf{WN}(\xi) \tag{3.18}$$

The basis function for a NURBS surface is given by

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_{i,p}(\xi)M_{j,q}(\eta)w_{i,j}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m}N_{\hat{i},p}(\xi)M_{\hat{j},q}(\eta)w_{\hat{i},\hat{j}}} = \frac{1}{W(\xi,\eta)}\mathbf{WN}(\xi,\eta) \tag{3.19}$$

The NURBS basis functions display the same properties as the B-spline basis functions, see Section 3.2.2.

The first derivative of the NURBS basis function is found by applying the quotient rule to Eq. (3.16),

$$\frac{d}{d\xi}R_i^p(\xi) = w_i\frac{W(\xi)N'_{i,p}(\xi) - W'(\xi)N_{i,p}(\xi)}{W^2(\xi)} \tag{3.20}$$

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

When the NURBS basis functions are determined, a NURBS curve is found in the similar way as for its B-spline counterpart (Eq. (3.8)),

$$\mathbf{C}(\xi) = \sum_{i=1}^{n}R_i^p(\xi)\mathbf{P}_i \tag{3.21}$$

The NURBS surface is given by

$$\mathbf{S}(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m}R_{i,j}^{p,q}(\xi,\eta)\mathbf{P}_{i,j} \tag{3.22}$$

3.3.3 Constructing Curve from Basis Functions - An Example

The aim is to draw a semicircle using the four quadratic NURBS basis functions from Section 3.2.3. A semicircle can be constructed using the four control points and weights [19]

$$\mathbf{P} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1/2 \\ 1/2 \\ 1 \end{bmatrix} \tag{3.23}$$

Say the NURBS basis functions and curve is to be evaluated at the point $\xi = 3/2$. Using Eqs. (3.7a) - (3.7d), the B-spline basis functions become

$$\begin{aligned} &N_{1,2}(\xi = \tfrac{3}{2}) = 0 \\ &N_{2,2}(\xi = \tfrac{3}{2}) = \tfrac{1}{2}(2 - \tfrac{3}{2})^2 = \tfrac{1}{8} \\ &N_{3,2}(\xi = \tfrac{3}{2}) = \tfrac{1}{2}\cdot\tfrac{3}{2}(2 - \tfrac{3}{2}) + (2 - \tfrac{3}{2})(\tfrac{3}{2} - 1) = \tfrac{5}{8} \\ &N_{4,2}(\xi = \tfrac{3}{2}) = (\tfrac{3}{2} - 1)^2 = \tfrac{1}{4} \end{aligned} \tag{3.24}$$

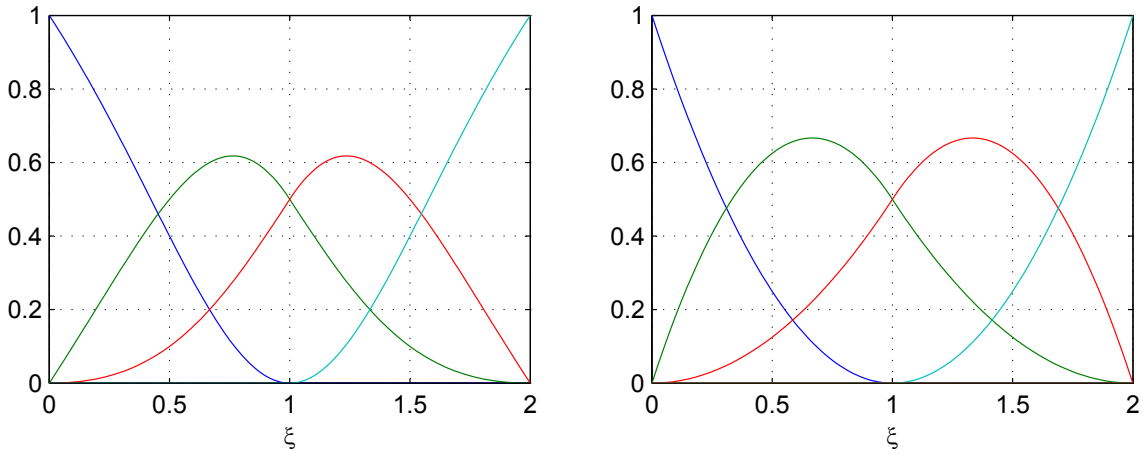
Eq. (3.16) gives the weighting function. The value of the weighting function at $\xi = 3/2$ is

$$W(\xi = \frac{3}{2}) = \sum_{i=1}^{4} N_{i,2}(\xi = \frac{3}{2}) w_i = 0 \cdot 1 + \frac{1}{8} \cdot \frac{1}{2} + \frac{5}{8} \cdot \frac{1}{2} + \frac{1}{4} \cdot 1 = \frac{5}{8} \tag{3.25}$$

The NURBS basis functions are then obtained for $\xi = 3/2$,

$$\begin{aligned}
R_1^2(\xi = \tfrac{3}{2}) &= \frac{N_{1,2}(\xi=3/2) \cdot w_1}{W(\xi=3/2)} = \frac{0 \cdot 1}{5/8} = 0 \\
R_2^2(\xi = \tfrac{3}{2}) &= \frac{N_{2,2}(\xi=3/2) \cdot w_2}{W(\xi=3/2)} = \frac{1/8 \cdot 1/2}{5/8} = \frac{1}{10} \\
R_3^2(\xi = \tfrac{3}{2}) &= \frac{N_{3,2}(\xi=3/2) \cdot w_3}{W(\xi=3/2)} = \frac{5/8 \cdot 1/2}{5/8} = \frac{1}{2} \\
R_4^2(\xi = \tfrac{3}{2}) &= \frac{N_{4,2}(\xi=3/2) \cdot w_4}{W(\xi=3/2)} = \frac{1/4 \cdot 1}{5/8} = \frac{2}{5}
\end{aligned} \tag{3.26}$$

The resulting NURBS basis functions (when evaluated at many ξ values) are shown in Figure 3.7a. To illustrate the difference, the B-spline basis functions are again pictured in Figure 3.7b. Note how the weights w_2 and w_3 affect not only the basis functions R_2^2 and R_3^2 , but also R_1^2 and R_4^2.



(a) NURBS basis functions, $w_1 = w_4 = 1$ and $w_2 = w_3 = 1/2$

(b) B-spline basis functions

Figure 3.7: NURBS and B-spline basis functions from $\Xi = \{0, 0, 0, 1, 2, 2, 2\}$

The NURBS curve is then given by x and y coordinates in pairs,

$$\begin{aligned}
\mathbf{C}(\xi = \tfrac{3}{2}) &= \begin{bmatrix} x \\ y \end{bmatrix} (\xi = \tfrac{3}{2}) = \textstyle\sum_{i=1}^{n} R_i^2(\xi = \frac{3}{2}) \begin{bmatrix} x_i \\ y_i \end{bmatrix} \\
&= \begin{bmatrix} 0 \cdot (-1) + \frac{1}{10} \cdot (-1) + \frac{1}{2} \cdot 1 + \frac{2}{5} \cdot 1 \\ 0 \cdot 0 + \frac{1}{10} \cdot 1 + \frac{1}{2} \cdot 1 + \frac{2}{5} \cdot 0 \end{bmatrix} = \begin{bmatrix} \frac{4}{5} \\ \frac{3}{5} \end{bmatrix}
\end{aligned} \tag{3.27}$$

For comparison, the B-spline curve at the same parametric coordinate may also be evaluated,

$$\begin{aligned} \mathbf{C}(\xi = \tfrac{3}{2}) &= \begin{bmatrix} x \\ y \end{bmatrix} (\xi = \tfrac{3}{2}) = \textstyle\sum_{i=1}^{n} N_{i,2}(\xi = \tfrac{3}{2}) \begin{bmatrix} x_i \\ y_i \end{bmatrix} \\ &= \begin{bmatrix} 0 \cdot (-1) + \frac{1}{8} \cdot (-1) + \frac{5}{8} \cdot 1 + \frac{1}{4} \cdot 1 \\ 0 \cdot 0 + \frac{1}{8} \cdot 1 + \frac{5}{8} \cdot 1 + \frac{1}{4} \cdot 0 \end{bmatrix} = \begin{bmatrix} \frac{3}{4} \\ \frac{3}{4} \end{bmatrix} \end{aligned} \tag{3.28}$$

The resulting curves when evaluated at several ξ values are illustrated in Figure 3.8. Since B-splines do not contain weights (all weights are equal to 1), the proportion of the control points $(-1, 1)$ and $(1, 1)$ will become too large, making the arc square shaped.

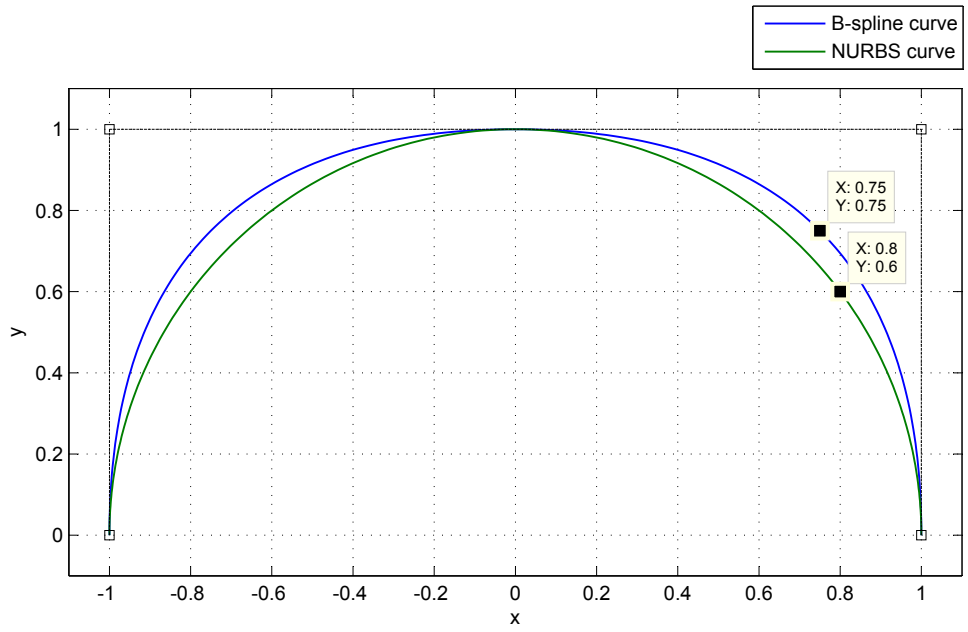


Figure 3.8: Semicircle built by NURBS basis and 180° arc built by B-spline basis

Note that this semicircle is constructed using four control points and C^1 continuity over the vertex $(0, 1)$. It is more common to use the knot vector $\Xi = \{0, 0, 0, 1, 1, 2, 2, 2\}$ to construct a semicircle, which gives C^0 continuity over the vertex and five basis functions and control points. The semicircle in Figure 3.6 is an example of the latter procedure, and the C^0 continuity may be seen on its B-spline counterpart.

3.4 T-Splines

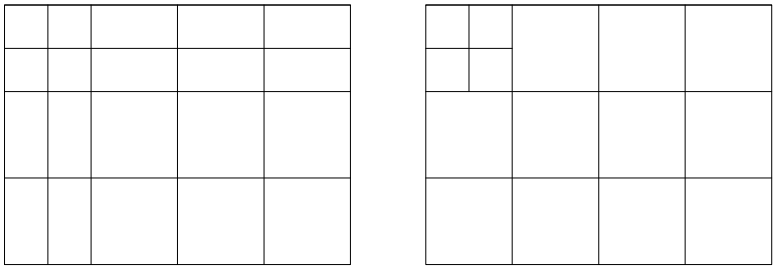
3.4.1 Overview

In 2003, Sederberg et al. [24] introduced T-splines as a generalization of NURBS technology, because of the restrictions associated with the tensor product of NURBS. Local refinement had been investigated in the CAD community using hierarchical B-splines involving a multilevel spline space among others [24]. However, truly local refinement was not established before the introduction of T-splines.

Sederberg et al. [24] starts with *point-based* splines (PB-splines) to define T-splines. The PB-spline is a surface whose control points has no topological relationship with each other, and is

therefore the collection of *blending functions*. Each blending function is a B-spline basis function generated from a set of *local* non-decreasing knot vectors. The expression *blending* is due to linear dependence of the functions. T-splines adapt these local knot vectors, but as opposed to the point based grid, a control grid similarly to NURBS is chosen.

While NURBS control points lie in a rectangular grid, rows and columns of T-spline control points may be incomplete as illustrated in Figure 3.9b, forming *T-junctions* in the *T-mesh*. NURBS are therefore a restricted subset of T-splines. By overcoming the tensor product restriction, T-splines allow local refinement. Note that T-splines imply T-spline surfaces, since there must be (at least) two dimensions for a T-junction to exist.

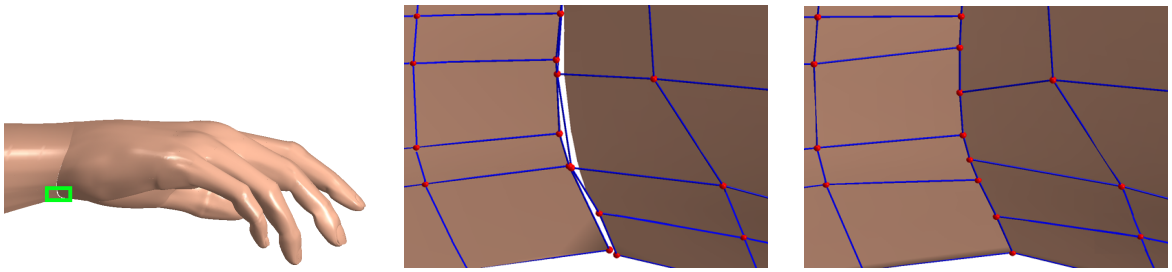


(a) Global refinement of a NURBS mesh

(b) Local refinement of a T-mesh

Figure 3.9: Global and local refinement of NURBS and T-splines

Local refinement has many benefits. For the same geometric representation, T-splines give fewer control points compared to NURBS, implying lower computational cost when performing analyses. An analysis cannot be performed on a model containing gaps, which are often non-avoidable in a NURBS model, because closing a gap requires refinement of the whole model. The refinement process increases the number of control points drastically and is therefore usually not performed. In contrast to this, using T-spline control net, the gaps may be locally refined, keeping the number of control points low while still giving an analysis-suitable model. Local refinement of gaps is often referred to as T-spline merging and an example to this is shown in Figure 3.10.



(a) Hand modelled with NURBS surfaces

(b) NURBS control grids of highlighted region

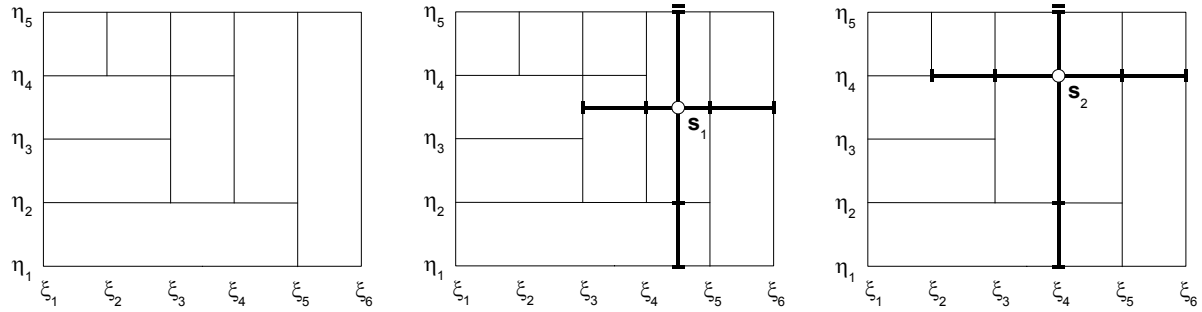
(c) T-spline control grid

Figure 3.10: Gap closed using T-spline merging [24]

Since NURBS are a special case of T-splines, the two bases are compatible with each other. Every NURBS is a T-spline and every T-spline may be converted into one or more NURBS surfaces by performing local refinement to eliminate all T-junctions. This is usually not of interest; however, compatibility is of importance for T-spline adoption in a commercial view.

3.4.2 T-Mesh and Local Knot Vectors

The T-mesh is often illustrated as a grid between the control points, but for topological purposes let us define the T-mesh in the *index space*. Like a NURBS index space, each knot line represents a knot value, but a T-mesh allows vertices connecting at three edges, forming T-junctions. In FEA, this corresponds "hanging nodes". Figure 3.11a illustrates a simple T-mesh.



(a) T-mesh (b) Anchor, $p = 2$ (c) Anchor, $p = 3$

Figure 3.11: T-mesh, anchors of even and odd polynomial degrees

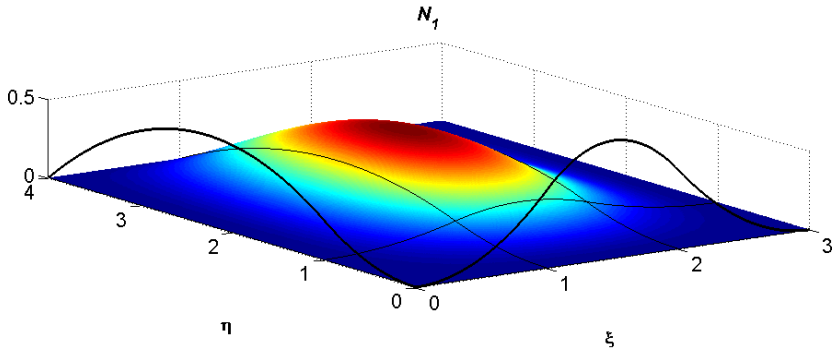
A valid T-mesh defines a T-spline basis function to each anchor and corresponding control point. Since a uniform index space is assumed, the anchors are the mid-points of the *T-mesh elements* if the polynomial order is even and coincide with the T-mesh vertices for odd polynomial degrees, cf. anchors in Section 3.2.6. T-mesh elements are rectangles in the T-mesh. To obtain a valid T-mesh, *local knot vectors* must be defined for each anchor.

Consider first the example in Figure 3.11b where the polynomial order $p = 2$. The example makes use of a T-mesh in Bazilevs et al. [1]. The local knot vectors are found by marching horizontally and vertically from the anchor $\mathbf{s}_i$ until $p/2 + 1$ 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 $\mathbf{s}_1 = ((\xi_4+\xi_5)/2, (\eta_3+\eta_4)/2)$ are therefore $\Xi_1 = \{\xi_3, \xi_4, \xi_5, \xi_6\}$ and $\mathcal{H}_1 = \{\eta_1, \eta_2, \eta_5, \eta_5\}$. Note that the length of the local knot vector is $(p + 2)$.

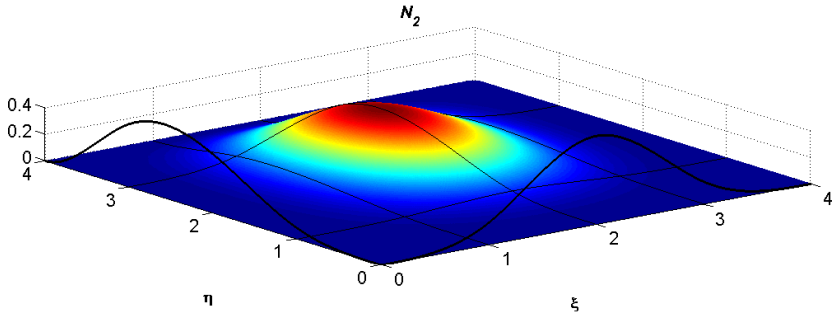
If the polynomial order is odd, the anchor's placing will coincide with the T-mesh vertices and therefore the control points, see Figure 3.11c where the polynomial order $p = 3$. The length of knot vector is still $(p + 2)$, i.e. for $p = 3$ $|\Xi_i| = |\mathrm{H}_i| = 5$. The local knot vectors are now found by marching $(p+1)/2$ in each direction. Hence, the local knot vectors to $\mathbf{s}_2 = (\xi_4, \eta_4)$ are $\Xi_2 = \{\xi_2, \xi_3, \xi_4, \xi_5, \xi_6\}$ and $\mathcal{H}_2 = \{\eta_1, \eta_2, \eta_4, \eta_5, \eta_5\}$. Note that the anchor $\mathbf{s}_2$ is also a control point.

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, $\Delta\Xi = \{\Delta\xi_1, \Delta\xi_2, \ldots, \Delta\xi_{p+1}\}$, such that $\Delta\Xi = \xi_{i+1} - \xi_i$. The *local basis function domain* may then always be placed at the origin: $\hat{\Omega}_A = [0, \Delta\xi_1 + \Delta\xi_2 + \ldots + \Delta\xi_{p+1}] \times [0, \Delta\eta_1 + \Delta\eta_2 + \ldots + \Delta\eta_{p+1}]$, $A = 1, 2, ...n$, where n is the number of control points. Over each local basis function domain, the T-spline basis function in the parameter space is found similarly to a NURBS basis function, Eq. (3.19).

For the example above, if $\xi_1 = 1, \xi_2 = 2, \ldots, \xi_6 = 6$ and $\eta_1 = 1, \eta_2 = 2, \ldots, \eta_5 = 5$, the local knot interval vectors to $\mathbf{s}_1$ are $\Delta\Xi_1 = \{1, 1, 1\}$ and $\Delta\mathcal{H}_1 = \{1, 3, 0\}$ and to $\mathbf{s}_2$ $\Delta\Xi_2 = \{1, 1, 1, 1\}$ and $\Delta\mathcal{H}_2 = \{1, 2, 1, 0\}$. Note that this knot value configuration is fictitious as the index space will not be interpolatory at the boundaries. The resulting T-spline basis functions N_1 and N_2 (when all weights are equal to 1) will be as illustrated in Figure 3.12a for $\mathbf{s}_1$ and as shown in Figure 3.12b for $\mathbf{s}_2$.



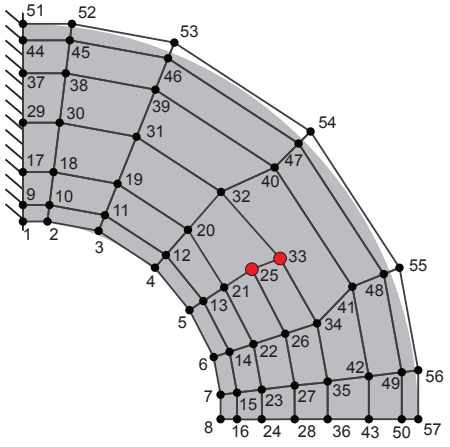
(a) T-spline basis function N_1 $(p = 2)$



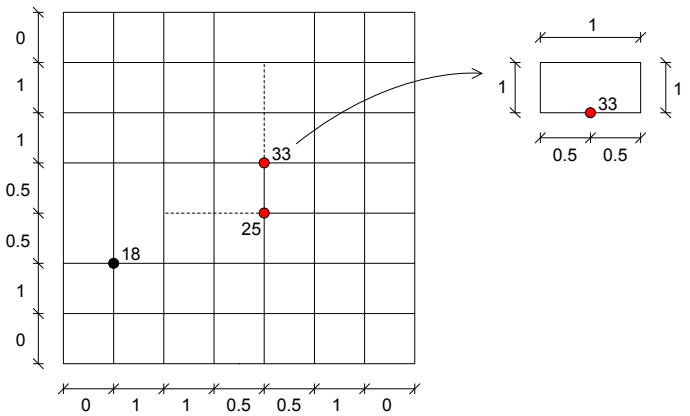
(b) T-spline basis function N_2 $(p = 3)$

Figure 3.12: T-spline basis functions over local domains

The notation "anchor" is adapted from Bazilevs et al. [1] due to presenting both odd and even polynomial orders. Arbitrary degree T-splines are also reviewed in Finnigan [10]. Many papers (including Sederberg et al. [24, 23], Dörfel et al. [9], Li et al. [15], Scott et al. [21, 20], Li and Scott [14]) on T-spline studies for odd polynomial degrees consider the T-mesh mainly as a control grid since it equals the T-mesh vertices in the index space (the anchors), see Figure 3.13a. In that case, edges in the control grid equal knot intervals. A *knot interval configuration* may therefore be assigned directly to the control grid without taking the detour to the local knots vectors. To obtain a valid knot interval configuration, it is required that the knot intervals on opposite sides of every T-mesh elements sum to the same value, see Figure 3.13b.



(a) Control grid in the physical space [21]



(b) The knot interval configuration drawn in the index space with a selection of control points

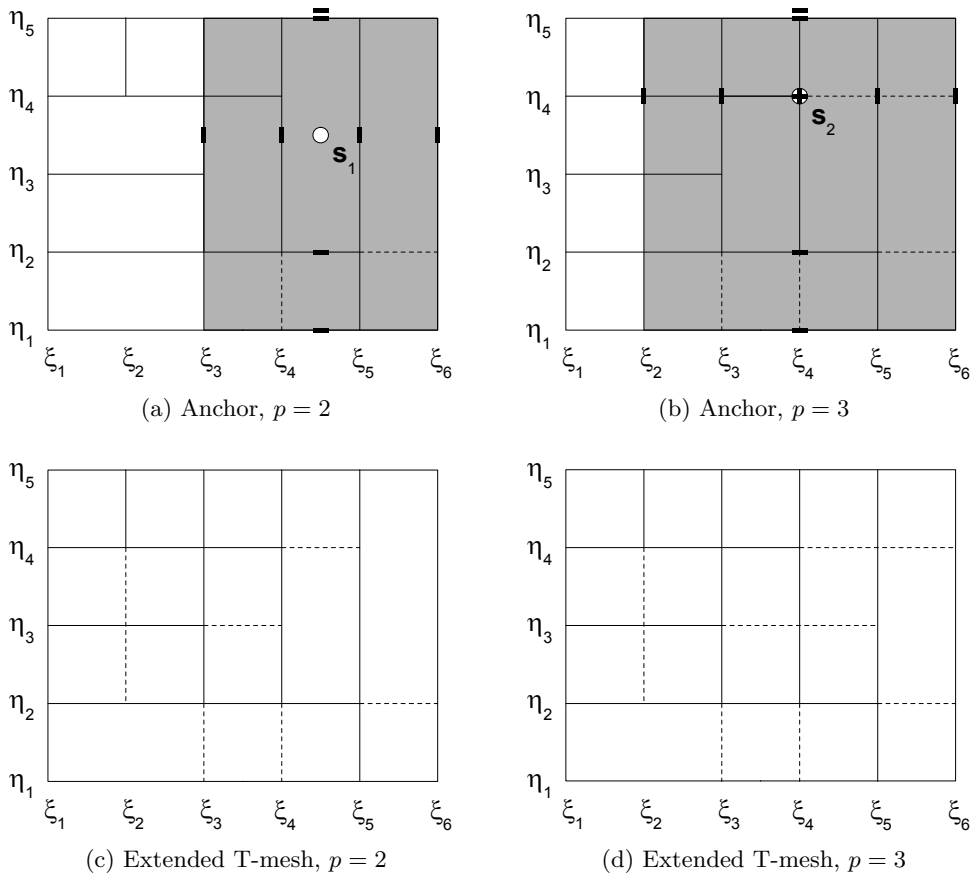
Figure 3.13: The T-mesh as a control grid

3.4.3 The Extended T-Mesh

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 extend the knot lines from the T-junctions. These lines are typically marked as dashed 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, the anchors and local knot vectors from Section 3.4.2 are useful. Again, consider the T-mesh in Figure 3.11a. For $p = 2$ and the anchor $\mathbf{s}_1$, the support of the corresponding T-spline basis function N_1 will be the region which the local knot vectors span, see Figure 3.14a. Note how the support here compares to the support of a NURBS basis function in Section 3.2.5. Reduced continuity for the local basis function N_1 will logically enough appear at all local knot values for that basis function. If there is not already a knot line in this grid of local knot values, the existing knot lines must be extended in the support area. Repeating this for all anchors of the T-mesh, the extended T-mesh is obtained, as shown in Figure 3.14c.

The same procedure may be approached to find the extended mesh for $p = 3$ for the same T-mesh, but now the anchors are at the vertices. The resulting extended T-mesh illustrated in Figure 3.14d will differ slightly from the extended T-mesh for $p = 2$, Figure 3.14c. However, the extended T-mesh for odd polynomial degrees may be found using an easier method. At each T-junction, simply extend the line until $(p+1)/2$ orthogonal edges are encountered.



(a) Anchor, $p = 2$

(b) Anchor, $p = 3$

(c) Extended T-mesh, $p = 2$

(d) Extended T-mesh, $p = 3$

Figure 3.14: Lines of reduced continuity

3.4.4 Analysis-Suitable T-Splines

Defining a basis function to each anchor does not make the T-mesh *analysis-suitable* without further inspections. The notation "basis function" for a T-spline instead of "blending function" is neither an obviousness. Li et al. [15] investigated the class of T-splines for which no perpendicular *T-junction extensions* intersect (intersect here includes touching). T-junction extensions are almost like lines of reduced continuity for odd polynomial orders. In addition to the line of reduced continuity, which is the *face extension* part of the T-junction extension, the T-junction extension is also composed of a possible *edge extension*, which is the line in the opposite direction to the first-encountered T-mesh vertex or orthogonal edge.

Figure 3.15 shows the T-junction extensions for the T-mesh example from Sections 3.4.2 and 3.4.3. The face extensions terminate in an open arrow, while the edge extensions terminate in a closed arrow. Since there are several T-junction extensions that touch, the T-mesh is not analysis-suitable in the way Li et al. [15] define it.

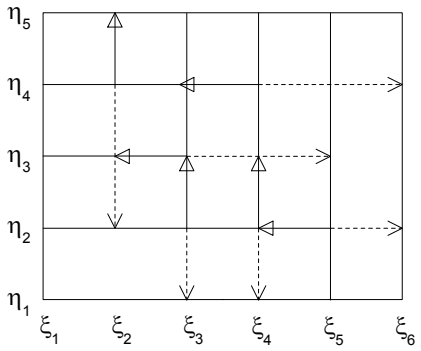
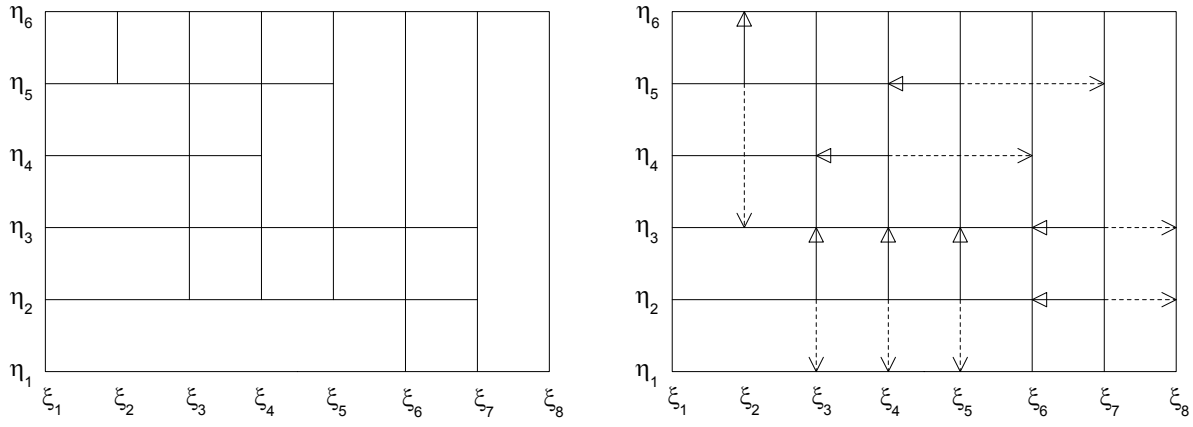


Figure 3.15: T-junction extensions, $p = 3$

Figure 3.16 shows an analysis-suitable T-mesh and its T-junction extensions.



(a) Analysis-suitable T-mesh

(b) T-junction extensions, $p = 3$

Figure 3.16: Analysis-suitable T-mesh

Analysis-suitable T-splines form a mildly restricted set of T-splines, but guarantees linear independence of the T-splines for all choices of knots [15]. When a T-spline is linearly independent, not till then is its blending functions a set of *basis* functions, indicating that the functions may be used for analysis as linear dependence leads to singular matrices. Linear independence of

T-splines is complex, since it is a function of both the T-mesh topology and the knot values. In Buffa et al. [4], examples of linearly dependent T-splines with multiple knots are described.

Note however that linearly dependent T-splines are very rare; a necessary and sufficient condition for a T-spline to be linearly independent is that the so-called T-spline-to-NURBS transform matrix M is full rank [15]. The vast majority of T-splines are linearly independent [15], therefore the T-spline basis functions of the T-mesh presented in Sections 3.4.2 and 3.4.3 can be referred to as "basis functions" directly.

When the T-spline forms a set of basis functions, it constitutes all the properties of B-splines, see Section 3.2.2. In addition, affine covariance of an analysis-suitable T-spline implies that all "patch tests" are satisfied *a priori* [20]. Scott et al. [20] newly showed that analysis-suitable T-splines have another benefit. They may be locally refined without producing excessive propagation of control points. Since T-splines do not have a regular grid as NURBS, refinement algorithms can be a challenge. Inserting a new knot requires others to also be inserted to maintain the T-mesh valid, but simultaneously the algorithm must prevent the local refinement from propagating over the entire mesh.

The first local knot insertion rules suggested by Sederberg et al. [24] were not particularly elegant. Shortly after this, Sederberg et al. [23] introduced an algorithm based on three violations which proved to be much more efficient. Later Dörfel et al. [9] made use of these rules in an adaptive refinement study. Although it proved to give good results for most occurrences, some special cases lead to an extensive refinement. This shortcoming has not been observed using analysis-suitable T-splines [20]. Recently, Li and Scott [14] established the nesting behaviour of analysis-suitable T-spline spaces, which provides a theoretical foundation for the local refinement algorithm presented in Scott et al. [20].

Analysis-suitable T-splines may therefore be more promising for isogeometric analysis than general T-splines, even though they seem to be a bit restricted at first sight. The initial T-mesh may contain more control points than a general T-spline, but a stable refinement assures that the refined mesh will show to advantage.

Refinement algorithms for T-splines are not considered in this thesis.

Note that also L-junctions, I-junctions and isolated nodes exist for the T-mesh. However, Li et al. [15] also proved that only T-junctions are permitted for T-splines to be analysis-suitable.

Chapter 4

Bézier Extraction of NURBS and T-Splines

B-splines and NURBS, as presented in Chapter 3, span a parameter space which consists of several elements. This global structure complicates implementation in a traditional finite element context.

This chapter presents the Bézier extraction operator for NURBS and T-splines, a tool which decomposes a set of NURBS or T-spline basis functions to the Bernstein polynomials. This will allow for generation of C^0 continuous Bézier elements, giving a local representation of the basis functions. An element structure for isogeometric analysis similarly to FEA is provided, which eases the implementation of isogeometric analysis in a finite element setting.

The formulas in Section 4.1 are taken from Borden et al. [3], while the theory in Section 4.2 is extracted from Scott et al. [21].

4.1 Bézier Extraction of NURBS

This section presents how NURBS can be constructed using Bézier elements and the Bézier extraction operator.

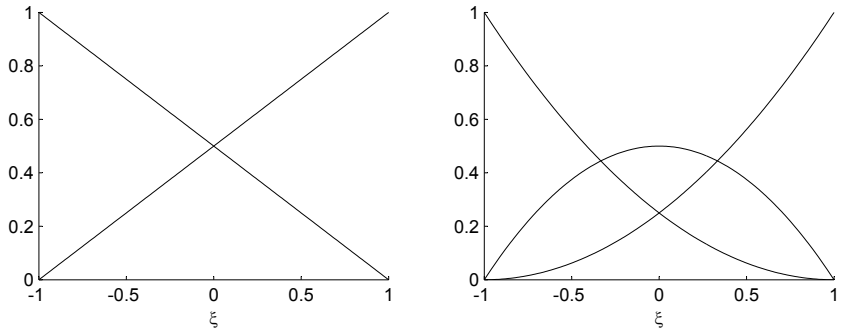
4.1.1 Bézier Elements and Bernstein Polynomials

A Bézier element spans traditionally $[0, 1]$ (in each parametric dimension) and is formed by a knot vector with no inner knot values; i.e. the knot vector contains $(p + 1)$ zeros and $(p + 1)$ ones, where p is the polynomial order. The basis functions formed by this knot vector are called the Bernstein polynomials and have many similarities to the Lagrange polynomials. Here, the Bernstein polynomials are defined over the interval $[-1, 1]$ such that the Bézier element spans the same interval as the quadrilateral elements of FEA. The Bernstein polynomials for polynomial order 1 to 4 are pictured in Figure 4.1.

The Bernstein polynomials can be formed using the B-spline definitions in Eqs. (3.2) and (3.3). However, since the knot values are restricted to -1 and 1, each repeated $(p + 1)$ times, the Bernstein polynomials can be more compactly defined as

$$B_{i,p}(\xi) = \frac{1}{2}(1 - \xi)B_{i,p-1}(\xi) + \frac{1}{2}(1 + \xi)B_{i-1,p-1}(\xi) \tag{4.1}$$

where



(a) $p = 1$ (b) $p = 2$

(c) $p = 3$ (d) $p = 4$

Figure 4.1: The Bernstein polynomials

$$B_{1,0}(\xi) \equiv 1 \qquad \text{and} \qquad B_{i,p}(\xi) \equiv 0 \quad \text{if} \quad i < 1 \quad \text{or} \quad i > p+1 \tag{4.2}$$

These Bernstein polynomials, like B-splines, constitute the partition of unity and are non-negative over the entire domain. The Bernstein polynomials are also symmetric and interpolatory at the endpoints, similar to the Lagrange polynomials.

The i^{th} derivative is

$$\frac{d}{d\xi}B_{i,p}(\xi) = \frac{p}{2}\left\{B_{i-1,p-1}(\xi) - B_{i,p-1}(\xi)\right\} \tag{4.3}$$

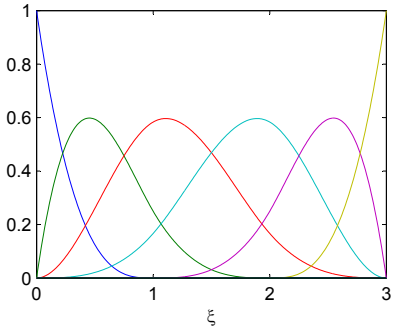
A Bézier curve is a linear combination of Bernstein polynomials and control points,

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

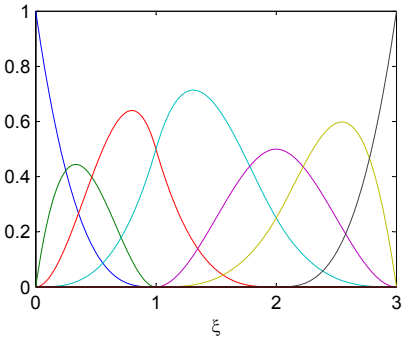
4.1.2 Bézier Decomposition and the Bézier Extraction Operator

The Bézier extraction operator maps linear combinations of Bernstein polynomials onto a NURBS basis. This transformation makes it possible to use piecewise C^0 Bézier elements as the finite element representation in isogeometric analysis. To decompose a set of NURBS basis functions to its Bézier elements, called Bézier decomposition, all interior knots of a knot vector are repeated until they have a multiplicity equal to p. Theoretically, the interior knots should have multiplicity of $(p+1)$ to form truly separated Bézier elements. However, a multiplicity of p is sufficient to represent the Bernstein polynomials, which in this context are also referred to as Bézier basis functions.

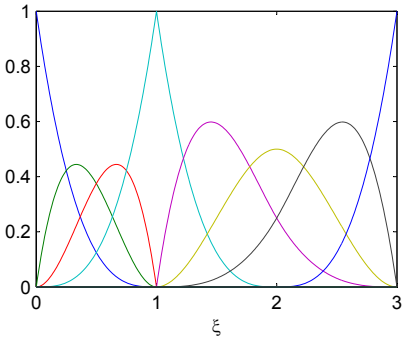
An example illustrates Bézier decomposition well. Starting with the knot vector $\Xi = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$, the set of B-splines (or NURBS) basis functions is decomposed to its Bézier elements by inserting the knots $\{1, 1, 2, 2\}$ after turn. The number of basis functions increases from 6 to 10. The knot insertion process is shown in Figure 4.2.



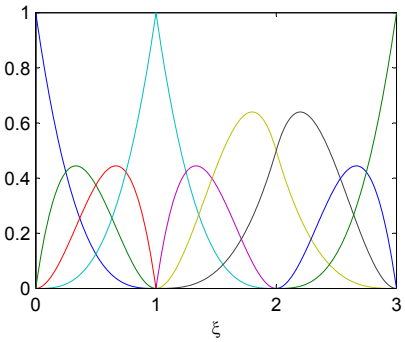
(a) $\Xi = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$



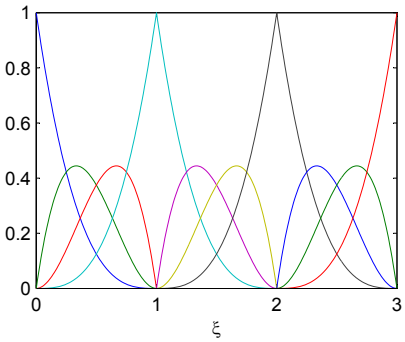
(b) $\Xi = \{0, 0, 0, 0, 1, 1, 2, 3, 3, 3, 3\}$



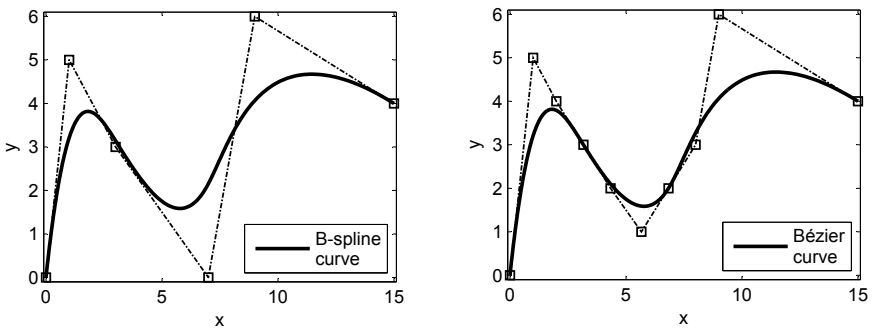
(c) $\Xi = \{0, 0, 0, 0, 1, 1, 1, 2, 3, 3, 3, 3\}$



(d) $\Xi = \{0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 3, 3\}$



(e) $\Xi = \{0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3\}$



(f) B-spline curve drawn from the basis functions in Figure 4.2a and the equal Bézier curve drawn from the basis functions in Figure 4.2e.

Figure 4.2: Bézier decomposition of $\Xi = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$

Figure 4.2f shows that the curves drawn from the B-spline basis functions and the Bézier basis functions are geometrically equal, but the number of control points increases from 6 to 10 like the number of basis functions.

Since the Bézier decomposition is a knot insertion operation, the Bézier extraction operator is based on the formulas for new control points formed from original control points when a knot is inserted, Eqs. (3.12) and (3.13). Let $\left\{\bar{\xi}_1, \bar{\xi}_2, \ldots, \bar{\xi}_j, \ldots, \bar{\xi}_m\right\}$ be the set of knots required to produce the Bézier decomposition of a B-spline. Define α_i^j, $i = 1, 2, \ldots, n+j$, to be the i^{th} α to the j^{th} knot inserted (α_i as defined in Eq. (3.13)). Then defining

$$\mathbf{C}^j = \begin{bmatrix} \alpha_1 & 1-\alpha_2 & 0 & \ldots & & 0 \\ 0 & \alpha_2 & 1-\alpha_3 & 0 & \ldots & 0 \\ \vdots & & & \ddots & & \\ 0 & \ldots & & & \alpha_{n+j-1} & 1-\alpha_{n+j} \end{bmatrix} \tag{4.5}$$

Eq. (3.12) can be rewritten in matrix form to represent the sequence of control variables formed from the knot insertion process,

$$\bar{\mathbf{P}}^{j+1} = (\mathbf{C}^j)^T \bar{\mathbf{P}}^j \qquad \text{where} \qquad \bar{\mathbf{P}}^1 = \mathbf{P} \tag{4.6}$$

The final set of control points $\bar{\mathbf{P}}^{m+1} = \mathbf{P}^b$. Defining

$$\mathbf{C}^T = (\mathbf{C}^m)^T (\mathbf{C}^{m-1})^T ... (\mathbf{C}^1)^T \tag{4.7}$$

The relation between the new Bézier control points and the original B-splines control points is obtained,

$$\mathbf{P}^b = \mathbf{C}^T \mathbf{P} \tag{4.8}$$

Note that for a two-dimensional physical space, the dimension of $\mathbf{P}$ is $n \times 2$, while $\mathbf{P}^b$ has dimension $(n+m) \times 2$, making the dimension of $\mathbf{C}$ $n \times (n+m)$. Here, n is the number of basis functions or control points before Bézier decomposition and m is the number of knots inserted.

Writing the B-spline curve representation, $\mathbf{C}(\xi)$, in matrix form,

$$\mathbf{C}(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) \mathbf{P}_i = \mathbf{P}^T \mathbf{N}(\xi) \tag{4.9}$$

and recalling that knot insertion causes no geometric nor parametric changes to the curve (Section 3.2.7), the following is obtained,

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

This gives the relation between the B-spline basis functions and the Bernstein polynomials,

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

$\mathbf{C}$ is called the Bézier extraction operator. Note that the only input required to construct $\mathbf{C}$ is the knot vector. The Bézier extraction operator is therefore independent of control points and basis functions, meaning that the operator is identical for B-splines and NURBS.

To obtain the NURBS basis functions, the weighting function is rewritten,

$$W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) w_i = \mathbf{w}^T \mathbf{N}(\xi) = \mathbf{w}^T \mathbf{C}\mathbf{B}(\xi) = (\mathbf{C}^T \mathbf{w})^T \mathbf{B}(\xi) = (\mathbf{w}^b)^T \mathbf{B}(\xi) = W^b(\xi) \quad (4.12)$$

where $\mathbf{w}^b = \mathbf{C}^T \mathbf{w}$ are the weights associated with the Bézier basis functions, given as a column vector. Substituting Eq. (4.11) into Eq. (3.18), the NURBS basis functions using the Bézier extraction operator become

$$\mathbf{R}(\xi) = \frac{1}{W^b(\xi)} \mathbf{W}\mathbf{C}\mathbf{B}(\xi) \quad (4.13)$$

where $\mathbf{W}$ are the NURBS weights.

As with knot insertion, Bézier decomposition of control points is performed directly to the B-spline curve which defines the NURBS curve. Geometrically this is done by projecting the NURBS control points into $\mathbb{R}^{d+1}$, then apply the Bézier extraction operator to the B-spline control points, and finally project back into $\mathbb{R}^d$ to obtain the relation between Bézier control points and NURBS control points,

$$\mathbf{P}^b = (\mathbf{W}^b)^{-1} \mathbf{C}^T \mathbf{W}\mathbf{P} \quad (4.14)$$

where $\mathbf{W}^b$ are the Bézier weights $\mathbf{w}^b$ given as a diagonal matrix. Multiply Eq. (4.14) by $\mathbf{W}^b$,

$$\mathbf{W}^b \mathbf{P}^b = \mathbf{C}^T \mathbf{W}\mathbf{P} \quad (4.15)$$

Combining Eqs. (4.13) and (4.15), the NURBS curve in terms of C^0 Bézier elements become

$$\mathbf{C}(\xi) = \mathbf{P}^T \mathbf{R}(\xi) = \frac{1}{W^b(\xi)} \mathbf{P}^T \mathbf{W}\mathbf{C}\mathbf{B}(\xi) = \frac{1}{W^b(\xi)} (\mathbf{C}^T \mathbf{W}\mathbf{P})^T \mathbf{B}(\xi) = \frac{1}{W^b(\xi)} (\mathbf{W}^b \mathbf{P}^b)^T \mathbf{B}(\xi) \quad (4.16)$$

4.1.3 Localizing the Extraction Operator

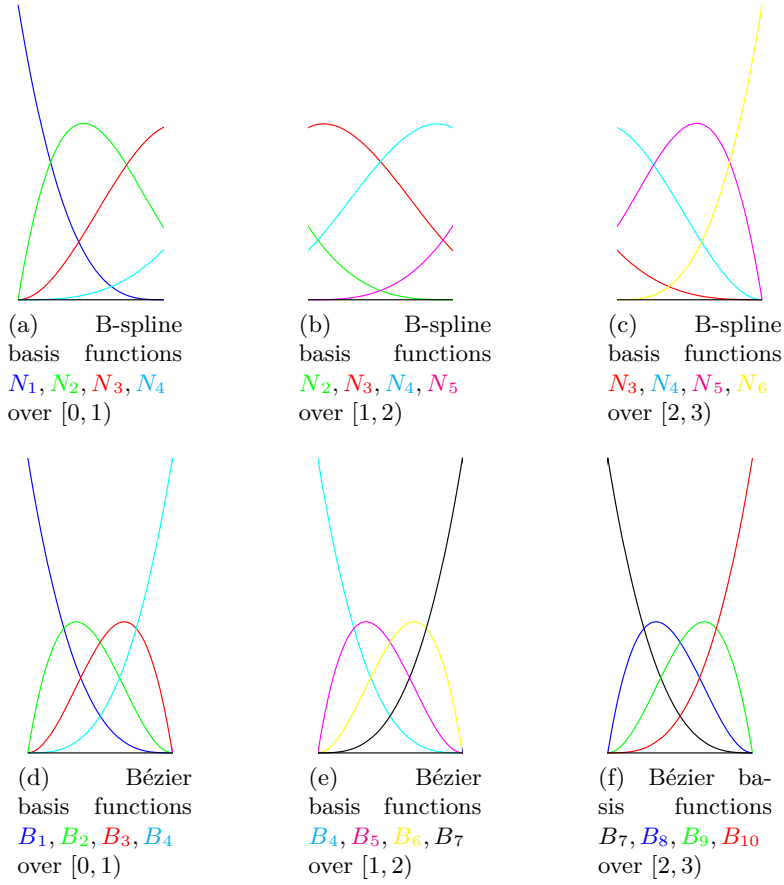
If the Bézier extraction operator for the example in Figure 4.2 is computed using Eqs. (4.5) and (4.7), Eq. (4.11) becomes

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \end{bmatrix} \tag{4.17}$$

when dropping the index for the polynomial order, which is $p = 3$. Bézier decomposition creates a Bézier element over each knot interval. Considering the first knot span, $[0, 1)$, it may be seen that the B-spline basis functions N_1, N_2, N_3 and N_4 can be represented as linear combinations of the Bézier basis functions over the same interval, B_1, B_2, B_3 and B_4,

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix} = \begin{bmatrix} N_1^1 \\ N_2^1 \\ N_3^1 \\ N_4^1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} B_1^1 \\ B_2^1 \\ B_3^1 \\ B_4^1 \end{bmatrix} \tag{4.18}$$

where the superscript denotes the knot interval or element number. Thus, the localized extraction operator for the knot span $[0, 1)$ can be separated as shown Figures 4.3a and 4.3d.



(a) B-spline basis functions N_1, N_2, N_3, N_4 over $[0, 1)$

(b) B-spline basis functions N_2, N_3, N_4, N_5 over $[1, 2)$

(c) B-spline basis functions N_3, N_4, N_5, N_6 over $[2, 3)$

(d) Bézier basis functions B_1, B_2, B_3, B_4 over $[0, 1)$

(e) Bézier basis functions B_4, B_5, B_6, B_7 over $[1, 2)$

(f) Bézier basis functions B_7, B_8, B_9, B_{10} over $[2, 3)$

Figure 4.3: Bézier decomposition over the knot intervals $[0, 1)$, $[1, 2)$ and $[2, 3)$

Following the same principal, the Bézier decompositions over the intervals $[1, 2)$ and $[2, 3)$ become

$$\begin{bmatrix} N_2 \\ N_3 \\ N_4 \\ N_5 \end{bmatrix} = \begin{bmatrix} N_1^2 \\ N_2^2 \\ N_3^2 \\ N_4^2 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} B_1^2 \\ B_2^2 \\ B_3^2 \\ B_4^2 \end{bmatrix} \tag{4.19}$$

$$\begin{bmatrix} N_3 \\ N_4 \\ N_5 \\ N_6 \end{bmatrix} = \begin{bmatrix} N_1^3 \\ N_2^3 \\ N_3^3 \\ N_4^3 \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_7 \\ B_8 \\ B_9 \\ B_{10} \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_1^3 \\ B_2^3 \\ B_3^3 \\ B_4^3 \end{bmatrix} \tag{4.20}$$

The B-spline basis functions and corresponding Bézier basis functions over the interval $[1, 2)$ are shown in Figures 4.3b and 4.3e, and over the knot span $[2, 3)$ in Figures 4.3c and 4.3f.

The localized element form of Eq. (4.11) is therefore

$$\mathbf{N}^e(\xi) = \mathbf{C}^e \mathbf{B}^e(\xi) \tag{4.21}$$

where the superscript e denotes the element number. Note that the size of $\mathbf{C}^e$ is $(p+1)\times(p+1)$. This means that the global extraction operator $\mathbf{C}$ does not need to be established, only the localized extraction operators $\mathbf{C}^e$ for each element.

Further, Eq. (4.13) is rewritten to the localized NURBS basis functions,

$$\mathbf{R}^e(\xi) = \frac{1}{W^b(\xi)}\mathbf{W}^e\mathbf{C}^e\mathbf{B}^e(\xi) \tag{4.22}$$

where $W^b(\xi)$ may be calculated as

$$W^b(\xi) = \sum_{i=1}^{(p+1)^{d_p}} B_{i,p}(\xi)w_i^b \tag{4.23}$$

and d_p is the number of parametric dimensions. Applying the quotient rule to Eq. (4.22), the derivatives of the localized NURBS basis functions are obtained,

$$\frac{\partial\mathbf{R}^e(\xi)}{\partial\xi} = \mathbf{W}^e\mathbf{C}^e\frac{\partial}{\partial\xi}(\frac{1}{W^b(\xi)}\mathbf{B}^e(\xi)) = \mathbf{W}^e\mathbf{C}^e(\frac{1}{W^b(\xi)}\frac{\partial\mathbf{B}^e(\xi)}{\partial\xi} - \frac{\partial W^b(\xi)}{\partial\xi}\frac{\mathbf{B}^e(\xi)}{(W^b(\xi))^2}) \tag{4.24}$$

where the derivative of the weighting function is evaluated by

$$\frac{\partial W^b(\xi)}{\partial\xi} = \sum_{i=1}^{(p+1)^{d_p}} B'_{i,p}(\xi)w_i^b \tag{4.25}$$

The relation between the localized Bézier control points and localized NURBS control points is similar to its global form (Eq. (4.14)),

$$\mathbf{P}^{b,e} = (\mathbf{W}^{b,e})^{-1}(\mathbf{C}^e)^T\mathbf{W}^e\mathbf{P}^e \tag{4.26}$$

where the localized Bézier weights are

$$\mathbf{W}^{b,e} = \begin{bmatrix} w_1^{b,e} & & & \\ & w_2^{b,e} & & \\ & & \ddots & \\ & & & w_n^{b,e} \end{bmatrix} \quad \text{and} \quad \mathbf{w}^{b,e} = (\mathbf{C}^e)^T\mathbf{w}^e \tag{4.27}$$

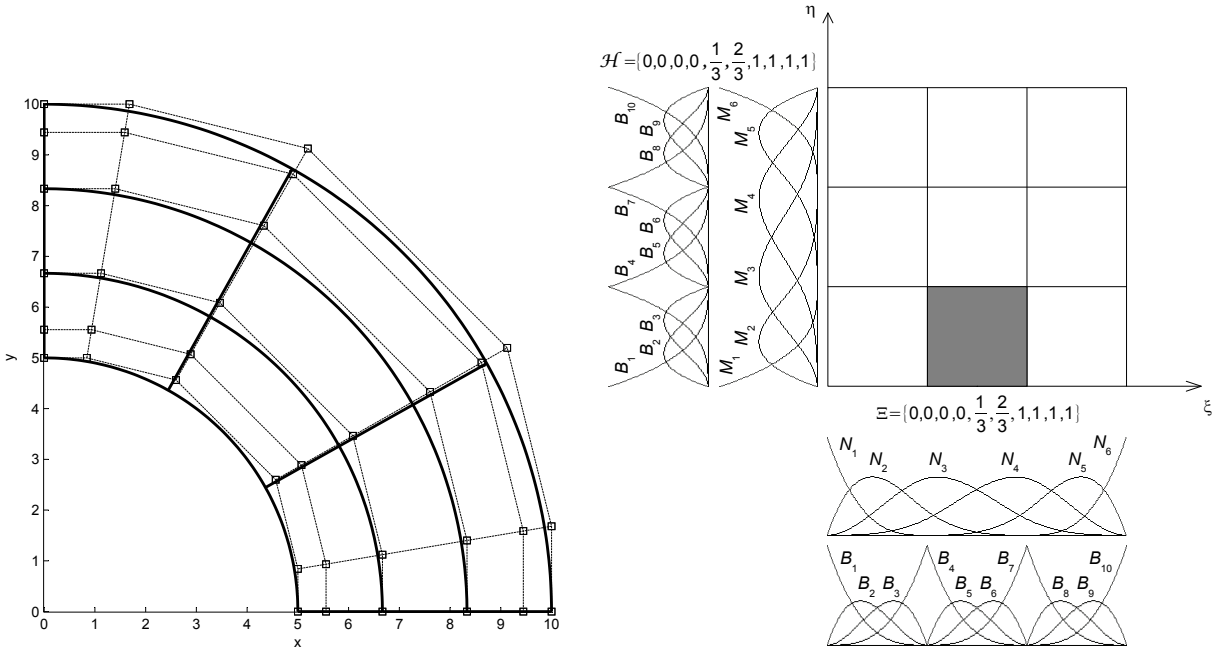
4.1.4 The Bivariate Extraction Operator

The localized bivariate extraction operator is defined as the tensor product of the univariate extraction operators,

$$\mathbf{C}^e = \mathbf{C}_\eta^j \otimes \mathbf{C}_\xi^i = \begin{bmatrix} C_{\eta,11}^j\mathbf{C}_\xi^i & C_{\eta,12}^j\mathbf{C}_\xi^i & \cdots \\ C_{\eta,21}^j\mathbf{C}_\xi^i & C_{\eta,22}^j\mathbf{C}_\xi^i & \\ \vdots & & \ddots \end{bmatrix} \tag{4.28}$$

where $\mathbf{C}_\xi^i$ and $\mathbf{C}_\eta^j$ is the i^{th} and j^{th} univariate element extraction operators in the ξ and η direction, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$ and $e = 1, 2, \ldots, n \times m$.

Figure 4.4a shows the physical mesh of a circular beam. This beam may be analysed using cubic NURBS basis functions. If three elements are chosen for both the radial direction and the tangential direction, the parameter space may defined by the knot vectors $\Xi = \left\{0, 0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1, 1\right\}$ and $\mathcal{H} = \left\{0, 0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1, 1\right\}$ as illustrated in Figure 4.4b. These knot vectors will render the same localized extraction operators as the univariate example in Section 4.1.3, Eqs. (4.18) - (4.20).



(a) Physical mesh of a circular beam with NURBS control mesh

(b) The circular beam in the parameter space

Figure 4.4: Bivariate extraction operator, the physical space and the parameter space

For the shaded element (number 2 in the tangential direction, number 1 in the radial direction), the univariate extraction operators are

$$\begin{bmatrix} N_2 \\ N_3 \\ N_4 \\ N_5 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} B_4 \\ B_5 \\ B_6 \\ B_7 \end{bmatrix}, \qquad \mathbf{C}_\xi^2 = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \tag{4.29}$$

and

$$\begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix}, \qquad \mathbf{C}_\eta^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \tag{4.30}$$

making the bivariate extraction operator

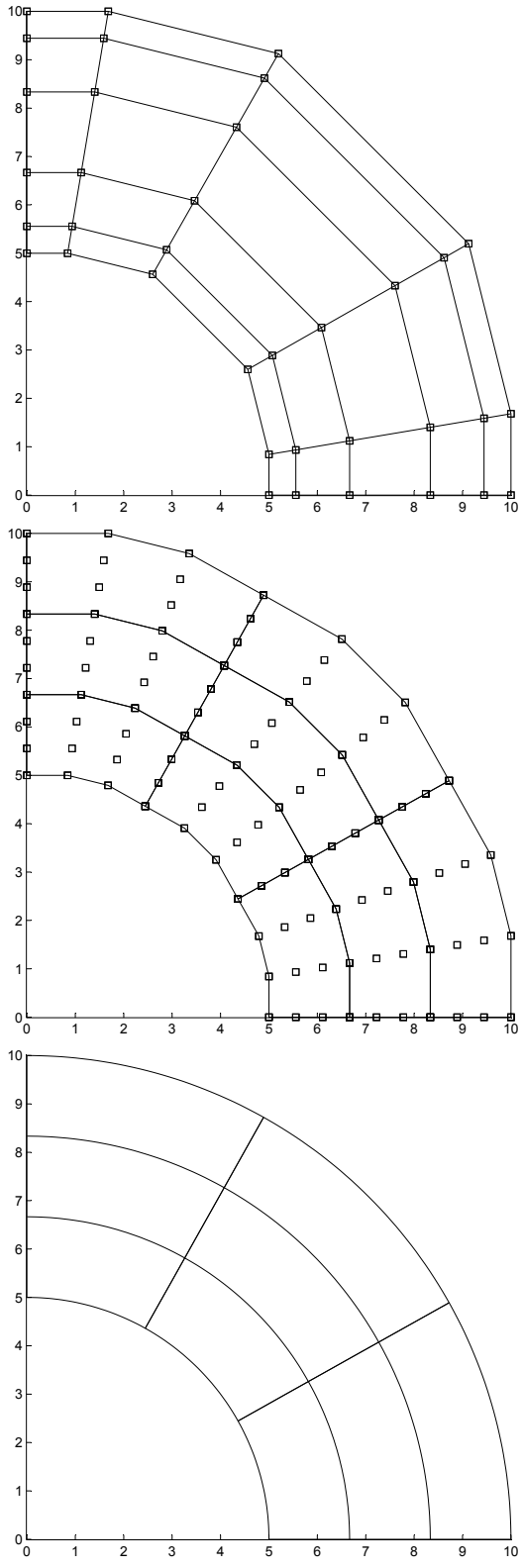
$$\mathbf{C}^2 = \mathbf{C}_\eta^1 \otimes \mathbf{C}_\xi^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \frac{7}{12} & \frac{2}{3} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix}$$

$$= \begin{bmatrix} 1/4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7/12 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/6 & 1/3 & 2/3 & 7/12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 & 0 & 1/8 & 0 & 0 & 0 & 1/16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7/12 & 2/3 & 1/3 & 1/6 & 7/24 & 1/3 & 1/6 & 1/12 & 7/48 & 1/6 & 1/12 & 1/24 \\ 0 & 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 7/12 & 1/12 & 1/6 & 1/3 & 7/24 & 1/24 & 1/12 & 1/6 & 7/48 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 0 & 0 & 0 & 1/8 & 0 & 0 & 0 & 1/16 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/8 & 0 & 0 & 0 & 7/48 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7/24 & 1/3 & 1/6 & 1/12 & 49/144 & 7/18 & 7/36 & 7/72 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/12 & 1/6 & 1/3 & 7/24 & 7/72 & 7/36 & 7/18 & 49/144 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/8 & 0 & 0 & 0 & 7/48 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/24 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7/72 & 1/9 & 1/18 & 1/36 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/36 & 1/18 & 1/9 & 7/72 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/24 \end{bmatrix} \tag{4.31}$$

With all the bivariate extraction operators at hand, the Bézier control points may be computed using Eq. (4.26) for each element. The Bézier physical mesh is then similarly to Eq. (4.16), determined by

$$\mathbf{C}(\xi,\eta) = \frac{1}{W^b(\xi,\eta)} (\mathbf{W}^b \mathbf{P}^b)^T \mathbf{B}(\xi,\eta) \tag{4.32}$$

The mapping from the NURBS control mesh to the Bézier control mesh, and finally to the Bézier physical mesh, is illustrated in Figure 4.5. Note that the Bézier elements equal the knot spans (elements) of NURBS, see Figure 4.4a. This is because Bézier decomposition leaves the geometry unchanged, but the surface contains more control points.



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

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

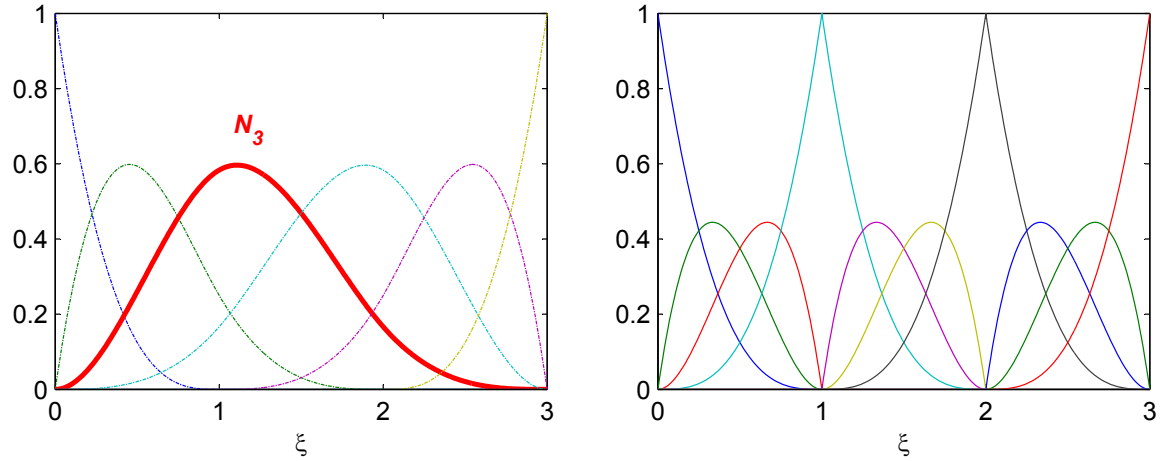
Figure 4.5: From NURBS control mesh to Bézier control mesh to Bézier physical mesh

4.2 The Bézier Extraction Operator for T-Splines

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 as reviewed in Section 3.4.2. Thus, the element extraction operators are not computed as a tensor product as in the case 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 4.6a shows the univariate T-spline basis function N_3 to the local knot vector $\Xi = \{0,0,1,2,3\}$ with $p=3$. The thin dashed 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 4.6b.



(a) Basis function N_3 to the local knot vector $\Xi = \{0,0,1,2,3\}$ and the additional basis functions (thin dashed lines) to the extended knot vector $\bar{\Xi} = \{0,0,0,0,1,2,3,3,3,3\}$

(b) Bézier basis functions after Bézier decomposition of $\bar{\Xi} = \{0,0,0,0,1,2,3,3,3,3\}$

Figure 4.6: Bézier decomposition of a univariate T-spline basis function

The extraction operators will therefore be equal to the operators in the case for NURBS,

$$\begin{bmatrix} N_1 \\ N_2 \\ \boldsymbol{N_3} \\ N_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{4} \\ \mathbf{0} & \mathbf{0} & \mathbf{\frac{1}{2}} & \mathbf{\frac{7}{12}} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} \boldsymbol{B_1} \\ \boldsymbol{B_2} \\ \boldsymbol{B_3} \\ \boldsymbol{B_4} \end{bmatrix} \tag{4.33}$$

$$\begin{bmatrix} N_2 \\ \boldsymbol{N_3} \\ N_4 \\ N_5 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ \mathbf{\frac{7}{12}} & \mathbf{\frac{2}{3}} & \mathbf{\frac{1}{3}} & \mathbf{\frac{1}{6}} \\ \frac{1}{6} & \frac{1}{3} & \frac{2}{3} & \frac{7}{12} \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} \boldsymbol{B_4} \\ \boldsymbol{B_5} \\ \boldsymbol{B_6} \\ \boldsymbol{B_7} \end{bmatrix} \tag{4.34}$$

$$\begin{bmatrix} \boldsymbol{N_3} \\ N_4 \\ N_5 \\ N_6 \end{bmatrix} = \begin{bmatrix} \mathbf{\frac{1}{6}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \frac{7}{12} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{B_7} \\ \boldsymbol{B_8} \\ \boldsymbol{B_9} \\ \boldsymbol{B_{10}} \end{bmatrix} \tag{4.35}$$

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 4.6b to the global T-spline basis function N_3 in Figure 4.6a. Thus, an algorithm to find the Bézier extraction operators for T-splines does not compute the redundant rows.

Recalling that T-splines in one dimension are pointless, the localized *bivariate* extraction operator for T-splines contains one row of length $(p+1)^2$ to each basis function in support of the T-spline element. The length of the row reflects the number of Bernstein polynomials for the bivariate element. The number of basis functions in support may be $(p+1)^2$ *or more* (in most cases) because T-junctions affect the element structure. This implies that the number of rows in the extraction operator for T-splines may be more than the $(p+1)^2$ rows in the bivariate extraction operator for NURBS.

Note that the extraction operator for T-splines handles the "hanging nodes" of FEA.

Chapter 5

Computational Procedures for Isogeometric Analysis

Isogeometric analysis employs the same mathematical foundation as FEA when it comes to the method for numerical solution of differential equations. For solid and structural mechanics, the nature of compatibility between displacements and strains, and equilibrium between forces and stresses, also applies. This means that the formulations given in Section 2.1 prevails (except for specific equations for Q4 and Q9 elements), and that the abstract computational structure for isogeometric analysis is quite similar to FEA. However, although the main difference lies compactly enough in the set basis functions used, this change influences all steps of traditional FEA: Preprocessing, solving and postprocessing.

This chapter reviews the various computational procedures for isogeometric analysis. In Section 5.1, computational procedures adapted straightforward from the theory of B-splines are reviewed. The FE solver based on this is referred to as the conventional isogeometric analysis program. Thereafter isogeometric data structures based on Bézier extraction of NURBS are presented in Section 5.2. In Section 5.3, modifications to the Bézier based program to be able to run isogeometric analysis based on Bézier extraction of T-splines on imported T-meshes from Rhino are described.

5.1 Computational Procedures in Two Dimensions using B-Splines

Figure 5.1 shows a flow chart for the conventional isogeometric analysis program. Compared to the flow chart for FEA, Figure 2.3, the element loop is replaced by a double loop. The reason for this is because the shape functions of FEA are given directly as bivariate functions, while the basis functions of isogeometric analysis are defined for each direction separately. The isogeometric analysis program is designed for single patch problems only. If multiple patches are part of the problem, an additional loop through the patches outside the element loops is required. There are changes in all the main steps of the analysis compared to FEA, and these are reviewed subsequently.

The MATLAB files based on conventional isogeometric analysis are given in Appendix C, and the verification of isogeometric analysis using B-splines may be found in Appendix A.

5.1.1 Preprocessing

For the READ INPUT box, several changes to how the mesh is structured are involved. Since the basis functions of isogeometric analysis are related to control points, and not nodes as

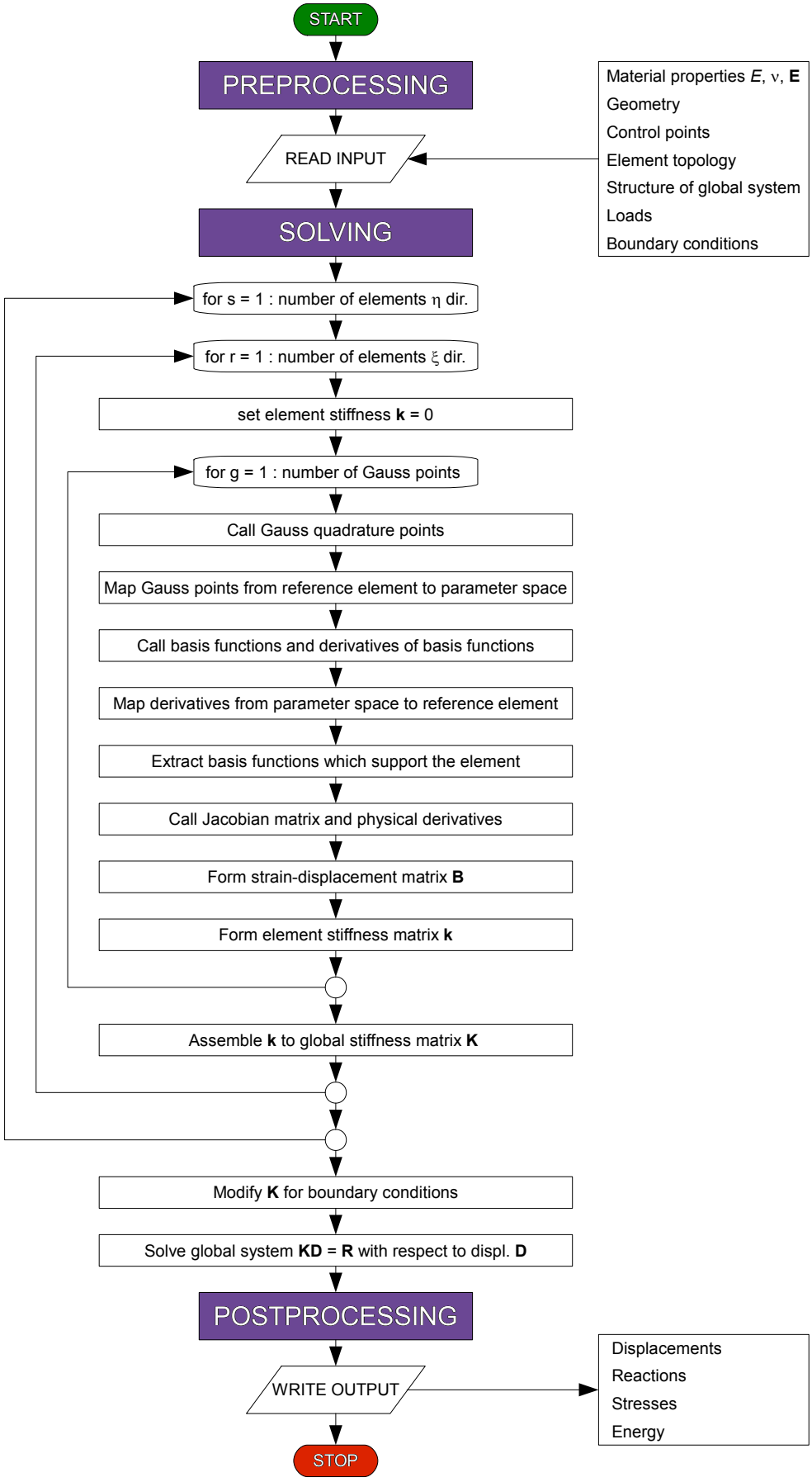
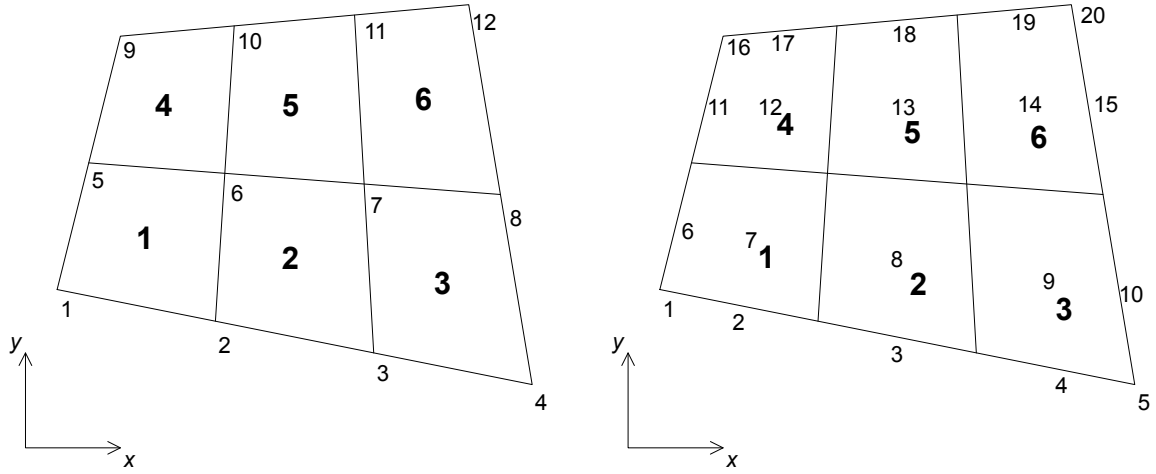


Figure 5.1: Flow chart for the conventional isogeometric analysis program

shape functions of FEA, the input of node coordinates is replaced by coordinates of control points. The element topology is therefore related to the numbering of control points rather than the numbering of element nodes. Figure 5.2 illustrates the numbering of elements and control points in the isogeometric analysis program. Recalling Figure 2.4a, note that the numbering for isogeometric analysis using 1st order elements is the same as for FEA using Q4 elements. The alignment of control points also coincides with nodes of the Q4 element. However, this is a special occurrence for the 1st polynomial order elements only.

For 2nd order elements and higher, control points of isogeometric analysis and nodes of FEA are not analogous. First, for the same number of elements, the mesh of isogeometric analysis contains fewer control points than nodes in the equal mesh of FEA. The consequence is fewer global degrees of freedom for the same mesh. Furthermore, the control points are not uniformly spaced as nodes are, see Figure 5.2b. They rather tend to gather at the boundaries of the physical space, and are in general independent of where the element boundaries are. This is because elements are defined by knots, while control points are determined by the number of basis functions and placed according to the knot insertion formulas (see Section 3.2.7).



(a) 1st order isogeometric elements (b) 2nd order isogeometric elements

Figure 5.2: Elements and control points

In FEA, the geometry needed for analysis is simply given as x, y coordinates of nodes. For isogeometric analysis, this input is a bit trickier because the geometry is defined by knot vectors and control points, i.e. a B-spline surface. Like FEA, the input of a simple square region is the four corners, but following a script generates initial knot vectors and control points given the polynomial order, see Table 5.1. The script may be found in Appendix B.4.14.

The number of initial control points in each direction is determined by the length of the knot vectors, and these control points must be equally spaced over the physical domain. For example, for $p = 2$, 3 initial control points must be given in each direction: The two corner points and the midpoint between them.

Table 5.1: Initial knot vectors and control points for a square region

Polynomial order	Initial knot vectors	Initial control points
$p = 1$	$\Xi = \{0, 0, 1, 1\}$ $\mathcal{H} = \{0, 0, 1, 1\}$	ξ direction: 2 η direction: 2
$p = 2$	$\Xi = \{0, 0, 0, 1, 1, 1\}$ $\mathcal{H} = \{0, 0, 0, 1, 1, 1\}$	ξ direction: 3 η direction: 3
$p = 3$	$\Xi = \{0, 0, 0, 0, 1, 1, 1, 1\}$ $\mathcal{H} = \{0, 0, 0, 0, 1, 1, 1, 1\}$	ξ direction: 4 η direction: 4
$p = 4$	$\Xi = \{0, 0, 0, 0, 0, 1, 1, 1, 1, 1\}$ $\mathcal{H} = \{0, 0, 0, 0, 0, 1, 1, 1, 1, 1\}$	ξ direction: 5 η direction: 5

The alignment of control points when the mesh is refined is then determined from a knot insertion routine based on Eqs. (3.12) and (3.13), enclosed in Appendix B.4.16. Since the formulas are based on a single knot inserted, the routine must be repeated to give the desired number of elements. The global knot vectors expand as knots are inserted. As mentioned in Section 3.2.7, knot insertion in this manner is necessary to leave the parametrization and geometry unchanged. Since control points and elements of isogeometric analysis are not directly connected in the way nodes and elements of FEA are related, the physical understanding may be intricate.

Nevertheless, a similarity to FEA exists in that the number of control points related to a single two-dimensional element is $(p + 1)^2$. This is the same as for FEA, where there are $(1 + 1)^2 = 4$ nodes for a Q4 element (polynomial order 1) and $(2+1)^2 = 9$ nodes for a Q9 element (polynomial order 2). This crystallizes to the fact that in isogeometric analysis (for elements greater than 1st order), more than one control point is "shared" in each direction between elements, in contrast to FEA where only the nodes on the element boundaries "belong" to elements on both sides. The sharing behaviour of control points is connected to the local support of the basis functions, as reviewed in Section 3.2.5.

The IEN arrays for the geometry in Figure 5.2 are therefore

$$\texttt{IEN}_{\texttt{P1}} = \begin{bmatrix} 1 & 2 & 5 & 6 \\ 2 & 3 & 6 & 7 \\ 3 & 4 & 7 & 8 \\ 5 & 6 & 9 & 10 \\ 6 & 7 & 10 & 11 \\ 7 & 8 & 11 & 12 \end{bmatrix} \quad \text{and} \quad \texttt{IEN}_{\texttt{P2}} = \begin{bmatrix} 1 & 2 & 3 & 6 & 7 & 8 & 11 & 12 & 13 \\ 2 & 3 & 4 & 7 & 8 & 9 & 12 & 13 & 14 \\ 3 & 4 & 5 & 8 & 9 & 10 & 13 & 14 & 15 \\ 6 & 7 & 8 & 11 & 12 & 13 & 16 & 17 & 18 \\ 7 & 8 & 9 & 12 & 13 & 14 & 17 & 18 & 19 \\ 8 & 9 & 10 & 13 & 14 & 15 & 18 & 19 & 20 \end{bmatrix} \tag{5.1}$$

and the coordinates of the control points are stored similarly to FEA,

$$\texttt{P}_{\texttt{P1}} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{12} & y_{12} \end{bmatrix} \quad \text{and} \quad \texttt{P}_{\texttt{P2}} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{20} & y_{20} \end{bmatrix}$$

The program considers only knot vectors with no repeated inner knot values, cf. k refinement in Section 3.2.7. The number of basis functions and thereof control points is then *the number of elements plus the polynomial order* in each direction. The argument for this, when considered the first parametric dimension ξ, is

$$n = |\Xi| - p - 1 = (n_x + 2p + 1) - p - 1 = n_x + p \tag{5.2}$$

where n is the number of basis functions, Ξ is the knot vector, p is the polynomial order and n_x is the number of elements in ξ direction. The same polynomial order is used for both directions. A mesh with 3rd order elements has therefore only one more control point in each direction compared to 2nd order elements. For FEA, replacing Q9 elements with Q16 elements would increase the number of nodes extensively, from $(2n_x + 1)$ to $(3n_x + 1)$ in each direction. This is why in FEA, increasing the order of the elements has limited use, at least in the ordinary Lagrange family. It is however an excellent option in isogeometric analysis to obtain solutions with higher accuracy.

Forming the load vector of consistent nodal loads involves more general formulation than FEA. Since the basis functions are defined for the whole model, there is no specific formulation for the reference element how the distributed load is dispersed at the control points. Eqs. (2.36), (2.37) and (2.38) can therefore not be employed and numerical integration must be used for all cases.

For traction at a vertical boundary in the parameter space (not necessarily vertical in the physical space) this becomes

$$\mathbf{r}_e = \int_0^{n_y} \mathbf{N}^T \mathbf{N} \mathbf{q} \left| \begin{matrix} \frac{\partial x}{\partial \eta} \\ \\ \frac{\partial y}{\partial \eta} \end{matrix} \right| d\eta \approx \sum_{j=1}^{N_{Gauss,\eta}} \mathbf{N}^T(\eta_j) \mathbf{N}(\eta_j) \mathbf{q}(\eta_j) \left| \begin{matrix} \frac{\partial x}{\partial \eta} \\ \\ \frac{\partial y}{\partial \eta} \end{matrix} \right| W_j \tag{5.3}$$

Similarly, for traction at a horizontal boundary in the parameter space, the consistent nodal loads are found by

$$\mathbf{r}_e = \int_0^{n_x} \mathbf{N}^T \mathbf{N} \mathbf{q} \left| \begin{matrix} \frac{\partial x}{\partial \xi} \\ \\ \frac{\partial y}{\partial \xi} \end{matrix} \right| d\xi \approx \sum_{i=1}^{N_{Gauss,\xi}} \mathbf{N}^T(\xi_i) \mathbf{N}(\xi_i) \mathbf{q}(\xi_i) \left| \begin{matrix} \frac{\partial x}{\partial \xi} \\ \\ \frac{\partial y}{\partial \xi} \end{matrix} \right| W_i \tag{5.4}$$

Here, $\mathbf{N}$ are the basis functions at the respective boundary and $\mathbf{q}$ is the load vector with values of the distributed load at the Gauss boundary points. $\mathbf{N}$ is a row vector with length $(p + 1)$, while $\mathbf{q}$ is a column vector with length $(p+1)$. The basis functions at the boundary are extracted from the the basis functions for the element, and the Jacobian for the boundary is extracted from the Jacobian matrix $\mathbf{J}$ for the element.

Boundary conditions for isogeometric analysis are assigned in a similar way as for FEA.

5.1.2 Solving

The local support of the basis functions is particularly important for the solving step because, unlike FEA, the basis functions of isogeometric analysis are defined for the whole model (if single patch), and not just a single element. When forming $\mathbf{k}$ in the element loop (see Figure 5.1), knowledge about which basis functions and control points that support the element currently looped over is necessary because the Jacobian matrix must be found by summarizing over these particular basis functions and control points,

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \sum_r \sum_s N'_r(\xi) M_s(\eta) x_{r,s} & \sum_r \sum_s N'_r(\xi) M_s(\eta) y_{r,s} \\ \\ \sum_r \sum_s N_r(\xi) M'_s(\eta) x_{r,s} & \sum_r \sum_s N_r(\xi) M'_s(\eta) y_{r,s} \end{bmatrix} \tag{5.5}$$

Here, r is the index of the supported basis function for the element in ξ direction and s is the index of the supported basis function for the element in η direction. $x_{r,s}$ and $y_{r,s}$ are the physical

coordinates of the corresponding control point to the r, s basis function. The support is actually *the current element number* in the respective direction *plus the polynomial order*. This may be seen in Figure 5.2b; for instance element number 3 in ξ direction, the index $r = 3, 4, 5$.

Recalling from Section 2.1.5, the Jacobian matrix represents the mapping of the element between the physical space and the parameter space. Because the basis functions span the parameter space which consist of several elements, a mapping between each reference element and the parameter space is also needed. The reference element in isogeometric analysis is equal to the familiar element in FEA, except that it has no nodes. It is on the reference element level the Gaussian quadrature for numerical integration is defined. Since the parameter space is rectangular, the mapping between the reference element and the parameter space involves a constant Jacobian, together with linear relations between the Gauss points and their corresponding parameter value,

$$\xi = \xi_i + (\hat{\xi} + 1)\frac{\xi_{i+1} - \xi_i}{2} \qquad \text{and} \qquad \eta = \eta_i + (\hat{\eta} + 1)\frac{\eta_{i+1} - \eta_i}{2} \tag{5.6}$$

where the hat indicates coordinates on the reference element. The mapping processes in isogeometric analysis are illustrated in Figure 5.3.

More precisely, the sequence of the mapping processes between the reference element and the parameter space is this:

1. Call Gauss points for the reference element.
2. **Map Gauss points from the reference element to the parameter space.**
3. Evaluate basis functions and derivatives in the parameter space.
4. **Map derivatives from the parameter space back to the reference element with a constant Jacobian.** The value of the constant Jacobian is $J = {}^{=\partial\xi}/_{\partial\hat{\xi}} = {}^{l}/_{2}$, where l is the length of the element in the parameter space (the reference element has length 2).
5. The Jacobian matrix and the physical derivatives may now be evaluated on the reference element level (after the supported basis functions and derivatives are extracted).

Finally, the global stiffness matrix $\mathbf{K}$ is constructed similarly to FEA. Note that the items in bold typing involve mapping, and are therefore different from FEA.

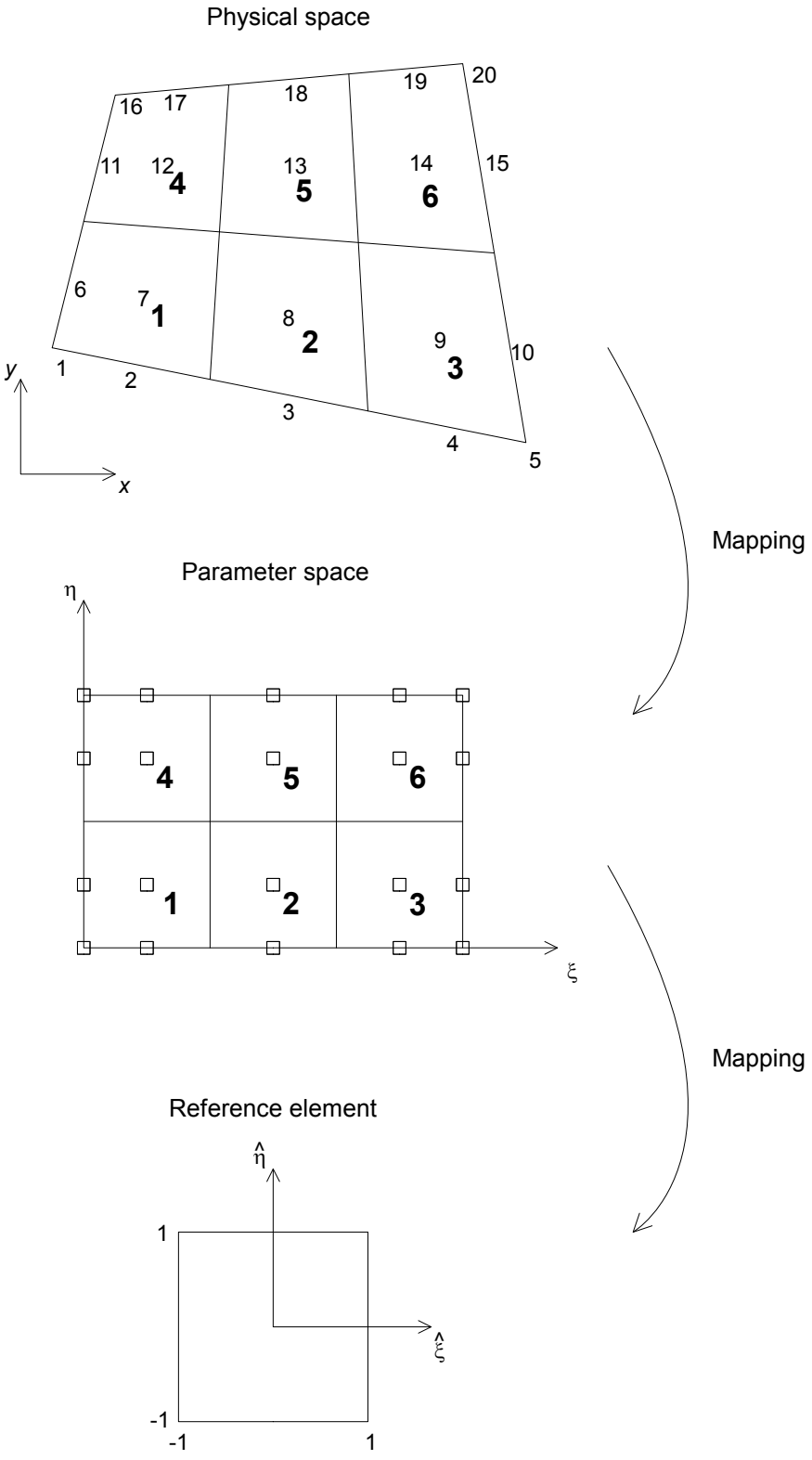


Figure 5.3: Mapping of elements in isogeometric analysis

5.1.3 Postprocessing

In the postprocessing step, isogeometric analysis will write displacements and reactions at the control points, in contrast to FEA which writes these quantities at the nodes. Since the alignment of the control points are not at the element boundaries, and the displacements at these lines or points are often of interest, the solution field must be interpolated using the basis functions,

$$\begin{bmatrix} u \\ v \end{bmatrix} (\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_i(\xi) M_j(\eta) \begin{bmatrix} u_{i,j} \\ v_{i,j} \end{bmatrix} \tag{5.7}$$

Hence the displacement components u and v are obtained as functions of (ξ, η), given the solution $\begin{bmatrix} u_{i,j} & v_{i,j} \end{bmatrix}^T$ at the control points. This interpolation may also be evaluated only for a single element for necessary points of interest. Note that for isogeometric analysis using 1st order elements the solution field is equal to FEA using Q4 elements, and interpolation is not necessary.

Like FEA, stresses are evaluated at the Gauss points. However, a stress recovery field for isogeometric analysis found by extrapolating the stresses in a traditional manner using the basis functions cannot be done. This is because basis functions in most cases are different from 1 at their corresponding control point, as opposed to shape functions and nodes of FEA. Stresses at element boundaries may be found by evaluating the stresses directly at points of interest.

5.2 Data Structures based on Bézier Extraction of NURBS

The original isogeometric analysis program is modified from B-splines as the set of basis functions used to NURBS based on Bézier extraction. NURBS can represent conical sections, and how such geometries are applied for analysis is reviewed. In addition, by employing Bézier extraction, the isogeometric FE data structure will to some extent turn away from the global structure as described in Section 5.1 and be closer to the localized structure of FEA. This will make isogeometric analysis easier to implement into existing FE codes, since the superior changes are confined to the shape function routine.

Figure 5.4 shows the flow chart for the isogeometric analysis program based on Bézier extraction. Recalling from Section 5.1, there were two major changes in conventional isogeometric analysis compared to FEA: The double element loop and the mappings between the reference element and the parameter space. These changes are vanished for isogeometric analysis based on Bézier extraction, and the flow chart resembles the flow chart for FEA more than the flow chart for conventional isogeometric analysis. This is owing to the new basis functions used. However, splines are still the basis for analysis, and therefore this program shares the majority of the computational formulations for conventional isogeometric analysis.

The MATLAB files for isogeometric analysis based on Bézier extraction of NURBS are given in Appendix B.

5.2.1 Preprocessing

The generation of control points and element topology is unchanged compared to the original isogeometric analysis program. This is because the global control points, and not the Bézier control points, serve as the degrees of freedom for the Bézier physical mesh.

NURBS geometry is utilized by modelling a quarter of a circular beam. For this geometry, the initial control points and weights will be as shown in Figure 5.5 for $p = 2$ and $p = 3$. The

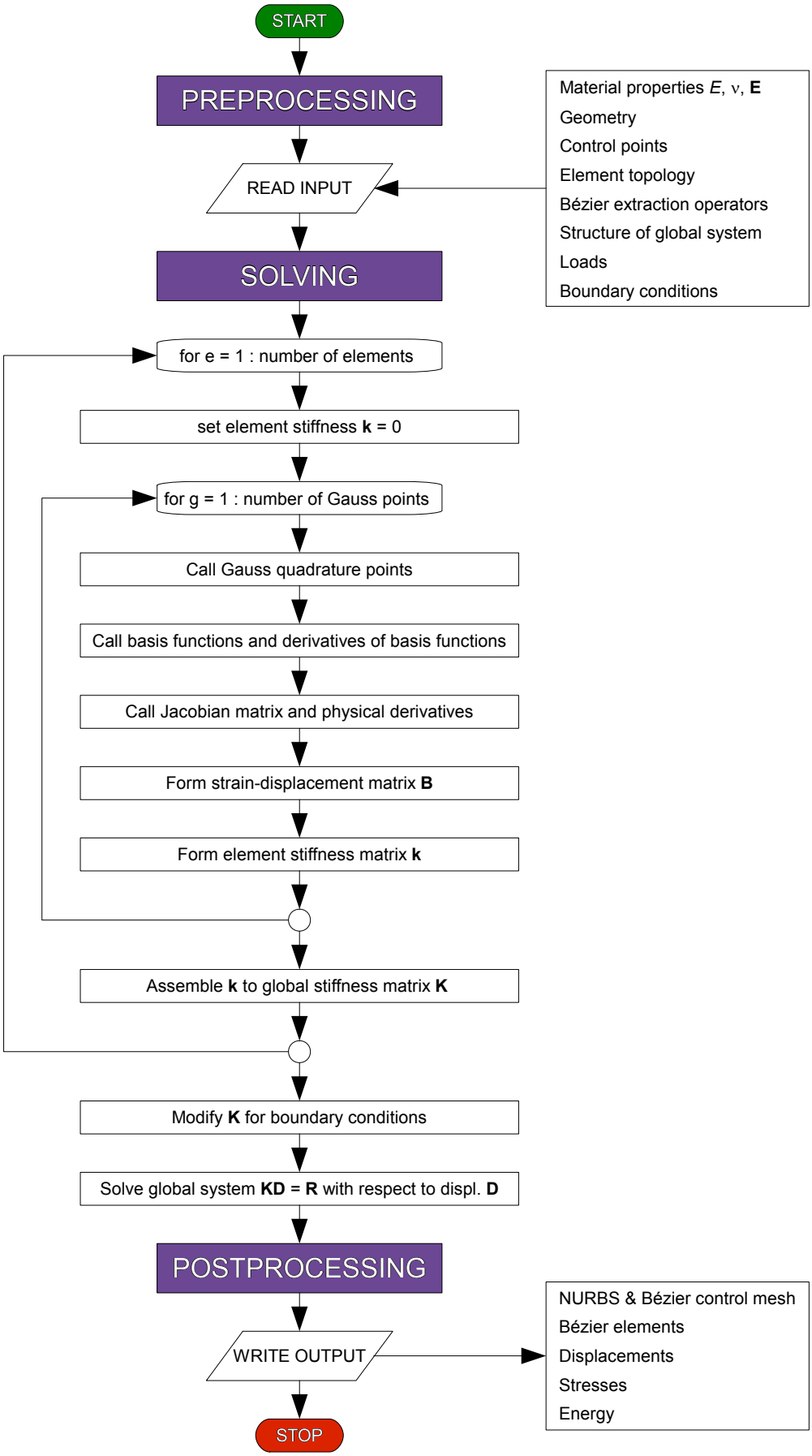
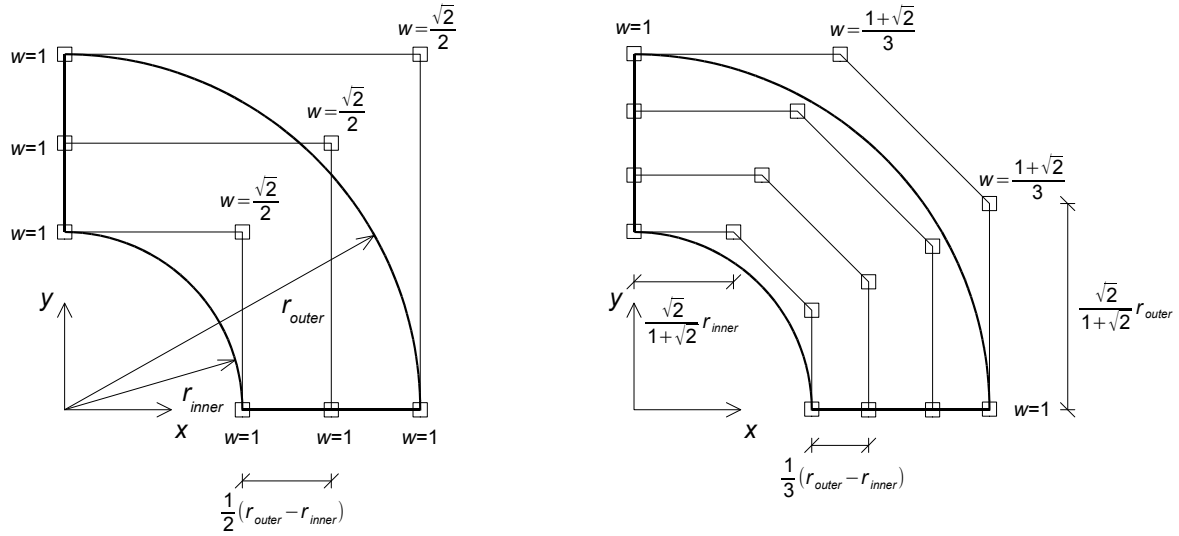


Figure 5.4: Flow chart for the isogeometric analysis program based on Bézier extraction of NURBS

initial knot vectors are equal to the square geometry, see Table 5.1. Both control points and weights will then be refined likewise conventional isogeometric analysis using the knot insertion algorithm.

Note how polar geometries are modelled with NURBS in contrast to FEA. In FEA, this geometry would be modelled by simply evaluating physical coordinates $x = \cos(\theta)$ and $y = \sin(\theta)$ at all nodes. Also, note that this geometry cannot be modelled using a 1st order NURBS surface. This is because the initial geometry in this case is a trapezoid, and refinement by knot insertion will not convert the geometry to a quarter disk.



(a) $p = 2$ (Based on examples from Piegl and Tiller [19]) (b) $p = 3$ (Calculated from formulas in Li [13])

Figure 5.5: Initial geometry for a quarter of a circular beam

The isogeometric analysis program based on Bézier extraction requires an extra input, namely the Bézier extraction operators. Piegl and Tiller [19] developed a Bézier decomposition algorithm corresponding Eqs. (3.12) and (3.13). Borden et al. [3] modified this algorithm to compute the localized extraction operators $\mathbf{C}^e$ directly with the knot vector as the only input. This algorithm may be found in Appendix B.4.5. The extraction operators may therefore be pre-calculated before solving. There will be one bivariate extraction operator $\mathbf{C}^e$ for each element e.

Boundary conditions for Bézier elements are applied equally to conventional isogeometric analysis since the element topology is unchanged. Numerical integration of loads must still be executed, but the process is simplified due to the new shape function routine.

5.2.2 Solving

Figure 5.6 shows a flow chart for the shape function routine which generates NURBS basis functions and derivatives in the isogeometric analysis program based on Bézier extraction. Recalling Section 4.1.3, the basis functions and derivatives are given on localized form and therefore directly for the element.

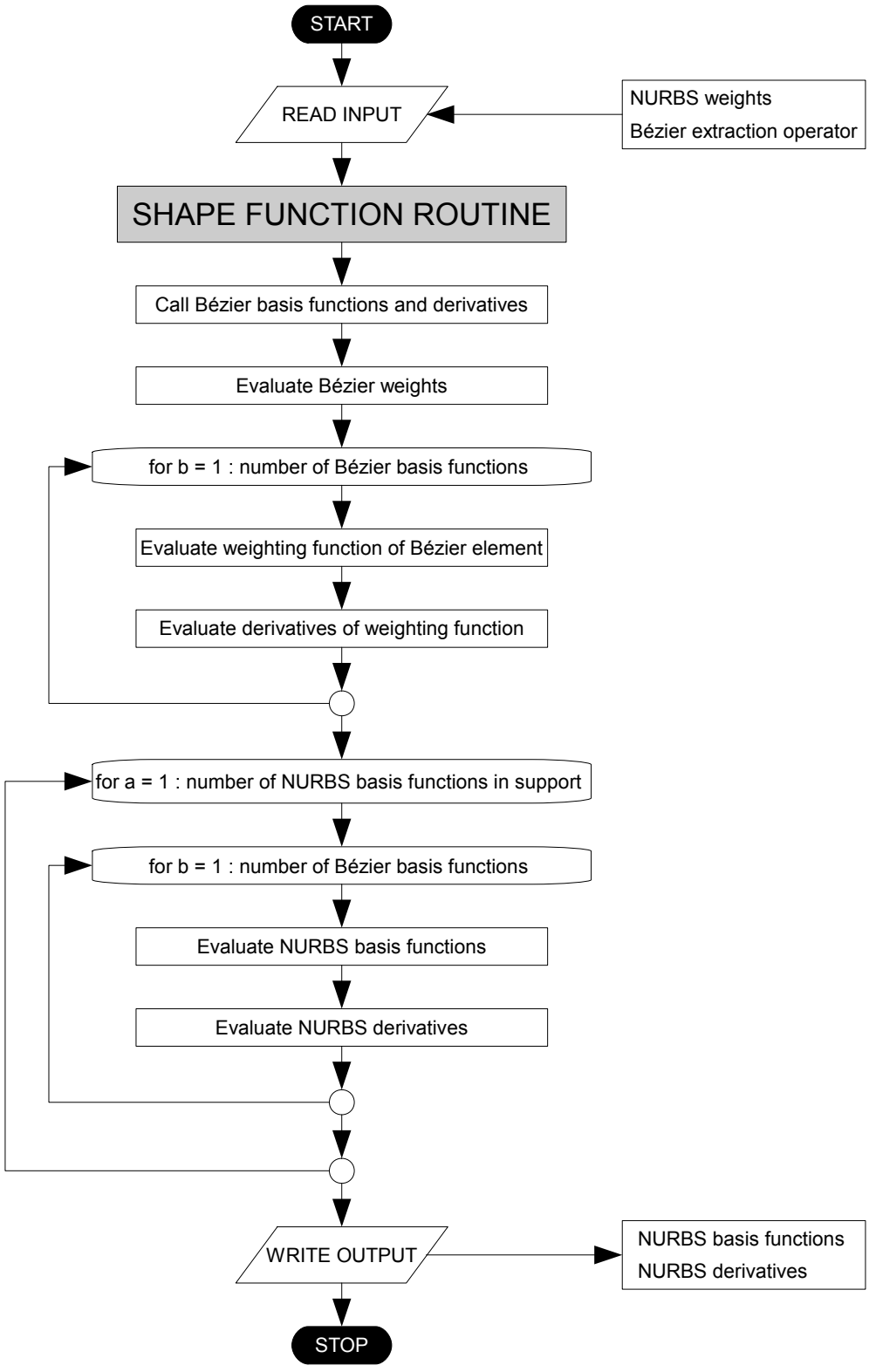


Figure 5.6: Flow chart for the shape function routine adapted to Bézier extraction of NURBS

To form NURBS basis functions based on Bézier extraction using this shape function routine, first the NURBS weights and the element extraction operators from the preprocessing step are called in. The Bézier basis functions and derivatives are calculated with Eqs. (4.1) and (4.3) in a separate routine, and are also called into the shape function routine. Then the Bézier weights are evaluated using Eq. (4.27), before the Bézier weighting function and derivatives are found

using Eqs. (4.23) and (4.25). Finally, the NURBS basis functions and derivatives are obtained with Eqs. (4.22) and (4.24).

The process to find the basis functions involves much more equations and subfunctions than conventional isogeometric analysis, which employs only Eqs. (3.2) and (3.3). However, because the Bézier extraction operator maps the global NURBS basis functions to the element level, actual mapping equations (Eq. (5.6)) between the reference element and the parameter space when the Gaussian quadrature rule is put to use disappear. For the same reason, the previously extraction of global basis functions which support the element is unnecessary. The mapping between the reference element and the parameter space still exists, but the extraction operators provide for that the mapping is performed in the shape function routine.

This new routine is also general in its structure, meaning that the tensor product property of NURBS is not utilized in the shape function routine, but beforehand in the forming of the Bernstein polynomials. The outcome is a single element loop instead of the previously observed double loop. The calculation of the Jacobian matrix becomes simplified and in fact equal to FEA (except that the derivatives vary for the element),

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{(p+1)^2} \frac{\partial R_i(\xi,\eta)}{\partial \xi} x_i & \sum_{i=1}^{(p+1)^2} \frac{\partial R_i(\xi,\eta)}{\partial \xi} y_i \\ \\ \sum_{i=1}^{(p+1)^2} \frac{\partial R_i(\xi,\eta)}{\partial \eta} x_i & \sum_{i=1}^{(p+1)^2} \frac{\partial R_i(\xi,\eta)}{\partial \eta} y_i \end{bmatrix} \tag{5.8}$$

All these changes result in a data structure where the forming of the element stiffness matrix $\mathbf{k}$ is almost identical to traditional FE structure, because the basis functions are given for the element similarly to FEA. This means that it will be much easier to implement isogeometric analysis into existing FE codes.

The drawback of isogeometric analysis based on Bézier extraction compared to conventional structures, is a slightly higher computational cost due to computation of the extraction operators and the additional processes in the evaluation of the basis functions. However, by replacing for-loops with matrix multiplication, MATLAB's framework is utilized better, decreasing the computational effort somewhat.

5.2.3 Postprocessing

The postprocessing step as presented in Section 5.1.3 still prevails. In addition, the program plots NURBS control mesh, Bézier control mesh and Bézier physical mesh. The displacement field is evaluated according to Eq. (5.7) at the parametric coordinates which equal the nodes of FEA (for the same polynomial order), and plotted in the physical space.

A modification is made to the storing of stress values to be able to plot contour plots of the stresses σ_x, σ_y, τ_{xy}, the von Mises stress and the Jacobian at the Gauss points. Instead of organizing the values as Eq. (2.43) shows, the stresses are stored reflecting the Gauss points' alignment relative to each other in the parameter space. Note that the stress values are plotted at the Gauss points directly, and hence no stress field is evaluated. In isogeometric analysis, the stress and strain cannot be extrapolated in the same manner as FEA. There is ongoing research on the field, and the preliminary results suggest the Greville points as a good choice for evaluation of stress and strain fields, but real applications in more detail must be studied [25].

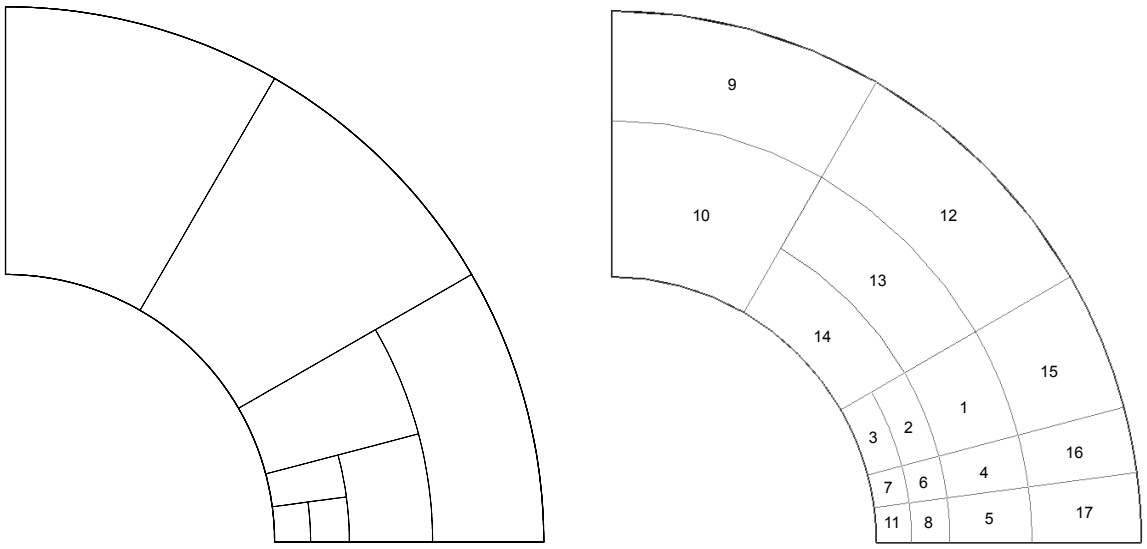
5.3 Isogeometric Analysis based on Bézier Extraction of T-Splines

If the NURBS code was developed in a traditional manner without Bézier extraction as described in Section 5.1 for B-splines, an extension to T-splines would involve a large amount of changes.

The reason for this is because NURBS have its source in the global tensor product domain, while T-splines exist only in the local tensor product domain. Traditionally, NURBS involve a mapping from the reference element to the (global) parameter space where the NURBS basis functions are defined. T-spline elements require a mapping from the reference element to a T-spline element domain before the T-spline basis functions are defined in the local basis function domains. The mappings are obviously not compatible.

However, if Bézier extraction is utilized, both NURBS and T-splines have an equal local tensor product domain, that is the Bézier element. FE data structures for T-splines based on Bézier extraction are therefore just a generalization of the data structures based on Bézier extraction of NURBS. For that reason, the latter may with only small modifications, also be used for isogeometric analysis of a T-mesh.

Bézier extraction of T-splines is in this study performed by importing the T-mesh of a quarter disk modelled in Rhino 4.0 with T-splines plug-in (T-Splines 3 for Rhino), into the FE solver in MATLAB. T-splines modelled with T-Splines for Rhino are 3rd order geometries. Note that the T-mesh in Figure 5.7a is the modelled geometry in the CAD program, but the imported data are for the extended T-mesh in Figure 5.7b.

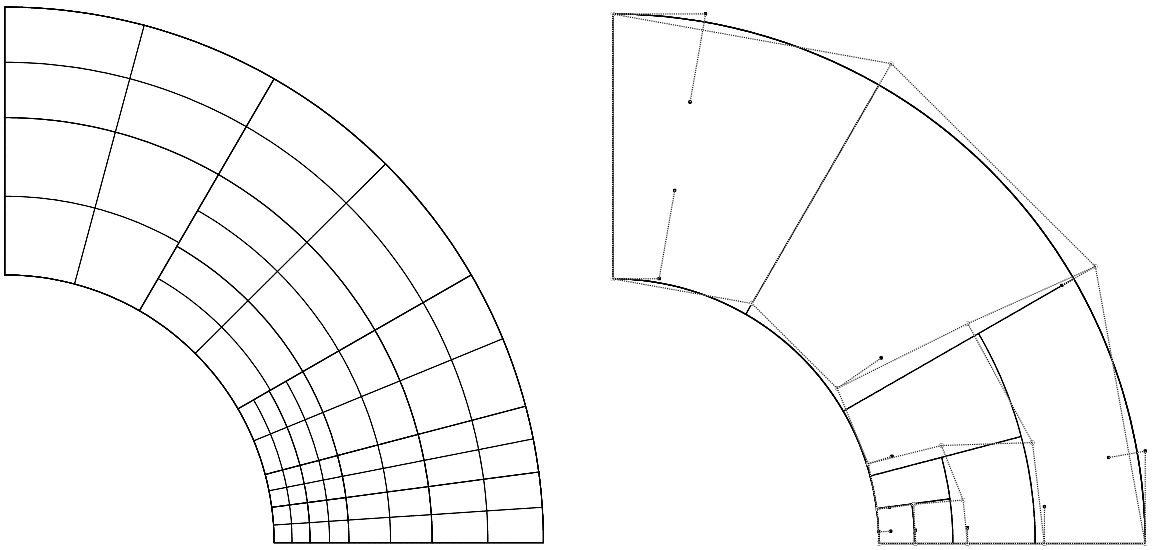


(a) T-mesh modelled in Rhino (b) The extended T-mesh

Figure 5.7: The T-mesh imported into the FE solver based on Bézier extraction of NURBS

The input from Rhino for this geometry is 39 control points and corresponding weights, together with Bézier extraction operators for the 17 T-spline elements. The flow chart for the isogeometric analysis program based on Bézier extraction of NURBS (Figure 5.4) is prevailing, except that geometry, control points and extraction operators are not generated by the program. These are instead inputs from T-Splines for Rhino. A parsing script creates the IEN array for the extended T-mesh based on the relations given by the extraction operators. The script also modifies the input data to be compatible with the program.

Figure 5.8a shows the Bézier elements for the quarter disk. Each Bézier element is marked by crossing lines, and there are 17 of them. The Bézier element boundaries equals the T-spline elements just like in the case for NURBS (see Section 4.1.4). Figure 5.8b illustrates the alignment of the 39 control points.



(a) Bézier elements (b) Control points

Figure 5.8: Bézier elements and control points for the quarter disk

Eqs. (4.22) and (4.24) are still employed to find the basis functions and derivatives for the element, but the number of basis functions in support of the element varies. For B-splines and NURBS, the number of basis functions in support of a two-dimensional element is $(p+1)^2$. For parts of the T-mesh that are not affected by the local refinement, the number of basis functions that support the T-spline element is the same as for NURBS. However, T-junctions cause nearby elements to be supported by extra basis functions, complicating the element topology of an extended T-mesh. The number of Bézier control points for the element is nevertheless equal to the case of NURBS. Since the number of basis functions in support of the element varies, the size of the element stiffness matrix $\mathbf{k}$ also varies. However, the assembling to the global stiffness matrix $\mathbf{K}$ is still arranged according to the IEN array.

Table 5.2 shows the number of basis functions in support of the elements for the geometry in Figure 5.7b. The random numbering of elements is caused by the input data from Rhino, which is a bit mixed-up in proportion to the geometry. This also makes it difficult to specify boundary conditions which cannot be defined and traced by a tolerance interval. Especially loads to be numerically integrated along boundaries require a search of control points related to the boundary beforehand.

Table 5.2: Number of basis functions in support of the T-spline elements in Figure 5.7b

Element	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Basis func.	16	16	16	17	17	18	18	19	16	16	20	16	16	16	16	16	16

The extra supported basis functions are handled in the NURBS based program by letting the number of basis functions and derivatives which support the element be determined by the length of the IEN array for the element instead of the constant length $(p+1)^2$ for NURBS. The size of the strain-displacement matrix $\mathbf{B}$ and the element stiffness matrix $\mathbf{k}$ should also be determined by length of the IEN array, and not what is expected for the order of the elements. These changes will not affect the NURBS analyses since the IEN array will only have constant length $(p+1)^2$ for all elements.

This illustrates that T-mesh analysis may with some modifications, be performed in a finite element solver with a shape function routine adapted to Bézier extraction of NURBS. The only

input needed is the control points and the extraction operators.

The main code for isogeometric analysis based on Bézier extraction of T-splines are given in Appendix B together with the NURBS based program. Particularly, the parsing script may be found in Appendix B.4.20.

Chapter 6

Verification of Isogeometric Analysis

6.1 Overview

Isogeometric analysis displays results which are numerically more accurate than FEA for polynomial order 2 and greater. The reason for this lies in the different approximation fields for the two methods of analysis. When using a knot vector without repeated inner knot values to form the basis functions of isogeometric analysis, higher continuity across element boundaries compared to FEA is achieved. The elements of FEA presented in this thesis display only C^0 continuity across element boundaries, while isogeometric analysis with basis functions as described, will give C^{p-1} continuity across element boundaries.

This means that continuity across element boundaries when performing isogeometric analysis using 1st order elements equals continuity in FEA. Since the control points in isogeometric analysis equals the nodes in FEA in both number and alignment, the solution using 1st order elements in isogeometric analysis and FEA (Q4 elements) will be exactly the same.

However, when the polynomial order is 2, the continuity across elements is C^1 for isogeometric analysis, while still C^0 for FEA (Q9 elements). In addition, increasing the order in isogeometric analysis involves much less effort than increasing the order in FEA. The algorithm includes only more repeated first and last knot values (k refinement) and one more control point for each increased step of order, preventing the number of global degrees of freedom to grow extensively. Therefore much higher continuity is easily obtained in isogeometric analysis, which ends up being superior to FEA regarding continuity across element boundaries.

The signification of higher order continuity means a smoother solution field, which usually resembles the physical problem better. Therefore the numerical solution will be closer to the physical problem. Still, if a discontinuous physical problem is desired, reduced continuity can be achieved by inserting inner knots which already exist. This makes the knot values repeated and hence reduces the order of continuity. Another possibility is to define the problem with several patches. However, application of various traditional finite elements may be just as convenient for discontinuous problems.

Higher continuity over element boundaries means also continuous stress and strain fields across such interfaces, as opposed to FEA where these quantities are discontinuous.

To verify that isogeometric analysis displays numerically better results than FEA, two examples of plane stress problems, Cook's problem and the end loaded beam, have previously been evaluated in the project work [17]. The results can be found in Appendix A.

The numerical results of Cook's problem and the end loaded beam are used to verify the isogeometric analysis program based on Bézier extraction. Giving results identical as before, the

isogeometric analysis program based on Bézier extraction is concluded to be equivalent to the conventional isogeometric analysis program in computational results.

To utilize the implementation of NURBS instead of B-splines, two examples involving conical sections are evaluated. Here isogeometric analysis has 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, the plate is chosen to be modelled as a quarter of a disk, which gives a geometry without singularities.

Finally, some studies on T-meshes modelled in Rhino 4.0 with T-Splines 3 for Rhino are performed. Numerical accuracy of manually refined T-meshes of the circular beam is investigated, and a machine part which is not possible to cite with a single NURBS surface is examined.

The results for isogeometric analysis in this chapter are obtained with the MATLAB code given in Appendix B. The results for FEA are also obtained with a self-made MATLAB code, but these files are not included. The results can be obtained with any FEA software, and the MATLAB code for FEA has previously been verified in the project work to display the correct results [17]. In addition, the FEA results for the circular beam are validated with the results in Zienkiewicz et al. [29].

6.2 Circular Beam

The circular beam is a cantilevered beam shaped as a quarter of a circular disk. The beam is subjected to a prescribed displacement $u_0 = -0.01$ at the free end. Its geometry, boundary conditions and material properties are given in Figure 6.1, and the material is linear-elastic and in a state of plane stress. The analytical solution to the problem is given in Timoshenko and Goodier [27] based on use of a stress function. The exact solution for the strain energy is given in Zienkiewicz et al. [29],

$$U = \frac{1}{\pi}(\ln 2 - 0.6) \approx 0.029649668442377 \tag{6.1}$$

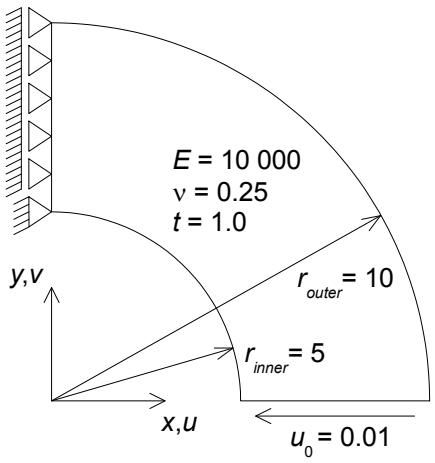
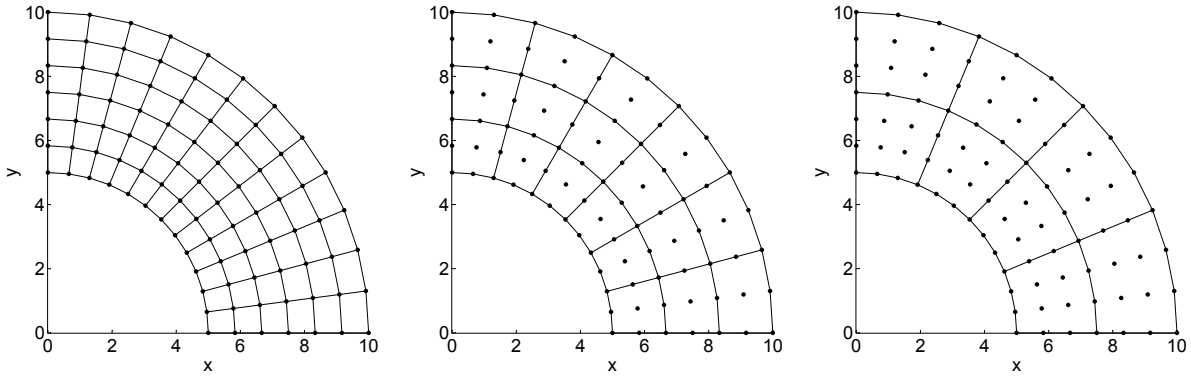


Figure 6.1: The geometry of the circular beam with material properties, boundary conditions and end shear.

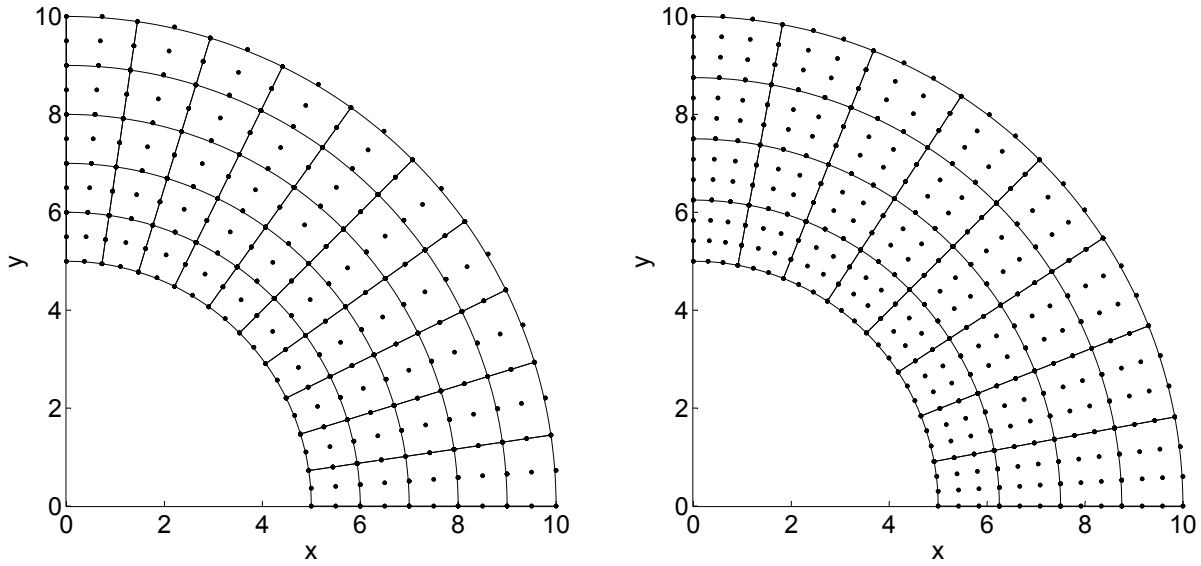
The circular beam is modelled with five different meshes using 4-, 9- and 16-node Lagrange elements for FEA, and quadrilateral NURBS Bézier elements of polynomial order 2 and 3 for

isogeometric analysis. The number of elements in the tangential direction is chosen to be twice the number of elements in the radial direction for all meshes. The coarsest meshes are shown in Figures 6.2 and 6.3. Isogeometric analysis polynomial order 1 is not considered, since it is not possible to model this geometry using a 1st order NURBS surface, see Section 5.2.1. Uniform refinement is chosen for the FEA meshes. The meshes for isogeometric analysis are chosen so that the number of global degrees of freedom is as close up to FEA as possible, while still keeping the number of elements in the tangential direction twice the number of elements in the radial direction.



(a) Q4 elements (b) Q9 elements (c) Q16 elements

Figure 6.2: Circular beam, coarsest meshes of FEA



(a) 2nd order Bézier elements (b) 3rd order Bézier elements

Figure 6.3: Circular beam, coarsest meshes of isogeometric analysis, Bézier physical mesh with Bézier control points.

Figure 6.4 shows a contour plot of the displacement u for the circular beam. Note that the horizontal displacement is zero at the left boundary due to the boundary condition illustrated in Figure 6.1. Also, $u = -0.01$ at the bottom edge, which agrees with the prescribed displacement u_0.

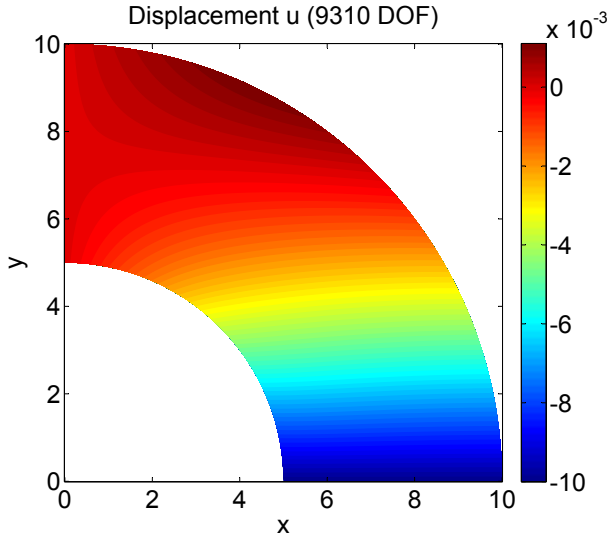


Figure 6.4: Displacement u for the circular beam

Table 6.1 shows the results of the strain energy with 14 decimal places (13 significant digits) for the different meshes and methods. The results for FEA are validated with Zienkiewicz et al. [29]. As expected, isogeometric analysis display energy closer to the exact solution than FEA for approximately the same number of global degrees of freedom and the same order of elements. Higher order elements decrease the error. FEA using Q4 elements is farthest off the exact result, while isogeometric analysis using 3rd order elements is very close to the exact solution for the finest mesh. This result is therefore shown with 15 decimal places.

Table 6.1: Strain energy of the circular beam (exact solution $U = 0.029649668442377$)

Lagrange Q4		
Mesh	DOF	U^h
6x12	182	0.03042038175071
12x24	650	0.02984351371323
24x48	2450	0.02969820784232
48x96	9506	0.02966180825828
96x192	37442	0.02965270370808

Lagrange Q9			Lagrange Q16		
Mesh	DOF	U^h	Mesh	DOF	U^h
3x6	182	0.02970101373401	2x4	182	0.02965327376971
6x12	650	0.02965318188484	4x8	650	0.02964975296446
12x24	2450	0.02964989418870	8x16	2450	0.02964966996157
24x48	9506	0.02964968266120	16x32	9506	0.02964966846707
48x96	37442	0.02964966933301	32x64	37442	0.02964966844276

NURBS $p = 2$			NURBS $p = 3$		
Mesh	DOF	U^h	Mesh	DOF	U^h
5x10	168	0.02965740783282	4x8	154	0.02964986407434
11x22	624	0.02964999723578	10x20	598	0.02964966945433
23x46	2400	0.02964968556157	22x44	2350	0.02964966845279
47x94	9408	0.02964966942255	46x92	9310	0.02964966844251
95x190	37248	0.02964966850106	94x188	37054	0.029649668442378

Figure 6.5 shows an error plot of these results. The error in the strain energy is defined as

$$\|e\|_E^2 = \frac{\left|U - U^h\right|}{U} \tag{6.2}$$

where U is the exact strain energy and U^h is the corresponding strain energy of the FE solution.

Again, the results using isogeometric analysis are more effective in terms of less error compared to FEA. Since an exact solution is available, the meshes of different element types will converge towards this value with mesh refinement. As expected, the slopes of the error lines for FEA and isogeometric analysis of equal element orders are approximately the same, which means that the convergence rates are the same. However, the accuracy is better using isogeometric analysis as opposed to traditional FEA. Note that the error is plotted against global degrees of freedom, and not element size, to be able to compare isogeometric analysis and FEA. This causes the slope of the error lines to not be exactly 1, 2 and 3 for 1st, 2nd and 3rd order elements, respectively. Also note that the error lines for isogeometric analysis are not completely straight due to non-uniform mesh refinement.

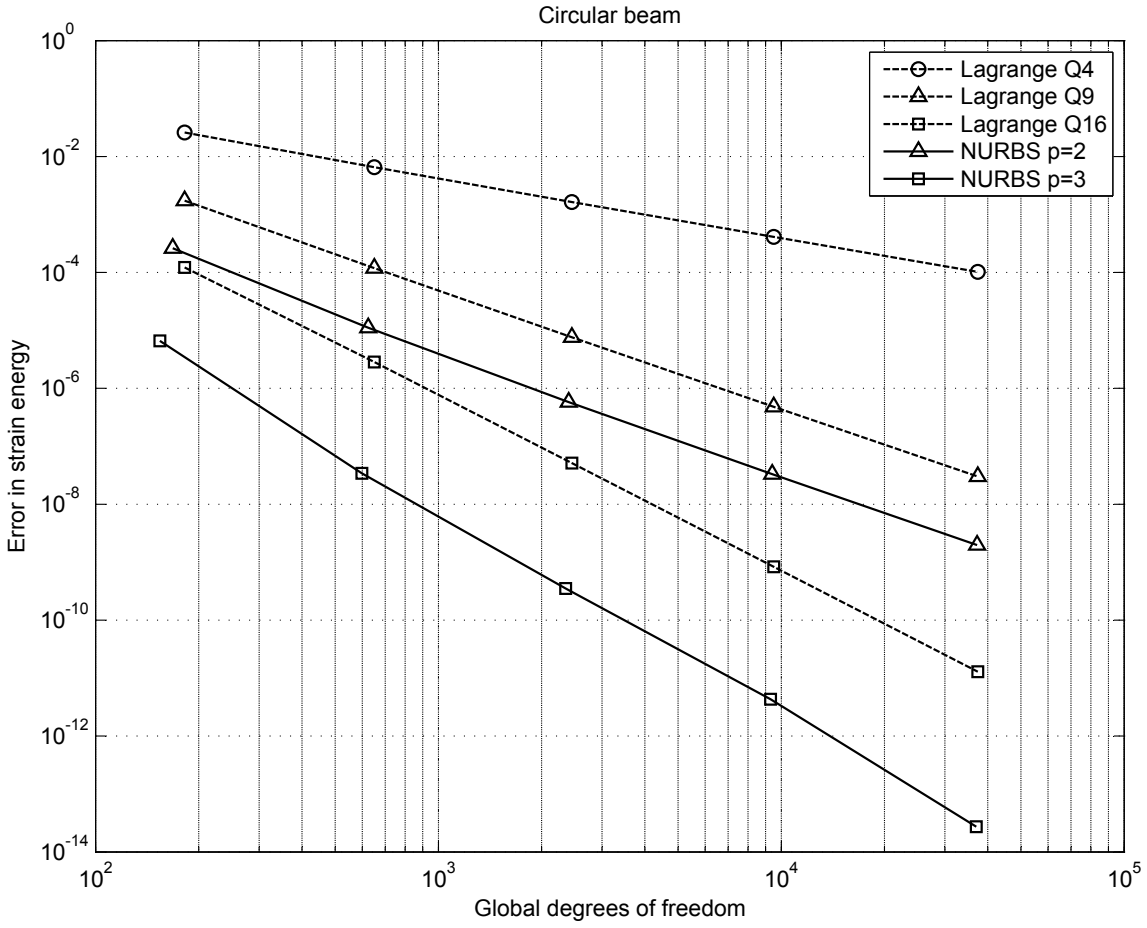


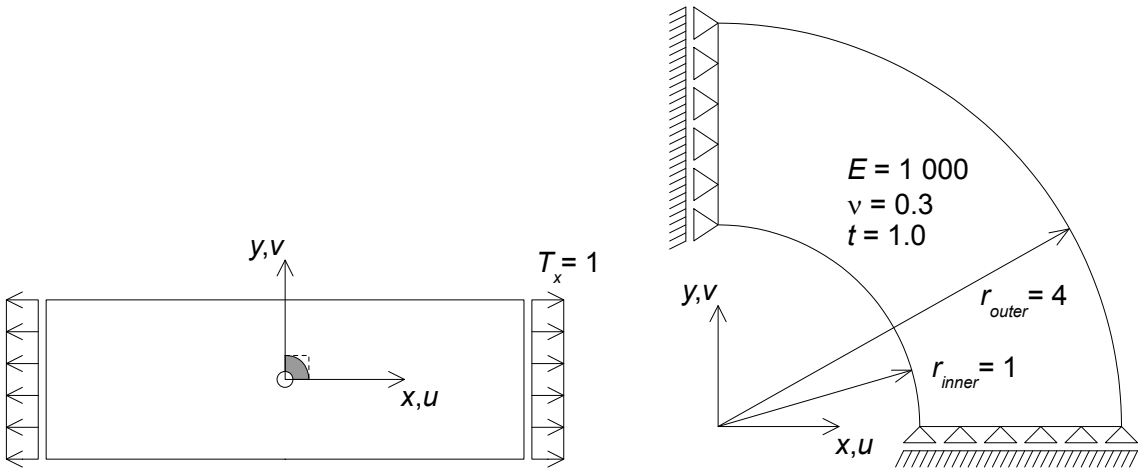
Figure 6.5: Error in strain energy of the circular beam

6.3 Infinite Plate with Circular Hole

The problem shown in Figure 6.6a consist of a plate which is infinitely large in the x and y direction, with a hole with radius $r_{inner} = 1$ in the centre of the plate. The plate is loaded

with far-field uniaxial tension, $T_x = 1$, and Figure 6.6b illustrates the part of the problem that is analysed due to symmetry. Note that this problem is often modelled as a quarter of the geometry with the outer edge square shaped [6, 11, 9, 30]. Here, a circular outer edge is chosen to avoid singularity in the geometric representation. Because of the convenient geometry, a total solution of the plate may be contemplated in terms of the strain energy. A parameter which reflects the total solution is often a better choice for numerical studies than a single value.

The plate is linear-elastic, and in a state of plane strain. Material properties and boundary conditions are given in Figure 6.6b.



(a) Infinite plate with circular hole subjected to far-field uniaxial tension

(b) Part of the infinite plate with circular hole that is analysed with material properties and boundary conditions.

Figure 6.6: The infinite plate with circular hole

The analytical solution to the problem is given in Timoshenko and Goodier [27] based on use of a stress function. The Cartesian stresses at an arbitrary point in the plate is given in Zienkiewicz and Zhu [30],

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x(r,\theta) \\ \sigma_y(r,\theta) \\ \tau_{xy}(r,\theta) \end{bmatrix} = \begin{bmatrix} T_x \left\{ 1 - (\frac{r_{inner}}{r})^2 (\frac{3}{2}\cos 2\theta + \cos 4\theta) + \frac{3}{2}(\frac{r_{inner}}{r})^4 \cos 4\theta \right\} \\ T_x \left\{ -(\frac{r_{inner}}{r})^2 (\frac{1}{2}\cos 2\theta - \cos 4\theta) - \frac{3}{2}(\frac{r_{inner}}{r})^4 \cos 4\theta \right\} \\ T_x \left\{ -(\frac{r_{inner}}{r})^2 (\frac{1}{2}\sin 2\theta + \sin 4\theta) + \frac{3}{2}(\frac{r_{inner}}{r})^4 \sin 4\theta \right\} \end{bmatrix} \tag{6.3}$$

From the stresses, the exact strain energy of the analysed part may be evaluated,

$$U = \int_V \boldsymbol{\sigma}^T \boldsymbol{\varepsilon} dV = t \int_0^{\frac{\pi}{2}} \int_1^4 \boldsymbol{\sigma}^T \mathbf{E}^{-1} \boldsymbol{\sigma} r dr d\theta = -\frac{135}{32768} \frac{\pi(1024\nu^2 + 5\nu - 1019)}{E} \approx 0.01197664128784 \tag{6.4}$$

where $\mathbf{E}$ is the constitutive matrix of plane strain, defined by Eq. (2.2).

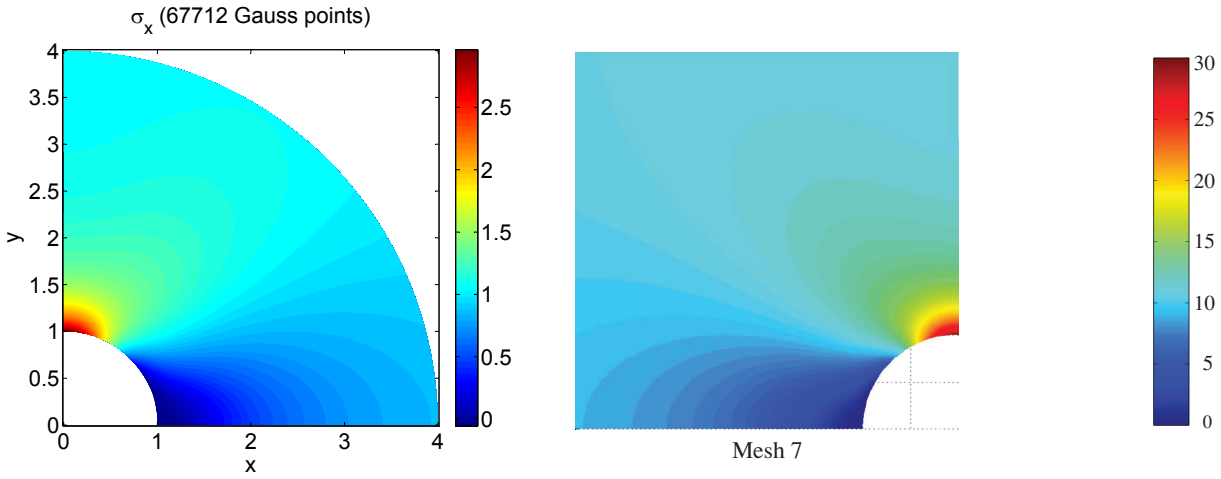
The exact stresses are applied to the system as a traction field at the outer edge,

$$\mathbf{\Phi} = \boldsymbol{\sigma}\hat{\mathbf{n}} = \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \frac{1}{\sqrt{x^2+y^2}} \begin{bmatrix} x \\ y \end{bmatrix} \tag{6.5}$$

where $\hat{\mathbf{n}}$ is the unit outward normal vector.

The infinite plate is modelled with five different meshes using 4-, 9- and 16-node Lagrange elements for FEA, and quadrilateral NURBS Bézier elements of polynomial order 2 and 3 for isogeometric analysis. The number of elements in the tangential direction is chosen to be twice the number of elements in the radial direction for all meshes. The coarsest meshes are similar to the circular beam (except that $r_{inner} = 1$ and $r_{outer} = 4$), see Figures 6.2 and 6.3. Uniform refinement is chosen for the FEA meshes. The meshes for isogeometric analysis are chosen so that the number of global degrees of freedom is as close up to FEA as possible, while still keeping the number of elements in the tangential direction twice the number of elements in the radial direction.

Figure 6.7a shows a contour plot of the normal stress σ_x for the second finest mesh using 3rd order NURBS elements. The stress distribution appears reasonable compared to the results obtained in Cottrell et al. [6], Figure 6.7b. By employing Eq. (6.3), the stress concentration $\sigma_x(r = r_{inner}, \theta = \frac{\pi}{2}) = 3$, which seems right. Note that the values from Cottrell et al. [6] are ten times higher due to that the traction value $T_x = 10$, as opposed to $T_x = 1$ in Figure 6.7a.



(a) Solution using 3rd order NURBS elements ($T_x = 1$)

(b) Solution from Cottrell et al. [6] ($T_x = 10$)

Figure 6.7: Normal stress σ_x for the infinite plate

Table 6.2 shows the results of the strain energy with 14 decimal places (13 significant digits) for the different meshes and methods. Isogeometric analysis display energy closer to the exact solution than FEA for approximately the same number of global degrees of freedom and the same order of elements. Higher order elements decrease the error. FEA using Q4 elements is farthest off the exact result, while isogeometric analysis using 3rd order elements is very close to the exact solution for the finest mesh. Note that the Lagrange Q16 elements perform poorer than the 2nd order NURBS elements for the three coarsest meshes, as opposed to the circular beam, see Table 6.1.

Table 6.2: Strain energy of the infinite plate (exact solution $U = 0.01197664128784$)

Lagrange Q4		
Mesh	DOF	U^h
6x12	182	0.01187038060514
12x24	650	0.01194415876726
24x48	2450	0.01196792769570
48x96	9506	0.01197441931851
96x192	37442	0.01197608294562

Lagrange Q9			Lagrange Q16		
Mesh	DOF	U^h	Mesh	DOF	U^h
3x6	182	0.01194105981301	2x4	182	0.01196018040464
6x12	650	0.01197075473017	4x8	650	0.01197503568669
12x24	2450	0.01197606693368	8x16	2450	0.01197657296440
24x48	9506	0.01197659949914	16x32	9506	0.01197663967925
48x96	37442	0.01197663856205	32x64	37442	0.01197664126006

NURBS $p = 2$			NURBS $p = 3$		
Mesh	DOF	U^h	Mesh	DOF	U^h
5x10	168	0.01196367734967	4x8	154	0.01197273772528
11x22	624	0.01197570293841	10x20	598	0.01197659081089
23x46	2400	0.01197658800256	22x44	2350	0.01197664069655
47x94	9408	0.01197663824978	46x92	9310	0.01197664127917
95x190	37248	0.01197664110484	94x188	37054	0.01197664128770

Figure 6.8 shows an error plot of these results. The error in the strain energy is calculated using Eq. (6.2). The performance of isogeometric analysis is better than FEA in terms of higher accuracy. Since an exact solution is available, the meshes of different element types will converge towards this value with mesh refinement, but this error plot differ from the error plot for the circular beam in Figure 6.5. Although uniform mesh refinement is carried out for FEA, clearly the error lines for Q9 and especially Q16 elements are not straight. The traditional finite elements seem to converge better with increased degrees of freedom, which gives downward bent error lines.

An explanation to this may be that in FEA, as opposed to isogeometric analysis, the circular edge where the load is applied cannot be represented exactly. This disadvantage for FEA is more significant for coarse meshes, perhaps resulting in that 3rd order Lagrange elements perform poorer than 2nd order NURBS elements to start with. When the number of nodes increases along the circular boundary, the consistent nodal loads are also applied to a set of nodes with more accurate location. This may give additional increased accuracy, and the convergence rate for FEA appears to approach the convergence rate for isogeometric analysis for elements of equal order.

Also, the error is less widespread for the coarse meshes of both FEA and isogeometric analysis compared to the circular beam. This may also affect the alignment of the error lines relative to each other.

The error lines for isogeometric analysis are fairly straight despite of not completely uniform mesh refinement. NURBS elements can represent the exact circular edge regardless of the number of elements and the polynomial order (if 2nd order or higher). In this example, it may seem like the benefit of no geometrical error in isogeometric analysis shows to advantage.

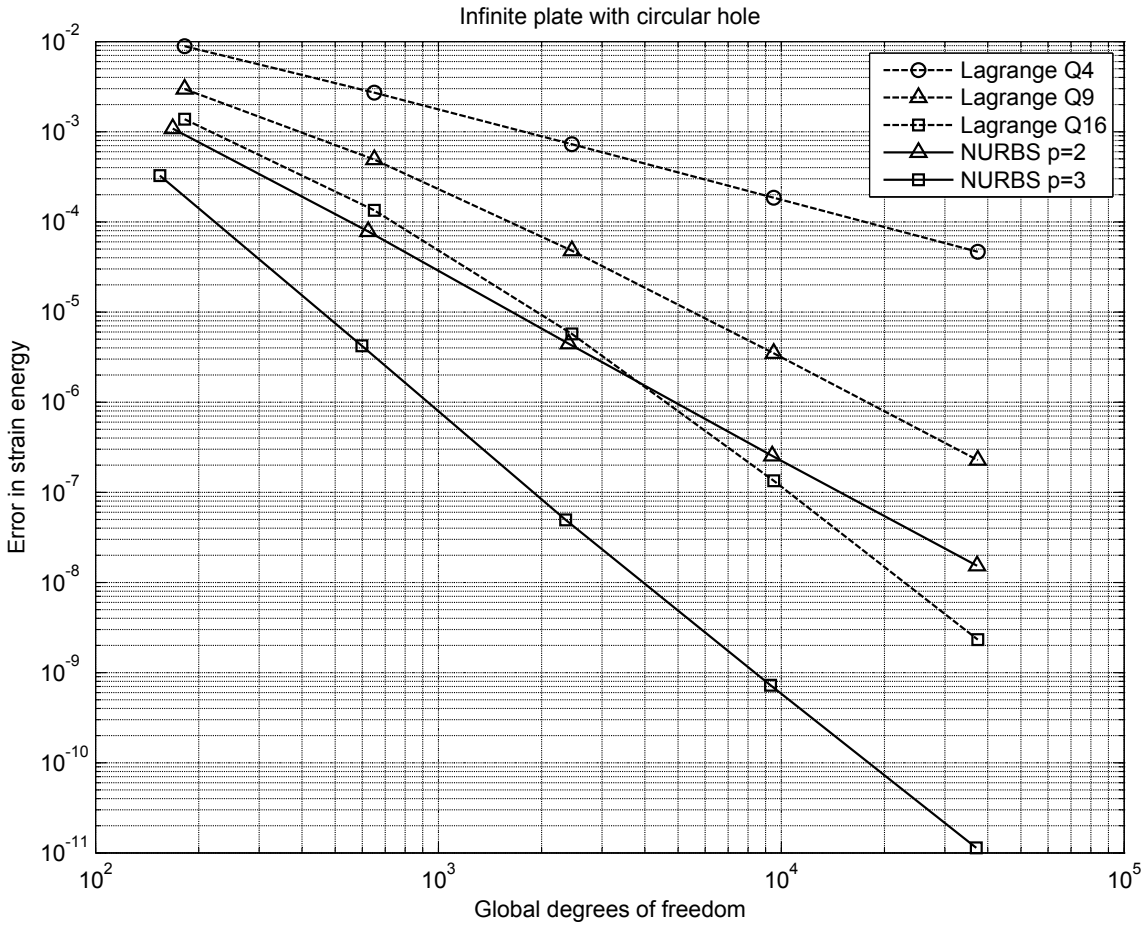
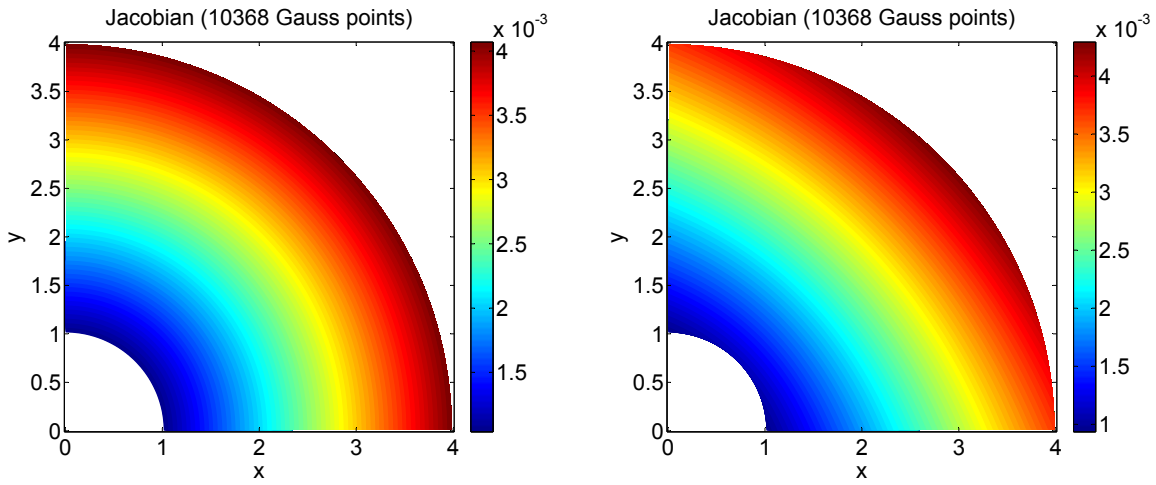


Figure 6.8: Error in strain energy of the infinite plate with circular hole

A note should be made on the Jacobian for curved geometries modelled with isogeometric elements as opposed to traditional finite elements. This may also explain the difference observed between FEA and isogeometric analysis regarding convergence. In Figure 6.9, the Jacobian for the infinite plate is plotted for FEA and isogeometric analysis using 2nd order elements. The number of elements is equal for both meshes such that the number of Gauss points is equal, but the number of global degrees of freedom is less for the isogeometric mesh.

The Jacobian for the FE mesh is constant along a circular line, which means that there is a linear mapping between the reference element and the physical space. However, this is not the case for the isogeometric mesh as illustrated in Figure 6.9b. The Jacobian along a circular line for this mesh will be slightly higher at the middle than close to the boundaries. Therefore, there will not be a linear mapping between the parameter space and the physical space for the isogeometric mesh.

For the infinite plate, it is important to evaluate the physical coordinates x, y by interpolating the basis functions when forming the consistent nodal loads for the traction field. The angle θ is then $\theta = \arctan(y/x)$. For FEA, evaluating the angle by $\theta = \arctan(y/x)$ or directly as the aperture angle has no significance, but the latter method will give the wrong solution for isogeometric analysis.



(a) Lagrange Q9 (1152 elements, 9506 DOF, 10368 Gauss points)

(b) NURBS $p = 2$ (1152 elements, 2600 DOF, 10368 Gauss points)

Figure 6.9: The Jacobian for the infinite plate, Lagrange Q9 and NURBS $p = 2$

The reason for the non-constant Jacobian for isogeometric analysis lies in the nature of the control points. For a curved line, the control points (both NURBS and Bézier) are aligned a bit outside the actual curve, even when the mesh is refined and this is barely visual (see Figure 6.3). Since there is a linear mapping in this case for FEA but not for isogeometric analysis, this leads to Bézier elements not being aligned with traditional finite elements. Figure 6.10 illustrates this by plotting a coarse mesh of the infinite plate using 2nd order elements for both FEA and isogeometric analysis. Black is used for Bézier elements and control points, while red lines indicate where the traditional FE element boundaries depart from the Bézier elements. The red dots are nodes of the Q9 elements. Here it may be seen that the angle θ is not equal for all elements in isogeometric analysis.

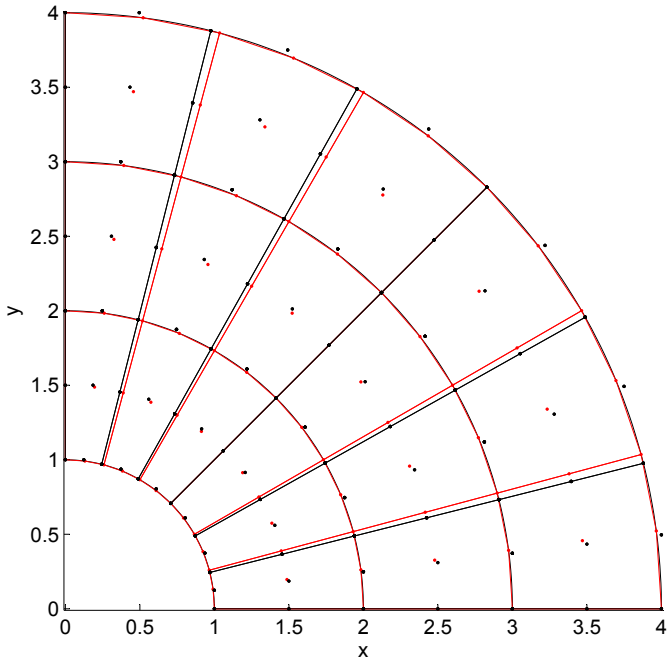
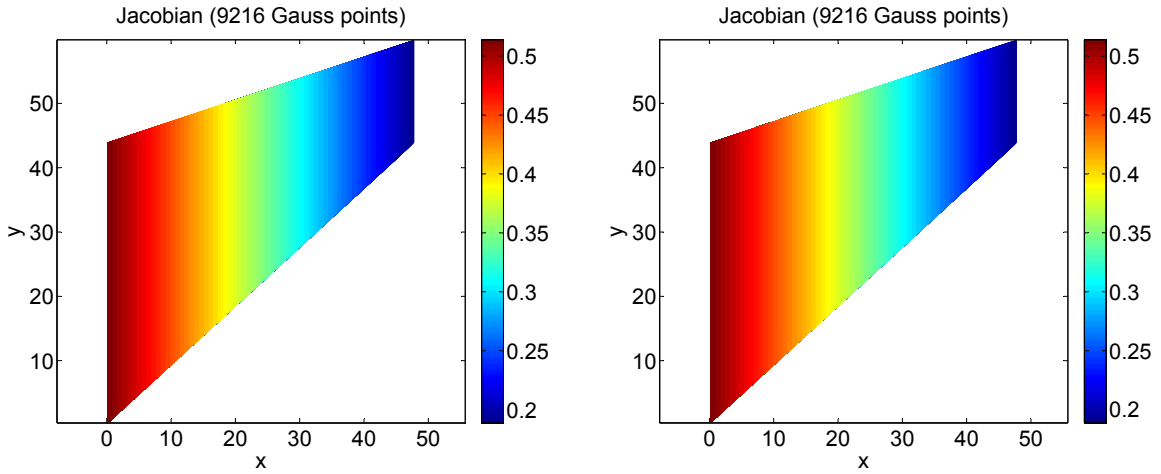


Figure 6.10: Infinite plate, meshes of 2nd order elements. Bézier physical mesh with Bézier control points (black) and traditional FE mesh with nodes (red).

This difference has no affect on neither the results (provided that the angle is evaluated correctly isogeometric analysis still performs better) nor the isogeometric data structures based on Bézier extraction. Locally represented basis functions are still provided for isogeometric analysis, and this eases the implementation.

Along straight lines, the Jacobian for both FEA and isogeometric analysis will be constant as shown in Figure 6.11 for Cook's problem as an example. Linear mapping is provided for both FEA and isogeometric analysis. For such geometries, the Bézier elements and the traditional finite elements will coincide completely.



(a) Lagrange Q9 (1024 elements, 8450 DOF, 9216 Gauss points)

(b) NURBS $p = 2$ (1024 elements, 2312 DOF, 9216 Gauss points)

Figure 6.11: The Jacobian for Cook's problem, Lagrange Q9 and NURBS $p = 2$

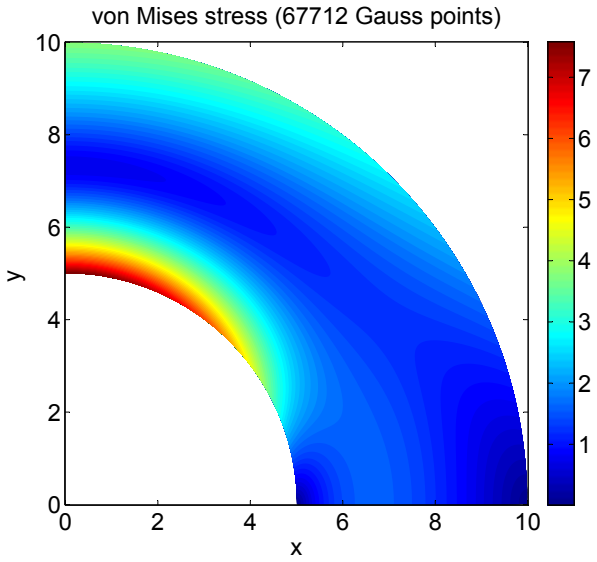
6.4 Studies on T-Meshes from Rhino

Some studies are performed on T-meshes modelled in Rhino 4.0 with T-Splines 3 for Rhino to investigate performance and applications.

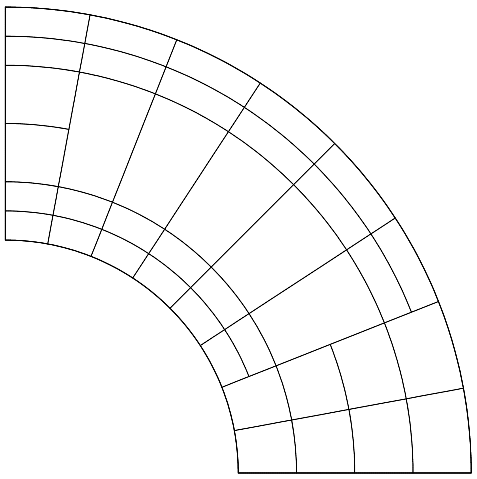
6.4.1 Circular Beam

The circular beam is modelled with different meshes using T-Splines for Rhino to study accuracy. This is done by first modelling a NURBS surface and then converting it to a T-spline, before refining the mesh by inserting points. When converting to a T-spline, it is important to choose degree elevation of the surface (from $p = 2$ for the NURBS surface to $p = 3$ for the T-spline), as opposed to rebuilding of the surface, which is the standard option [22]. This is because the latter option will give an approximated surface.

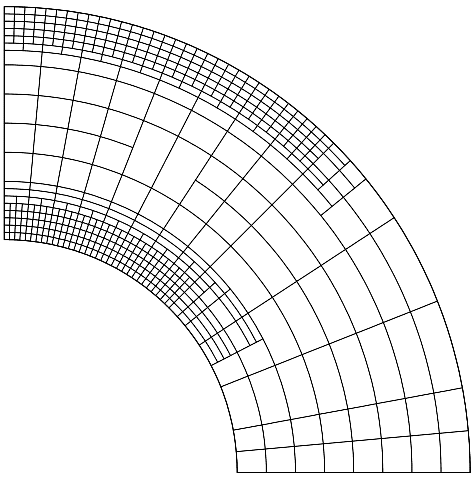
The advantage of T-splines compared to NURBS is the possibility to locally refine the mesh at necessary regions. How will a T-mesh manually refined in Rhino perform compared to a NURBS mesh? A look at the von Mises stress for the circular beam in Figure 6.12a suggests that the T-mesh should be refined especially at the inner and outer boundary for about half the arc length. The resulting T-meshes are illustrated in Figures 6.12b - 6.12d.



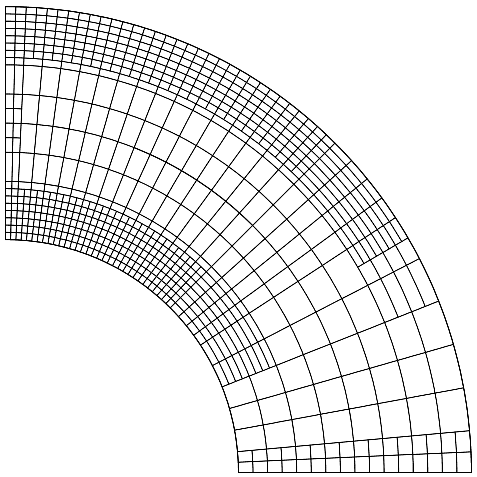
(a) von Mises stress for the circular beam



(b) Coarse T-mesh



(c) Fine T-mesh 1



(d) Fine T-mesh 2

Figure 6.12: T-mesh considerations for the circular beam

The modelling strategies for the fine T-meshes are a bit different. For fine T-mesh 1, refinements are performed basically for areas with stress values of signification. Fine T-mesh 2 is also refined for areas elsewhere to ensure that the total solution will not be delayed because of a too coarse mesh outside the areas of signification.

Table 6.3 shows the strain energy of the three T-meshes. These results are plotted in the error plot for the circular beam in Figure 6.13. Considering the number of global degrees of freedom, the performance of the T-meshes appears to be good compared to 1st and 2nd order Lagrange and NURBS elements. However, compared to Q16 Lagrange elements, it is uncertain whether the manually refined T-meshes are better or not. Perhaps fine T-mesh 1 is too coarse outside the areas of signification. Compared to 3rd order NURBS elements, none of the manually refined T-meshes manage to perform better.

Table 6.3: Strain energy of T-meshes, circular beam (exact solution $U = 0.02964966844238$)

Mesh	Elements	DOF	U^h
Figure 6.12b	47	178	0.02965039206827
Figure 6.12c	614	1366	0.02964968477295
Figure 6.12d	844	1924	0.02964966930160

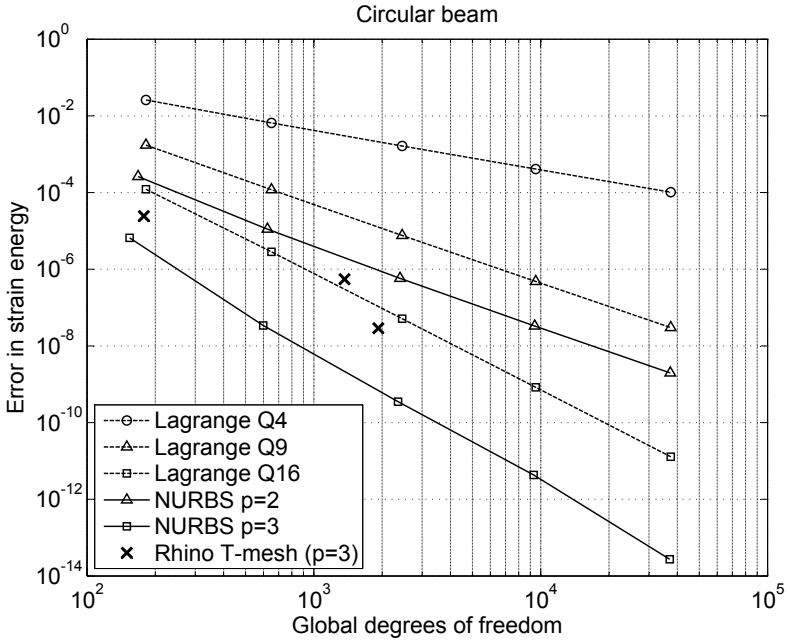
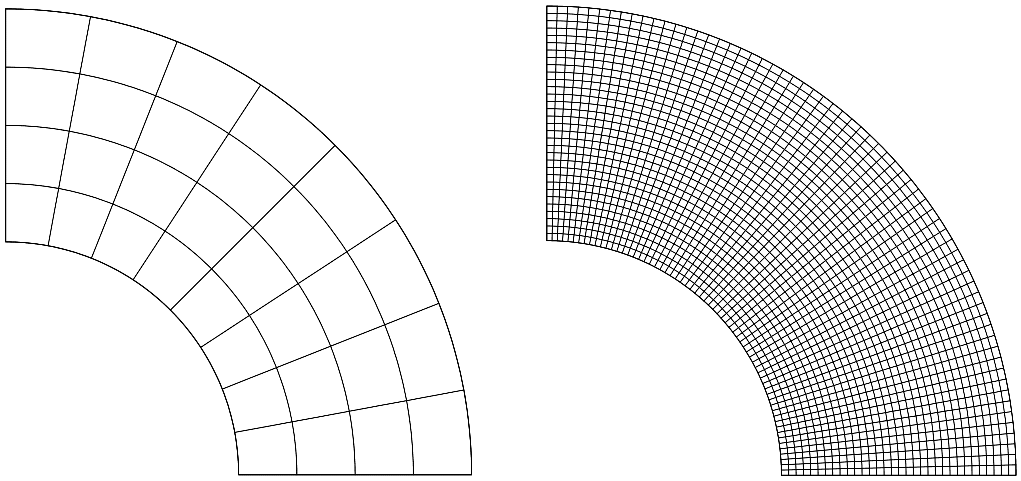


Figure 6.13: Error in strain energy of T-meshes for the circular beam

This is studied closer by modelling 3rd order NURBS meshes using T-Splines for Rhino and comparing these with equal NURBS meshes from the MATLAB program. The 3rd coarsest and the finest mesh from Rhino are illustrated in Figure 6.14.



(a) Coarse mesh from Rhino (4 x 8 elements) (b) Fine mesh from Rhino (32 x 64 elements)

Figure 6.14: 3rd order NURBS meshes modelled with T-Splines for Rhino

The results for the strain energy are given in Table 6.4. The strain energies of the two coarsest

meshes are equal.

Table 6.4: Strain energy of the circular beam, MATLAB and Rhino 3rd order NURBS meshes (exact solution $U = 0.02964966844238$)

Mesh		U^h
1 x 2 elements (40 DOF)	MATLAB	0.02981823414793
	Rhino	0.02981823414793
2 x 4 elements (70 DOF)	MATLAB	0.02965926069638
	Rhino	0.02965926069638
4 x 8 elements (154 DOF)	MATLAB	0.02964986407434
	Rhino	0.02964986407966
8 x 16 elements (418 DOF)	MATLAB	0.02964967209557
	Rhino	0.02964967209765
16 x 32 elements (1330 DOF)	MATLAB	0.02964966850911
	Rhino	0.02964966850430
32 x 64 elements (4690 DOF)	MATLAB	0.02964966844353
	Rhino	0.02964966844242

The error in the strain energy for the NURBS meshes is plotted in Figure 6.15. There are small variations except for the last point. The number of elements is doubled in both directions for each step of mesh refinement because of practical modelling reasons in Rhino. This causes the error lines to flatten out due to non-uniform mesh refinement (in isogeometric analysis).

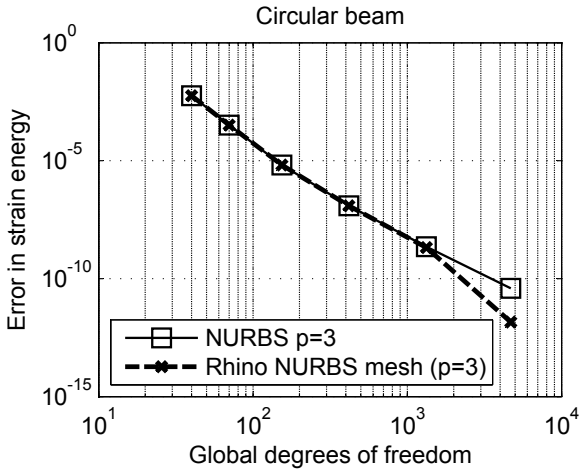


Figure 6.15: Error in strain energy of circular beam, MATLAB and Rhino meshes

A comparison of the control points and weights reveals small distinctions for the four finest meshes, with an order of magnitude of 10^{-9} for the worst cases. The reason may be numerical inaccuracy when splitting the elements over and over. This indicates that the better performance of the finest mesh may just be an accidental occurrence. The distinction in the strain energy in Table 6.4 is relatively constant for these meshes, equal to 10 decimal places when rounded. Geometrical inaccuracy is present, but increasing the tolerance of the units in Rhino does not change the results.

Although this explains why there is a small difference between the error of the meshes, the poorer performance of the T-meshes in Figure 6.12 is more significant than this distinction and can therefore not be caused by the geometrical data alone.

A final attempt of local refinement of the circular beam is done systematically by using the second finest NURBS mesh (16 x 32 elements) as the basis and perform local refinement at areas of signification (areas with higher von Mises stress and areas applied to boundary conditions), see Figure 6.16. The T-mesh consists of 1136 elements and 2602 DOF, and the strain energy is $U^h = 0.02964966845969$.

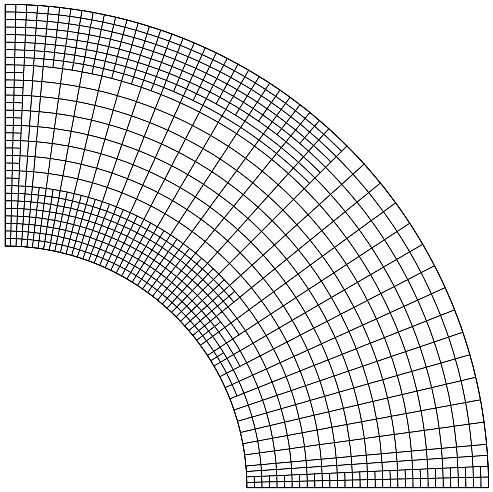


Figure 6.16: T-mesh refined using NURBS mesh 16 x 32 elements as basis

The result of this action is again plotted in the error plot to investigate performance, see Figure 6.17. The step from the NURBS 16 x 32 mesh to the T-mesh in Figure 6.16 is marked with a thick dashed line in Figure 6.17. The gain in reduced error from the local refinement compared to the increased degrees of freedom is negative compared to equal global refinement. For this simple geometry with no geometrical nor stress singularities, local refinement seems unnecessary. The indication is that better performance is obtained with a regular mesh than a locally refined mesh.

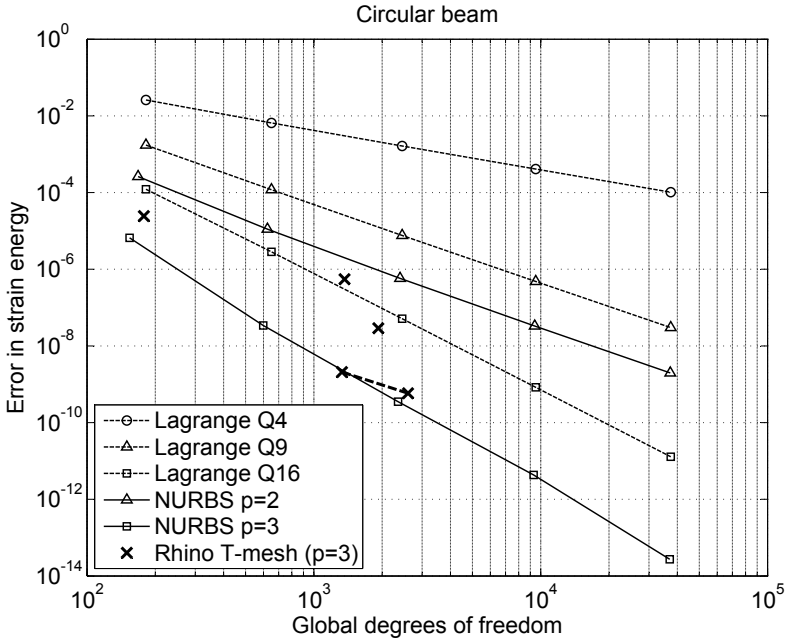


Figure 6.17: Error in strain energy of circular beam, systematically refined T-mesh

6.4.2 Machine Part

Another advantage of T-splines compared to NURBS is the possibility to create single surfaces as opposed to multiple-patch surfaces. The machine part in Figure 6.18 is a reconstruction of a problem pictured in Zienkiewicz et al. [29]. Because of the interior holes, the geometry cannot be modelled with a single NURBS surface. However, this is possible using T-splines.

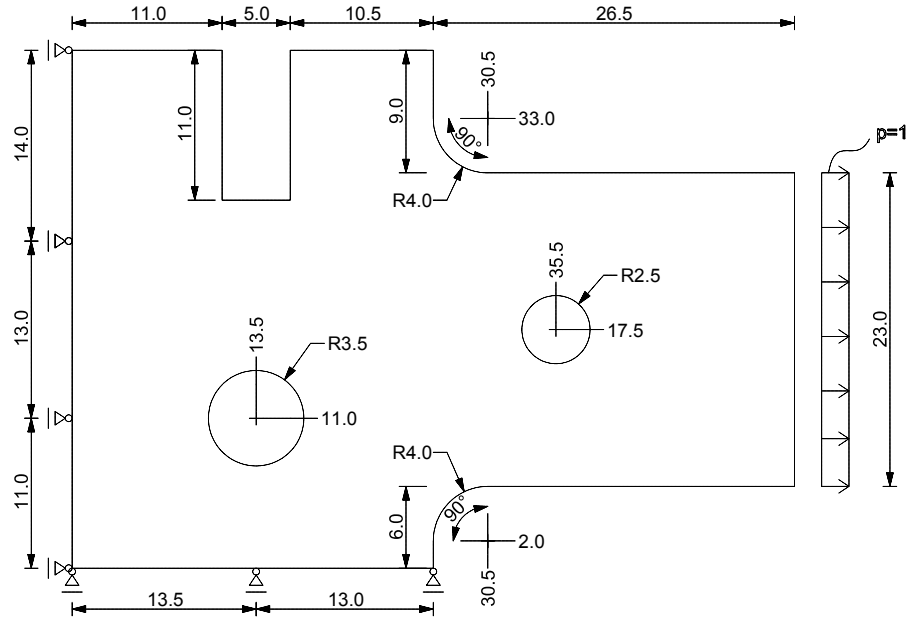
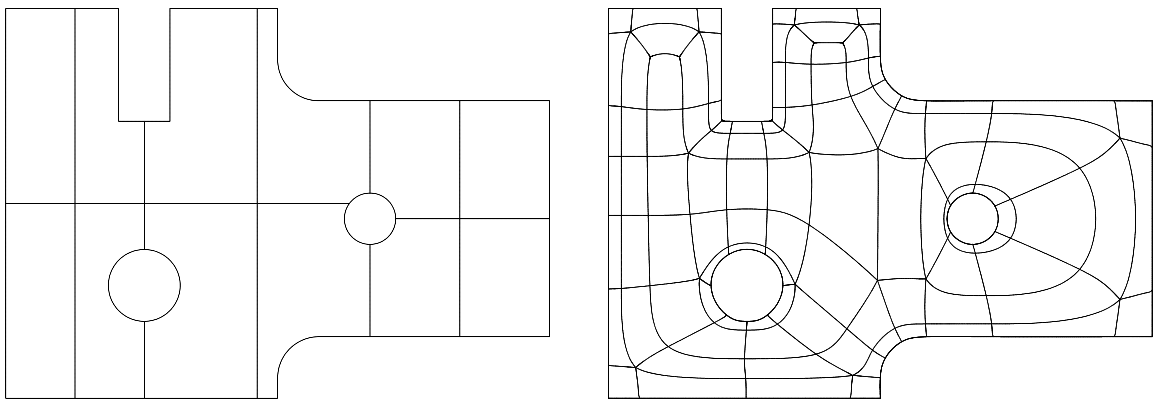


Figure 6.18: The geometry of the machine part with boundary conditions and surface traction

Figure 6.19a shows the machine part modelled with 3 NURBS surfaces (a NURBS polysurface) and Figure 6.19b illustrates how this can be done with a single T-spline. The T-spline surface is quite difficult to model compared to the NURBS model. This is because T-Splines for Rhino does not support direct conversion of trimmed NURBS surfaces [22], although there are ongoing work on this [26]. Therefore, the T-spline had to be modelled directly as a T-spline surface. This can be accomplished by using the T-spline command tsFromLines.

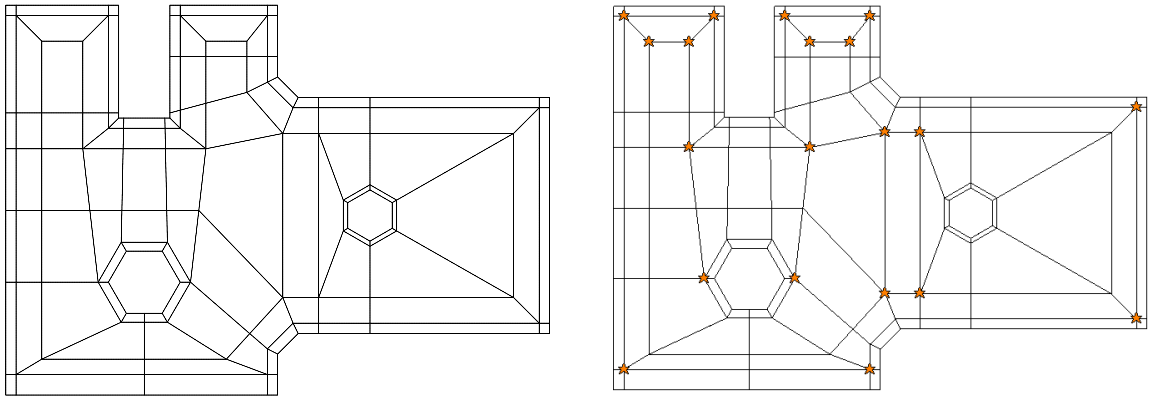


(a) NURBS surfaces (b) Single T-spline surface

Figure 6.19: NURBS and T-spline surface of the machine part

To use this command, first the control polygon of the T-spline is modelled, which is the boundaries of the geometry defined by the control points. Thereafter, the control polygon must be

connected correctly as described in Sederberg [22]. This is not straightforward, and the writer thanks an experienced industrial modeller on the T-splines forum [26] for helping out. The result is illustrated in Figure 6.20a. Note that each vertex equals a control point. Finally, all lines must be split into single segments using the tsSplitCurves command before the tsFromLines command can be employed. The command tsPull may then be used to adjust the surface and control points.



(a) The control polygon of the machine part (b) Star points in the control polygon

Figure 6.20: The control polygon of the machine part

The machine part is imported into the FE solver for the purpose of analysis. The geometry turned out to be quite complex, as the model consists of 266 DOF and a total of 1948 elements. Figure 6.21 shows the Bézier elements for the machine part. The surface contains no T-junctions (do not mistake the crossing lines of the Bézier elements for T-junctions), however 20 extraordinary points called star points (illustrated in Figure 6.20b).

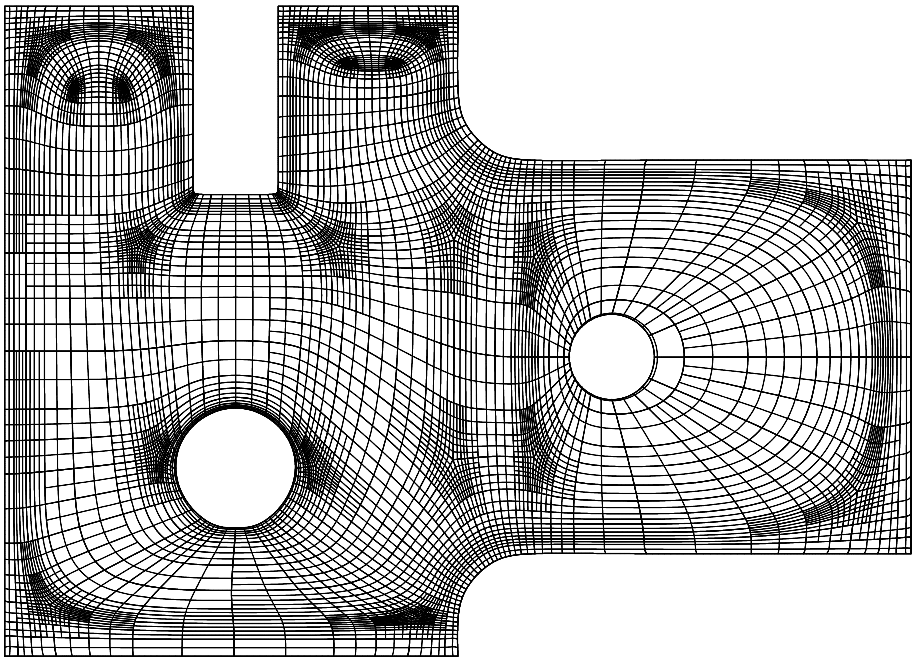


Figure 6.21: Bézier elements of the machine part

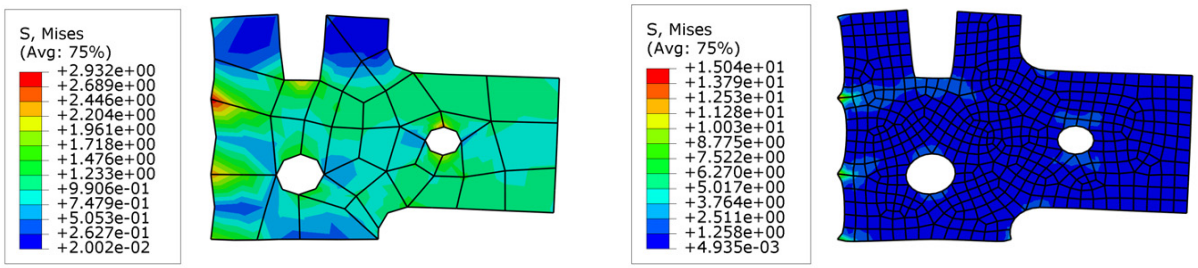
These star points allow a T-spline to be non-rectangular and therefore enables untrimmed holes in the surface to be created. However, the points are complex as they are related to so-called

subdivision surfaces and not NURBS as T-junctions are [22]. They cause areas around them to be extra dense with elements as seen in Figure 6.21. The IEN array also reveals that some elements are supported by up to 24 basis functions, while others are only supported by 11. Recall the quarter disk from Section 5.3 which has 16 - 20 supported basis functions. This range is more usual for simple T-meshes, and often for regular T-meshes there are just either 16 or 17 supported basis functions. Thus the T-spline for the machine part is not actually defined by a T-mesh.

For the analysis, the machine part is assumed linear elastic and in a state of plane stress, and material properties are set to $E = 1000$ and $\nu = 0.3$.

To apply the uniformly distributed load at the right hand side (see Figure 6.18), the script for numerical integration of the consistent nodal loads must loop through all element (because of no tensor product structure) and include a search of the control points aligned at this boundary. This increases the computational cost compared to a structured NURBS surface.

The T-spline for the machine part is compared to solutions from Abaqus/CAE 6.9-2 using CPS8 (8-node biquadratic plane stress quadrilateral) elements with full integration. Two coarse meshes and one fine mesh are modelled. The very coarse and fine mesh are illustrated in Figure 6.22.



(a) Very coarse mesh (36 elements, 278 DOF) (b) Fine mesh (515 elements, 3332 DOF)

Figure 6.22: The machine part modelled in Abaqus

Table 6.5 shows the results for the strain energy and the displacement u at the top of the right boundary for the different meshes. The T-spline mesh and the very coarse Abaqus mesh contain approximately the same number of global degrees of freedom, and the T-spline mesh performs considerably better than this Abaqus mesh. The T-spline mesh also displays displacement u at the top of the right boundary closer to the fine Abaqus mesh solution than the medium coarse mesh. Although the order of the T-spline elements is one degree higher than the order of the CPS8 elements, the T-spline mesh for the machine part can be said to perform satisfactorily. However, a NURBS surface modelled with multiple patches may be more convenient for this problem at the present state.

Table 6.5: Strain energy and displacement u at the top right boundary for the machine part

Mesh	Elements	DOF	Strain energy	Displ. u top right boundary
T-spline mesh	1948	266	1.45	0.0744
Very coarse (Abaqus)	36	278	1.39	0.0660
Coarse (Abaqus)	80	578	1.53	0.0732
Fine (Abaqus)	515	3332	1.66	0.0783

Chapter 7

Concluding Remarks

Isogeometric analysis resembles FEA much in its abstract formulation, meaning that the majority of the foundation as presented for FEA is persistent. The main distinction is that Lagrange shape functions are replaced by NURBS basis functions. This action brings along some considerable modifications regarding the mesh build up and an additional space for mapping. Nodes are replaced by control points, and the connection between control points and elements is not unique for the element. In general, common element properties connected to its shape functions like in FEA are lost, forcing a more general formulation of various parts in the construction of the global system matrices.

The Bézier extraction operator is significantly easing the implementation of isogeometric analysis in an existing finite element framework, since the necessary changes are confined to the shape function routine. From this routine, the basis functions are defined on the element level similar to FEA. Data structures for FEA may therefore be utilized for isogeometric analysis directly, as the latter no longer has mappings to the parameter space nor the tensor product structure in the construction of the stiffness matrix. The eased implementation is at the cost of a slight increase in computational effort compared to conventional isogeometric analysis.

As shown in two plane condition cases, use of NURBS in analysis display increased accuracy compared to traditional FEA. Generally, the convergence rates are the same for elements of equal order, but for element meshes of approximately equal degrees of freedom, isogeometric analysis will produce a smaller error compared to FEA.

The introduction of T-splines as a generalization of NURBS allows for local refinement. This provides for complex geometries to be created and an adaptive isogeometric analysis. However, there is still much to explore about properties and refinement of T-splines.

Chapter 8

Further Work

Isogeometric analysis is a large field, making the possibilities for further work numerous. The subject is also of great immediate interest, continuously revealing new topics to study.

From this thesis' perspective, it would be natural to implement Bézier extraction of T-splines for two-dimensional problems. T-splines are a challenge compared to NURBS especially when it comes to refinement schemes. Refinement of analysis-suitable T-splines proposed by Scott et al. [20] seems promising. There is much to explore about T-splines concerning adaptive refinement since they are not predictable in the way a NURBS mesh is. Adaptive refinement is dynamic gridding based on features of the results as the computation progresses. Such refinement schemes require acute assumptions and intelligent algorithms.

In the step towards integration of CAD and FEA, there is also need for efficient methods for converting a T-spline surface to a solid for analysis. How are the inner elements of a T-spline solid? A code which handles three-dimensional problems is desired. If the code is based on NURBS Bézier elements, an expansion to T-splines is roughly already available. Three-dimensional objects from T-Splines for Rhino may then be studied further.

Locally refined B-splines (LR B-splines) are an alternative to T-splines. A study on adaptive refinement using LR B-splines is performed by Kvamsdal et al. [12], and LR B-splines are promising for analysis because they do not rely on a specific grid structure as T-splines. LR B-splines allow for local refinement of the B-spline basis functions directly. Refinement is therefore performed on the mesh defined by the knots rather than on the control grid defined by the control points. The results seem to indicate that regularity is of more importance than locality, which is quite interesting. As for T-splines, partition of unity and linearly independence are also issues for LR B-splines.

Isogeometric analysis based on Bézier extraction provides a data structure which can be implemented into existing FE codes. This renders possibility to define distinct isogeometric elements in a FEA software similar to that one may choose quadrilateral elements, triangular elements, plate or shell elements etc. with appurtenant element topology and shape functions. The idea may be clear, but details in the build-up of a specific FEA program, for example Abaqus, must be studied to survey which changes and specifications that need to be made.

Isogeometric analysis may be applied to all fields where FEA is used today to investigate feasibility and performance. Examples are linear fracture mechanics, explicit and implicit dynamic analyses, isogeometric plate and shell elements and non-linear analyses like contact problems and finite deformation. The latter is under ongoing research by Mathisen et al. [16]. An approach to these topics may be done by investigating benchmark problems. The use of T-splines may be more of current interest than NURBS for isogeometric analysis, but one must be aware of that far from all problems are best analysed with T-splines.

From an industrial perspective, isogeometric analysis applied to engineering problems like marine or offshore structures, is particularly interesting. The investigation can be performed based on for example *a posteriori* error estimation. However, note that there are still restrictions to what can be modelled due to limited commercial and also educational software.

Bibliography

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

[2] Bell, K. (1994). *Matrisestatikk* (2 ed.). Tapir, Trondheim.

[3] Borden, M. J., M. A. Scott, J. A. Evans, and T. J. R. Hughes (2011). Isogeometric finite element data structures based on Bézier extraction of NURBS. *International Journal for Numerical Methods in Engineering 86*, n/a.

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

[5] Cook, R. D., D. S. Malkus, M. E. Plesha, and R. J. Witt (2002). *Concepts and Applications of Finite Element Analysis* (4 ed.). John Wiley & Sons, Hoboken.

[6] Cottrell, J. A., T. J. Hughes, and Y. Bazilevs (2009). *Isogeometric Analysis: Towards Integration of CAD and FEA*. John Wiley & Sons, Chichester.

[7] Cox, M. G. (1972). The Numerical Evaluation of *B*-Splines. *Journal of the Institute of Mathematics and its Applications 10*, 134 – 149.

[8] de Boor, C. (1972). On Calculating with *B*-Splines. *Journal of Approximation Theory 6*, 50 – 62.

[9] Dörfel, M. R., B. Jüttler, and B. Simeon (2010). Adaptive isogeometric analysis by local h-refinement with T-splines. *Computer Methods in Applied Mechanics and Engineering 199*, 264 – 275.

[10] Finnigan, G. T. (2008). Arbitrary Degree T-Splines. Master's thesis, Department of Computer Science, Brigham Young University.

[11] Hughes, T., J. Cottrell, and Y. Bazilevs (2005). Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering 194*, 4135 – 4195.

[12] Kvamsdal, T., K. A. Johannessen, and T. Dokken (2011). Adaptive refinement in isogeometric analysis using LR B-splines. In B. Skallerud and H. I. Andersson (Eds.), *MekIT'11 Sixth National Conference on Computational Mechanics*, pp. 157 – 170. Tapir Academic Press, Trondheim.

[13] Li, J. (2009). Isogeometric Finite Element Analysis Using T-Splines. Master's thesis, Department of Civil and Environmental Engineering, Brigham Young University.

[14] Li, X. and M. Scott (2011). On the Nesting Behavior of T-splines. Technical report, ICES REPORT 11-13, The Institute for Computational Engineering and Sciences, The University of Texas at Austin.

[15] Li, X., J. Zheng, T. Sederberg, T. Hughes, and M. Scott (2010). On Linear Independence of T-splines. Technical report, ICES REPORT 10-40, The Institute for Computational Engineering and Sciences, The University of Texas at Austin.

[16] Mathisen, K. M., K. M. Okstad, T. Kvamsdal, and S. B. Raknes (2011). Isogeometric Analysis of Finite Deformation Elastic and Elastoplastic Solid Problems. In B. Skallerud and H. I. Andersson (Eds.), *MekIT'11 Sixth National Conference on Computational Mechanics*, pp. 209 – 222. Tapir Academic Press, Trondheim.

[17] Nguyen, T. N. (2010). Isogeometric Analysis vs. Finite Element Analysis in Solid and Structural Mechanics. Project work in Computational Mechanics, Department of Structural Engineering, Norwegian University of Science and Technology (NTNU).

[18] Nguyen, T. N., O. J. Fredheim, K. M. Mathisen, and K. A. Johannessen (2011). Isogeometric finite element analysis based on Bézier extraction of NURBS and T-splines. In B. Skallerud and H. I. Andersson (Eds.), *MekIT'11 Sixth National Conference on Computational Mechanics*, pp. 239 – 256. Tapir Academic Press, Trondheim.

[19] Piegl, L. and W. Tiller (1997). *The NURBS Book* (2 ed.). Springer-Verlag, Berlin Heidelberg.

[20] Scott, M., X. Li, T. Sederberg, and T. Hughes (2011). Local Refinement of Analysis-Suitable T-splines. Technical report, ICEC REPORT 11-06, The Institute for Computational Engineering and Sciences, The University of Texas at Austin.

[21] Scott, M. A., M. J. Borden, C. V. Verhoosel, T. W. Sederberg, and T. J. R. Hughes (2011). Isogeometric finite element data structures based on Bézier extraction of T-splines. *International Journal for Numerical Methods in Engineering 86*, n/a.

[22] Sederberg, M. (2011). *T-Splines 3 User manual*. T-Splines, Inc.

[23] Sederberg, T. W., D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, and T. Lyche (2004). T-spline Simplification and Local Refinement. *ACM Transactions on Graphics 23*, 276 – 283.

[24] Sederberg, T. W., J. Zheng, A. Bakenov, and A. Nasri (2003). T-splines and T-NURCCs. *ACM Transactions on Graphics 22*, 477 – 484.

[25] Stahl, A. and T. Kvamsdal (2011). B-Spline Approximation for Visualisation of Isogeometric Analysis Results. In B. Skallerud and H. I. Andersson (Eds.), *MekIT'11 Sixth National Conference on Computational Mechanics*, pp. 313 – 328. Tapir Academic Press, Trondheim.

[26] T-Splines, Inc. (2004). T-Splines Forum. http://www.tsplines.com/forum/ [May 24-29, 2011].

[27] Timoshenko, S. P. and J. N. Goodier (1970). *Theory of Elasticity* (3 ed.). McGraw-Hill, New York.

[28] Yunus, S. M., S. Saigal, and R. D. Cook (1989). On improved hybrid finite elements with rotational degrees of freedom. *International Journal for Numerical Methods in Engineering 28*, 785 – 800.

[29] Zienkiewicz, O. C., R. L. Taylor, and J. Z. Zhu (2005). *The Finite Element Method: Its Basis and Fundamentals* (6 ed.). Elsevier Butterworth-Heinemann, Oxford.

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

Appendix A

Verification of Isogeometric Analysis using B-Splines

A.1 Cook's Problem

Cook's problem is a problem proposed by Cook as a test case for plane stress elements, and the outline of the problem is taken from Yunus et al. [28]. Geometry, boundary conditions and loads are shown in Figure A.1. The vertical displacement at point C is chosen for consideration. Since there is no known analytical solution of this problem, the result for a fine mesh modelled in COMSOL Multiphysics 3.5a using linear strain triangle (LST) is used for comparison. The model in COMSOL consists of 10248 elements and 41586 global degrees of freedom, giving the vertical displacement of point C to be 23.965979. Due to the singularity in the upper left corner as may be seen in Figure A.2, the solution will not converge towards any fixed value. This is a drawback of Cook's problem.

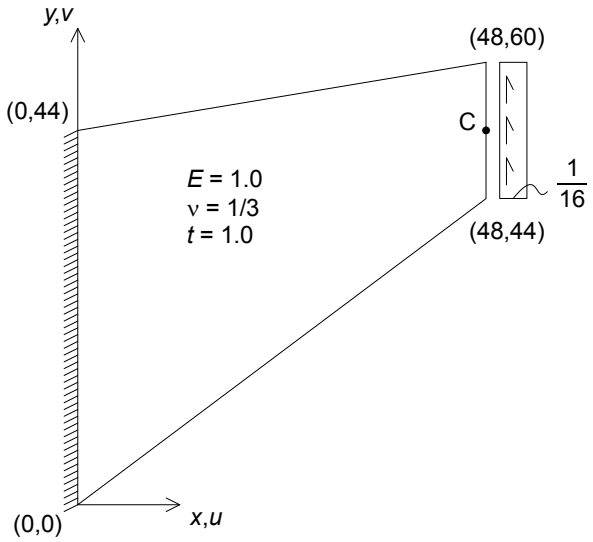


Figure A.1: The geometry of Cook's problem with material properties, boundary conditions and shear traction.

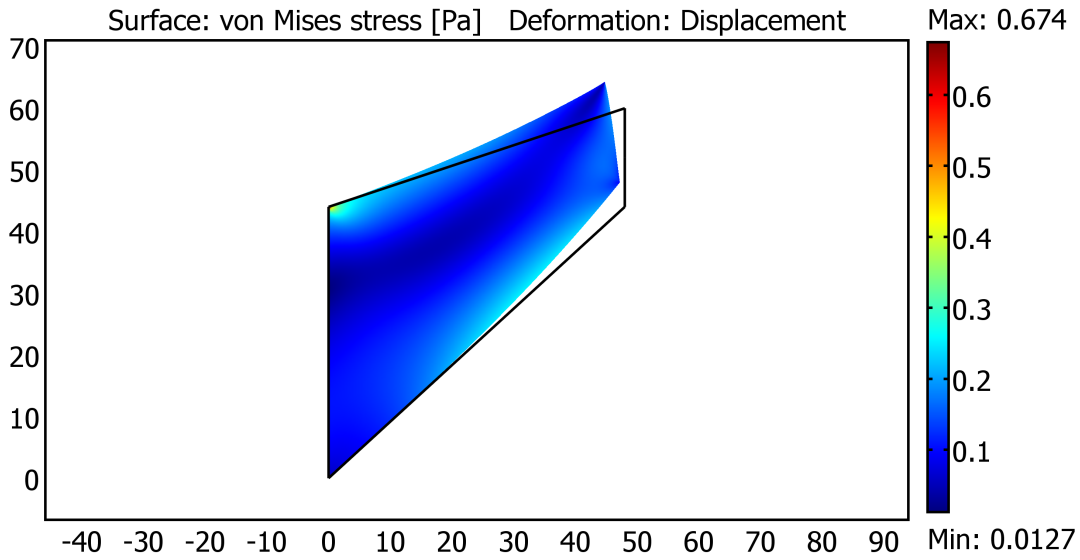


Figure A.2: Reference solution used for Cook's problem

Cook's problem is modelled with five different meshes using Q4 and Q9 elements for FEA, and elements of polynomial order 1 and 2 for isogeometric analysis. Uniform mesh refinement is carried out for FEA and also isogeometric analysis using 1st order elements, which means that the element size is halved in horizontal and vertical direction for each step of mesh refinement. For isogeometric analysis using 2nd order elements, the mesh refinement can also be regarded as uniform, since the number of global degrees of freedom equals the other cases. The shear force is modelled as a uniformly distributed load with value $1/16$ over the whole height, which sums up to a shear force equal to 1.

Table A.1 shows the results with 3 decimal places (5 significant digits) for the different meshes and methods. As expected, FEA using Q4 elements and isogeometric analysis using polynomial order 1 display the same results, while isogeometric analysis using 2nd order elements shows displacements closer to the reference result than FEA using Q9 elements for the same number of global degrees of freedom. The vertical displacement at point C in isogeometric analysis is found by employing Eq. (5.7) for $N_{i,p}(\xi) = 1$ (right boundary value) and interpolate the displacements using the basis functions $M_j(\eta)$ for the middle element in η direction.

Table A.1: Vertical displacement at point C of Cook's problem (reference solution $v_{C,ref} = 23.966$)

ALL	Lagrange Q4		B-splines $p = 1$		Lagrange Q9		B-splines $p = 2$	
DOF	Mesh	v_C	Mesh	v_C	Mesh	v_C	Mesh	v_C
50	4x4	18.299	4x4	18.299	2x2	23.289	3x3	23.621
162	8x8	22.079	8x8	22.079	4x4	23.840	7x7	23.905
578	16x16	23.430	16x16	23.430	8x8	23.925	15x15	23.942
2178	32x32	23.818	32x32	23.818	16x16	23.949	31x31	23.959
8450	64x64	23.925	64x64	23.925	32x32	23.961	63x63	23.964

Figure A.3 shows an error plot of these results. The error is calculated as

$$\|e\|^2 = \frac{v_{C,ref} - v_C}{v_{C,ref}} \tag{A.1}$$

where $v_{C,ref}$ is the reference solution and v_C is the corresponding vertical displacement at point C from the FE solution.

The singularity point causes the error lines when using 2nd order elements, i.e. Q9 or polynomial order 2, to be non-straight. However, the trend is still distinct: Isogeometric analysis is clearly more effective in terms of less error. Any possible difference between FEA using Q9 elements and isogeometric analysis using 2nd order elements regarding the convergence rate is difficult to observe since the lines are not straight. The error line of the meshes using 1st order elements on the other hand, seems linear in the logarithmic plot. A possible explanation to this may be that the singularity point is somehow not "observed" when using 1st order elements, meaning that the singularity is not distinct in the same way as when using 2nd order elements (or higher). The cause may be the restriction of linear elements, as they cannot exhibit pure bending [5].

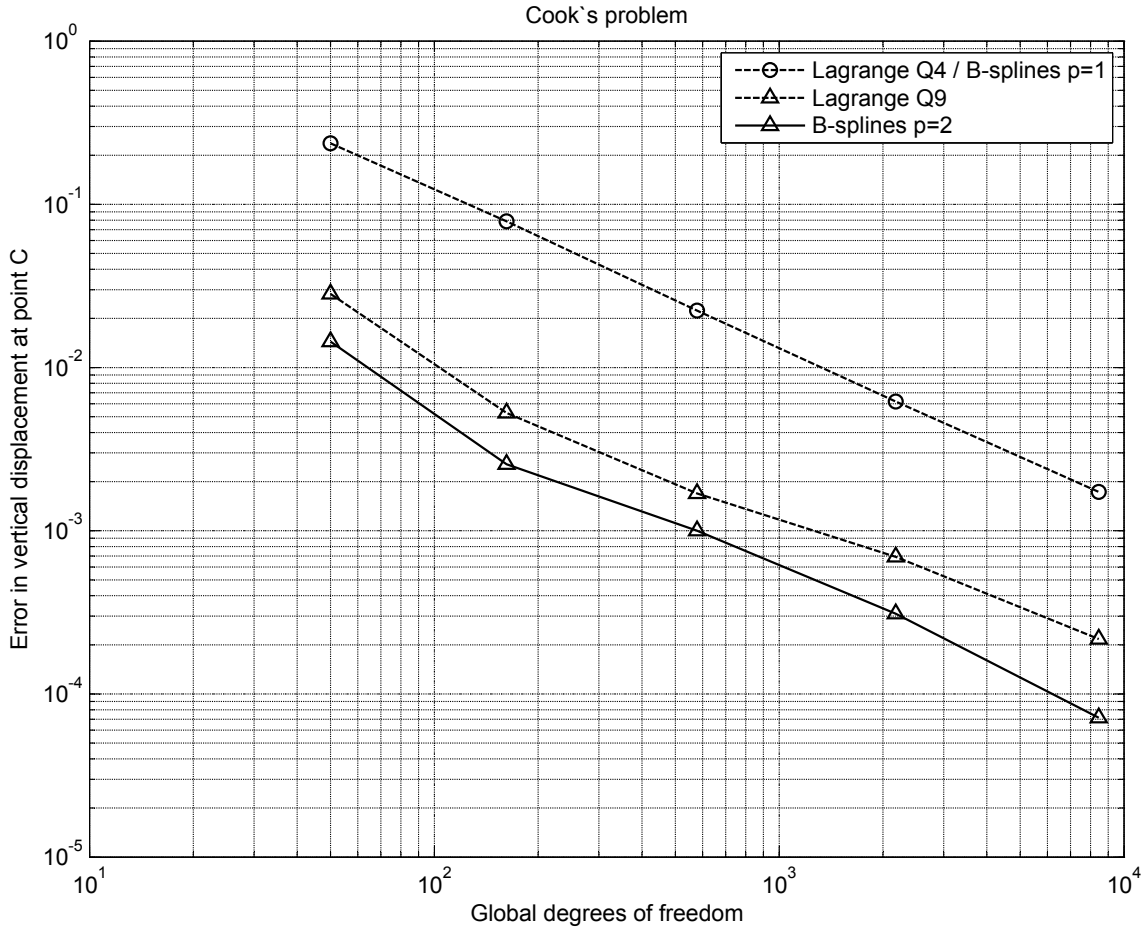


Figure A.3: Error in vertical displacement at point C of Cook's problem

A.2 End Loaded Beam

The end loaded beam in Figure A.4 is another benchmark problem for plane stress elements. The parameter chosen for consideration is the strain energy of the system. The analytical solution of the beam is given by Timoshenko and Goodier in Zienkiewicz et al. [29]. Since the exact solution for the displacements given by Timoshenko and Goodier contains all polynomial terms of degree 3 or less, the solution with a mesh of bicubic elements (Q16) will give the exact energy, which is $U = 3296$ [29].

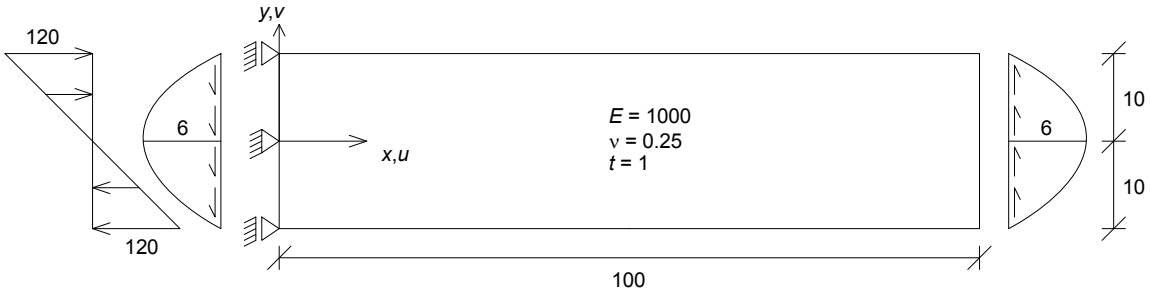


Figure A.4: The geometry of the end loaded beam with material properties, boundary conditions and surface tractions.

The end loaded beam is modelled with five different meshes using Q4 and and Q9 elements for FEA, and elements of polynomial order 1 and 2 for isogeometric analysis. The shear force is modelled as a parabolic distributed load given by the function $f(y) = 6 - {}^{3y^2}/_{50}$. The load integrated over the height sums up to a shear force with value 80. This load must also be applied in the opposite direction at the support side to be compatible with the exact solution, which is given by beam theory. In addition, a normal traction with value ± 120 is applied at the support side to be work equivalent to the moment caused by the shear traction at the free end.

The mesh for isogeometric analysis using 2nd order elements is chosen so that the number of elements in y direction is an odd number. Then the number of control points in this direction will also be odd, giving a control point at the middle, which is necessary for the middle support to be modelled. The number of elements in the horizontal direction is twice the number of elements in the vertical direction. Since the choice of the mesh for isogeometric analysis using 2nd order elements is restricted by these requirements, an entirely uniform mesh refinement procedure is not possible.

Table A.2 shows the results with 5 decimal places (9 significant digits) for the different meshes and methods. FEA using Q4 elements and isogeometric analysis using 1st order elements display the same results, while isogeometric analysis using 2nd order elements shows energy closer to the exact result than FEA using Q9 elements for approximately the same number of global degrees of freedom.

Table A.2: Strain energy of the end loaded beam (exact solution $U = 3296.00000$)

Lagrange Q4 / B-splines $p = 1$			Lagrange Q9			B-splines $p = 2$		
Mesh	DOF	U^h	Mesh	DOF	U^h	Mesh	DOF	U^h
6x12	182	3077.49865	3x6	182	3294.75118	5x10	168	3295.81975
12x24	650	3238.29153	6x12	650	3295.91739	11x22	624	3295.99229
24x48	2450	3281.34653	12x24	2450	3295.99469	23x46	2400	3295.99960
48x96	9506	3292.32062	24x48	9506	3295.99966	47x94	9408	3295.99998

Figure A.5 shows an error plot of these results. The error in the strain energy is defined as

$$\|e\|_E^2 = \frac{U - U^h}{U} \tag{A.2}$$

where U is the exact strain energy and U^h is the corresponding strain energy of the FE solution.

Again, the results using 2nd order isogeometric elements is more effective in terms of less error compared to FEA using Q9 elements. Since an exact solution exists, the meshes of different element types will converge towards this value with mesh refinement. The convergence rate is

approximately the same for the quadratic elements of FEA and isogeometric analysis, and better than the convergence rate for the linear elements.

The gradients are not integers because the error is plotted against global degrees of freedom and not element size. The reason for this choice is due to the fact that an isogeometric mesh with the same element size as an equal FE mesh generates fewer global degrees of freedom, making the methods non-comparable.

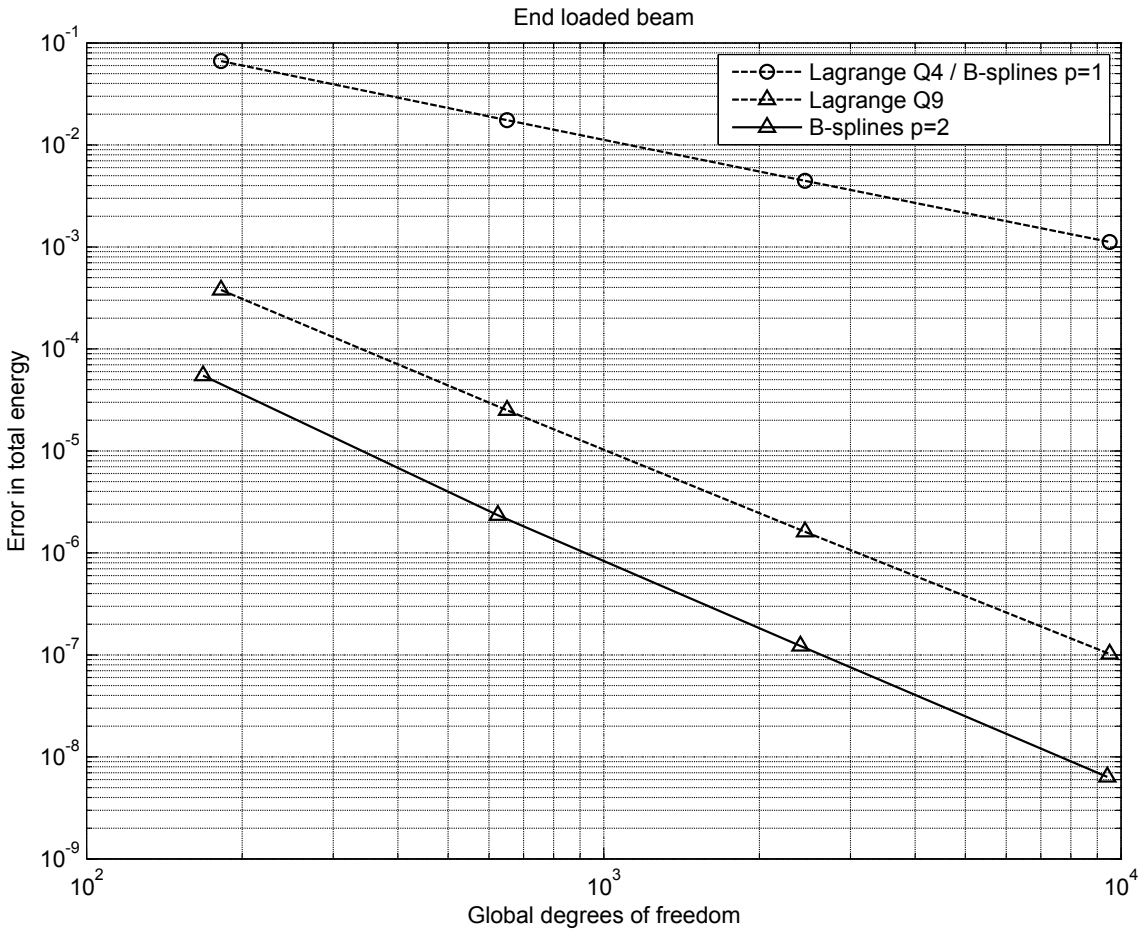


Figure A.5: Error in strain energy of the end loaded beam

Appendix B

MATLAB Code for Isogeometric Analysis based on Bézier Extraction

B.1 User Definition

This is a simple finite element solver based on Bézier extraction of NURBS. The code handles two-dimensional linear elastic problems modelled with quadrilateral elements, and is written in MATLAB R2010b with the purpose of self-tuition. The program is adapted to specific problems and has therefore not a general user definition. However, for beginners in isogeometric analysis with knowledge from FEA, the code is still a good source to understand the data structures in isogeometric analysis compared to FEA.

B.1.1 Material Properties

Material properties are defined by the elastic modulus `Emod` and the Poisson's ratio `nu`. The user may choose the constitutive matrix `E` for plane stress or plane strain. The user must be aware of consistent denomination himself (e.g. N, mm, MPa etc. or N, m, Pa etc.).

B.1.2 Geometry

The geometry may be

1. A square defined by coordinates of four corners, given from left to right and bottom to top.
2. A quarter of a circular disk, defined by an inner radius and an outer radius.
3. The geometry may also be the input from T-Splines for Rhino (two-dimensional objects). In that case, the following is needed from Rhino,
    - Control points and weights from the .tsm file, tag `0g`. Copy the numbers to a .mat file for MATLAB and label the set of variables `P_Rhino`. The data set should consist of 4 columns (1st column = x values, 2nd column = y values, 3rd column = z values = 0, 4th column = weights. Note that the control points are non-rational. To obtain the Cartesian coordinates the values are divided by the weights in the script ParseRhinoData.m).

- Bézier extraction operators from Rhino. Use the T-spline command `tsDumpBeziers` (the object must be a T-spline for this to work), select `DumpExtraction = Yes` and copy the values in `CombData(...)`. Delete all characters that are not numbers and copy the numbers to the same .mat file as the control points. Label the data `C_Rhino`. The data set should consist of Bézier values in odd columns and control point indices in even columns.

Label the .mat file for instance `NameOfProblem.mat`. Load the .mat file in the MATLAB program and use the script ParseRhinoData.m to make the geometry data compatible with the MATLAB program. Mesh refinement of the imported geometry is not possible in the MATLAB program.

The number of elements in ξ and η direction may be chosen by specifying the variables `nx` and `ny`, respectively.

The polynomial order of the elements may be for the different geometries:

1. A square: `poly = 1`, `poly = 2`, `poly = 3` or `poly = 4`. The restriction is due to that the highest Gauss order supported by the program is of degree 5, i.e. full integration of 4th order elements.

2. A quarter disk: `poly = 2` or `poly = 3`. The restriction is due to that a circular section must be modelled with at least 2nd order basis functions, and that the initial geometry using a 4th or higher order element requires a general order elevation algorithm, which is not supported by the program.

3. T-splines from Rhino: `poly = 3`. T-splines from Rhino are of degree 3.

Note that the element topology of the program is given from left to right and bottom to top, both locally for the element (counting of control points in support of the element) and globally for the whole domain (counting of elements in the physical/parameter space). This is also described in Section 5.1.1.

B.1.3 Loads

The following loads are supported by the program,

1. Direct point loads

2. Distributed loads on η edges; linear normal traction and constant or parabolic shear traction

3. Prescribed displacements along η edge (i.e. a load defined as a boundary condition)

4. Traction field on ξ edge defined by stresses from an analytical solution

Notice that the loads are highly for specific problems (Cook's problem, end loaded beam, circular beam, infinite plate with circular hole and machine part). Nevertheless, due to the variation of load types, they still illustrate how different loads are handled in an isogeometric environment.

B.1.4 Boundary Conditions

Different combinations of suppressed degrees of freedom define the boundary conditions available,

1. Single DOF suppressed
2. Fixed boundary along η edge; all DOF in x and y direction are suppressed
3. Partially suppressed boundary along η edge; all DOF in x direction and zero or one DOF in y direction are suppressed, or reversed
4. Prescribed displacements (i.e. a load) along η edge

The boundary conditions are restricted to η edges since they are modelled for specific problems, but reformulation to ξ edges is not difficult.

B.2 Variable Description

This section defines all variables used between the subfunctions and also important temporary variables used in the main code and/or the subfunctions.

Variable	Description
`aDof`	Active/free degrees of freedom (unknown degrees of freedom)
`alpha`	Knot insertion variable α
`B`	Strain-displacement matrix
`B`	NURBS control points and weights given as 3 matrices; x values, y values, weights
`B_old`	Old NURBS control points and weights when new knot is inserted
`Bb`	Bernstein basis functions over the interval [-1,1]
`Bb`	Bézier control points and weights for the element given as 3 matrices
`Bbeta`	Bernstein basis functions over the interval [-1,1] in η direction
`Bbxi`	Bernstein basis functions over the interval [-1,1] in ξ direction
`C`	Bézier extraction operators, univariate or bivariate
`C_Rhino`	Bézier extraction operators from T-Splines for Rhino
`Ceta`	Bézier extraction operators in η direction
`Cxi`	Bézier extraction operators in ξ direction
`D`	Global displacement vector
`dB`	Derivatives of Bernstein basis functions over [-1,1], univariate or bivariate
`dBeta`	Derivatives of Bernstein basis functions over [-1,1] in η direction
`dBxi`	Derivatives of Bernstein basis functions over [-1,1] in ξ direction
`deta`	Derivatives of B-spline basis functions, η direction
`dir`	Direction 1 ($= \xi$) or 2 ($= \eta$)
`dN`	Univariate derivative of B-spline basis functions
`dR`	Bivariate derivatives of NURBS basis functions
`dW`	Univariate derivative of NURBS weight function
`dWbeta`	η derivative of Bézier weight function

`dWbxi`	ξ derivative of Bézier weight function
`dxi`	Derivatives of B-spline basis functions, ξ direction
`dxy`	Physical derivatives
`E`	Constitutive matrix
`e`	Element number
`edgeDofxL`	Element topology in x direction for the left edge of the element
`edgeDofxT`	Element topology in x direction for the top edge of the element
`edgeDofyL`	Element topology in y direction for the left edge of the element
`edgeDofyR`	Element topology in y direction for the right edge of the element
`edgeDofyT`	Element topology in y direction for the top edge of the element
`eDof`	Element topology for the element in x and y direction
`Emod`	Elastic modulus
`eta`	2^{nd} parametric coordinate
`eta`	2^{nd} coordinate of reference element
`G`	Gauss points given from left to right and bottom to top; 1st, 2nd col. $= \xi, \eta$ values
`G`	Gauss boundary points given from left to right and bottom to top
`G`	Gauss points given as 3 matrices; ξ values, η values, weights
`gDof`	Number of global degrees of freedom
`IEN`	IEN array (element topology; numbering of control points)
`IEN_e`	IEN array for the element
`IEN_ey`	Element topology for the element in y direction
`J`	Jacobian matrix
`Jac`	Jacobian (the determinant of the Jacobian matrix)
`K`	Global stiffness matrix
`k`	Element stiffness matrix
`Kfs`	Stiffness matrix related to prescribed degrees of freedom
`knot`	Knot vectors given as a MATLAB structure; `knot.xi` and `knot.eta`
`knot_ins`	Value of knot inserted into the knot vector
`knot_old`	Old knot vector when new knot is inserted
`m`	Number of basis functions/control points in η direction
`Mises`	von Mises stress, $\sigma_{\text{e}} = \sqrt{\sigma_x^2 - \sigma_x\sigma_y + \sigma_y^2 + 3\tau_{xy}^2}$
`N`	Univariate B-spline basis functions
`Neta`	B-spline basis functions, η direction
`Nxi`	B-spline basis functions, ξ direction
`n`	Number of basis functions/control points in ξ direction
`n`	Unit outward normal vector
`ncp`	Total number of control points
`nel`	Total number of elements
`Nodes`	Parametric coordinates which equals nodes of FEA, 1st, 2nd col. $= \xi, \eta$ values
`nu`	Poisson's ration
`nx`	Number of elements in ξ direction
`ny`	Number of elements in η direction
`P`	Coordinates of NURBS control points
`p`	Normal traction
`P_Rhino`	T-spline control points and weights from T-Splines for Rhino

`Pb`	Coordinates of Bézier control points for the element
`poly`	Polynomial order
`prDof`	Prescribed/suppressed degrees of freedom (known degrees of freedom)
`q`	Shear traction
`R`	Global load vector
`r`	Polar coordinate, radial direction
`R_f`	Load vector related to active degrees of freedom (known loads)
`R_fn`	Load vector related to active DOF including loads from prescribed DOF
`r_inn`	Inner radius of a quarter of a circular disk
`r_out`	Outer radius of a quarter of a circular disk
`Rb`	NURBS basis functions
`RbL`	NURBS basis functions related to the left edge of the element
`RbR`	NURBS basis functions related to the right edge of the element
`RbT`	NURBS basis functions related to the top edge of the element
`sigma_x`	σ_x from analytical solution
`sigma_y`	σ_y from analytical solution
`strains`	Strains ε_x, ε_y and γ_{xy} given as a column vector
`stress`	Stresses σ_x, σ_y and τ_{xy} given as given as three matrices
`t`	Thickness
`tau_xy`	τ_{xy} from analytical solution
`theta`	Polar coordinate, tangential direction
`Tx`	Value of the far-field uniaxial tension for the infinite plate with circular hole
`u`	Displacement in x direction
`U1`	Strain energy of the system calculated from global displacements
`U2`	Strain energy of the system calculated from element stiffness matrices
`U3`	Strain energy of the system calculated from strains
`v`	Displacement in y direction
`vC`	Displacement in y direction at point C of Cook's problem
`W`	Weights of Gauss points
`W`	NURBS weight function
`w`	Weights of NURBS control points
`Wb`	Weight function of Bézier element
`wb`	Weights of Bézier control points
`X`	1st physical coordinates organized in a vector or matrix
`x`	1st physical coordinate (NURBS entity)
`xb`	1st physical coordinate, B-spline entity
`XI`	Knot vector in one dimension
`xi`	1st parametric coordinate
`xi`	1st coordinate of reference element
`XY`	Coord. of corners from left to right and bottom to top; 1st, 2nd col. $= x, y$ values
`Y`	2nd physical coordinates organized in a vector or matrix
`y`	2nd physical coordinate (NURBS entity)
`yb`	2nd physical coordinate, B-spline entity

B.3 Main Code

This section presents the main code of the program for specific problems; Cook's problem, end loaded beam, circular beam, infinite plate with circular hole and machine part.

B.3.1 Cook's Problem

Figure B.1 shows a flow chart of the subfunctions involved in the main code for Cook's Problem, IA_CooksProblem.m.

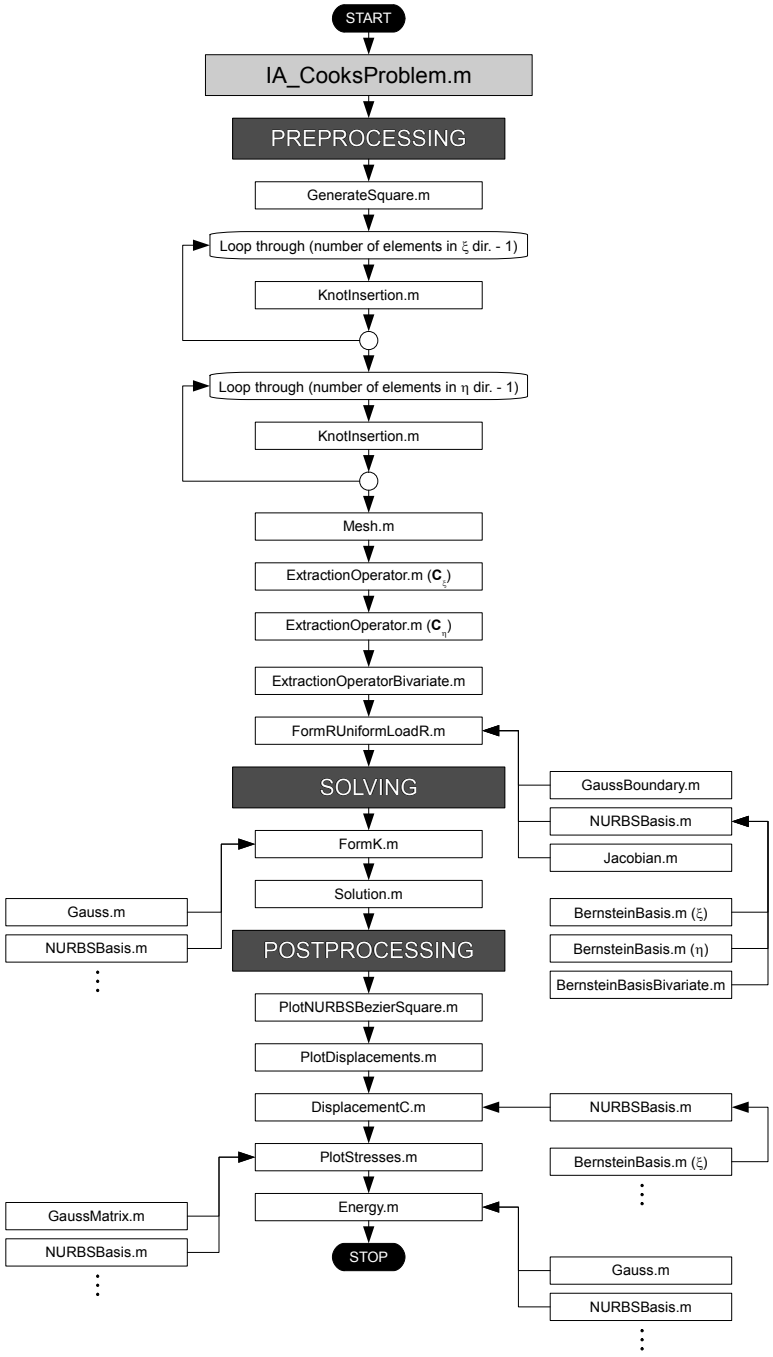


Figure B.1: Cook's problem, flow chart of the subfunctions involved

```
%-----------------ISOGEOMETRIC ANALYSIS - COOK'S PROBLEM-----------------%

close all; clear all
tStart=tic;

%-----------------------------------PRE-----------------------------------%

% MATERIAL PROPERTIES
Emod=1;
nu=1/3;
E=Emod/(1-nu^2)*[1 nu 0;nu 1 0;0 0 (1-nu)/2];   % Plane stress

% GEOMETRY, CONTROL POINTS AND ELEMENT TOPOLOGY
t=1;                              % Thickness
XY=[0 0;48 44;0 44;48 60];        % Coordinates of corners [X1 Y1;...;X4 Y4],
                                  % left to right and bottom to top
nx=13;          % Number of elements, xi direction
ny=13;          % Number of elements, eta direction
poly=4;         % Polynomial order, same in both directions
nel=nx*ny;      % Total number of elements
n=nx+poly;      % Number of basis functions/control points, xi direction
m=ny+poly;      % Number of basis functions/control points, eta direction
ncp=n*m;        % Total number of basis functions/control points
gDof=2*ncp;     % Global degrees of freedom

% The function GenerateSquare.m generates control points and weights
% from the given square
[B,knot]=GenerateSquare(XY,poly);

% The function KnotInsertion.m refines the mesh by knot insertion
for i=1:nx-1
    [B knot.xi]=KnotInsertion(B,knot.xi,i/nx,poly,1);
end
for j=1:ny-1
    [B knot.eta]=KnotInsertion(B,knot.eta,j/ny,poly,2);
end

% Transfer control points and weights from three matrices (B) to one matrix
% P for control points (x=1st column, y=2nd) and one vector w for weights
P=zeros(ncp,2);
w=zeros(ncp,1);
k=1;
for j=1:m
  for i=1:n
      P(k,1)=B(i,j,1);
      P(k,2)=B(i,j,2);
      w(k)=B(i,j,3);
      k=k+1;
  end
end

% ELEMENT TOPOLOGY
IEN=Mesh(nx,ny,poly);

% BÉZIER EXTRACTION OPERATOR
Cxi=ExtractionOperator(knot.xi,poly);
```

```
Ceta=ExtractionOperator(knot.eta,poly);
C=ExtractionOperatorBivariate(Cxi,Ceta,poly,nx,ny);

% FOR STRUCTURE
K=zeros(gDof);
D=zeros(gDof,1);
R=zeros(gDof,1);

% LOADS
% Uniformly distributed shear traction at right hand side (sum = 1)
% The function FormRUniformLoadR.m numerical integrates consistent nodal
% loads and assembles to R
q=1/16;
R=FormRUniformLoadR(IEN,P,R,nx,ny,ncp,poly,w,C,q);

% BOUNDARY CONDITIONS
% Fixed at left end
prDof=zeros(2*m,1);
for j=1:m
    prDof(j)=1+n*(j-1);
    prDof(m+j)=1+m*(j-1)+ncp;
end

%----------------------------------SOLVE---------------------------------%

% STIFFNESS MATRIX
% The function FormK.m forms k for each element and assembles to K
K=FormK(IEN,P,K,E,t,nel,ncp,poly,w,C);

% SOLVE SYSTEM
% The function Solution.m solves the system in terms of active DOFs
K=sparse(K); D=sparse(D); R=sparse(R);
[D,R]=Solution(gDof,prDof,K,D,R);

%----------------------------------POST----------------------------------%

disp(['IGA COOK`S PROBLEM P',num2str(poly),' (',num2str(gDof),' DOF)'])
SolutionTime=toc(tStart)

% Plot NURBS control mesh, Bézier control mesh and Bézier elements
PlotNURBSBezierSquare(IEN,B,P,nel,poly,XY,w,C);

% Plot displacements u and v
PlotDisplacements(IEN,P,D,n,m,nel,ncp,poly,w,C);

% Calculate vertical displacement at point C
vC=DisplacementC(IEN,D,nx,ny,ncp,poly,w,C)

% Calculate strains & stresses, plot contour plots of Jacobian + stresses
PlotStresses(IEN,P,D,E,nx,ny,ncp,poly,w,C);

% Calculate strain energy of the system
U1=D'*R
[U2,U3]=Energy(IEN,P,D,E,t,nel,ncp,poly,w,C)
toc(tStart)
```

B.3.2 End Loaded Beam

Figure B.2 shows a flow chart of the subfunctions involved in the main code for the end loaded beam, IA_EndLoadedBeam.m.

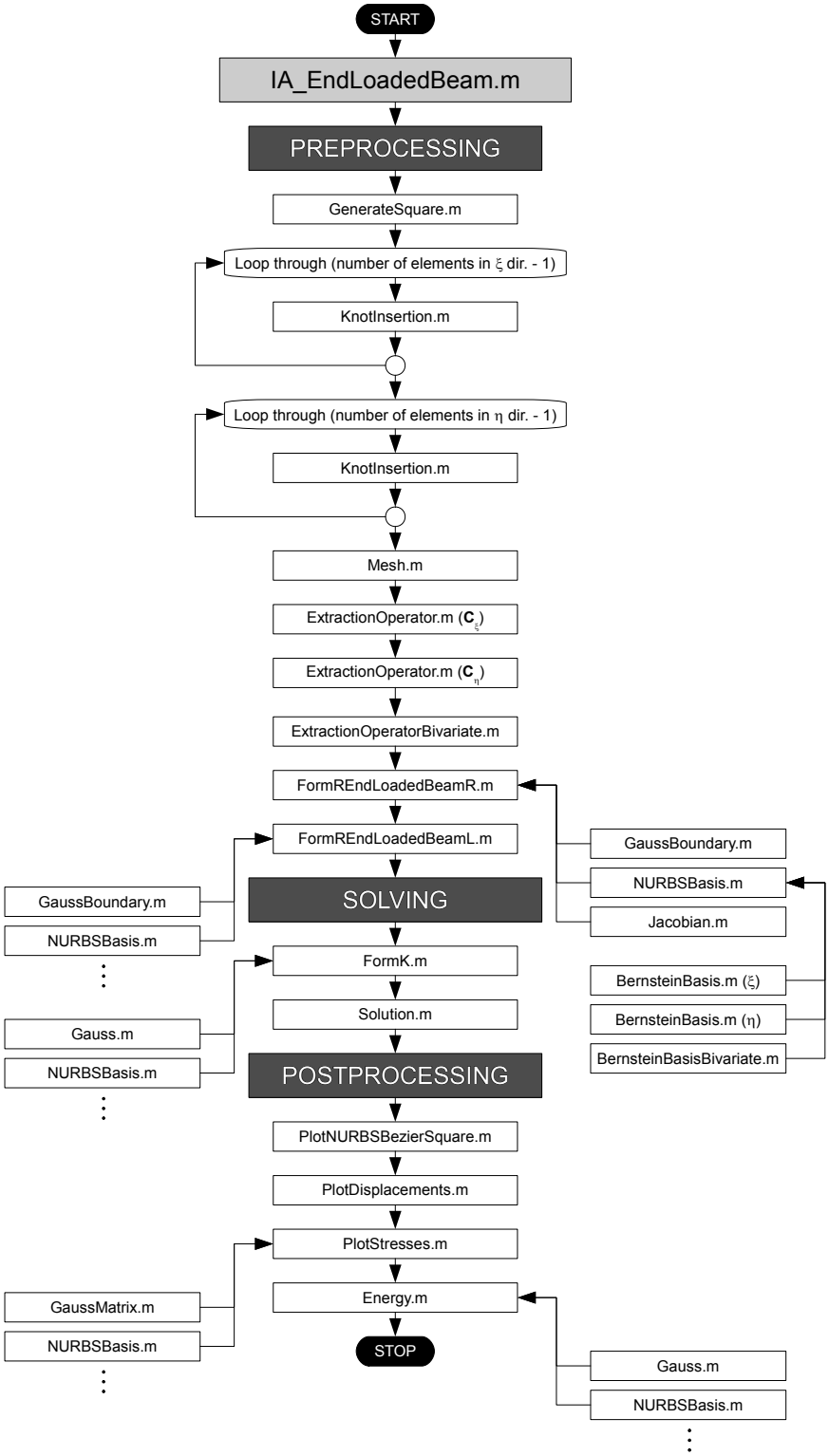


Figure B.2: End loaded beam, flow chart of the subfunctions involved

```
%-----------------ISOGEOMETRIC ANALYSIS - END LOADED BEAM-----------------%

close all; clear all
tStart=tic;

%-----------------------------------PRE-----------------------------------%

% MATERIAL PROPERTIES
Emod=1000;
nu=0.25;
E=Emod/(1-nu^2)*[1 nu 0;nu 1 0;0 0 (1-nu)/2];   % Plane stress

% GEOMETRY AND CONTROL POINTS
t=1;                              % Thickness
XY=[0 -10;100 -10;0 10;100 10]; % Coordinates of corners [X1 Y1;...;X4 Y4],
                                  % left to right and bottom to top
nx=22;          % Number of elements, xi direction
ny=11;          % Number of elements, eta direction
poly=2;         % Polynomial order, same in both directions
nel=nx*ny;      % Total number of elements
n=nx+poly;      % Number of basis functions/control points, xi direction
m=ny+poly;      % Number of basis functions/control points, eta direction
ncp=n*m;        % Total number of basis functions/control points
gDof=2*ncp;     % Global degrees of freedom

% The function GenerateSquare.m generates control points and weights
% from the given square
[B,knot]=GenerateSquare(XY,poly);

% The function KnotInsertion.m refines the mesh by knot insertion
for i=1:nx-1
    [B knot.xi]=KnotInsertion(B,knot.xi,i/nx,poly,1);
end
for j=1:ny-1
    [B knot.eta]=KnotInsertion(B,knot.eta,j/ny,poly,2);
end

% Transfer control points and weights from three matrices (B) to one matrix
% P for control points (x=1st column, y=2nd) and one vector w for weights
P=zeros(ncp,2);
w=zeros(ncp,1);
k=1;
for j=1:m
  for i=1:n
      P(k,1)=B(i,j,1);
      P(k,2)=B(i,j,2);
      w(k)=B(i,j,3);
      k=k+1;
  end
end

% ELEMENT TOPOLOGY
IEN=Mesh(nx,ny,poly);

% BÉZIER EXTRACTION OPERATOR
Cxi=ExtractionOperator(knot.xi,poly);
```

```
Ceta=ExtractionOperator(knot.eta,poly);
C=ExtractionOperatorBivariate(Cxi,Ceta,poly,nx,ny);

% FOR STRUCTURE
K=zeros(gDof);
D=zeros(gDof,1);
R=zeros(gDof,1);

% LOADS
% Parabolic shear traction (sum = 80) at both sides (shear force equivalent)
% Normal traction (+- 120 top/bottom) at support side (moment equivalent)
% The function FormREndLoadedBeamR.m numerical integrates consistent
% nodal loads at right hand side and assembles to R
% The function FormREndLoadedBeamL.m numerical integrates consistent
% nodal loads at left hand side and assembles to R
R=FormREndLoadedBeamR(IEN,P,R,nx,ny,ncp,poly,w,C);
R=FormREndLoadedBeamL(IEN,P,R,nx,ny,ncp,poly,w,C);

% BOUNDARY CONDITIONS
% u(0,-10) = u(0,0) = u(0,10) = v(0,0) = 0
prDof=[1 1+((m-1)/2)*n 1+(m-1)*n 1+((m-1)/2)*n+ncp]';

%----------------------------------SOLVE----------------------------------%

% STIFFNESS MATRIX
% The function FormK.m forms k for each element and assembles to K
K=FormK(IEN,P,K,E,t,nel,ncp,poly,w,C);

% SOLVE SYSTEM
% The function Solution.m solves the system in terms of active DOFs
K=sparse(K); D=sparse(D); R=sparse(R);
[D,R]=Solution(gDof,prDof,K,D,R);

%----------------------------------POST-----------------------------------%

disp(['IGA END LOADED BEAM P',num2str(poly),' (',num2str(gDof),' DOF)'])
SolutionTime=toc(tStart)

% Plot NURBS control mesh, Bézier control mesh and Bézier elements
PlotNURBSBezierSquare(IEN,B,P,nel,poly,XY,w,C);

% Plot displacements u and v
PlotDisplacements(IEN,P,D,n,m,nel,ncp,poly,w,C);

% Calculate strains & stresses, plot contour plots of Jacobian + stresses
PlotStresses(IEN,P,D,E,nx,ny,ncp,poly,w,C);

% Calculate strain energy of the system
U1=D'*R
[U2,U3]=Energy(IEN,P,D,E,t,nel,ncp,poly,w,C)

toc(tStart)
```

B.3.3 Circular Beam

Figure B.3 shows a flow chart of the subfunctions involved in the main code for the circular beam, IA_CircularBeam.m.

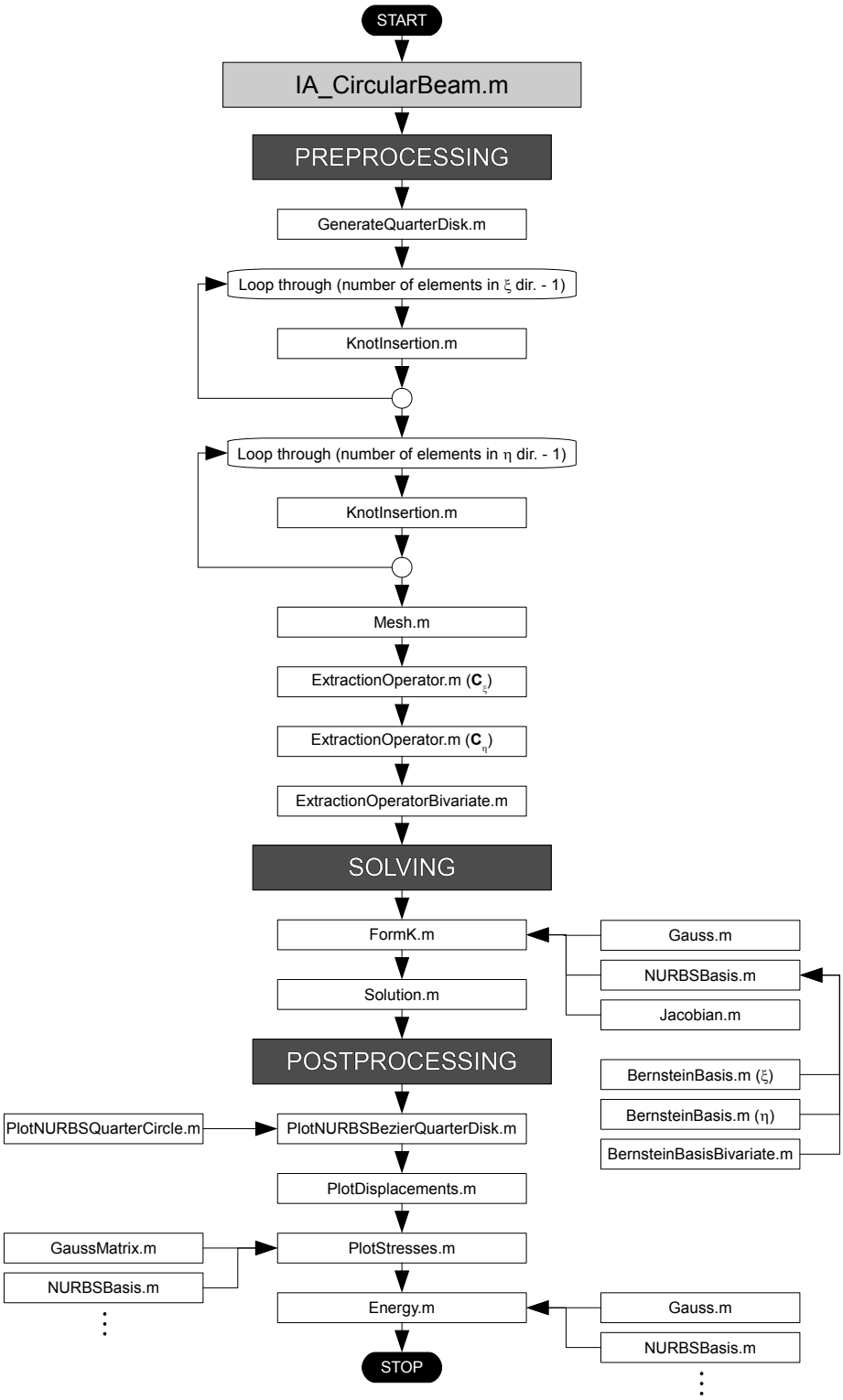


Figure B.3: Circular beam, flow chart of the subfunctions involved

```
%------------------ISOGEOMETRIC ANALYSIS - CIRCULAR BEAM------------------%

close all; clear all
tStart=tic;

%-----------------------------------PRE-----------------------------------%

% MATERIAL PROPERTIES
Emod=10000;
nu=0.25;
E=Emod/(1-nu^2)*[1 nu 0;nu 1 0;0 0 (1-nu)/2];    % Plane stress

% GEOMETRY AND CONTROL POINTS
t=1;              % Thickness
r_inn=5;          % Inner radius
r_out=10;         % Outer radius
nx=22;            % Number of elements, xi direction
ny=11;            % Number of elements, eta direction
poly=2;           % Polynomial order, same in both directions
nel=nx*ny;        % Total number of elements
n=nx+poly;        % Number of basis functions/control points, xi direction
m=ny+poly;        % Number of basis functions/control points, eta direction
ncp=n*m;          % Total number of basis functions/control points
gDof=2*ncp;       % Global degrees of freedom

% The function GenerateCircularBeam.m generates control points and
% weights for the circular beam from the given radii
[B,knot]=GenerateQuarterDisk(r_inn,r_out,poly);

% The function KnotInsertion.m refines the mesh by knot insertion
for i=1:nx-1
    [B,knot.xi]=KnotInsertion(B,knot.xi,i/nx,poly,1);
end
for j=1:ny-1
    [B,knot.eta]=KnotInsertion(B,knot.eta,j/ny,poly,2);
end

% Transfer control points and weights from three matrices (B) to one matrix
% P for control points (x=1st column, y=2nd) and one vector w for weights
P=zeros(ncp,2);
w=zeros(ncp,1);
k=1;
for j=1:m
  for i=1:n
      P(k,1)=B(i,j,1);
      P(k,2)=B(i,j,2);
      w(k)=B(i,j,3);
      k=k+1;
  end
end

% ELEMENT TOPOLOGY
IEN=Mesh(nx,ny,poly);

% BÉZIER EXTRACTION OPERATOR
Cxi=ExtractionOperator(knot.xi,poly);
```

```
Ceta=ExtractionOperator(knot.eta,poly);
C=ExtractionOperatorBivariate(Cxi,Ceta,poly,nx,ny);

% FOR STRUCTURE
K=zeros(gDof);
D=zeros(gDof,1);
R=zeros(gDof,1);

% LOADS
% Prescribed displacement at bottom edge (shear force equivalent)
for i=n:n:ncp
    D(i)=-0.01;
end

% BOUNDARY CONDITIONS
% Zero displacement in all x plus y at bottom node, at left end
prDof=zeros(2*m+1,1);
for j=1:m
    prDof(j)=1+n*(j-1);
    prDof(m+j)=n*j;                 % Due to prescribed displacement
end
prDof(2*m+1)=1+ncp;
prDof=sort(prDof);

%----------------------------------SOLVE-----------------------------------%

% STIFFNESS MATRIX
% The function FormK.m forms k for each element and assembles to K
K=FormK(IEN,P,K,E,t,nel,ncp,poly,w,C);

% SOLVE SYSTEM
% The function Solution.m solves the system in terms of active DOFs
K=sparse(K); D=sparse(D); R=sparse(R);
[D,R]=Solution(gDof,prDof,K,D,R);

%----------------------------------POST------------------------------------%

disp(['IGA CIRCULAR BEAM P',num2str(poly),' (',num2str(gDof),' DOF)'])
SolutionTime=toc(tStart)

% Plot NURBS control mesh, Bézier control mesh and Bézier elements
PlotNURBSBezierQuarterDisk(IEN,B,P,ny,nel,poly,r_inn,r_out,w,C);

% Plot displacements u and v
PlotDisplacements(IEN,P,D,n,m,nel,ncp,poly,w,C);

% Calculate strains & stresses, plot contour plots of Jacobian + stresses
PlotStresses(IEN,P,D,E,nx,ny,ncp,poly,w,C);

% Calculate strain energy of the system
U1=D'*R
[U2,U3]=Energy(IEN,P,D,E,t,nel,ncp,poly,w,C)

toc(tStart)
```

The circular beam is also analysed using geometries modelled in T-Splines for Rhino. Figure B.4 shows a flow chart of the subfunctions (and .mat file) involved in the main code for the imported circular beam, IA_CircularBeamRhino.m.

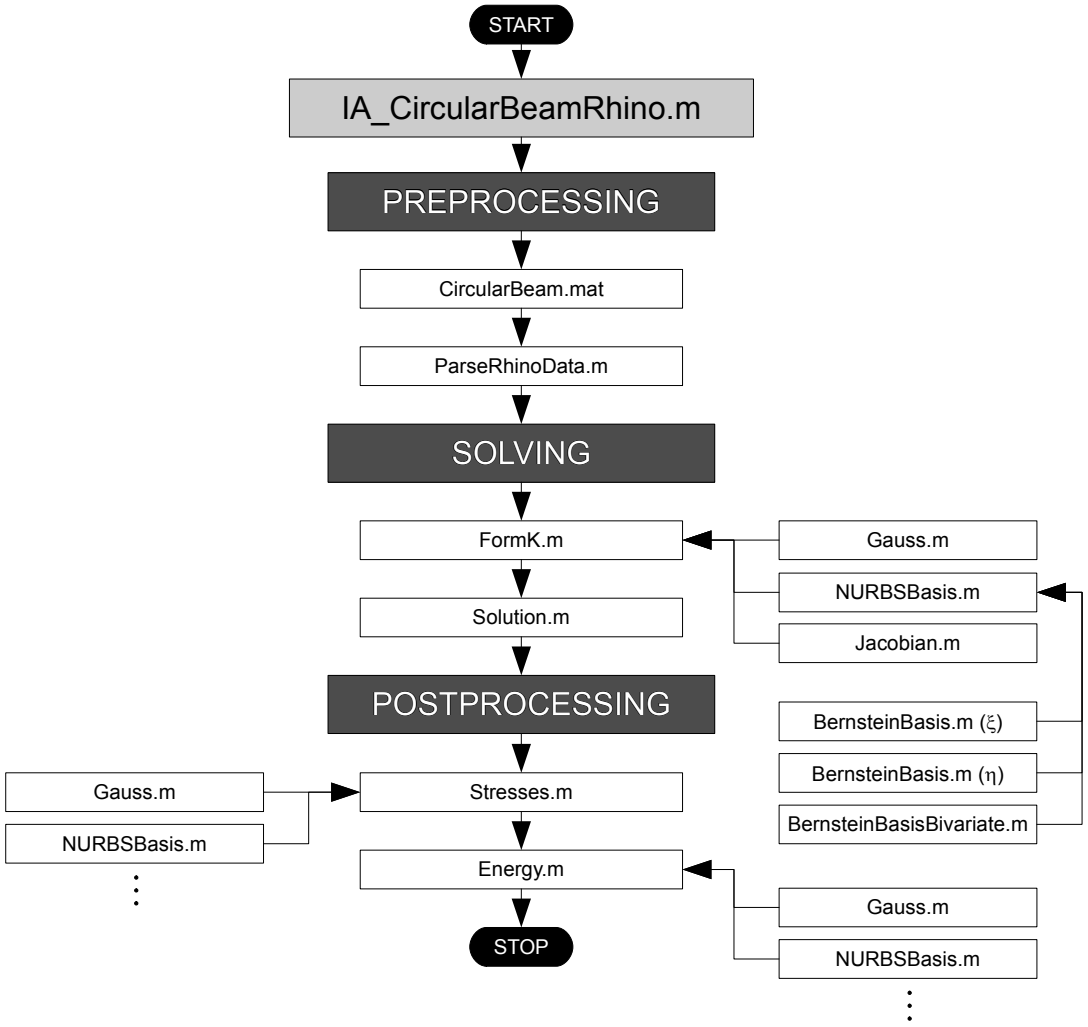


Figure B.4: Circular beam modelled in Rhino, flow chart of the subfunctions involved

```
%--------------ISOGEOMETRIC ANALYSIS - CIRCULAR BEAM RHINO---------------%

close all; clear all
tStart=tic;

%-----------------------------------PRE-----------------------------------%

% MATERIAL PROPERTIES
Emod=10000;
nu=0.25;
E=Emod/(1-nu^2)*[1 nu 0;nu 1 0;0 0 (1-nu)/2];    % Plane stress

% GEOMETRY, CONTROL POINTS, ELEMENT TOPOLOGY AND BÉZIER EXTRACTION OPERATOR
t=1;             % Thickness
r_inn=5;         % Inner radius
r_out=10;        % Outer radius

% Load control points P_Rhino and extraction operators C_Rhino from
% Rhinoceros. The function ParseRhinoData.m reads and edits the data
load CircularBeam.mat
[P,w,C,IEN]=ParseRhinoData(P_Rhino,C_Rhino);

poly=3;              % Polynomial order, same in both directions
nel=size(C,3);       % Total number of elements
ncp=size(P,1);       % Total number of basis functions/control points
gDof=2*ncp;          % Global degrees of freedom

% FOR STRUCTURE
K=zeros(gDof);
D=zeros(gDof,1);
R=zeros(gDof,1);

% LOADS
% Prescribed displacement at bottom edge (shear force equivalent)
D(P(:,2)<1e-5)=-0.01;

% BOUNDARY CONDITIONS
% Zero displacement in all x plus y at bottom node, at left end
prDof1=find(P(:,1)<1e-5);    % DOF in x direction, left end
% DOF in y direction, bottom left node
prDof2=find(P(:,1)<1e-5 & P(:,2)>r_inn-1e-5 & P(:,2)<r_inn+1e-5);
prDof3=find(P(:,2)<1e-5);    % DOF in x dir., prescribed displ. bottom edge
prDof=[prDof1;prDof2+ncp;prDof3];

%---------------------------------SOLVE-----------------------------------%

% STIFFNESS MATRIX
% The function FormK.m forms k for each element and assembles to K
K=FormK(IEN,P,K,E,t,nel,ncp,poly,w,C);

% SOLVE SYSTEM
% The function Solution.m solves the system in terms of active DOFs
[D,R]=Solution(gDof,prDof,K,D,R);

%----------------------------------POST-----------------------------------%
```

```
disp(['IGA CIRCULAR BEAM RHINO P3 (',num2str(gDof),' DOF)'])
SolutionTime=toc(tStart)

% Write displacements and reactions
Displacements=[(1:gDof)' D];
Reactions=[prDof R(prDof)];

% Calculate strains and stresses and write stresses at Gauss points
stress=Stresses(IEN,P,D,E,nel,ncp,poly,w,C);

% Calculate strain energy of the system
U1=D'*R
[U2,U3]=Energy(IEN,P,D,E,t,nel,ncp,poly,w,C)

toc(tStart)
```

B.3.4 Infinite Plate with Circular Hole

Figure B.5 shows a flow chart of the subfunctions involved in the main code for the infinite plate with circular hole, IA_InfinitePlate.m.

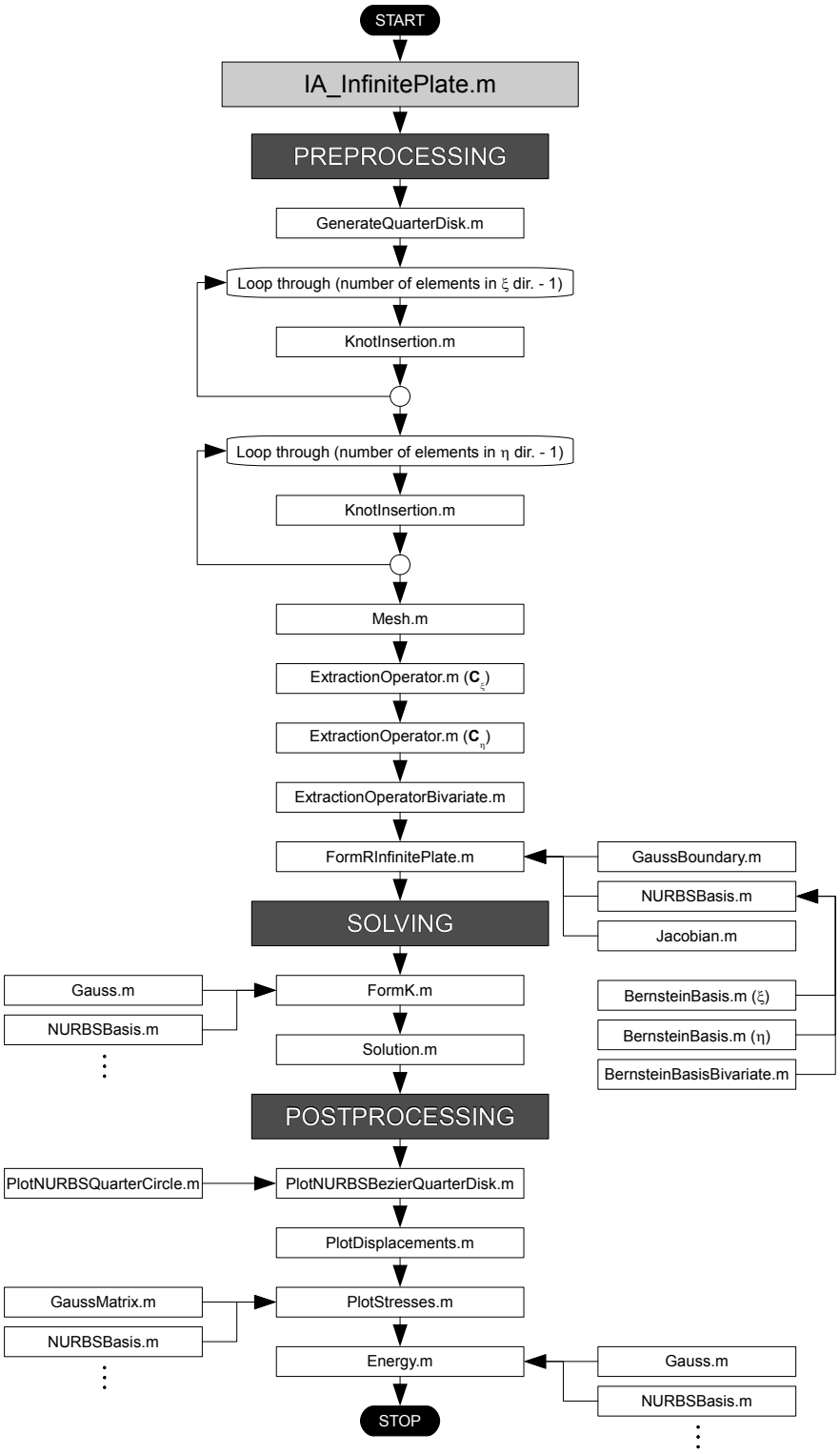


Figure B.5: Infinite plate with circular hole, flow chart of the subfunctions involved

```
%-----------------ISOGEOMETRIC ANALYSIS - INFINITE PLATE-----------------%

close all; clear all
tStart=tic;

%-----------------------------------PRE-----------------------------------%

% MATERIAL PROPERTIES
Emod=1000;
nu=0.3;
% Plane strain
E=(Emod/((1+nu)*(1-2*nu)))*[1-nu nu 0;nu 1-nu 0;0 0 (1-2*nu)/2];

% GEOMETRY AND CONTROL POINTS
t=1;             % Thickness
r_inn=1;         % Inner radius
r_out=4;         % Outer radius
nx=20;           % Number of elements, xi direction
ny=10;           % Number of elements, eta direction
poly=3;          % Polynomial order, same in both directions
nel=nx*ny;       % Total number of elements
n=nx+poly;       % Number of basis functions/control points, xi direction
m=ny+poly;       % Number of basis functions/control points, eta direction
ncp=n*m;         % Total number of basis functions/control points
gDof=2*ncp;      % Global degrees of freedom

% The function GenerateCircularBeam.m generates control points and
% weights for the circular beam from the given radii
[B,knot]=GenerateQuarterDisk(r_inn,r_out,poly);

% The function KnotInsertion.m refines the mesh by knot insertion
for i=1:nx-1
    [B,knot.xi]=KnotInsertion(B,knot.xi,i/nx,poly,1);
end
for j=1:ny-1
    [B,knot.eta]=KnotInsertion(B,knot.eta,j/ny,poly,2);
end

% Transfer control points and weights from three matrices (B) to one matrix
% P for control points (x=1st column, y=2nd) and one vector w for weights
P=zeros(ncp,2);
w=zeros(ncp,1);
k=1;
for j=1:m
  for i=1:n
      P(k,1)=B(i,j,1);
      P(k,2)=B(i,j,2);
      w(k)=B(i,j,3);
      k=k+1;
  end
end

% ELEMENT TOPOLOGY
IEN=Mesh(nx,ny,poly);

% BÉZIER EXTRACTION OPERATOR
```

```
Cxi=ExtractionOperator(knot.xi,poly);
Ceta=ExtractionOperator(knot.eta,poly);
C=ExtractionOperatorBivariate(Cxi,Ceta,poly,nx,ny);

% FOR STRUCTURE
K=zeros(gDof);
D=zeros(gDof,1);
R=zeros(gDof,1);

% LOADS
% Traction from analytical solution of infinite plate with circular hole
% The function FormRInfinitePlate.m numerical integrates consistent nodal
% loads and assembles to R
Tx=1;                          % Traction value
R=FormRInfinitePlate(IEN,P,R,nx,ny,ncp,poly,w,C,r_inn,r_out,Tx);

% BOUNDARY CONDITIONS
% Symmetry: quarter of infinite plate with circular hole
prDof=zeros(2*m,1);
for j=1:m
    prDof(j)=1+n*(j-1);      % Left side: prescribed in x direction
    prDof(m+j)=n*j+ncp;      % Bottom side: prescribed in y direction
end

%----------------------------------SOLVE----------------------------------%

% STIFFNESS MATRIX
% The function FormK.m forms k for each element and assembles to K
K=FormK(IEN,P,K,E,t,nel,ncp,poly,w,C);

% SOLVE SYSTEM
% The function Solution.m solves the system in terms of active DOFs
K=sparse(K); D=sparse(D); R=sparse(R);
[D,R]=Solution(gDof,prDof,K,D,R);

%----------------------------------POST-----------------------------------%

disp(['IGA INFINITE PLATE P',num2str(poly),' (',num2str(gDof),' DOF)'])
SolutionTime=toc(tStart)

% Plot NURBS control mesh, Bézier control mesh and Bézier elements
PlotNURBSBezierQuarterDisk(IEN,B,P,ny,nel,poly,r_inn,r_out,w,C);

% Plot displacements u and v
PlotDisplacements(IEN,P,D,n,m,nel,ncp,poly,w,C);

% Calculate strains & stresses, plot contour plots of Jacobian + stresses
PlotStresses(IEN,P,D,E,nx,ny,ncp,poly,w,C);

% Calculate strain energy of the system
U1=D'*R
[U2,U3]=Energy(IEN,P,D,E,t,nel,ncp,poly,w,C)

toc(tStart)
```

B.3.5 Machine Part

Figure B.4 shows a flow chart of the subfunctions (and .mat file) involved in the main code for the machine part imported from T-Splines for Rhino, IA_MachinePart.m.

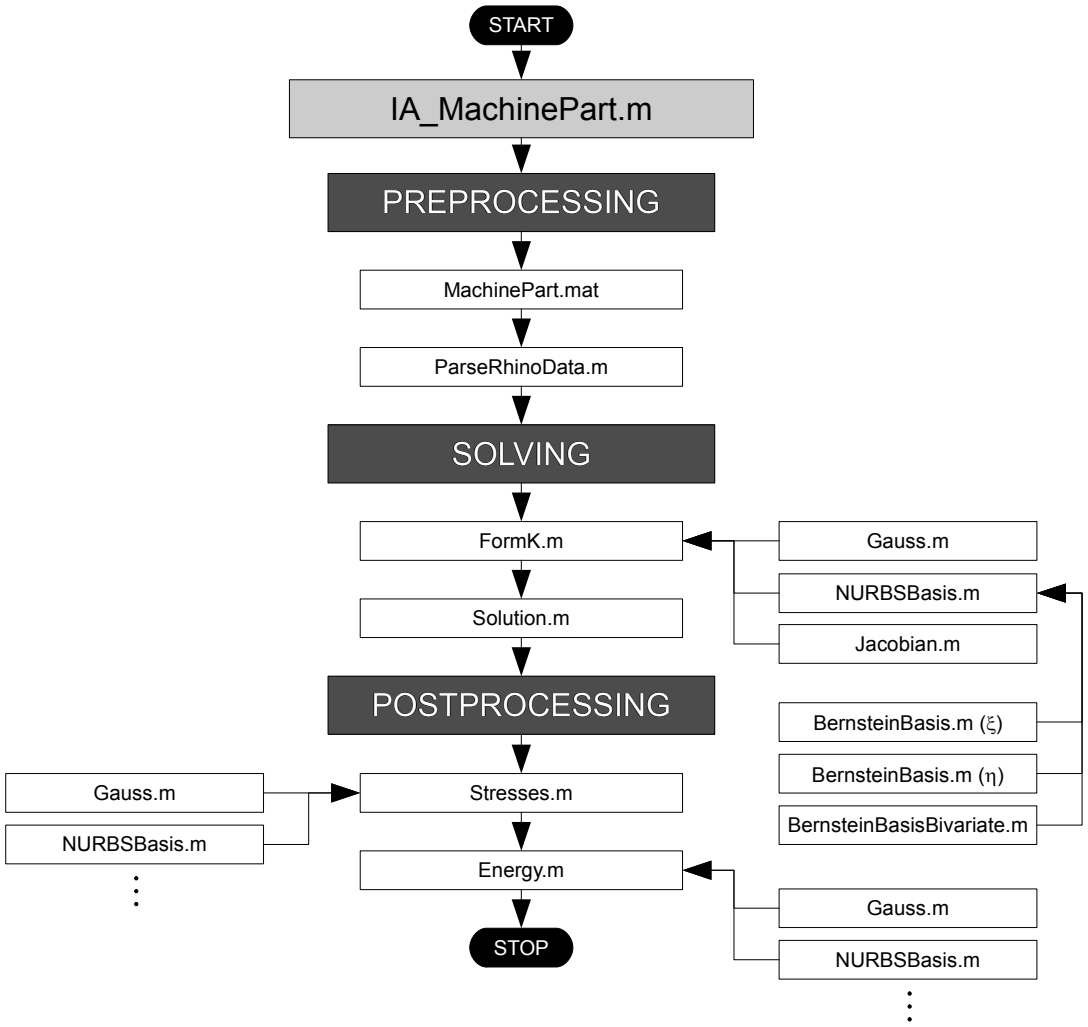


Figure B.6: Machine part modelled in Rhino, flow chart of the subfunctions involved

```
%--------------ISOGEOMETRIC ANALYSIS - MACHINE PART RHINO----------------%

close all; clear all
tStart=tic;

%-----------------------------------PRE-------------------------------------%

% MATERIAL PROPERTIES
Emod=1000;
nu=0.3;
E=Emod/(1-nu^2)*[1 nu 0;nu 1 0;0 0 (1-nu)/2];   % Plane stress

% GEOMETRY, CONTROL POINTS, ELEMENT TOPOLOGY AND BÉZIER EXTRACTION OPERATOR
t=1;                % Thickness

% Load control points P_Rhino and extraction operators C_Rhino from
% Rhinoceros. The function ParseRhinoData.m reads and edits the data
load MachinePart.mat
[P,w,C,IEN]=ParseRhinoData(P_Rhino,C_Rhino);

poly=3;             % Polynomial order, same in both directions
nel=size(C,3);      % Total number of elements
ncp=size(P,1);      % Total number of basis functions/control points
gDof=2*ncp;         % Global degrees of freedom

% FOR STRUCTURE
K=zeros(gDof);
D=zeros(gDof,1);
R=zeros(gDof,1);

% LOADS
% Uniformly distributed load p=1 at right boundary
p=1;
R=FormRUniformLoadR_MP(IEN,P,R,nel,poly,w,C,p);

% BOUNDARY CONDITIONS
% Zero displacement in x direction at 4 locations left boundary
% Zero displacement in y direction at 3 locations bottom edge
prDofL1=find(P(:,1)<1e-5 & P(:,2)>38-1e-5 & P(:,2)<38+1e-5);
prDofL2=find(P(:,1)<1e-5 & P(:,2)>24-1e-5 & P(:,2)<24+1e-5);
prDofL3=find(P(:,1)<1e-5 & P(:,2)>11-1e-5 & P(:,2)<11+1e-5);
prDofB1=find(P(:,2)<1e-5 & P(:,1)<1e-5); % B1=L4
prDofB2=find(P(:,2)<1e-5 & P(:,1)>13.5-1e-5 & P(:,2)<13.5+1e-5);
prDofB3=find(P(:,2)<1e-5 & P(:,1)>26.5-1e-5 & P(:,2)<26.5+1e-5);
prDof=[prDofL1;prDofL2;prDofL3;prDofB1;prDofB1+ncp;prDofB2+ncp;prDofB3+ncp];

%----------------------------------SOLVE------------------------------------%

% STIFFNESS MATRIX
% The function FormK.m forms k for each element and assembles to K
K=FormK(IEN,P,K,E,t,nel,ncp,poly,w,C);

% SOLVE SYSTEM
% The function Solution.m solves the system in terms of active DOFs
[D,R]=Solution(gDof,prDof,K,D,R);
```

```
%----------------------------------POST-----------------------------------%

disp(['IGA MACHINE PART RHINO P3 (',num2str(gDof),' DOF)'])
SolutionTime=toc(tStart)

% Write displacements and reactions
Displacements=[(1:gDof)' D];
Reactions=[prDof R(prDof)];

% Calculate strains and stresses and write stresses at Gauss points
stress=Stresses(IEN,P,D,E,nel,ncp,poly,w,C);

% Calculate strain energy of the system
U1=D'*R
[U2,U3]=Energy(IEN,P,D,E,t,nel,ncp,poly,w,C)

toc(tStart)
```

B.4 Subfunctions

This section presents the subfunctions in alphabetical order, with the following properties listed:

Output:	Output variables from the subfunction, described in Appendix B.2
Input:	Input variables from the subfunction, described in Appendix B.2
Subfunctions:	Possible other subfunctions applied (called) in the subfunction

B.4.1 Bernstein Basis

This subfunction forms univariate Bernstein basis functions and derivatives using Eqs. (4.1) and (4.2) from Section 4.1.1.

Output:	`Bb, dB`
Input:	`xi, poly`

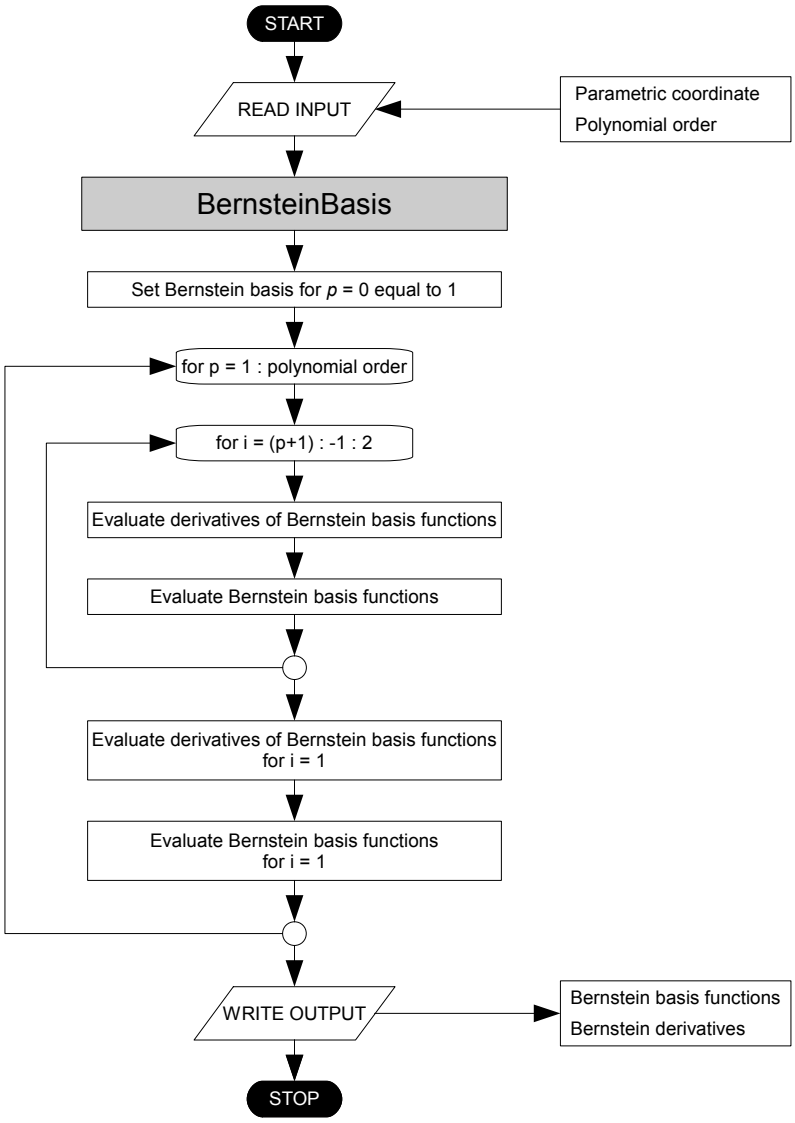


Figure B.7: Flow chart for BernsteinBasis.m

```
% This function forms univariate Bernstein basis functions and derivatives
%
% Output:   Bb - Bernstein basis functions over the interval [-1,1]
%           dB - derivatives of Bernstein basis functions over [-1,1]
%
% Input:    xi - parametric coordinate
%           poly - polynomial order

function [Bb,dB]=BernsteinBasis(xi,poly)

Bb=zeros(1,poly+1);
dB=zeros(1,poly+1);

% p=0
Bb(1)=1;

% p=1,2,3,...
for p=1:poly
    for i=(p+1):-1:2
        dB(i)=0.5*p*(Bb(i-1)-Bb(i));
        Bb(i)=0.5*(1-xi)*Bb(i)+0.5*(1+xi)*Bb(i-1);
    end
    dB(1)=-0.5*p*Bb(1);          % To avoid Bb(0)
    Bb(1)=0.5*(1-xi)*Bb(1);      % To avoid Bb(0)
end

end
```

B.4.2 Bernstein Basis Bivariate

This subfunction forms bivariate Bernstein basis functions and derivatives over the interval $[-1,1]\times[-1,1]$. The inputs are the univariate basis functions, found by employing the subfunction from Appendix B.4.1 twice.

Output: `Bb`, `dB`

Input: `Bbxi`, `Bbeta`, `dBxi`, `dBeta`, `poly`

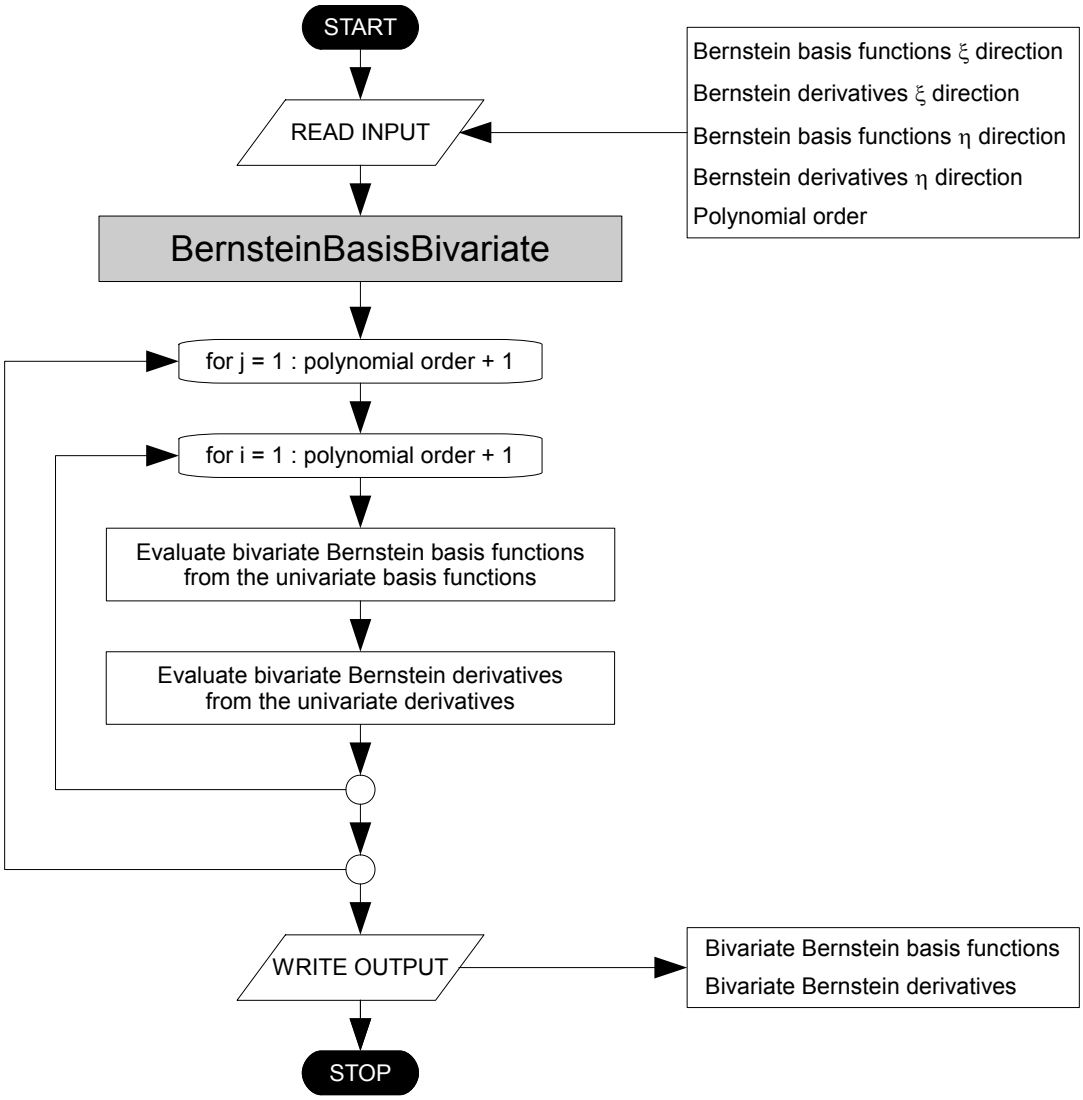


Figure B.8: Flow chart for BernsteinBasisBivariate.m

```
% This function forms bivariate Bernstein basis functions and derivatives
%
% Output:   Bb - Bernstein basis functions over the interval [-1,1]x[-1,1]
%           dB - derivatives of Bernstein basis functions
%
% Input:    Bbxi - Bernstein basis function in xi direction
%           Bbeta - Bernstein basis function in eta direction
%           dBxi - xi derivatives of Bernstein basis functions
%           dBeta - eta derivatives of Bernstein basis functions
%           poly - polynomial order

function [Bb,dB]=BernsteinBasisBivariate(Bbxi,Bbeta,dBxi,dBeta,poly)

Bb=zeros(1,(poly+1)^2);
dB=zeros(2,(poly+1)^2);

k=1;
for j=1:poly+1
    for i=1:poly+1
        Bb(k)=Bbxi(i)*Bbeta(j);
        dB(1,k)=dBxi(i)*Bbeta(j);
        dB(2,k)=Bbxi(i)*dBeta(j);
        k=k+1;
    end
end

end
```

B.4.3 Displacement C

This subfunction calculates vertical displacement at point C of Cook's problem by interpolating the basis functions over the middle element at the right hand side boundary. The number of elements in η direction should therefore be odd, `ny` =an odd number. An exception is for polynomial order 1 where the basis functions are exactly equal to 1 at the control points and no interpolation is necessary, but in return there must be a control point at the middle. The number of elements in η direction for `poly` = `1` should therefore be even, `ny` =an even number.

Note that the ξ and η values are given for the reference element level. Basis functions extracted are the bivariate NURBS basis functions which correspond to the right boundary.

Output: `vC`

Input: `IEN`, `D`, `nx`, `ny`, `ncp`, `poly`, `w`, `C`

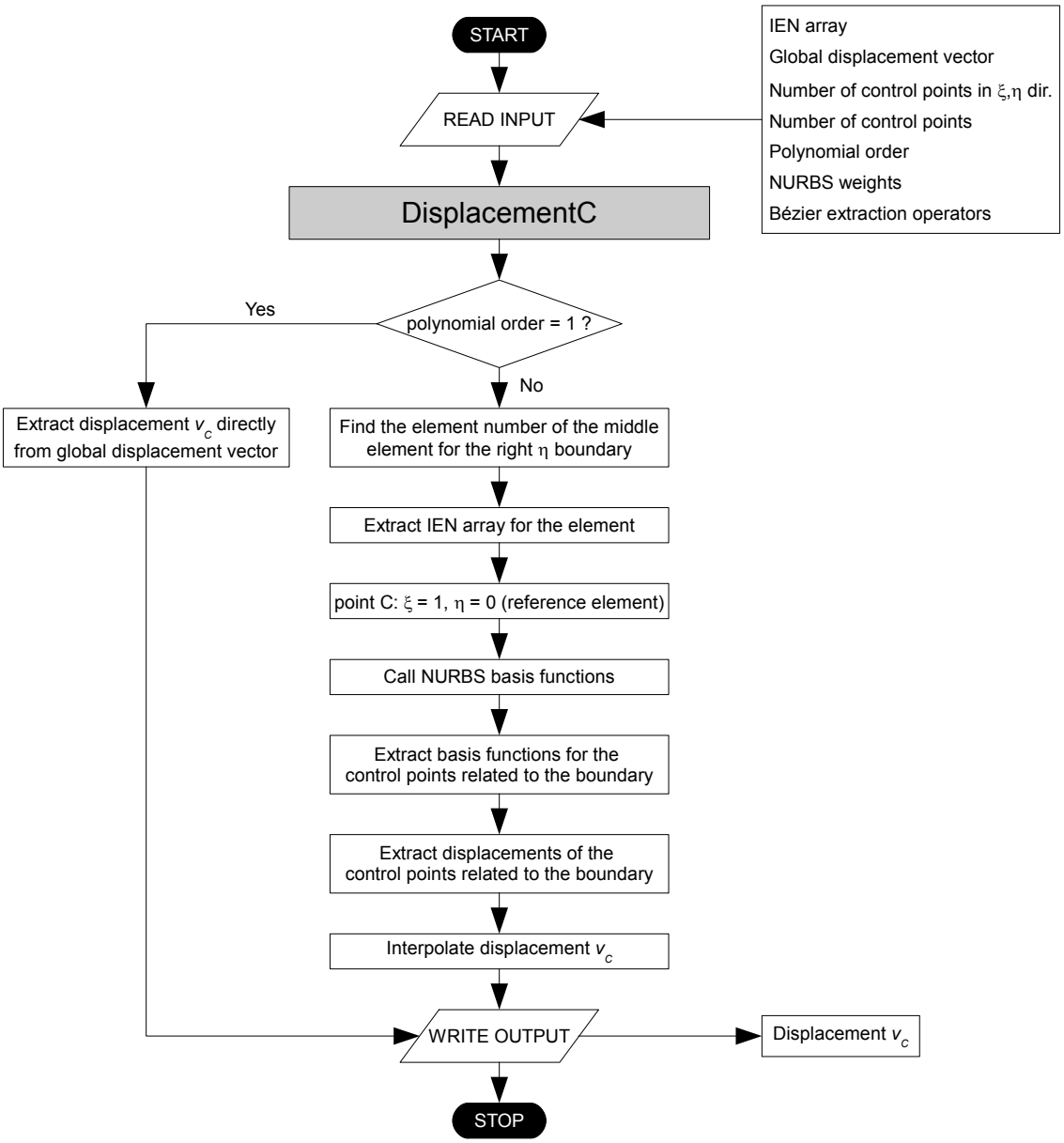


Figure B.9: Flow chart for DisplacementC.m

```
% This function calculates vertical displ. of point C in Cook's problem
% by interpolating the basis functions over the middle element at the
% right hand side boundary. The *number of elements in eta direction*
% should therefore be *odd*.
%
% An exception is for *IGA P1* where the basis functions are exactly equal
% to 1 at the control points and no interpolation is necessary, but in
% return there must be a control point at the middle. The *number of
% elements in eta direction* for poly=1 should therefore be *even*.
%
% Output:   vC - vertical displacement at point C
%
% Input:    IEN - element topology: numbering of control points
%           D - global displacement vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function vC=DisplacementC(IEN,D,nx,ny,ncp,poly,w,C)

if poly==1
    vC=D((ny/2+1)*(nx+poly)+ncp);
else
    s=(ny+1)/2;
    e=nx*s;
    IEN_e=IEN(e,:);

    xi=1;
    eta=0;

    [Rb,~]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);
    RbR=Rb((poly+1):(poly+1):(poly+1)^2);

    D_ey=D(IEN_e+ncp);
    D_eyR=D_ey((poly+1):(poly+1):(poly+1)^2);
    vC=RbR*D_eyR;
end

end
```

B.4.4 Energy

This subfunction calculates strain energy of the system by applying Eqs. (2.30) and (2.31).

Output:	`U2`, `U3`
Input:	`IEN`, `P`, `D`, `E`, `t`, `nel`, `ncp`, `poly`, `w`, `C`
Subfunctions:	Gauss.m, NURBSBasis.m, Jacobian.m

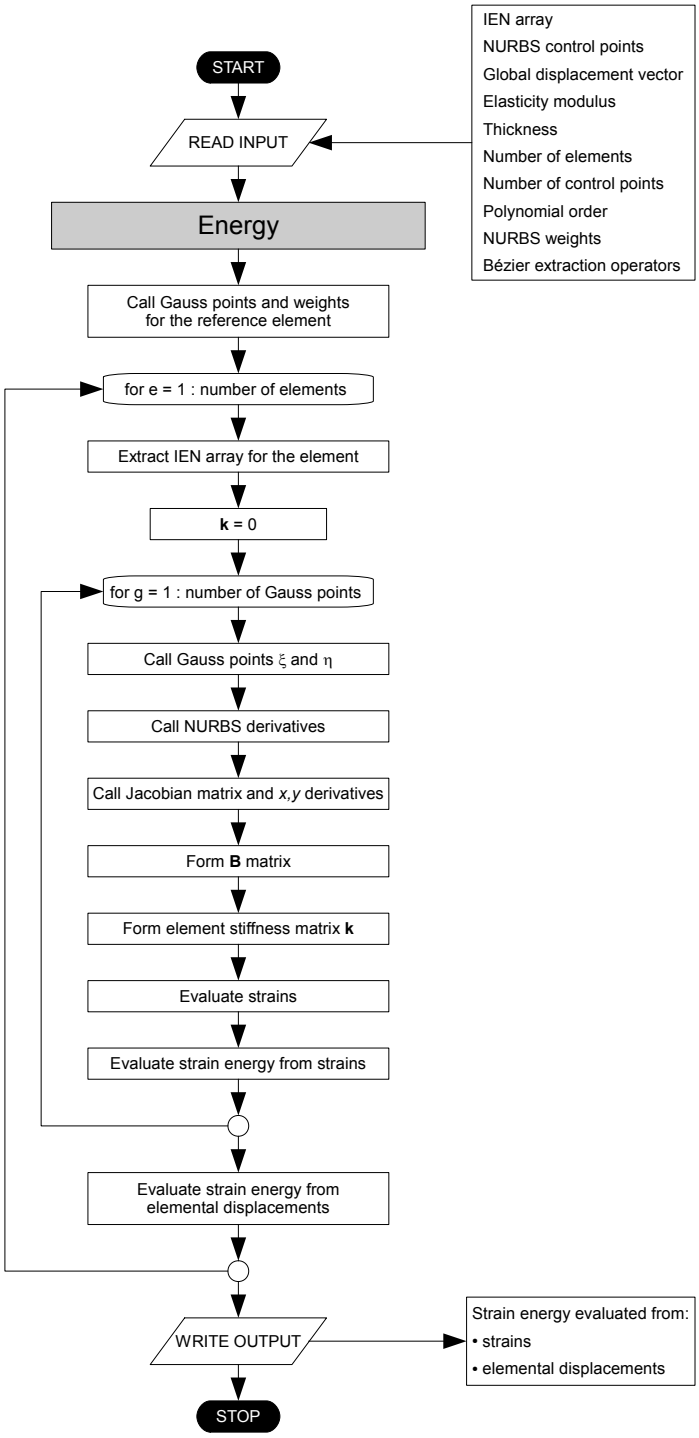


Figure B.10: Flow chart for Energy.m

```
% This function calculates strain energy of the system
%
% Output:   U2 - strain energy calculated from element stiffness matrices
%           U3 - strain energy calculated from strains
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           D - global displacement vector
%           E - constitutive matrix
%           t - thickness
%           nel - total number of elements
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function [U2,U3]=Energy(IEN,P,D,E,t,nel,ncp,poly,w,C)

[G,W]=Gauss(poly);                      % Call Gauss points and weights

U2=0;
U3=0;

for e=1:nel
    IEN_e=nonzeros(IEN(e,:))';          % Element topology of current el.
    eDof=[IEN_e IEN_e+ncp];             % eDof: first x, then y

    k=0;

    for g=1:size(G,1)                   % For each Gauss point
        xi=G(g,1);                      % Gauss coord. reference element
        eta=G(g,2);                     % Gauss coord. reference element

        [~,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);

        [J,dxy]=Jacobian(dR,P,IEN_e);

        B=zeros(3,2*length(IEN_e));     % B matrix
        B(1,1:length(IEN_e))=dxy(1,:);
        B(2,length(IEN_e)+1:2*length(IEN_e))=dxy(2,:);
        B(3,1:length(IEN_e))=dxy(2,:);
        B(3,length(IEN_e)+1:2*length(IEN_e))=dxy(1,:);

        k=k+B'*E*B*t*det(J)*W(g);       % Numerical integration of k
        strains=B*D(eDof);              % Strains (3x1) in each Gauss point
        U3=U3+t*strains'*E*strains*det(J)*W(g);
    end

    U2=U2+D(eDof)'*k*D(eDof);
end

end
```

B.4.5 Extraction Operator

This subfunction computes the localized univariate extraction operator as described in Borden et al. [3].

Output: `C`
Input: `XI`, `poly`

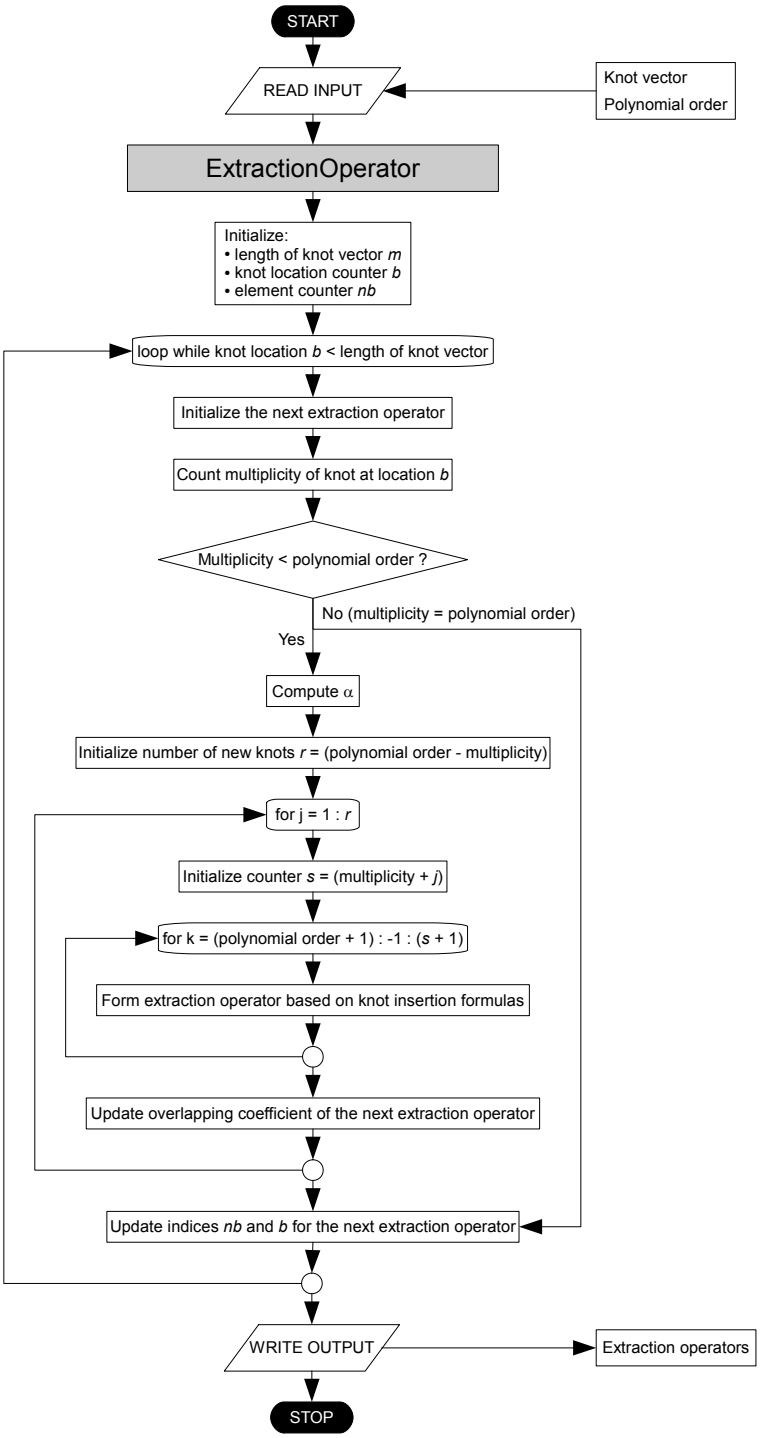


Figure B.11: Flow chart for ExtractionOperator.m

```
% This function computes the localized univariate extraction operators
%
% Output:   C(:,:,e) - extraction operators, e=1,2,..,nel+1 in xi direction
%           C(:,:,nel+1) - Identity matrix due to the algorithm, not used
%
% Input:    XI - knot vector
%           poly - polynomial order

function C=ExtractionOperator(XI,poly)

% Initializations
m=length(XI);
a=poly+1;
b=a+1;
nb=1;
C(:,:,1)=eye(poly+1);

while b<m
    C(:,:,nb+1)=eye(poly+1);    % Initialize the next extraction operator
    i=b;

    % Count multiplicity of the knot at location b
    while b<m && XI(b+1)==XI(b)
        b=b+1;
    end
        mult=b-i+1;

    if mult<poly
        numer=XI(b)-XI(a);

        for j=poly:-1:mult+1
            alphas(j-mult)=numer/(XI(a+j)-XI(a));
        end

        r=poly-mult;

        % Update the matrix coefficient for r new knots
        for j=1:r
            save=r-j+1;
            s=mult+j;

            for k=poly+1:-1:s+1
                alpha=alphas(k-s);
                % Form extraction operator
                C(:,k,nb)=alpha*C(:,k,nb)+(1-alpha)*C(:,k-1,nb);
            end

            if b<m
                % Update overlapping coefficients of the next operator
                C(save:j+save,save,nb+1)=C(poly-j+1:poly+1,poly+1,nb);
            end
        end

        % Finished with the current operator.
        % Update indices for the next operator.
        nb=nb+1;
```

```
        if b<m
            a=b;
            b=b+1;
        end

    elseif mult==poly   % In case multiplicity of knot is already p,

        % update indices for the next operator.
        nb=nb+1;
        if b<m
            a=b;
            b=b+1;
        end
    end
end

end
```

B.4.6 Extraction Operator Bivariate

This subfunction computes the localized bivariate extraction operators by taking the tensor product of the univariate extraction operators. The univariate extraction operators are found by employing the subfunction from Appendix B.4.5 twice.

Output: `C`

Input: `Cxi`, `Ceta`, `poly`, `nx`, `ny`

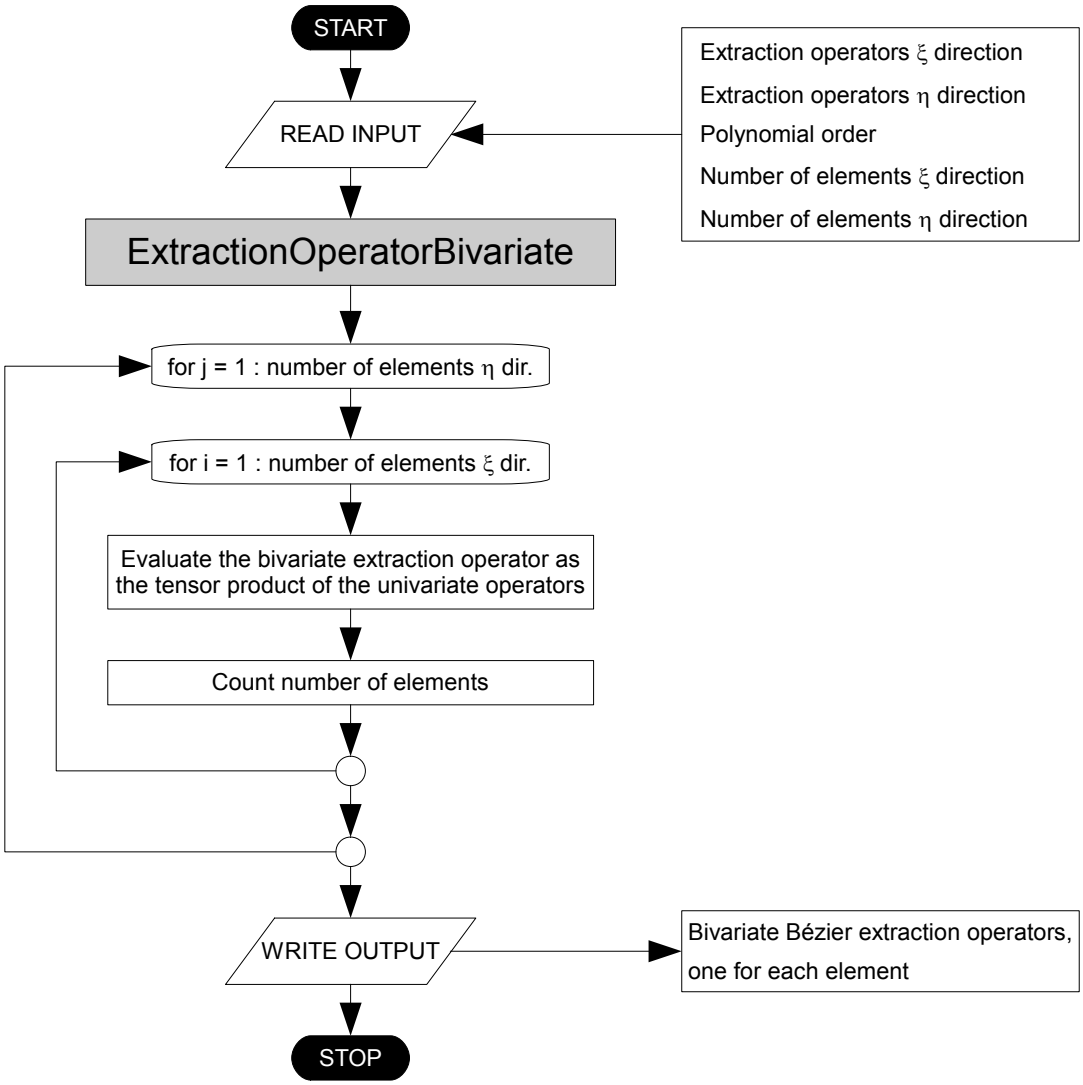


Figure B.12: Flow chart for ExtractionOperatorBivariate.m

```
% This function computes the localized bivariate extraction operators
%
% Output:   C(:,:,e) - bivariate extraction operators, e=1,2,..,nel
%
% Input:    Cxi(:,:,e) - univariate extraction operators, xi direction
%           Ceta(:,:,e) - univariate extraction operators, eta direction

function C=ExtractionOperatorBivariate(Cxi,Ceta,poly,nx,ny)

C=zeros((poly+1)^2,(poly+1)^2,nx*ny);

k=1;
for j=1:ny
    for i=1:nx
        C(:,:,k)=kron(Ceta(:,:,j),Cxi(:,:,i));
        k=k+1;
    end
end

end
```

B.4.7 Form K

This subfunction forms the global stiffness matrix for plane stress or plane strain by employing Eq. (2.21).

Output:	`K`
Input:	`IEN`, `P`, `K`, `E`, `t`, `nel`, `ncp`, `poly`, `w`, `C`
Subfunctions:	Gauss.m, NURBSBasis.m, Jacobian.m

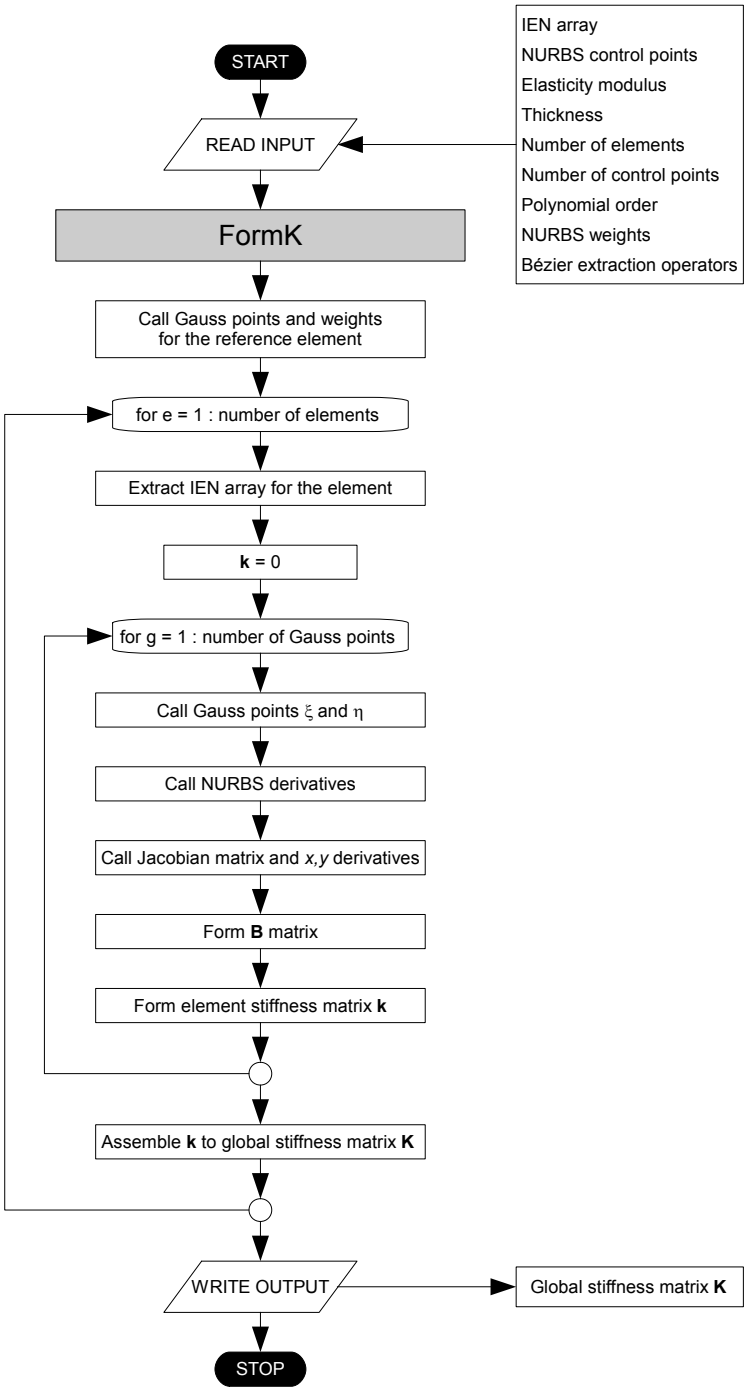


Figure B.13: Flow chart for FormK.m

```
% This function forms global stiffness matrix (plane stress/strain)
%
% Output:   K - global stiffness matrix
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           K - empty global stiffness matrix
%           E - constitutive matrix
%           t - thickness
%           nel - total number of elements
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function K=FormK(IEN,P,K,E,t,nel,ncp,poly,w,C)

[G,W]=Gauss(poly);                          % Call Gauss points and weights

for e=1:nel
    IEN_e=nonzeros(IEN(e,:))';              % Element topology of current el.
    eDof=[IEN_e IEN_e+ncp];                 % eDof: first x, then y

    k=0;

    for g=1:size(G,1)                       % For each Gauss point
        xi=G(g,1);                          % Gauss coord. reference element
        eta=G(g,2);                         % Gauss coord. reference element

        [~,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);

        [J,dxy]=Jacobian(dR,P,IEN_e);

        B=zeros(3,2*length(IEN_e));         % B matrix
        B(1,1:length(IEN_e))=dxy(1,:);
        B(2,length(IEN_e)+1:2*length(IEN_e))=dxy(2,:);
        B(3,1:length(IEN_e))=dxy(2,:);
        B(3,length(IEN_e)+1:2*length(IEN_e))=dxy(1,:);

        k=k+B'*E*B*t*det(J)*W(g);           % Numerical integration of k
    end

    K(eDof,eDof)=K(eDof,eDof)+k;
end

end
```

B.4.8 Form R End Loaded Beam L and Form R End Loaded Beam R

These subfunctions form global load vector for the end loaded beam for the left and right hand side as described in Appendix A.2.

Output:	`R`
Input:	`IEN`, `P`, `R`, `nx`, `ny`, `ncp`, `poly`, `w`, `C`
Subfunctions:	GaussBoundary.m, NURBSBasis.m, Jacobian.m

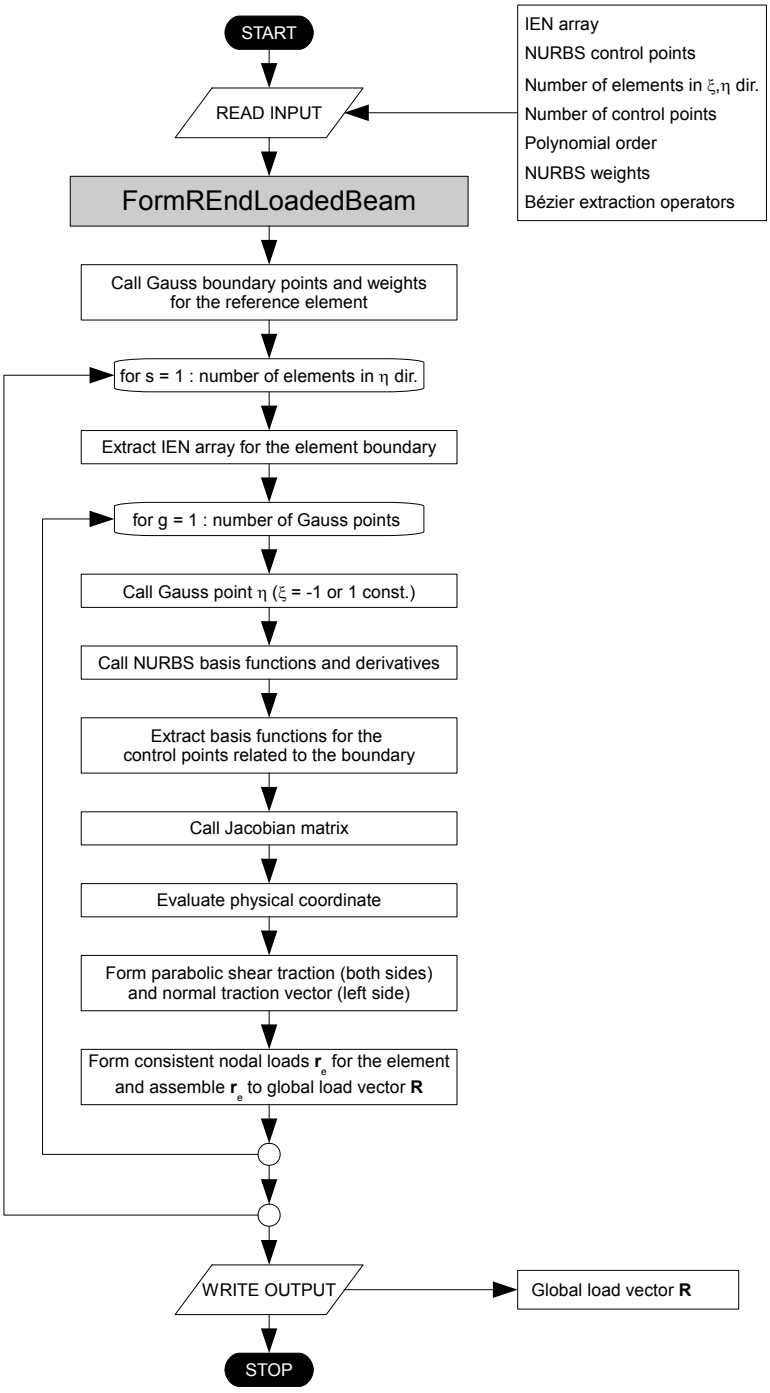


Figure B.14: Flow chart for FormREndLoadedBeamL.m and FormREndLoadedBeamR.m

```
% This function forms global load vector for a parabolic shear traction
% (sum = 80) and a normal traction (+- 120 top/bottom) at left hand side
%
% Output:   R - global load vector
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           R - empty global load vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function R=FormREndLoadedBeamL(IEN,P,R,nx,ny,ncp,poly,w,C)

[G,W]=GaussBoundary(poly);          % Call Gauss boundary points & weights

xi=-1;

for s=1:ny
    e=1+nx*(s-1);
    IEN_e=IEN(e,:);                 % Element topology of current element
    IEN_ey=IEN_e+ncp;
    % Control points x direction at left hand side for each element
    edgeDofxL=IEN_e(1:(poly+1):(poly+1)^2-poly);
    % Control points y direction at left hand side for each element
    edgeDofyL=IEN_ey(1:(poly+1):(poly+1)^2-poly);

    for g=1:size(G,1)               % For each Gauss boundary point
        eta=G(g);                   % Gauss coord. reference element

        [Rb,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);
        RbL=Rb(1:(poly+1):(poly+1)^2);
        [J,~]=Jacobian(dR,P,IEN_e);

        y=Rb*P(IEN_e,2);                    % Phys. coord. of Gauss point
        p=12*y; pv=[p p p p p]';            % Form normal traction vector
        q=6-(3/50)*y^2; qv=[-q -q -q -q -q]';% Form parabolic shear traction

        R(edgeDofxL)=R(edgeDofxL)+RbL'*RbL*pv(1:poly+1)*...
            sqrt(J(2,1)^2+J(2,2)^2)*W(g);
        R(edgeDofyL)=R(edgeDofyL)+RbL'*RbL*qv(1:poly+1)*...
            sqrt(J(2,1)^2+J(2,2)^2)*W(g);
    end
end

end
```

```
% This function forms global load vector for a parabolic shear traction
% (sum = 80) at right hand side
%
% Output:   R - global load vector
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           R - empty global load vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function R=FormREndLoadedBeamR(IEN,P,R,nx,ny,ncp,poly,w,C)

[G,W]=GaussBoundary(poly);          % Call Gauss boundary points & weights

xi=1;

for s=1:ny
    e=nx*s;
    IEN_e=IEN(e,:);                 % Element topology of current element
    IEN_ey=IEN_e+ncp;
    % Control points y direction at right hand side for each element
    edgeDofyR=IEN_ey((poly+1):(poly+1):(poly+1)^2);

    for g=1:size(G,1)               % For each Gauss boundary point
        eta=G(g);                   % Gauss coord. reference element

        [Rb,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);
        RbR=Rb((poly+1):(poly+1):(poly+1)^2);
        [J,~]=Jacobian(dR,P,IEN_e);

        y=Rb*P(IEN_e,2);                % Phys. coord. of Gauss point
        q=6-(3/50)*y^2; qv=[q q q q q]';% Form parabolic shear traction

        R(edgeDofyR)=R(edgeDofyR)+RbR'*RbR*qv(1:poly+1)*...
            sqrt(J(2,1)^2+J(2,2)^2)*W(g);
    end
end

end
```

B.4.9 Form R Infinite Plate

This subfunction forms global load vector for the infinite plate with circular hole as described in Section 6.3.

Output:	`R`
Input:	`IEN`, `P`, `R`, `nx`, `ny`, `ncp`, `poly`, `w`, `C`, `r_inn`, `r_out`, `Tx`
Subfunctions:	GaussBoundary.m, NURBSBasis.m, Jacobian.m

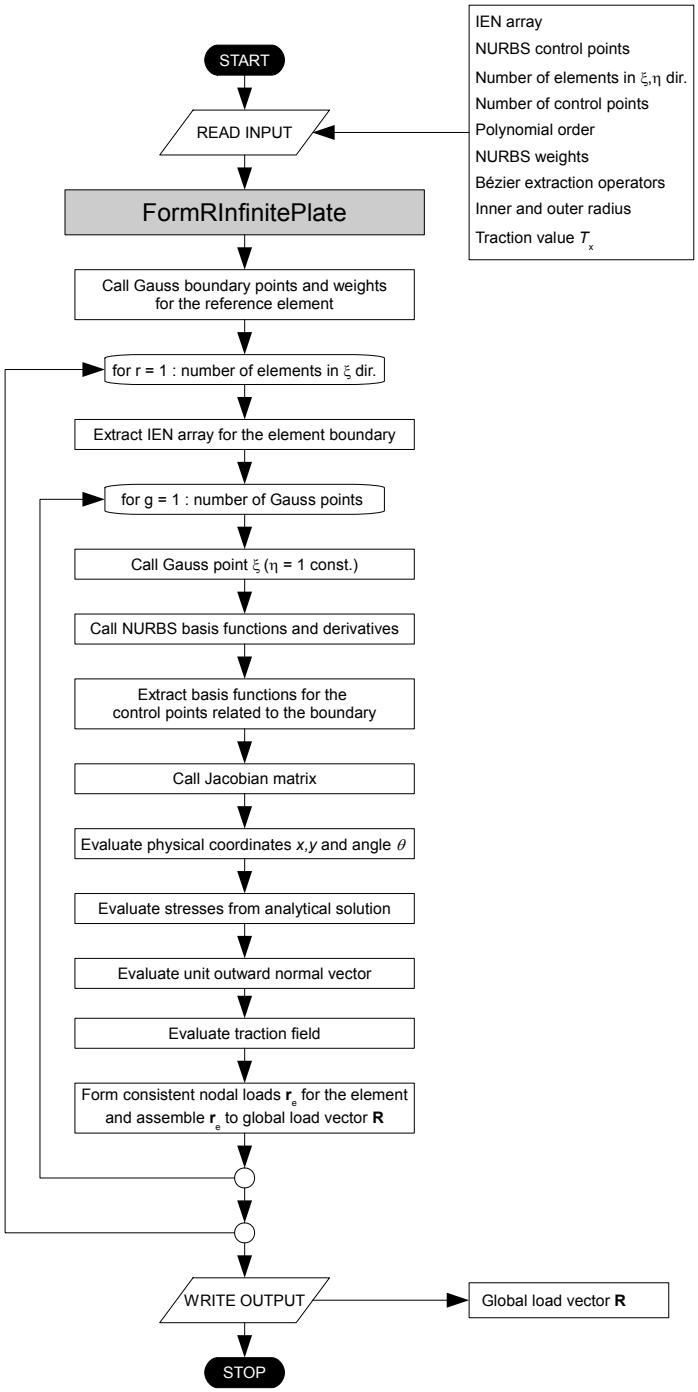


Figure B.15: Flow chart for FormRInfinitePlate.m

```
% This function forms global load vector for the infinite plate with
% circular hole at the outer edge
%
% Output:   R - global load vector
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           R - empty global load vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators
%           r_inn - inner radius
%           r_out - outer radius
%           Tx - traction value

function R=FormRInfinitePlate(IEN,P,R,nx,ny,ncp,poly,w,C,r_inn,r_out,Tx)

[G,W]=GaussBoundary(poly);          % Call Gauss boundary points & weights

eta=1;

for r=1:nx
    e=(ny-1)*nx+r;                  % Current element number
    IEN_e=IEN(e,:);                 % Element topology of current element
    IEN_ey=IEN_e+ncp;
    % Control points x direction at top side for each element
    edgeDofxT=IEN_e(((poly+1)^2-poly):(poly+1)^2);
    % Control points y direction at top side for each element
    edgeDofyT=IEN_ey(((poly+1)^2-poly):(poly+1)^2);

    for g=1:size(G,1)               % For each Gauss boundary point
        xi=G(g);                    % Gauss coord. reference element

        [Rb,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);
        RbT=Rb(((poly+1)^2-poly):(poly+1)^2);
        [J,~]=Jacobian(dR,P,IEN_e);

        x=Rb*P(IEN_e,1);
        y=Rb*P(IEN_e,2);
        theta=atan(y/x);

        % Stresses av given from analytical solution
        sigma_x=Tx*(1-(r_inn/r_out)^2*(1.5*cos(2*theta)+cos(4*theta))+ ...
            1.5*(r_inn/r_out)^4*cos(4*theta));
        sigma_y=Tx*(-(r_inn/r_out)^2*(0.5*cos(2*theta)-cos(4*theta))- ...
            1.5*(r_inn/r_out)^4*cos(4*theta));
        tau_xy=Tx*(-(r_inn/r_out)^2*(0.5*sin(2*theta)+sin(4*theta))+ ...
            1.5*(r_inn/r_out)^4*sin(4*theta));

        n=(1/sqrt(x^2+y^2))*[x y]';             % Unit outward normal vector
        phi1=sigma_x*n(1)+tau_xy*n(2);          % Traction 1
        phi2=tau_xy*n(1)+sigma_y*n(2);          % Traction 2

        R(edgeDofxT)=R(edgeDofxT)+RbT'*phi1*sqrt(J(1,1)^2+J(1,2)^2)*W(g);
```

```
        R(edgeDofyT)=R(edgeDofyT)+RbT'*phi2*sqrt(J(1,1)^2+J(1,2)^2)*W(g);
    end
end

end
```

B.4.10 Form R Uniform Load R

This subfunction forms global load vector for a uniform load at the η boundary (on the right hand side), described in Section 5.1.1.

Output:	`R`
Input:	`IEN`, `P`, `R`, `nx`, `ny`, `ncp`, `poly`, `w`, `C`, `q`
Subfunctions:	GaussBoundary.m, NURBSBasis.m, Jacobian.m

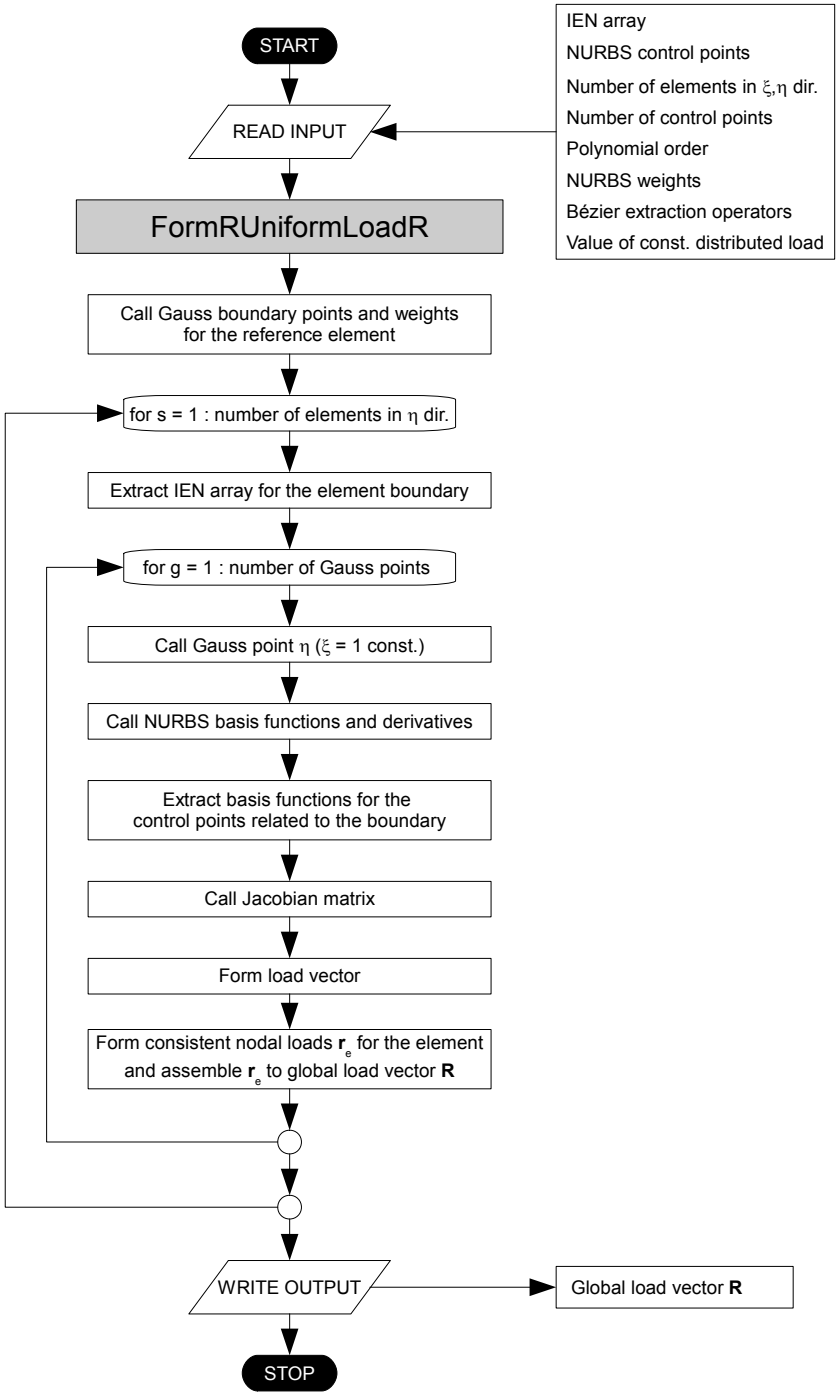


Figure B.16: Flow chart for FormRUniformLoadR.m

```
% This function forms global load vector for a uniformly distributed shear
% traction at right hand side
%
% Output:   R - global load vector
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           R - empty global load vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators
%           q - constant value of distributed load

function R=FormRUniformLoadR(IEN,P,R,nx,ny,ncp,poly,w,C,q)

[G,W]=GaussBoundary(poly);          % Call Gauss boundary points & weights

xi=1;

for s=1:ny
    e=nx*s;
    IEN_e=IEN(e,:);                 % Element topology of current element
    IEN_ey=IEN_e+ncp;
    % Control points y direction at right hand side for each element
    edgeDofyR=IEN_ey((poly+1):(poly+1):(poly+1)^2);

    for g=1:size(G,1)               % For each Gauss boundary point
        eta=G(g);                   % Gauss coord. reference element

        [Rb,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);
        RbR=Rb((poly+1):(poly+1):(poly+1)^2);
        [J,~]=Jacobian(dR,P,IEN_e);

        qv=[q q q q q]';            % Form load vector, (poly+1)x1

        R(edgeDofyR)=R(edgeDofyR)+RbR'*RbR*qv(1:poly+1)*...
            sqrt(J(2,1)^2+J(2,2)^2)*W(g);
    end
end

end
```

B.4.11 Form R Uniform Load R MP

This subfunction forms global load vector for a uniform load at the right hand side for the machine part. The subfunction differ from the function given in Appendix B.4.10 in that the loop through the elements on the η boundary (on the right hand side) is replaced by a loop through all the elements due to no tensor product for the T-mesh of the machine part. Because of this, each element must also be checked if it is on the boundary or not before the element topology for the boundary control points can be extracted.

Output:	`R`
Input:	`IEN`, `P`, `R`, `nel`, `poly`, `w`, `C`, `p`
Subfunctions:	GaussBoundary.m, NURBSBasis.m, Jacobian.m

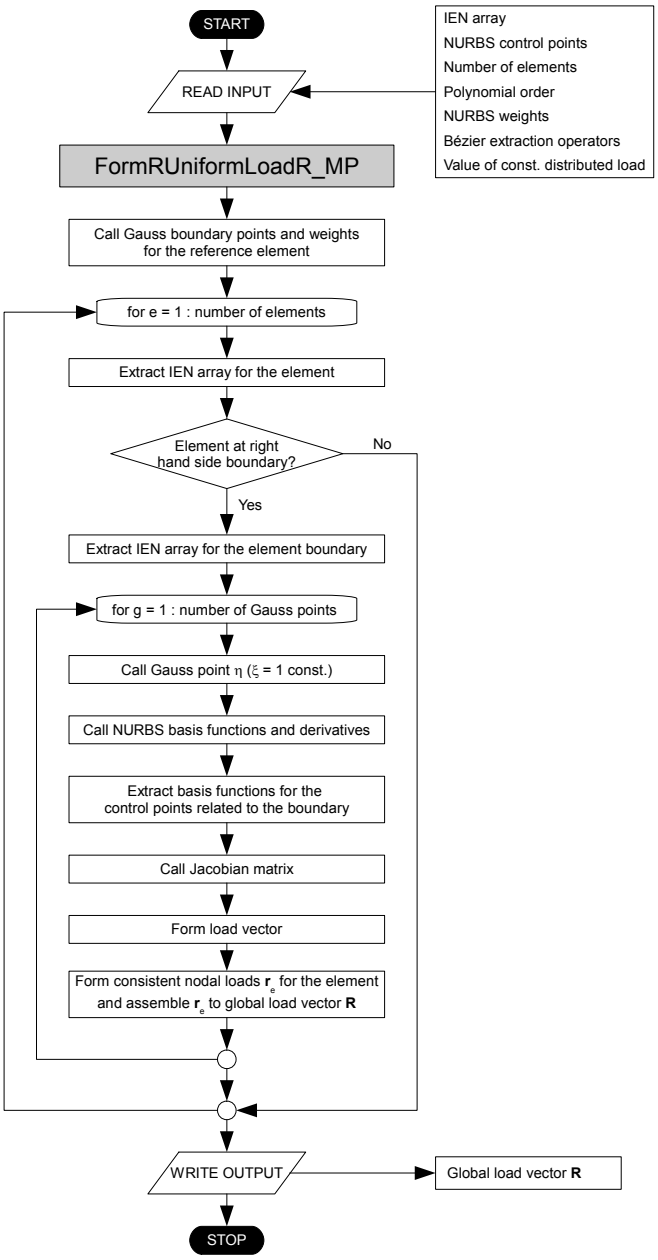


Figure B.17: Flow chart for FormRUniformLoadR_MP.m

```
% This function forms global load vector for a uniformly distributed normal
% traction at the right hand side for the machine part
%
% Output:   R - global load vector
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           R - empty global load vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators
%           p - constant value of distributed load

function R=FormRUniformLoadR_MP(IEN,P,R,nel,poly,w,C,p)

[G,W]=GaussBoundary(poly);          % Call Gauss boundary points & weights

xi=1;

for e=1:nel
    IEN_e=nonzeros(IEN(e,:))';      % Element topology of current element

    % Find if the element is at the right boundary of the machine part
    if size(find(P(IEN_e,1)>53-1e-5 & P(IEN_e,1)<53+1e-5),1)==4;

        % Control points x direction at right hand side for each element
        temp=P(IEN_e,1)>53-1e-5 & P(IEN_e,1)<53+1e-5;
        edgeDofxR=IEN_e(temp);

            for g=1:size(G,1)           % For each Gauss boundary point
                eta=G(g);               % Gauss coord. reference element

                [Rb,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);
                RbR=Rb(temp);
                [J,~]=Jacobian(dR,P,IEN_e);

                R(edgeDofxR)=R(edgeDofxR)+RbR'*RbR*[p p p p]'*...
                sqrt(J(2,1)^2+J(2,2)^2)*W(g);
            end
    end
end

end
```

B.4.12 Gauss, Gauss Boundary and Gauss Matrix

Both subfunctions Gauss.m and GaussMatrix.m contain two-dimensional Gauss quadrature for full numerical integration. Gauss.m gives the Gauss points from left to right and bottom to top; ξ values in 1st column; η values in 2nd column and corresponding weights in an own column vector. GaussMatrix.m gives the same Gauss points in three matrices; ξ values in 1st matrix, η values in 2nd matrix and corresponding weights in a 3^{rd} matrix. Rows are along ξ axis, columns are along η axis. The subfunction GaussBoundary.m contains one-dimensional Gauss quadrature for full numerical integration given as two column vectors; one for Gauss points and one for weights.

Output: `G` and possible `W`
Input: `poly`

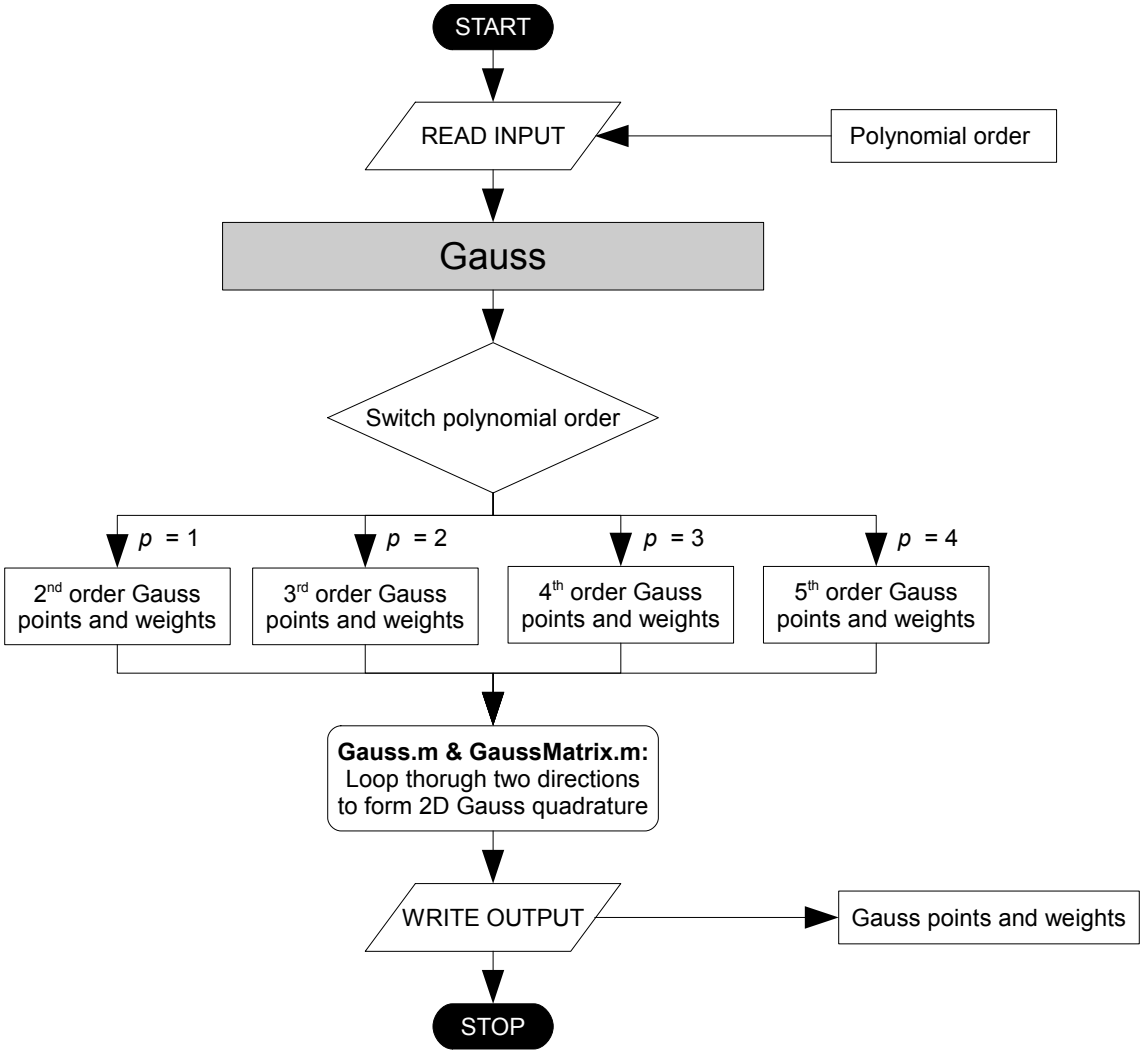


Figure B.18: Flow chart for Gauss.m, GaussBoundary.m and GaussMatrix.m

```
% This function contains 2D Gauss quadrature for full numerical integration
%
% Output:   G - Gauss points given from left to right and bottom to top,
%               1st column = xi value, 2nd column = eta value
%           W - corresponding weights to Gauss points
%
% Input:    poly - polynomial order

function [G,W]=Gauss(poly)

switch poly
    case 1 % Gauss order p=2 (2x2), for IGA P1
        G=[-1/sqrt(3) 1/sqrt(3)]';
        W_temp=[1 1]';
    case 2 % Gauss order p=3 (3x3), for IGA P2
        G=[-sqrt(0.6) 0 sqrt(0.6)]';
        W_temp=[5/9 8/9 5/9]';
    case 3 % Gauss order p=4 (4x4), for IGA P3
        G=[-sqrt((3+2*sqrt(1.2))/7) -sqrt((3-2*sqrt(1.2))/7) ...
            sqrt((3-2*sqrt(1.2))/7) sqrt((3+2*sqrt(1.2))/7)]';
        W_temp=[(0.5-sqrt(30)/36) (0.5+sqrt(30)/36) (0.5+sqrt(30)/36) ...
            (0.5-sqrt(30)/36)]';
    case 4 % Gauss order p=5 (5x5), for IGA P4
        G=[-(1/3)*sqrt(5+2*sqrt(10/7)) -(1/3)*sqrt(5-2*sqrt(10/7)) ...
            0 (1/3)*sqrt(5-2*sqrt(10/7)) (1/3)*sqrt(5+2*sqrt(10/7))]';
        W_temp=[(322-13*sqrt(70))/900 (322+13*sqrt(70))/900 128/225 ...
            (322+13*sqrt(70))/900 (322-13*sqrt(70))/900]';
end

W=zeros((poly+1)^2,1);
k=1;
for j=1:poly+1
    for i=1:poly+1
        G(k,1)=G(i);
        G(k,2)=G(j);
        W(k)=W_temp(i)*W_temp(j);
        k=k+1;
    end
end

end
```

```
% This function contains 1D Gauss quadrature for full numerical integration
%
% Output:   G - Gauss points given from left to right and bottom to top,
%               1st column = xi value, 2nd column = eta value
%           W - corresponding weights to Gauss points
%
% Input:    poly - polynomial order

function [G,W]=GaussBoundary(poly)

switch poly
    case 1 % Gauss order p=2 (2x2), for IGA P1
        G=[-1/sqrt(3) 1/sqrt(3)]';
        W=[1 1]';
    case 2 % Gauss order p=3 (3x3), for IGA P2
        G=[-sqrt(0.6) 0 sqrt(0.6)]';
        W=[5/9 8/9 5/9]';
    case 3 % Gauss order p=4 (4x4), for IGA P3
        G=[-sqrt((3+2*sqrt(1.2))/7) -sqrt((3-2*sqrt(1.2))/7) ...
            sqrt((3-2*sqrt(1.2))/7) sqrt((3+2*sqrt(1.2))/7)]';
        W=[(0.5-sqrt(30)/36) (0.5+sqrt(30)/36) (0.5+sqrt(30)/36) ...
            (0.5-sqrt(30)/36)]';
    case 4 % Gauss order p=5 (5x5), for IGA P4
        G=[-(1/3)*sqrt(5+2*sqrt(10/7)) -(1/3)*sqrt(5-2*sqrt(10/7)) ...
            0 (1/3)*sqrt(5-2*sqrt(10/7)) (1/3)*sqrt(5+2*sqrt(10/7))]';
        W=[(322-13*sqrt(70))/900 (322+13*sqrt(70))/900 128/225 ...
            (322+13*sqrt(70))/900 (322-13*sqrt(70))/900]';
end

end
```

```
% This function contains 2D Gauss quadrature for full numerical integration
%
% Output:   G - Gauss points given as three matrices. Rows are along
%               xi axis, columns are along eta axis
%               G(:,:,1) - xi values
%               G(:,:,2) - eta values
%               G(:,:,3) - weights
%
% Input:    poly - polynomial order

function G=GaussMatrix(poly)

switch poly
    case 1 % Gauss order p=2 (2x2), for IGA P1
        G=[-1/sqrt(3) 1/sqrt(3)]';
        W=[1 1]';
    case 2 % Gauss order p=3 (3x3), for IGA P2
        G=[-sqrt(0.6) 0 sqrt(0.6)]';
        W=[5/9 8/9 5/9]';
    case 3 % Gauss order p=4 (4x4), for IGA P3
        G=[-sqrt((3+2*sqrt(1.2))/7) -sqrt((3-2*sqrt(1.2))/7) ...
            sqrt((3-2*sqrt(1.2))/7) sqrt((3+2*sqrt(1.2))/7)]';
        W=[(0.5-sqrt(30)/36) (0.5+sqrt(30)/36) (0.5+sqrt(30)/36) ...
            (0.5-sqrt(30)/36)]';
    case 4 % Gauss order p=5 (5x5), for IGA P4
        G=[-(1/3)*sqrt(5+2*sqrt(10/7)) -(1/3)*sqrt(5-2*sqrt(10/7)) ...
            0 (1/3)*sqrt(5-2*sqrt(10/7)) (1/3)*sqrt(5+2*sqrt(10/7))]';
        W=[(322-13*sqrt(70))/900 (322+13*sqrt(70))/900 128/225 ...
            (322+13*sqrt(70))/900 (322-13*sqrt(70))/900]';
end

for j=1:poly+1
    for i=1:poly+1
        G(i,j,1)=G(i);
        G(i,j,2)=G(j);
        G(i,j,3)=W(i)*W(j);
    end
end

end
```

B.4.13 Generate Quarter Disk

This subfunction generates initial control points and weights for a quarter of a circular disk as described in Section 5.2.1.

Output: `B, knot`

Input: `r_inn, r_out, poly`

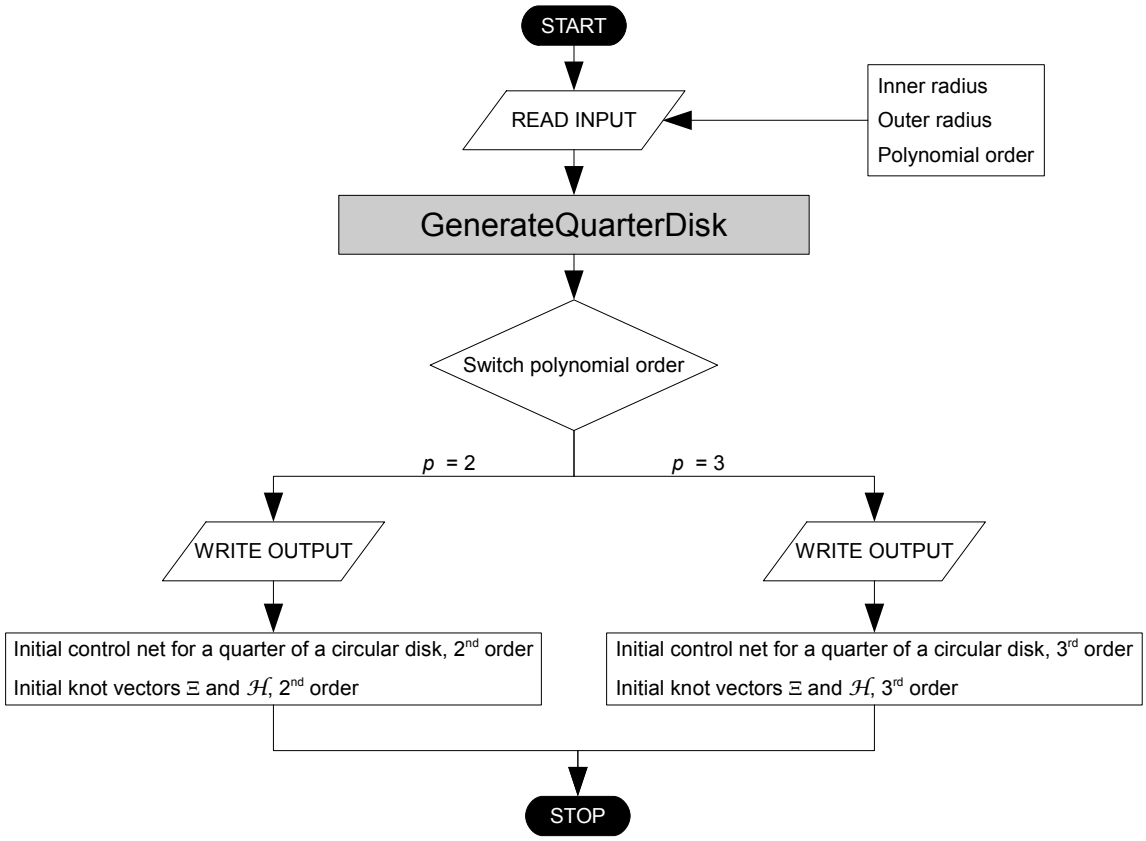


Figure B.19: Flow chart for GenerateQuarterDisk.m

```
% This function generates control points and weights for a quarter of a
% circular disk, degree 2 or 3
%
% Output:   B - control points given as a 3x3x3 matrix
%               B(:,:,1) - x coordinates
%               B(:,:,2) - y coordinates
%               B(:,:,3) - weights
%           knot - knot vector given as a Matlab structure
%               knot.xi - knot vector in xi direction
%               knot.eta - knot vector in eta direction
%
% Input:    r_inn - inner radius
%           r_out - outer radius
%           poly - polynomial order

function [B,knot]=GenerateQuarterDisk(r_inn,r_out,poly)

switch poly
    case 2 % IGA P2
        knot=struct('xi',[0 0 0 1 1 1],'eta',[0 0 0 1 1 1]);

        p1=[0 r_inn];       p4=[0 r_out];
        p2=[r_inn r_inn];   p5=[r_out r_out];
        p3=[r_inn 0];       p6=[r_out 0];

        B=zeros(3,3,3);

        B(1,1,1:2)=p1;      B(1,2,1:2)=(p1+p4)/2;   B(1,3,1:2)=p4;
        B(2,1,1:2)=p2;      B(2,2,1:2)=(p2+p5)/2;   B(2,3,1:2)=p5;
        B(3,1,1:2)=p3;      B(3,2,1:2)=(p3+p6)/2;   B(3,3,1:2)=p6;

        B(1,1,3)=1;         B(1,2,3)=1;             B(1,3,3)=1;
        B(2,1,3)=1/sqrt(2); B(2,2,3)=1/sqrt(2);     B(2,3,3)=1/sqrt(2);
        B(3,1,3)=1;         B(3,2,3)=1;             B(3,3,3)=1;

    case 3 % IGA P3
        knot=struct('xi',[0 0 0 0 1 1 1 1],'eta',[0 0 0 0 1 1 1 1]);

        e=sqrt(2)/(1+sqrt(2));
        r1=r_inn;
        r2=r_inn+(r_out-r_inn)/3;
        r3=r_inn+2*(r_out-r_inn)/3;
        r4=r_out;
        we=(1+sqrt(2))/3;

        B=zeros(4,4,3);

        B(1,1,1:2)=[0 r1]; B(1,2,1:2)=[0 r2]; B(1,3,1:2)=[0 r3];
            B(1,4,1:2)=[0 r4];
        B(2,1,1:2)=[e*r1 r1]; B(2,2,1:2)=[e*r2 r2]; B(2,3,1:2)=[e*r3 r3];
            B(2,4,1:2)=[e*r4 r4];
        B(3,1,1:2)=[r1 e*r1]; B(3,2,1:2)=[r2 e*r2]; B(3,3,1:2)=[r3 e*r3];
            B(3,4,1:2)=[r4 e*r4];
        B(4,1,1:2)=[r1 0]; B(4,2,1:2)=[r2 0]; B(4,3,1:2)=[r3 0];
            B(4,4,1:2)=[r4 0];
```

```
        B(1,1,3)=1; B(1,2,3)=1; B(1,3,3)=1; B(1,4,3)=1;
        B(2,1,3)=we; B(2,2,3)=we; B(2,3,3)=we; B(2,4,3)=we;
        B(3,1,3)=we; B(3,2,3)=we; B(3,3,3)=we; B(3,4,3)=we;
        B(4,1,3)=1; B(4,2,3)=1; B(4,3,3)=1; B(4,4,3)=1;
end

end
```

B.4.14 Generate Square

This subfunction generates initial control points and weights for a square as described in Section 5.2.1.

Output: `B`, `knot`

Input: `XY`, `poly`

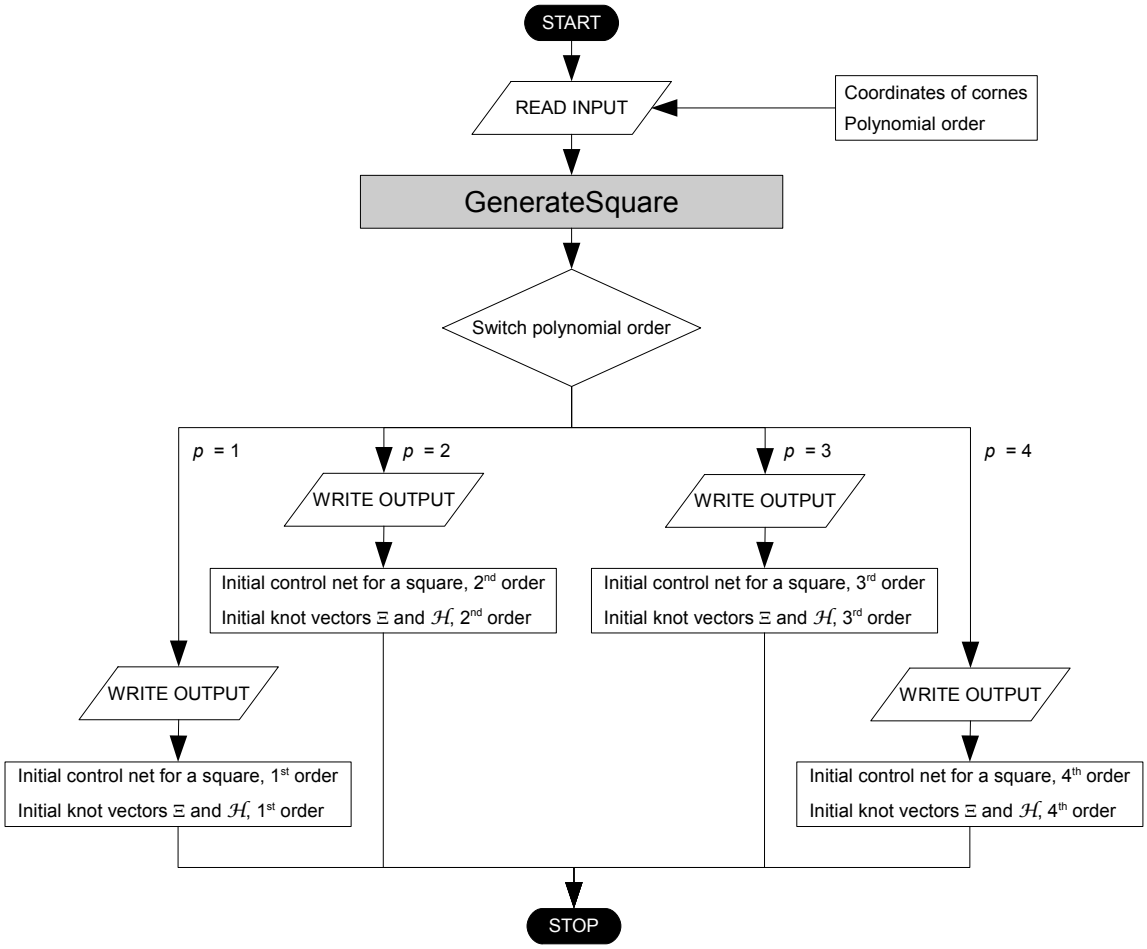


Figure B.20: Flow chart for GenerateSquare.m

```
% This function generates control points and weights for a square,
% degree 1, 2, 3 or 4
%
% Output:   B - control points given as a 3x3x3 matrix
%               B(:,:,1) - x coordinates
%               B(:,:,2) - y coordinates
%               B(:,:,3) - weights
%           knot - knot vector given as a Matlab structure
%               knot.xi - knot vector in xi direction
%               knot.eta - knot vector in eta direction
%
% Input:    XY - coordinates of corners [X1 Y1;...;X4 Y4], left to right
%               and bottom to top
%           poly - polynomial order

function [B,knot]=GenerateSquare(XY,poly)

p1=XY(1,:); p2=XY(2,:); p3=XY(3,:); p4=XY(4,:);
B=zeros(poly+1,poly+1,3);
B(:,:,3)=1;

switch poly
    case 1 % IGA P1
        knot=struct('xi',[0 0 1 1],'eta',[0 0 1 1]);

        B(1,1,1:2)=p1;  B(1,2,1:2)=p3;
        B(2,1,1:2)=p2;  B(2,2,1:2)=p4;

    case 2 % IGA P2
        knot=struct('xi',[0 0 0 1 1 1],'eta',[0 0 0 1 1 1]);

        B(1,1,1:2)=p1;          B(1,2,1:2)=(p1+p3)/2;       B(1,3,1:2)=p3;
        B(2,1,1:2)=(p1+p2)/2;   B(2,2,1:2)=(p1+p2+p3+p4)/4;
            B(2,3,1:2)=(p3+p4)/2;
        B(3,1,1:2)=p2;          B(3,2,1:2)=(p2+p4)/2;       B(3,3,1:2)=p4;

    case 3 % IGA P3
        knot=struct('xi',[0 0 0 0 1 1 1 1],'eta',[0 0 0 0 1 1 1 1]);

        B(1,1,1:2)=p1; B(1,2,1:2)=(2*p1+p3)/3; B(1,3,1:2)=(p1+2*p3)/3;
            B(1,4,1:2)=p3;
        B(2,1,1:2)=(2*p1+p2)/3; B(2,2,1:2)=(4*p1+2*p2+2*p3+p4)/9;
            B(2,3,1:2)=(2*p1+p2+4*p3+2*p4)/9; B(2,4,1:2)=(2*p3+p4)/3;
        B(3,1,1:2)=(p1+2*p2)/3; B(3,2,1:2)=(2*p1+4*p2+p3+2*p4)/9;
            B(3,3,1:2)=(p1+2*p2+2*p3+4*p4)/9; B(3,4,1:2)=(p3+2*p4)/3;
        B(4,1,1:2)=p2; B(4,2,1:2)=(2*p2+p4)/3; B(4,3,1:2)=(p2+2*p4)/3;
            B(4,4,1:2)=p4;

    case 4 % IGA P4
        knot=struct('xi',[0 0 0 0 0 1 1 1 1 1],'eta',[0 0 0 0 0 1 1 1 1 1]);

        B(1,1,1:2)=p1; B(1,2,1:2)=(3*p1+p3)/4; B(1,3,1:2)=(p1+p3)/2;
            B(1,4,1:2)=(p1+3*p3)/4; B(1,5,1:2)=p3;
        B(2,1,1:2)=(3*p1+p2)/4; B(2,2,1:2)=(9*p1+3*p2+3*p3+p4)/16;
            B(2,3,1:2)=(3*p1+p2+3*p3+p4)/8;
            B(2,4,1:2)=(3*p1+p2+9*p3+3*p4)/16; B(2,5,1:2)=(3*p3+p4)/4;
```

```
        B(3,1,1:2)=(p1+p2)/2; B(3,2,1:2)=(3*p1+3*p2+p3+p4)/8;
            B(3,3,1:2)=(p1+p2+p3+p4)/4; B(3,4,1:2)=(p1+p2+3*p3+3*p4)/8;
            B(3,5,1:2)=(p3+p4)/2;
        B(4,1,1:2)=(p1+3*p2)/4; B(4,2,1:2)=(3*p1+9*p2+p3+3*p4)/16;
            B(4,3,1:2)=(p1+3*p2+p3+3*p4)/8;
            B(4,4,1:2)=(p1+3*p2+3*p3+9*p4)/16; B(4,5,1:2)=(p3+3*p4)/4;
        B(5,1,1:2)=p2; B(5,2,1:2)=(3*p2+p4)/4; B(5,3,1:2)=(p2+p4)/2;
            B(5,4,1:2)=(p2+3*p4)/4; B(5,5,1:2)=p4;
end

end
```

B.4.15 Jacobian

This subfunction calculates the Jacobian matrix and physical derivatives on a general basis as described in Section 5.2.2.

Output: `J, dxy`

Input: `dR, P, IEN_e`

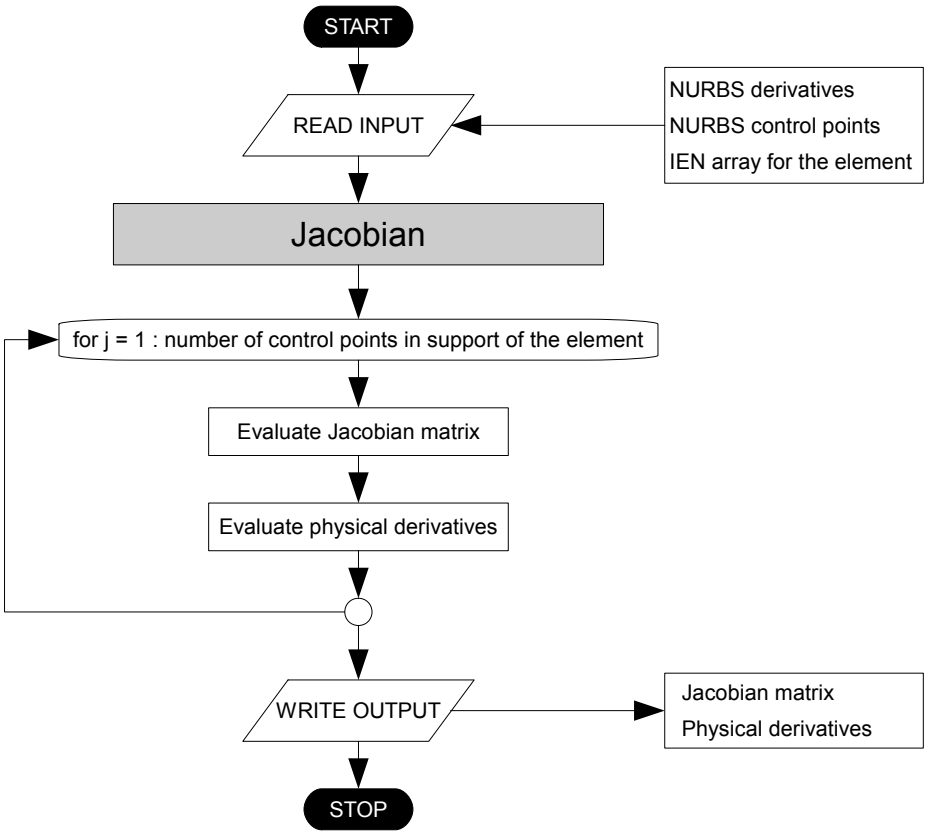


Figure B.21: Flow chart for Jacobian.m

```
% This function calculates the Jacobian matrix and x,y derivatives
%
% Output:   J - Jacobian matrix
%           dxy - x,y derivatives
%
% Input:    dR - natural derivatives of NURBS basis
%           P - coordinates of NURBS control points
%           IEN_e - element topology of the current element

function [J,dxy]=Jacobian(dR,P,IEN_e)

% P(IEN_e(a),1) = x value of control points which support the element,
% P(IEN_e(a),2) = y value
J(1,1)=dR(1,:)*P(IEN_e,1);
J(1,2)=dR(1,:)*P(IEN_e,2);
J(2,1)=dR(2,:)*P(IEN_e,1);
J(2,2)=dR(2,:)*P(IEN_e,2);

dxy=J\dR;

end
```

B.4.16 Knot Insertion

This subfunction generates new control points when a knot is inserted for a two-dimensional (parameter space) problem. Eqs. (3.12) and (3.13) from Section 3.2.7 are applied.

Output: `B, knot`

Input: `B_old, knot_old, knot_ins, poly, dir`

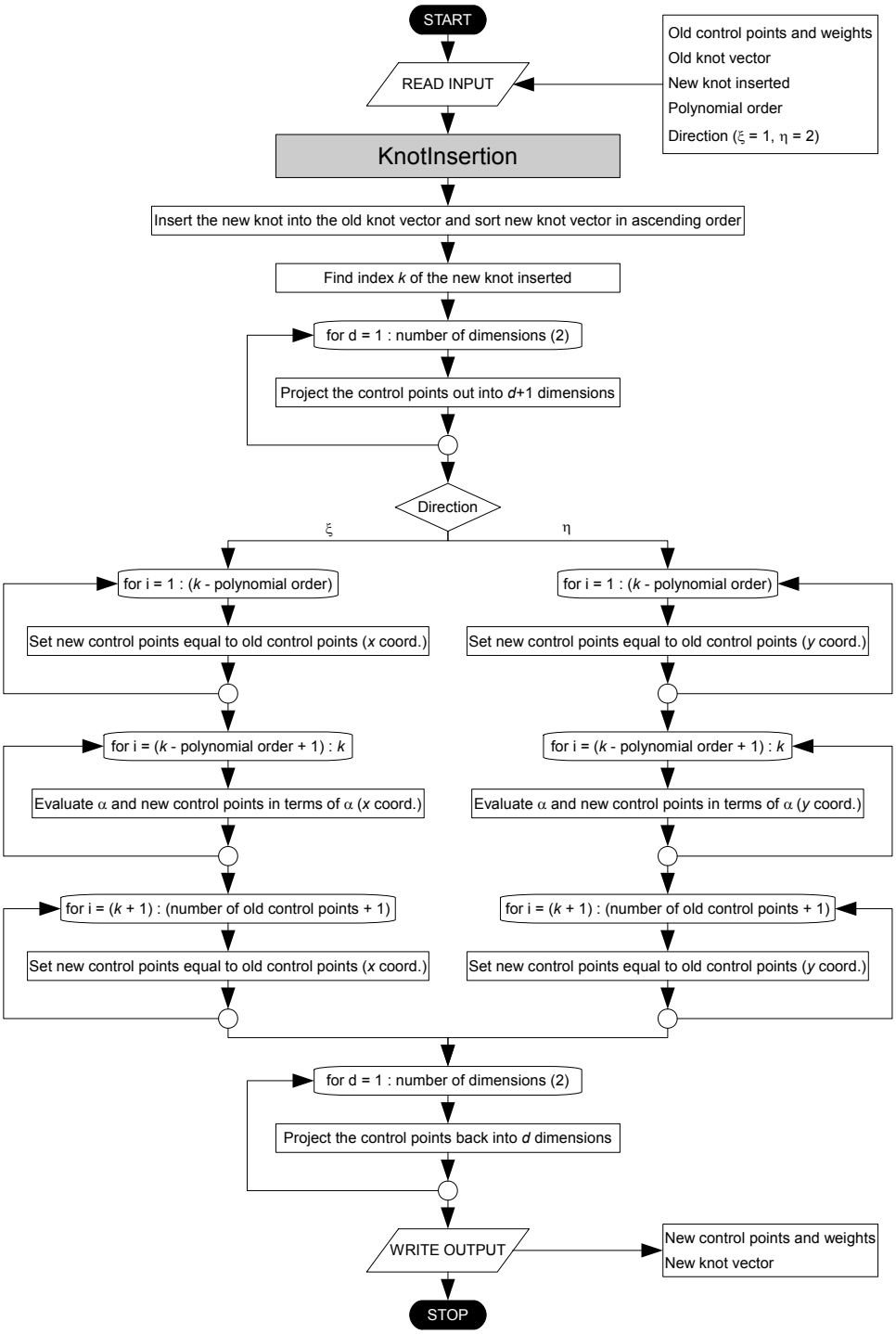


Figure B.22: Flow chart for KnotInsertion.m

```
% This function generates new control points when a knot is inserted for a
% NURBS surface
%
% Output:   B - control points and weights given as a three-dim. matrix
%               B(:,:,1) - x coordinates
%               B(:,:,2) - y coordinates
%               B(:,:,3) - weights
%           knot - knot vector given as a Matlab structure
%               knot.xi - knot vector in xi direction
%               knot.eta - knot vector in eta direction
%
% Input:    B_old - old B matrix
%           knot_old - old knot vector
%           knot_ins - new knot inserted
%           poly - polynomial order
%           dir - direction 1 (xi) or 2 (eta)

function [B,knot]=KnotInsertion(B_old,knot_old,knot_ins,poly,dir)

knot=[knot_old,knot_ins];
knot=sort(knot);
k=find(knot_ins<knot_old,1)-1;      % Index at which knot_ins was inserted
n=size(B_old);

% Project the control points out into d+1 dimensions
for i=1:2
    B_old(:,:,i)=B_old(:,:,i).*B_old(:,:,3);
end

% Evaluate new control points
if dir==1 % (xi direction)
    B=zeros(n(1)+1,n(2),3);
    B(1:k-poly,:,:)=B_old(1:k-poly,:,:);
    for i=k-poly+1:k
        alpha=(knot_ins-knot_old(i))/(knot_old(i+poly)-knot_old(i));
        B(i,:,:)=alpha*B_old(i,:,:)+(1-alpha)*B_old(i-1,:,:);
    end
    B(k+1:end,:)=B_old(k:end,:);
else % dir==2 (eta direction)
    B=zeros(n(1),n(2)+1,3);
    B(:,1:k-poly,:)=B_old(:,1:k-poly,:);
    for i=k-poly+1:k
        alpha=(knot_ins-knot_old(i))/(knot_old(i+poly)-knot_old(i));
        B(:,i,:)=alpha*B_old(:,i,:)+(1-alpha)*B_old(:,i-1,:);
    end
    B(:,k+1:end,:)=B_old(:,k:end,:);
end

% Project back into d dimensions
for i=1:2
    B(:,:,i)=B(:,:,i)./B(:,:,3);
end

end
```

B.4.17 Mesh

This subfunction creates mesh or element topology as described in Section 5.1.1 for an arbitrary degree tensor product mesh.

Output: `IEN`

Input: `nx`, `ny`, `nel`, `n`, `poly`

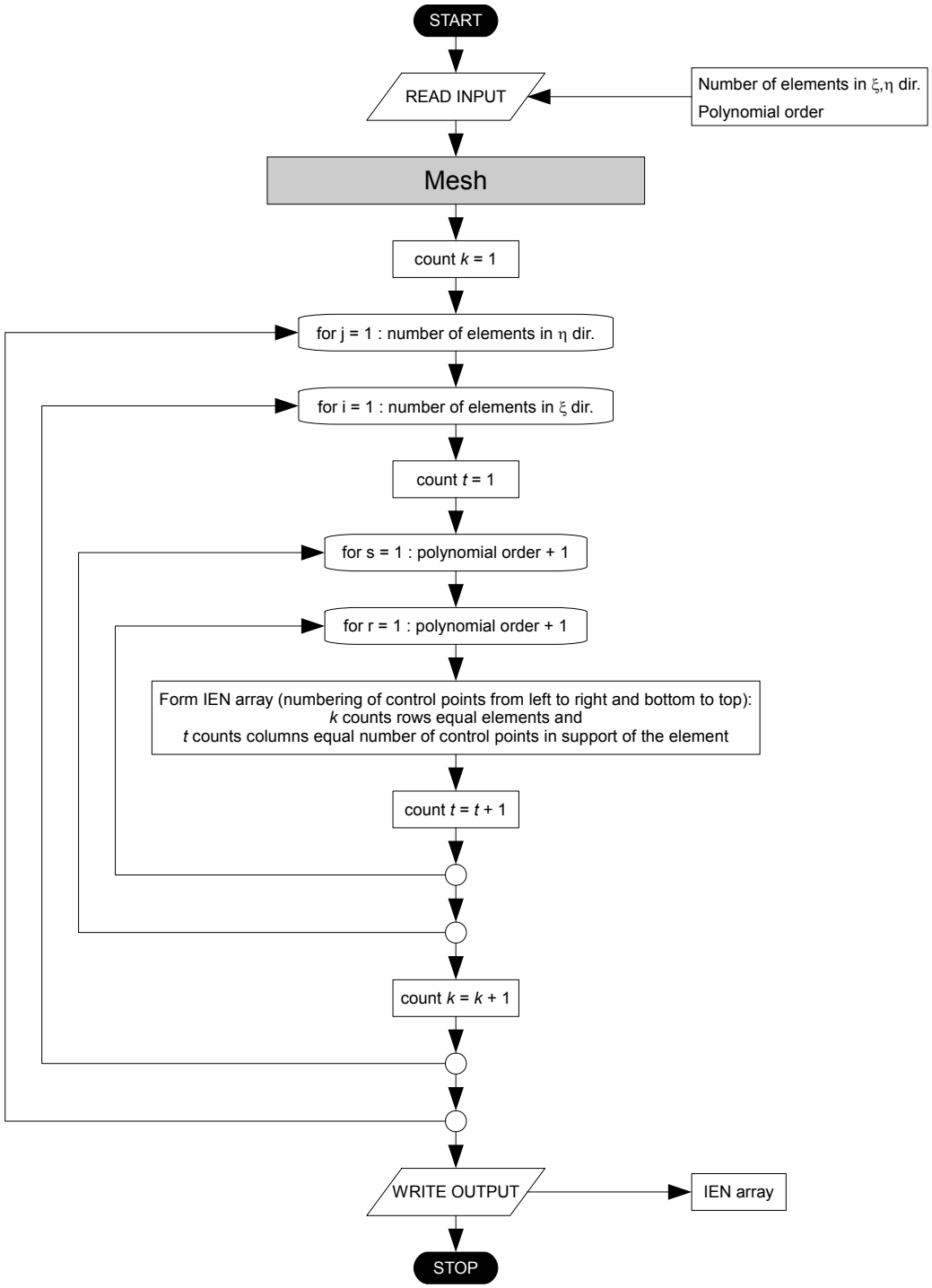


Figure B.23: Flow chart for Mesh.m

```
% This function creates mesh (element topology) for the IGA program
%
% Output:   IEN - element topology: numbering of control points from left
%               to right and bottom to top, 1 row for each element
%
% Input:    nx - number of elements, xi direction
%           ny - number of elements, eta direction
%           poly - polynomial order

function IEN=Mesh(nx,ny,poly)

IEN=zeros(nx*ny,(poly+1)^2);

k=1;
for j=1:ny
    for i=1:nx
        t=1;
        for s=1:poly+1
            for r=1:poly+1
                IEN(k,t)=(nx+poly)*(j+s-2)+i+r-1;
                t=t+1;
            end
        end
        k=k+1;
    end
end

end
```

B.4.18 Nodes FEA

This subfunction contains parametric coordinates which equals nodes of FEA for elements of the same polynomial order. The subfunction is applied in the function PlotDisplacements.m, see Appendix B.4.22.

Output: `Nodes`
Input: `poly`

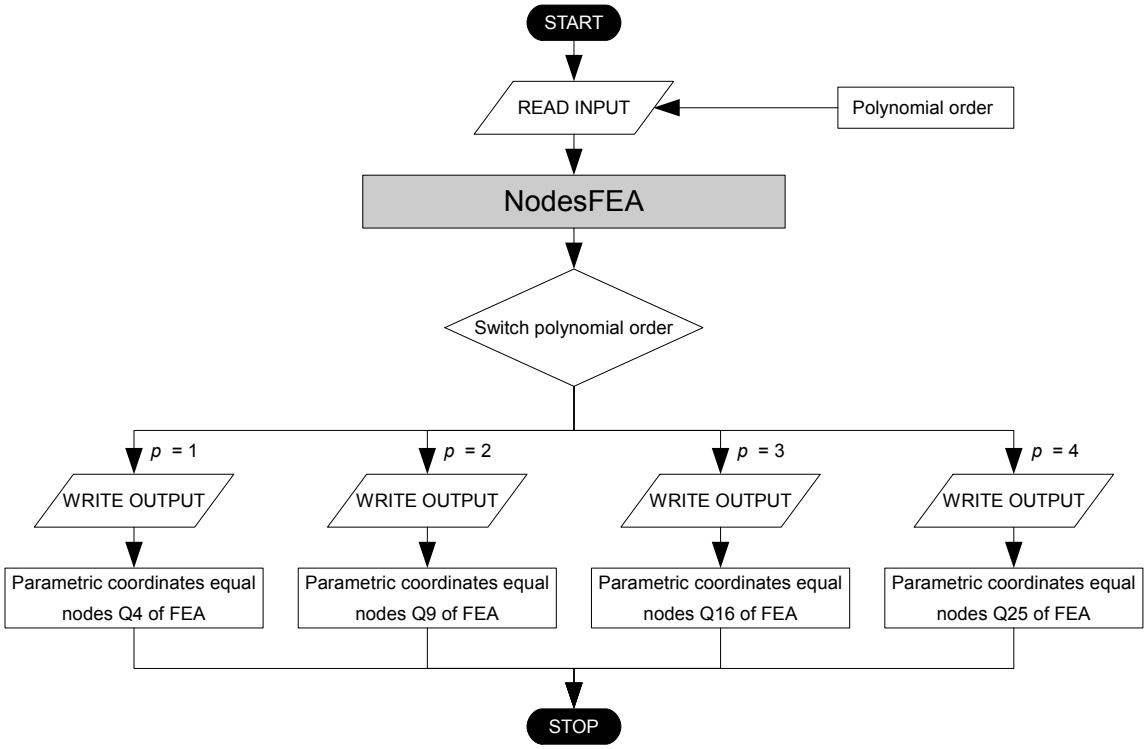


Figure B.24: Flow chart for NodesFEA.m

```
% This function contains parametric coordinates which equals nodes of FEA
% for elements of the same polynomial order
%
% Output:   Nodes - "Node points" from left to right and bottom to top,
%               1st column = xi value, 2nd column = eta value
%
% Input:    poly - polynomial order

function Nodes=NodesFEA(poly)

switch poly
    case 1 % 2x2, for IGA P1
        Nodes=[-1 -1;1 -1;-1 1;1 1];
    case 2 % 3x3, for IGA P2
        Nodes=[-1 -1;0 -1;1 -1;-1 0;0 0;1 0;-1 1;0 1;1 1];
    case 3 % 4x4, for IGA P3
        Nodes=[-1 -1;-1/3 -1;1/3 -1;1 -1;
            -1 -1/3;-1/3 -1/3;1/3 -1/3;1 -1/3;
            -1 1/3;-1/3 1/3;1/3 1/3;1 1/3;
            -1 1;-1/3 1;1/3 1;1 1];
    case 4 % 5x5, for IGA P4
        Nodes=[-1 -1;-1/2 -1;0 -1;1/2 -1;1 -1;
            -1 -1/2;-1/2 -1/2;0 -1/2;1/2 -1/2;1 -1/2;
            -1 0;-1/2 0;0 0;1/2 0;1 0;
            -1 1/2;-1/2 1/2;0 1/2;1/2 1/2;1 1/2;
            -1 1;-1/2 1;0 1;1/2 1;1 1];
end

end
```

B.4.19 NURBS Basis

This subfunction forms NURBS basis functions and derivatives in two dimensions by applying Eqs. (4.22) and (4.24) from Section 4.1.3. The basis is therefore formed using the Bézier extraction operator and localized to the element level. Note that this is equivalent to Eqs. (3.19) and (3.20) from Section 3.3.2. Since the bivariate Bernstein basis functions are applied, the output NURBS basis functions are also bivariate.

Output:	`Rb`, `dR`
Input:	`xi`, `eta`, `poly`, `e`, `IEN_e`, `w`, `C`
Subfunctions:	BernsteinBasis.m, BernsteinBasisBivariate.m

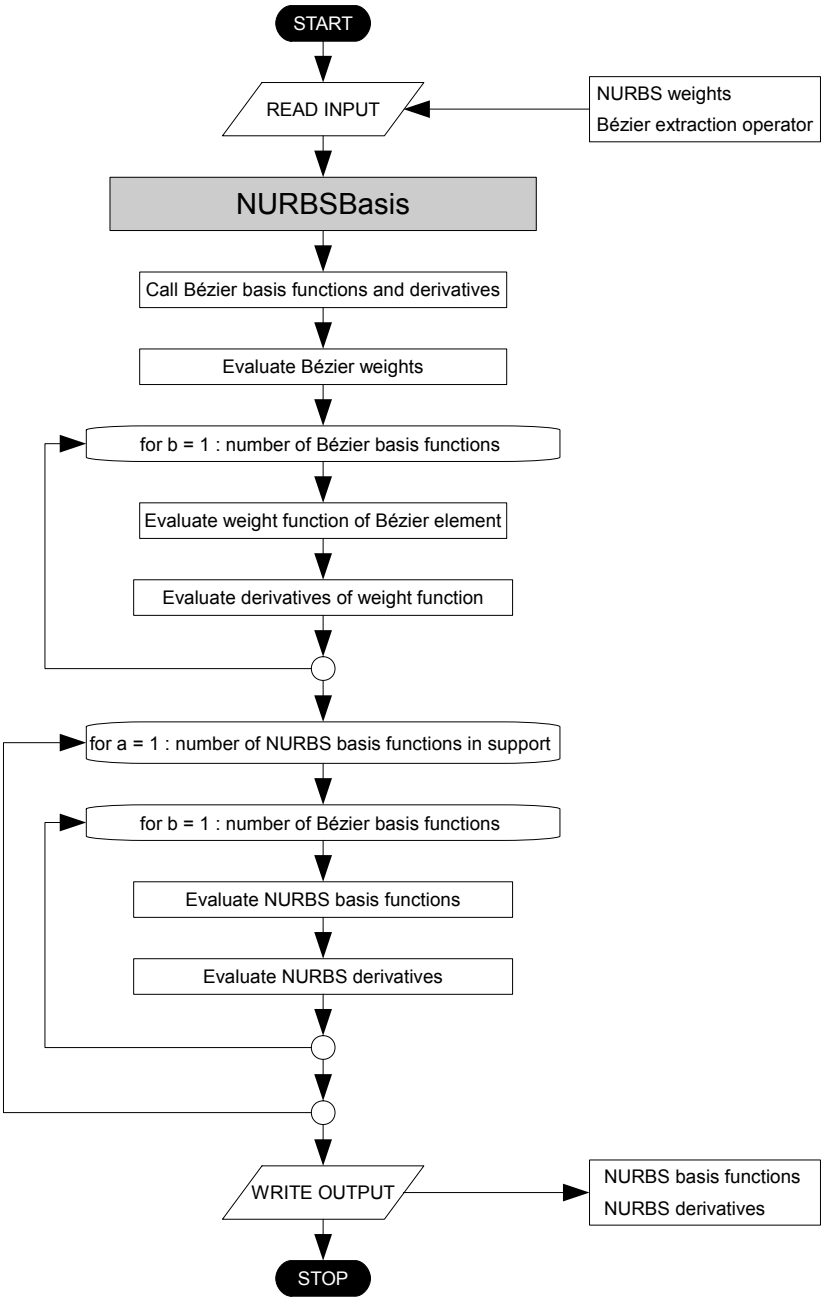


Figure B.25: Flow chart for NURBSBasis.m

```
% This function forms basis functions and derivatives
%
% Output:   Rb - NURBS basis functions
%           dR - derivatives of NURBS basis functions
%
% Input:    xi - coord. of 1st parametric direction
%           eta - coord. of 2nd parametric direction
%           poly - polynomial order
%           e - current element number
%           IEN_e - element topology of current element
%           w - NURBS weights
%           C - Bézier extraction operators

function [Rb,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C)

% Bernstein basis functions and derivatives
[Bbxi,dBxi]=BernsteinBasis(xi,poly);
[Bbeta,dBeta]=BernsteinBasis(eta,poly);
[Bb,dB]=BernsteinBasisBivariate(Bbxi,Bbeta,dBxi,dBeta,poly);

% Weight function and derivatives of weight function, Bézier element
wb=C(1:length(IEN_e),:,e)'*w(IEN_e);    % Bézier weights
Wb=Bb*wb;
dWbxi=dB(1,:)*wb;
dWbeta=dB(2,:)*wb;

% NURBS basis functions and derivatives
Rb=diag(w(IEN_e))*C(1:length(IEN_e),:,e)*Bb'/Wb;
Rb=Rb';
dR(:,1)=diag(w(IEN_e))*C(1:length(IEN_e),:,e)*(dB(1,:)'/Wb-dWbxi*Bb'/Wb^2);
dR(:,2)=diag(w(IEN_e))*C(1:length(IEN_e),:,e)*(dB(2,:)'/Wb-dWbeta*Bb'/Wb^2);
dR=dR';

end
```

B.4.20 Parse Rhino Data

This subfunction is a parsing script for reading and editing the geometrical data from T-Splines for Rhino. The input data should be as described in Appendix B.1.2. The number of T-spline elements from Rhino must be at least 2 for this script to work.

Output: `P`, `w`, `C`, `IEN`

Input: `P_Rhino`, `C_Rhino`

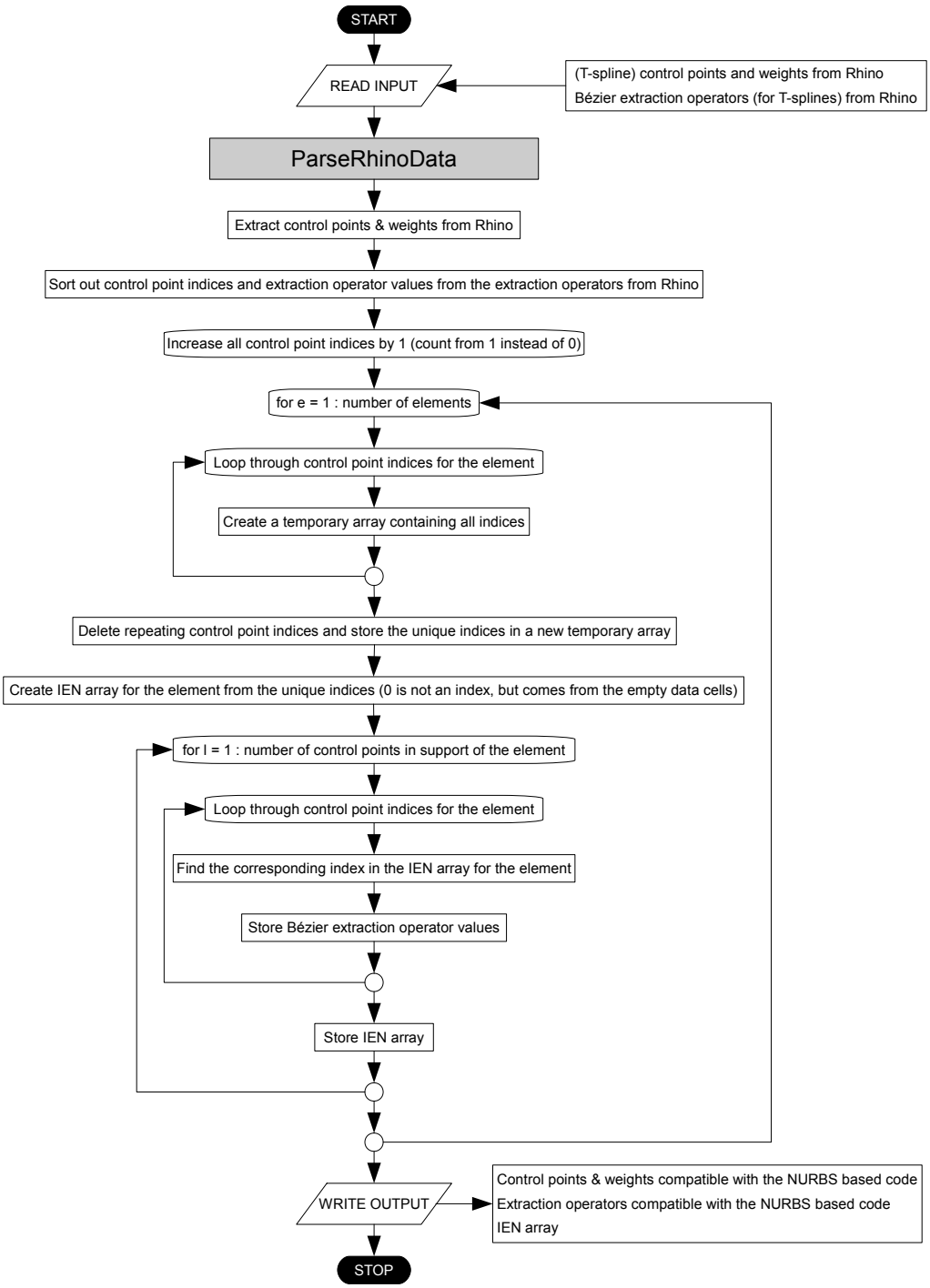


Figure B.26: Flow chart for ParseRhinoData.m

```
% Parsing script to read and edit the data from Rhinoceros
%
% Output:   P - physical coordinates of T-splines control points
%           w - corresponding weights
%           C - Bézier extraction operators for each element C(:,:,e)
%           IEN - element topology
%
% Input:    P_QD - T-spline control points from Rhino (non-rational)
%               P_Rhino(:,1) - x coordinates
%               P_Rhino(:,2) - y coordinates
%               P_Rhino(:,3) - z coordiantes = 0 (planar surface)
%               P_Rhino(:,4) - weights
%           C_Rhino - Bézier extraction operators from Rhino
%               C_Rhino(:,1:2:size(C_Rhino,2)-1) - linear operators
%               C_Rhino(:,2:2:size(C_Rhino,2)) - control point indices

function [P,w,C,IEN]=ParseRhinoData(P_Rhino,C_Rhino)

% Physical coordinates of control points and corresponding weights
w=P_Rhino(:,4);
P(:,1)=P_Rhino(:,1)./w;
P(:,2)=P_Rhino(:,2)./w;

% Sort out control point indices and Bézier extraction operators
indC=C_Rhino(:,2:2:size(C_Rhino,2));
numC=C_Rhino(:,1:2:size(C_Rhino,2)-1);

% Count control points from 1 instead of 0
for i=1:size(indC,1)
    indC(i,1)=indC(i,1)+1;
    for j=2:size(indC,2)
        if indC(i,j)>0
            indC(i,j)=indC(i,j)+1;
        end
    end
end

% Extract which control points that support each element (element topology)
% and build the extraction operator C for that element
temp=zeros(1,1);
C=zeros(1,1,1);
IEN=zeros(1,1);
for e=1:size(indC,1)/16
    % Create an array of control point indices for the element
    k=0;
    for i=(e-1)*16+1:e*16
        for j=1:size(indC,2)
            k=k+1;
            temp(e,k)=indC(i,j);
        end
    end
    % Find the unique control points for the element
    temp2=unique(temp(e,:));
    % Element topology
    IEN_e=temp2(2:end);
    for l=1:length(IEN_e)
```

```
        m=0;
        for i=(e-1)*16+1:e*16
            m=m+1;
            for j=1:size(indC,2)
                % Index of control point for current element
                n=IEN_e==indC(i,j);
                % Bezier extraction operator
                C(n,m,e)=numC(i,j);
            end
        end
        % Store element topology
        IEN(e,l)=IEN_e(l);
    end
end
```

B.4.21 Plot B-Splines NURBS

This function is a stand-alone function for plotting purposes. The evaluation of the basis functions and derivatives is based on Eqs. (3.2), (3.3) and (3.4) from Section 3.2.2 and Eqs. (3.19) and (3.20) from Section 3.3.2. Eqs. (3.8) and (3.21) are used to evaluate the B-spline and NURBS curve, respectively.

All basis functions and curves in this thesis are drawn using this function or minor modifications to this function.

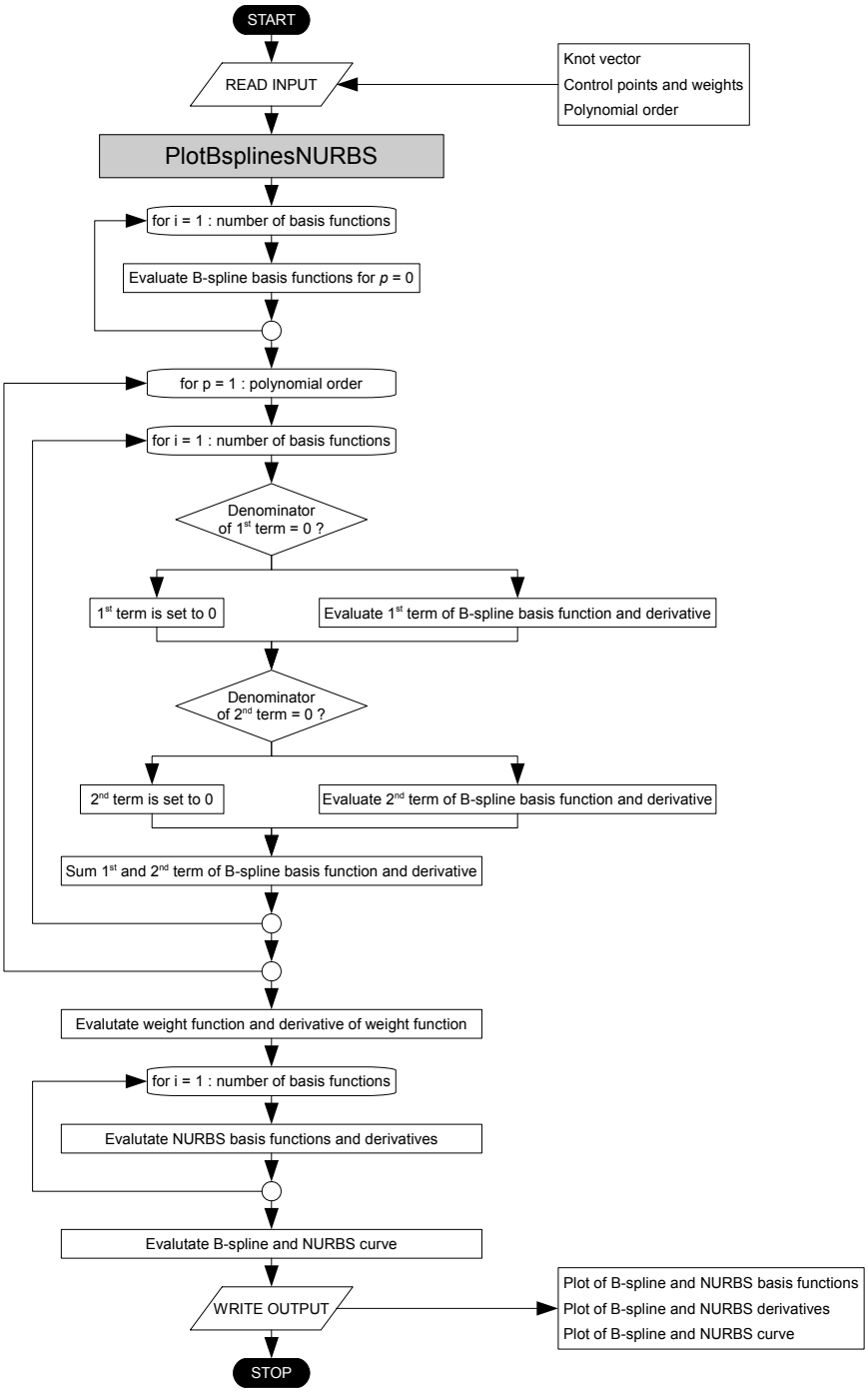


Figure B.27: Flow chart for PlotBsplinesNURBS.m

```
%-----------------------PLOT B-SPLINES AND NURBS-------------------------%

% Plot B-splines and NURBS basis functions and derivatives from non-uniform
% knot vector. Plot B-splines and NURBS curve from basis functions.

close all; clear all

% KNOT VECTOR
XI=[0 0 0 1 2 2 2];
xi=(0:0.01:XI(end));                % Assume that knot vector starts at 0

% CONTROL POINTS AND WEIGHTS
P=[-1 0;-1 1;1 1;1 0]; w=[1 1/2 1/2 1]';

% POLYNOMIAL ORDER
poly=length(find(XI(1,:)==0))-1;    % Assume that knot vector starts at 0

% FOR STRUCTURE
N=zeros(1,length(xi)); dN=zeros(1,length(xi));
W=zeros(1,length(xi)); dW=zeros(1,length(xi));
Rb=zeros(1,length(xi)); dR=zeros(1,length(xi));

% p=0
p=0;
n=length(XI)-p-1;                   % Number of functions Ni
for i=1:n
    for j=1:length(xi)
        if xi(j)>=XI(i) && xi(j)<XI(i+1)
            N(i,j)=1;
        else
            N(i,j)=0;
        end
    end
end

% p=1,2,3,...
for p=1:poly
n=length(XI)-p-1;                   % Number of functions Ni
    for i=1:n
        for j=1:length(xi)
            if (XI(i+p)-XI(i))==0
                A=0;
                a=0;
            else
                A=(xi(j)-XI(i))*N(i,j)/(XI(i+p)-XI(i));
                a=poly*N(i,j)/(XI(i+p)-XI(i));
            end
            if (XI(i+p+1)-XI(i+1))==0
                B=0;
                b=0;
            else
                B=(XI(i+p+1)-xi(j))*N(i+1,j)/(XI(i+p+1)-XI(i+1));
                b=poly*N(i+1,j)/(XI(i+p+1)-XI(i+1));
            end
            N(i,j)=A+B;             % B-spline basis functions
            dN(i,j)=a-b;            % B-spline derivatives
```

```
        end
    end
end
N(n,length(xi))=1;                  % For plotting: End value of Nn is set
                                    % to 1 (not 0)
for j=1:length(xi)
    W(j)=N(1:n,j)'*w(1:n);          % Weight function
    dW(j)=dN(1:n,j)'*w(1:n);        % Derivative of weight function
end

for i=1:n
    for j=1:length(xi)
        Rb(i,j)=N(i,j)*w(i)/W(j);   % NURBS basis functions
        dR(i,j)=w(i)*(dN(i,j)/W(j)-dW(j)*N(i,j)/W(j)^2);    % NURBS deriv.
    end
end

% EVALUATE B-SPLINE AND NURBS CURVE
xb=zeros(1,length(xi)); yb=zeros(1,length(xi));
x=zeros(1,length(xi)); y=zeros(1,length(xi));
for j=1:length(xi)
    xb(j)=N(1:n,j)'*P(:,1);
    yb(j)=N(1:n,j)'*P(:,2);
    x(j)=Rb(1:n,j)'*P(:,1);
    y(j)=Rb(1:n,j)'*P(:,2);
end

% PLOT B-SPLINE AND NURBS BASIS FUNCTIONS AND DERIVATIVES
figure
subplot(2,2,1)
plot(xi,N)
grid on
title('B-spline basis functions')
xlabel('\xi')
subplot(2,2,2)
plot(xi,dN)
grid on
xlabel('\xi')
subplot(2,2,3)
plot(xi,Rb)
grid on
title('NURBS basis functions')
xlabel('\xi')
subplot(2,2,4)
plot(xi,dR)
grid on
xlabel('\xi')

% PLOT B-SPLINE AND NUBRS CURVE
figure
hold on
plot(xb,yb,x,y)
plot(P(:,1),P(:,2),'sk')
plot([-1 -1],[0 1],'--k',[-1 1],[1 1],'--k',[1 1],[0 1],'--k')
legend('B-spline curve','NURBS curve')
w1=ones(1,length(xi));
```

```
axis('equal')
grid on
xlim([-1.1 1.1]); ylim([-0.1 1.1]);
xlabel('x'); ylabel('y');
```

B.4.22 Plot Displacements

This subfunction interpolates displacements according to Eq. (5.7) from Section 5.1.3 and plots contour plots of the displacements u and v as described in Section 5.2.3. Application of this subfunction requires a tensor product geometry.

Input:	`IEN`, `P`, `D`, `n`, `m`, `nel`, `ncp`, `poly`, `w`, `C`
Subfunctions:	NodesFEA.m, NURBSBasis.m

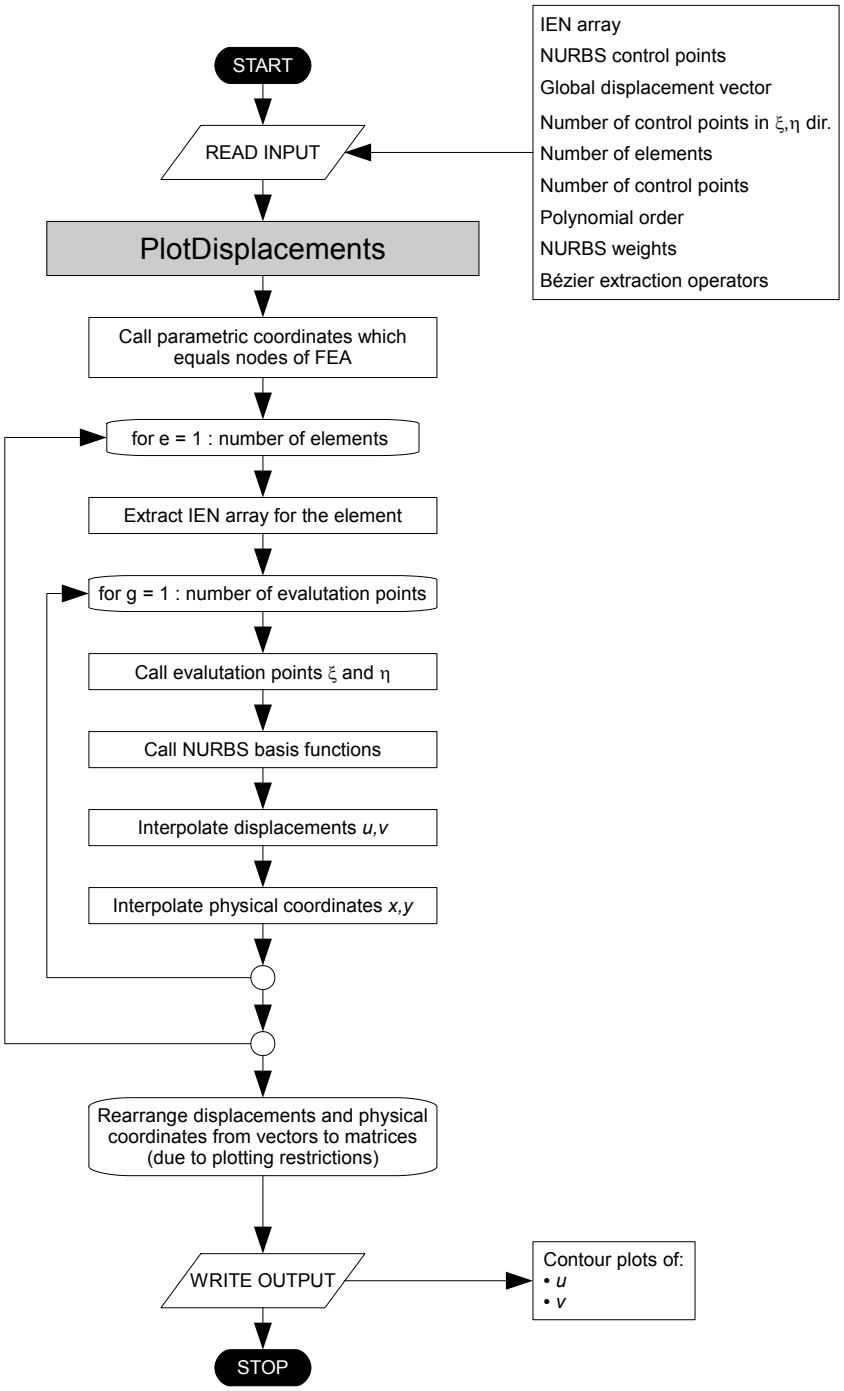


Figure B.28: Flow chart for PlotDisplacements.m

```
% This function interpolates the displacement field and plots contour plots
% of the displacements
%
% Output:   contour plot of displacement u
%           contour plot of displacement v
%
% Inut:     IEN - element topology: numbering of control points
%           P - coordinates of control points
%           D - global displacement vector
%           n - number of basis functions/control points, xi direction
%           m - number of basis functions/control points, eta direction
%           nel - total number of elements
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function PlotDisplacements(IEN,P,D,n,m,nel,ncp,poly,w,C)

Nodes=NodesFEA(poly);   % Call parametric coorc. which equals nodes of FEA

u_temp=zeros(ncp,1); v_temp=zeros(ncp,1);
X_temp=zeros(ncp,1); Y_temp=zeros(ncp,1);
for e=1:nel
    IEN_e=IEN(e,:);

    for g=1:size(Nodes,1);
        xi=Nodes(g,1);
        eta=Nodes(g,2);

        [Rb,~]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);

        u_temp(IEN_e(g))=Rb*D(IEN_e);
        v_temp(IEN_e(g))=Rb*D(IEN_e+ncp);

        X_temp(IEN_e(g))=Rb*P(IEN_e,1);
        Y_temp(IEN_e(g))=Rb*P(IEN_e,2);
    end
end

u=zeros(n,m); v=zeros(n,m);
X=zeros(n,m); Y=zeros(n,m);
k=1;
for j=1:m
    for i=1:n
        u(i,j)=u_temp(k);
        v(i,j)=v_temp(k);
        X(i,j)=X_temp(k);
        Y(i,j)=Y_temp(k);
        k=k+1;
    end
end

figure
contourf(X,Y,u,500,'LineStyle','none');
colorbar
```

```
axis('equal')
title(['Displacement u (',num2str(ncp*2),' DOF)'])
xlabel('x'); ylabel('y')

figure
contourf(X,Y,v,500,'LineStyle','none');
colorbar
axis('equal')
title(['Displacement v (',num2str(ncp*2),' DOF)'])
xlabel('x'); ylabel('y')

end
```

B.4.23 Plot NURBS Bézier QuarterDisk

This subfunction plots NURBS and Bézier control mesh and Bézier elements for the geometry of a quarter disk. Application of this subfunction requires a tensor product geometry.

Input: `IEN`, `B`, `P`, `ny`, `nel`, `poly`, `r__inn`, `r__out`, `w`, `C`
Subfunctions: PlotNURBSQuarterCircle.m

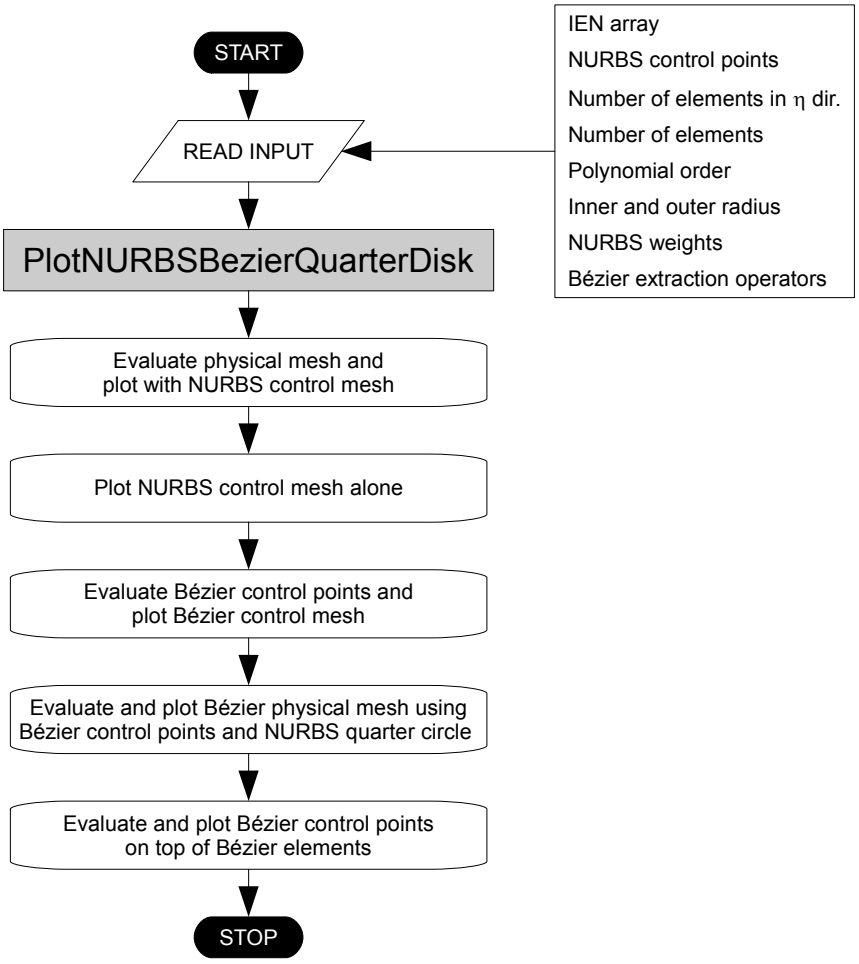


Figure B.29: Flow chart for PlotNURBSBezierQuarterDisk.m

```
% This function plots NURBS and Bézier control mesh and Bézier elements
% for the geometry of a (quarter of a) circular beam
%
% Output:   plot of physical mesh with NURBS control mesh
%           plot of NURBS control mesh
%           plot of Bézier control mesh
%           plot of Bézier physical mesh
%           plot of Bézier control points on top of Bézier elements
%
% Input:    IEN - element topology: numbering of control points
%           B - NURBS control points given as a 3x3x3 matrix
%               B(:,:,1) - x coordinates
%               B(:,:,2) - y coordinates
%           P - NURBS control points given as a (number of control
%           points)x2 matrix
%               1st column - x coordinates
%               2nd column - y coordinates
%           ny - number of elements, eta direction
%           nel - total number of elements
%           poly - polynomial order
%           r_inn - inner radius
%           r_out - outer radius
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function PlotNURBSBezierQuarterDisk(IEN,B,P,ny,nel,poly,r_inn,r_out,w,C)

figure % Physical mesh with NURBS control mesh
hold on
plot(B(:,:,1),B(:,:,2),'--sk')
plot(B(:,:,1)',B(:,:,2)','--sk')
[x,y]=PlotNURBSQuarterCircle(r_inn);
plot(x,y,'k')
[x,y]=PlotNURBSQuarterCircle(r_out);
plot(x,y,'k')
X(1,1)=0; X(1,2)=r_inn; X(2,1)=0; X(2,2)=r_out;
Y(1,1)=r_inn; Y(1,2)=0; Y(2,1)=r_out; Y(2,2)=0;
plot(X,Y,'k')
axis('equal')
xlim([-0.01 r_out+0.01]); ylim([-0.01 r_out+0.01])
xlabel('x'); ylabel('y')
title('Physical mesh with NURBS control mesh')
hold off

figure % NURBS control mesh
hold on
plot(B(:,:,1),B(:,:,2),'-sk')
plot(B(:,:,1)',B(:,:,2)','-sk')
axis('equal')
xlim([-0.01 r_out+0.01]); ylim([-0.01 r_out+0.01])
xlabel('x'); ylabel('y')
title('NURBS control mesh')
hold off

figure % Bézier control mesh
hold on
```

```
for e=1:nel
    IEN_e=IEN(e,:);
    Wb=diag(C(:,:,e)'*w(IEN_e));
    Pb=Wb\C(:,:,e)'*diag(w(IEN_e))*P(IEN_e,:);
    Bb=zeros(poly+1,poly+1,2);
    k=1;
    for j=1:poly+1
        for i=1:poly+1
            for d=1:2
                Bb(i,j,d)=Pb(k,d);
            end
            k=k+1;
        end
    end
    plot(Bb(:,:,1),Bb(:,:,2),'sk')
    plot(Bb(:,:,1)',Bb(:,:,2)','sk')
    for k=1:poly:size(Bb,1)
        plot(Bb(k,:,1)',Bb(k,:,2)','-sk')
    end
    for k=1:poly:size(Bb,2)
        plot(Bb(:,k,1),Bb(:,k,2),'-sk')
    end
end
axis('equal')
xlim([-0.01 r_out+0.01]); ylim([-0.01 r_out+0.01])
xlabel('x'); ylabel('y')
title('Bézier control mesh')
hold off

figure % Bézier physical mesh
hold on
for j=1:ny+1
    r=r_inn+(r_out-r_inn)/ny*(j-1);
    [x,y]=PlotNURBSQuarterCircle(r);
    plot(x,y,'k')
end
for e=1:nel
    IEN_e=IEN(e,:);
    Wb=diag(C(:,:,e)'*w(IEN_e));
    Pb=Wb\C(:,:,e)'*diag(w(IEN_e))*P(IEN_e,:);
    Bb=zeros(poly+1,poly+1,2);
    k=1;
    for j=1:poly+1
        for i=1:poly+1
            for d=1:2
                Bb(i,j,d)=Pb(k,d);
            end
            k=k+1;
        end
    end
    for k=1:poly:size(Bb,1)
    plot(Bb(k,:,1)',Bb(k,:,2)','-k')
    end
end
axis('equal')
xlim([-0.01 r_out+0.01]); ylim([-0.01 r_out+0.01])
```

```
xlabel('x'); ylabel('y')
title('Bézier physical mesh')
hold off

figure % Bézier control points on top of Bézier elements
hold on
for j=1:ny+1
    r=r_inn+(r_out-r_inn)/ny*(j-1);
    [x,y]=PlotNURBSQuarterCircle(r);
    plot(x,y,'k')
end
for e=1:nel
    IEN_e=IEN(e,:);
    Wb=diag(C(:,:,e)'*w(IEN_e));
    Pb=Wb\C(:,:,e)'*diag(w(IEN_e))*P(IEN_e,:);
    Bb=zeros(poly+1,poly+1,2);
    k=1;
    for j=1:poly+1
        for i=1:poly+1
            for d=1:2
                Bb(i,j,d)=Pb(k,d);
            end
            k=k+1;
        end
    end
    plot(Bb(:,:,1),Bb(:,:,2),'.k')
    plot(Bb(:,:,1)',Bb(:,:,2)','.k')
    for k=1:poly:size(Bb,1)
        plot(Bb(k,:,1)',Bb(k,:,2)','.k')
        plot(Bb(k,:,1)',Bb(k,:,2)','-k')
    end
    for k=1:poly:size(Bb,2)
        plot(Bb(:,k,1),Bb(:,k,2),'.k')
    end
end
axis('equal')
xlim([-0.01 r_out+0.01]); ylim([-0.01 r_out+0.01])
xlabel('x'); ylabel('y')
title('Bézier control points on top of Bézier elements')
hold off

end
```

B.4.24 Plot NURBS Bézier Square

This subfunction plots NURBS and Bézier control mesh and Bézier elements for the geometry of a square. Application of this subfunction requires a tensor product geometry.

Input: `IEN`, `B`, `P`, `nel`, `poly`, `XY`, `w`, `C`

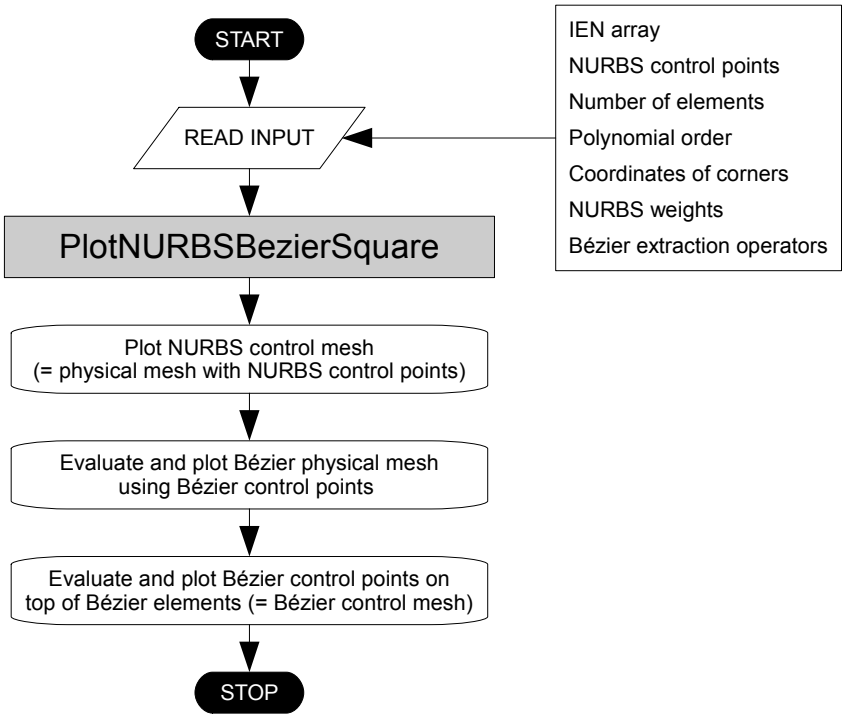


Figure B.30: Flow chart for PlotNURBSBezierSquare.m

```
% This function plots NURBS and Bézier control mesh and Bézier elements
% for the geometry of a square
%
% Output:   plot of NURBS control mesh
%               (=Physical mesh with NURBS control mesh)
%           plot of Bézier physical mesh
%           plot of Bézier control points on top of Bézier elements
%               (=Bézier control mesh)
%
% Input:    IEN - element topology: numbering of control points
%           B - NURBS control points given as a 3x3x3 matrix
%               B(:,:,1) - x coordinates
%               B(:,:,2) - y coordinates
%           P - NURBS control points given as a (number of control
%           points)x2 matrix
%               1st column - x coordinates
%               2nd column - y coordinates
%           nel - total number of elements
%           poly - polynomial order
%           XY - coordinates of corners [X1 Y1;...;X4 Y4]
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function PlotNURBSBezierSquare(IEN,B,P,nel,poly,XY,w,C)

figure % NURBS control mesh
hold on
plot(B(:,:,1),B(:,:,2),'-sk')
plot(B(:,:,1)',B(:,:,2)','-sk')
axis('equal')
xlim([-0.01+XY(1,1) XY(2,1)+0.01]); ylim([-0.01+XY(1,2) XY(4,2)+0.01])
xlabel('x'); ylabel('y')
title('NURBS control mesh')
hold off

figure % Bézier physical mesh
hold on
for e=1:nel
    IEN_e=IEN(e,:);
    Wb=diag(C(:,:,e)'*w(IEN_e));
    Pb=Wb\C(:,:,e)'*diag(w(IEN_e))*P(IEN_e,:);
    Bb=zeros(poly+1,poly+1,2);
    k=1;
    for j=1:poly+1
        for i=1:poly+1
            for d=1:2
                Bb(i,j,d)=Pb(k,d);
            end
            k=k+1;
        end
    end
    for k=1:poly:size(Bb,1)
    plot(Bb(k,:,1)',Bb(k,:,2)','-k')
    end
    for k=1:poly:size(Bb,2)
        plot(Bb(:,k,1),Bb(:,k,2),'-k')
```

```
    end
end
axis('equal')
xlim([-0.01+XY(1,1) XY(2,1)+0.01]); ylim([-0.01+XY(1,2) XY(4,2)+0.01])
xlabel('x'); ylabel('y')
title('Bézier physical mesh')
hold off

figure % Bézier control points on top of Bézier elements
hold on
for e=1:nel
    IEN_e=IEN(e,:);
    Wb=diag(C(:,:,e)'*w(IEN_e));
    Pb=Wb\C(:,:,e)'*diag(w(IEN_e))*P(IEN_e,:);
    Bb=zeros(poly+1,poly+1,2);
    k=1;
    for j=1:poly+1
        for i=1:poly+1
            for d=1:2
                Bb(i,j,d)=Pb(k,d);
            end
            k=k+1;
        end
    end
    plot(Bb(:,:,1),Bb(:,:,2),'.k')
    plot(Bb(:,:,1)',Bb(:,:,2)','.k')
    for k=1:poly:size(Bb,1)
        plot(Bb(k,:,1)',Bb(k,:,2)','.k')
        plot(Bb(k,:,1)',Bb(k,:,2)','-k')
    end
    for k=1:poly:size(Bb,2)
        plot(Bb(:,k,1),Bb(:,k,2),'.k')
        plot(Bb(:,k,1),Bb(:,k,2),'-k')
    end
end
axis('equal')
xlim([-0.01+XY(1,1) XY(2,1)+0.01]); ylim([-0.01+XY(1,2) XY(4,2)+0.01])
xlabel('x'); ylabel('y')
title('Bézier control points on top of Bézier elements')
hold off

end
```

B.4.25 Plot NURBS Quarter Circle

This subfunction evaluates the NURBS curve (degree 2) of a quarter circle for the purpose of plotting. The function employs the same equations and structure as PlotBsplinesNURBS.m (Appendix B.4.21) and is applied in the function PlotNURBSBezierQuarterDisk.m, see Appendix B.4.23.

Output: `x, y`

Input: `r`

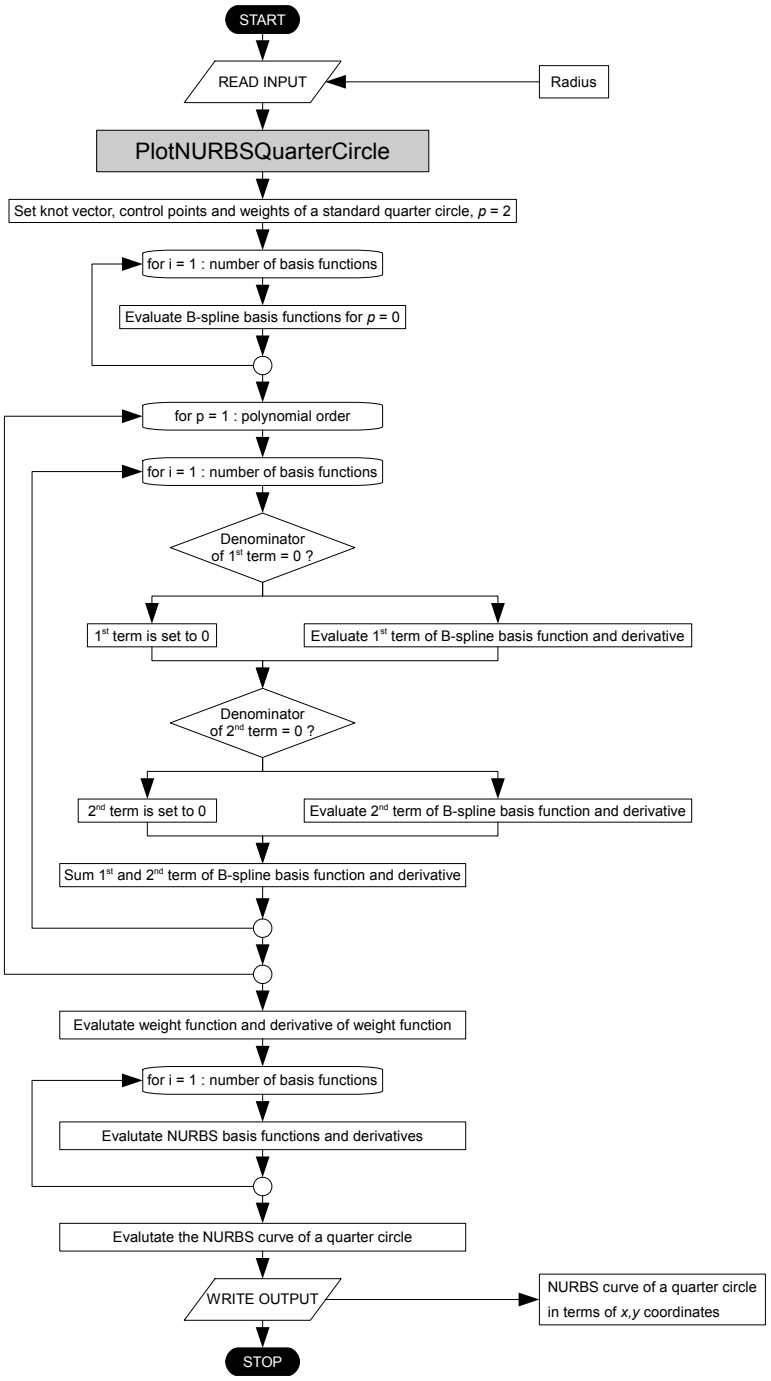


Figure B.31: Flow chart for PlotNURBSQuarterCircle.m

```
% This function evaluates the NURBS curve of a quarter circle for the
% purpose of plotting
%
% Output:   x,y - physical coordinates of curve
%
% Input:    r - radius of curve

function [x,y]=PlotNURBSQuarterCircle(r)

% KNOT VECTOR
XI=[0 0 0 1 1 1];
xi=(0:0.01:XI(end));                % Assume that knot vector starts at 0

% CONTROL POINTS AND WEIGHTS
P=[0 r;r r;r 0]; w=[1 1/sqrt(2) 1]';

% POLYNOMIAL ORDER
poly=length(find(XI(1,:)==0))-1;    % Assume that knot vector starts at 0

% FOR STRUCTURE
N=zeros(1,length(xi)); dN=zeros(1,length(xi));
W=zeros(1,length(xi)); dW=zeros(1,length(xi));
Rb=zeros(1,length(xi)); dR=zeros(1,length(xi));

% p=0
p=0;
n=length(XI)-p-1;                   % Number of functions Ni
for i=1:n
    for j=1:length(xi)
        if xi(j)>=XI(i) && xi(j)<XI(i+1)
            N(i,j)=1;
        else
            N(i,j)=0;
        end
    end
end

% p=1,2,3,...
for p=1:poly
n=length(XI)-p-1;                   % Number of functions Ni
    for i=1:n
        for j=1:length(xi)
            if (XI(i+p)-XI(i))==0
                A=0;
                a=0;
            else
                A=(xi(j)-XI(i))*N(i,j)/(XI(i+p)-XI(i));
                a=poly*N(i,j)/(XI(i+p)-XI(i));
            end
            if (XI(i+p+1)-XI(i+1))==0
                B=0;
                b=0;
            else
                B=(XI(i+p+1)-xi(j))*N(i+1,j)/(XI(i+p+1)-XI(i+1));
                b=poly*N(i+1,j)/(XI(i+p+1)-XI(i+1));
            end
```

```
            N(i,j)=A+B;             % B-spline basis functions
            dN(i,j)=a-b;            % B-spline derivatives
        end
    end
end
N(n,length(xi))=1;                  % End value of Nn is set to 1 (not 0)

for j=1:length(xi)
    W(j)=N(1:n,j)'*w(1:n);          % Weight function
    dW(j)=dN(1:n,j)'*w(1:n);        % Derivative of weight function
end

for i=1:n
    for j=1:length(xi)
        Rb(i,j)=N(i,j)*w(i)/W(j);   % NURBS basis functions
        dR(i,j)=w(i)*(dN(i,j)/W(j)-dW(j)*N(i,j)/W(j)^2);    % NURBS deriv.
    end
end

% EVALUATE NURBS CURVE
x=zeros(1,length(xi)); y=zeros(1,length(xi));
for j=1:length(xi)
    x(j)=Rb(1:n,j)'*P(:,1);
    y(j)=Rb(1:n,j)'*P(:,2);
end

end
```

B.4.26 Plot Stresses

This subfunction calculates strains according to Eq. (2.7) and stresses according to Eqs. (2.1) and (5.1.1) and plots contour plots of the stresses σ_x, σ_y and τ_{xy} (organized as described in Section 5.2.3). Application of this subfunction requires a tensor product geometry.

Input:	`IEN`, `P`, `D`, `E`, `nx`, `ny`, `ncp`, `poly`, `w`, `C`
Subfunctions:	GaussMatrix.m, NURBSBasis.m, Jacobian.m

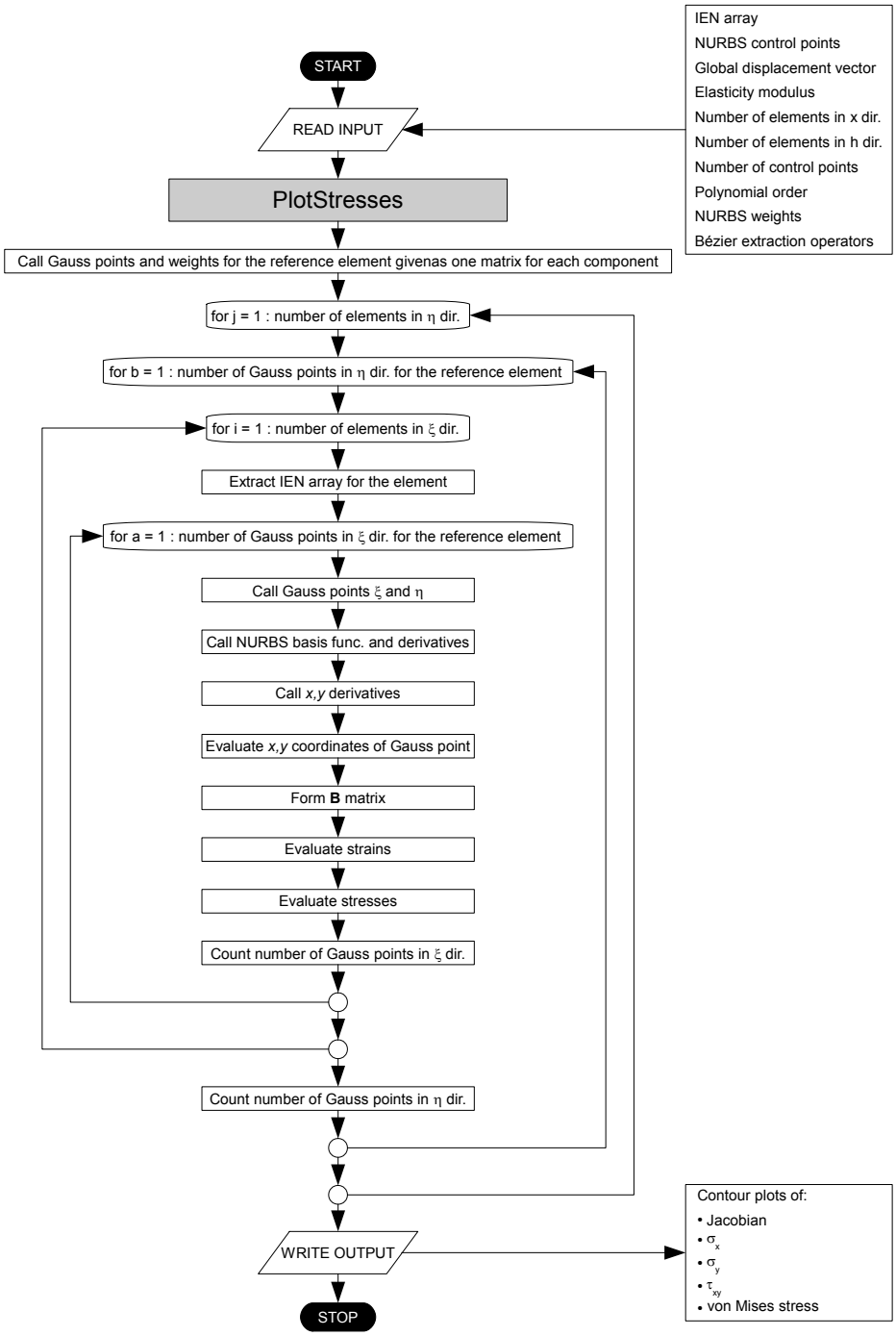


Figure B.32: Flow chart for PlotStresses.m

```
% This function calculates strains and stresses (plane stress) and plots
% contour plots of the stresses and also the Jacobian
%
% Output:   contour plot of Jacobian
%           contour plot of sigma x
%           contour plot of sigma y
%           contour plot of tau xy
%           contour plot of Mises stress
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of control points
%           D - global displacement vector
%           E - constitutive matrix
%           nx - number of elements, xi direction
%           ny - number of elements, eta direction
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function PlotStresses(IEN,P,D,E,nx,ny,ncp,poly,w,C)

G=GaussMatrix(poly);                    % Call Gauss points

X=zeros(nx*size(G,1),ny*size(G,2));
Y=zeros(nx*size(G,1),ny*size(G,2));
Jac=zeros(nx*size(G,1),ny*size(G,2));
stress=zeros(nx*size(G,1),ny*size(G,2),3);
Mises=zeros(nx*size(G,1),ny*size(G,2));

d=1;
for j=1:ny
    for b=1:size(G,2)                   % For each Gauss point in eta dir.
        c=1;

        for i=1:nx
            e=(j-1)*nx+i;               % Current element number
            IEN_e=IEN(e,:);             % Element topology of current el.
            eDof=[IEN_e IEN_e+ncp];     % eDof: first x, then y

            for a=1:size(G,1)           % For each Gauss point in xi dir.
                xi=G(a,b,1);            % Gauss coord. reference element
                eta=G(a,b,2);           % Gauss coord. reference element

                [Rb,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);
                [J,dxy]=Jacobian(dR,P,IEN_e);

                Jac(c,d)=det(J);

                X(c,d)=Rb*P(IEN_e,1);
                Y(c,d)=Rb*P(IEN_e,2);

                B=zeros(3,2*(poly+1)^2);        % B matrix
                B(1,1:(poly+1)^2)=dxy(1,:);
                B(2,(poly+1)^2+1:2*(poly+1)^2)=dxy(2,:);
                B(3,1:(poly+1)^2)=dxy(2,:);
```

```
                B(3,(poly+1)^2+1:2*(poly+1)^2)=dxy(1,:);

                strains=B*D(eDof);      % Strains (3x1) in each Gauss point
                stress(c,d,:)=E*strains;
                Mises(c,d)=sqrt(stress(c,d,1)^2-stress(c,d,1)*...
                    stress(c,d,2)+stress(c,d,2)^2+3*stress(c,d,3)^2);

                c=c+1;
            end

        end

        d=d+1;
    end
end

nrG=nx*size(G,1)*ny*size(G,2);

figure
contourf(X,Y,Jac(:,:),500,'LineStyle','none');
colorbar
axis('equal')
title(['Jacobian (',num2str(nrG),' Gauss points)'])
xlabel('x'); ylabel('y')

figure
contourf(X,Y,stress(:,:,1),500,'LineStyle','none');
colorbar
axis('equal')
title(['\sigma_x (',num2str(nrG),' Gauss points)'])
xlabel('x'); ylabel('y')

figure
contourf(X,Y,stress(:,:,2),500,'LineStyle','none');
colorbar
axis('equal')
title(['\sigma_y (',num2str(nrG),' Gauss points)'])
xlabel('x'); ylabel('y')

figure
contourf(X,Y,stress(:,:,3),500,'LineStyle','none');
colorbar
axis('equal')
title(['\tau_x_y (',num2str(nrG),' Gauss points)'])
xlabel('x'); ylabel('y')

figure
contourf(X,Y,Mises(:,:),500,'LineStyle','none');
colorbar
axis('equal')
title(['von Mises stress (',num2str(nrG),' Gauss points)'])
xlabel('x'); ylabel('y')

end
```

B.4.27 Solution

This function solves the global system matrices with respect to active degrees of freedom, described in Section 2.2.2.

Output: `D, R`

Input: `gDof, prDof, K, D, R`

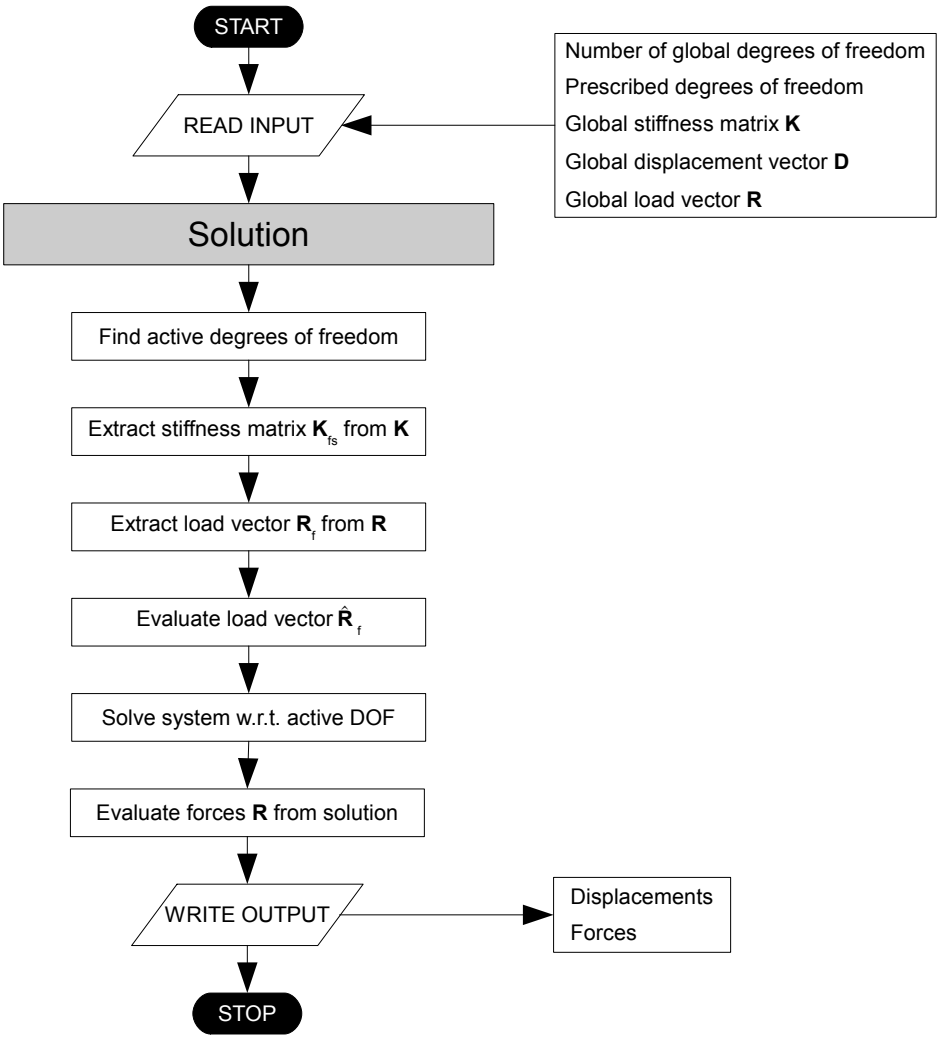


Figure B.33: Flow chart for Solution.m

```
% This function solves the global system matrices w.r.t. active DOF
%
% Output:   D - global displacement vector
%
% Input:    gDof - global degrees of freedom
%           prDof - prescribed degrees of freedom
%           K - global stiffness matrix
%           D - global displacement vector
%           R - global load vector

function [D,R]=Solution(gDof,prDof,K,D,R)

aDof=setdiff(1:gDof,prDof);     % Active DOF

Kfs=K(aDof,prDof);
Rf=R(aDof);
Rf_n=Rf-Kfs*D(prDof);

D(aDof)=K(aDof,aDof)\Rf_n;      % Solution

R=K*D;                          % Forces

end
```

B.4.28 Stresses

This subfunction calculates strains according to Eq. (2.7) and stresses according to Eqs. (2.1) and (2.2) and prints out stresses σ_x, σ_y and τ_{xy} as described in Section 2.2.3. The subfunction is applied to T-spline surfaces from Rhino since these geometries do not have a tensor product structure.

Output:	`stress`
Input:	`IEN`, `P`, `D`, `E`, `nel`, `ncp`, `poly`, `w`, `C`
Subfunctions:	Gauss.m, NURBSBasis.m, Jacobian.m

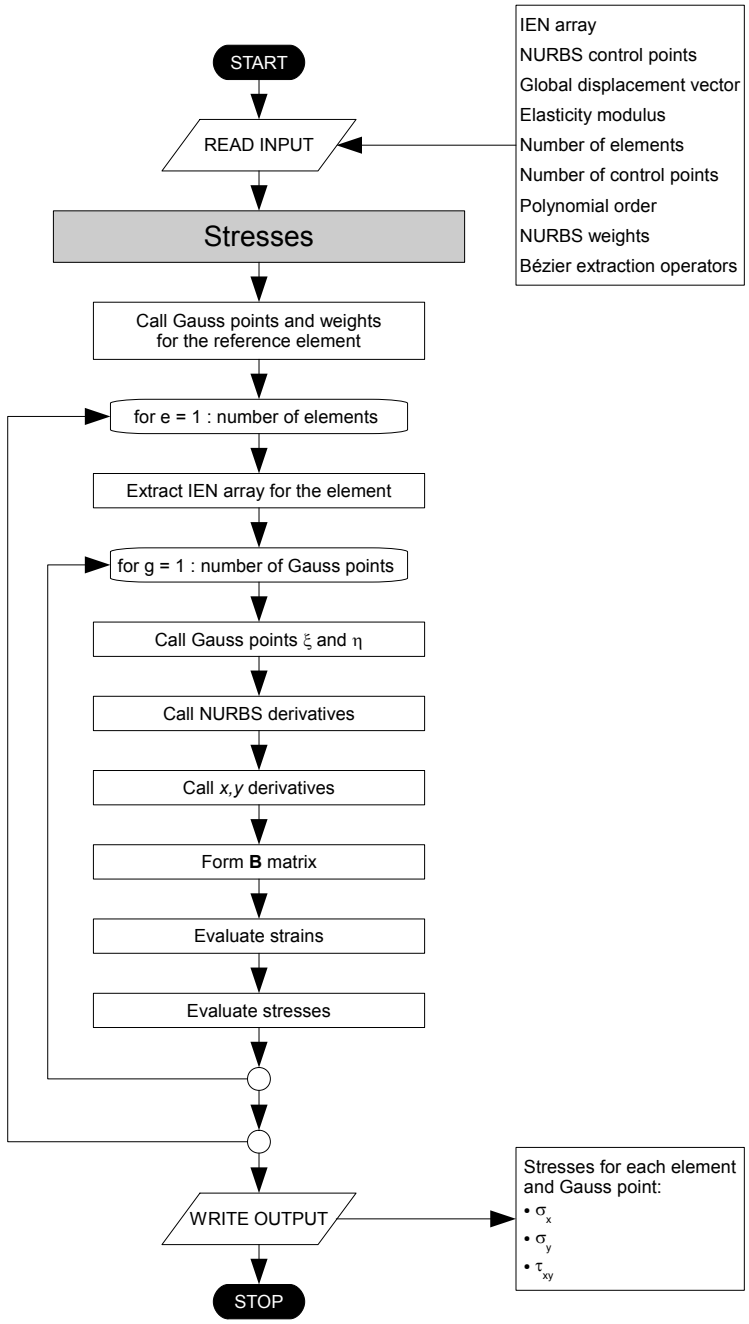


Figure B.34: Flow chart for Stresses.m

```
% This function calculates strains and stresses (plane stress/strain)
%
% Output:   stress - given as three matrices, one for each stress component
%               stress(e,g,1) - sigma x
%               stress(e,g,2) - sigma y
%               stress(e,g,3) - tau xy
%               rows repr. elements (e), columns repr. Gauss points (g)
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of control points
%           D - global displacement vector
%           E - constitutive matrix
%           nel - total number of elements
%           ncp - number of control points
%           poly - polynomial order
%           w - weights of NURBS control points
%           C - Bézier extraction operators

function stress=Stresses(IEN,P,D,E,nel,ncp,poly,w,C)

[G,~]=Gauss(poly);                      % Call Gauss points

stress=zeros(nel,size(G,1),3);

for e=1:nel
        IEN_e=nonzeros(IEN(e,:))';      % Element topology of current el.
        eDof=[IEN_e IEN_e+ncp];         % eDof: first x, then y

    for g=1:size(G,1)                   % For each Gauss point
        xi=G(g,1);                      % Gauss coord. reference element
        eta=G(g,2);                     % Gauss coord. reference element

        [~,dR]=NURBSBasis(xi,eta,poly,e,IEN_e,w,C);

        [~,dxy]=Jacobian(dR,P,IEN_e);

        B=zeros(3,2*length(IEN_e));     % B matrix
        B(1,1:length(IEN_e))=dxy(1,:);
        B(2,length(IEN_e)+1:2*length(IEN_e))=dxy(2,:);
        B(3,1:length(IEN_e))=dxy(2,:);
        B(3,length(IEN_e)+1:2*length(IEN_e))=dxy(1,:);

        strains=B*D(eDof);              % Strains (3x1) in each Gauss point
        stress(e,g,:)=E*strains;
    end

end

end
```

Appendix C

MATLAB Code for Conventional Isogeometric Analysis

This appendix presents the MATLAB code for conventional isogeometric analysis as opposed to isogeometric analysis based on Bézier extraction. The program was prepared in the project work [17] for the two specific problems Cook's problem and the end loaded beam.

The conventional program applies many of the same subfunctions as the program based on Bézier extraction. The differences are related to the subfunctions which make use of the basis functions, i.e. subfunctions to form load vector and stiffness matrix, and to evaluate displacements, stresses and energy.

Note that the program makes use of B-spline basis functions instead of NURBS basis functions, however, this is not of significance for the two problems since all weights are equal to 1.

The user definition and variable description given in Appendix B prevail, except for:

- The geometry may only by a square
- The load may not be a traction field

C.1 Main Code

C.1.1 Cook's Problem

Figure C.1 shows a flow chart of the subfunctions involved in the main code for Cook's Problem, IA_CooksProblem_.m.

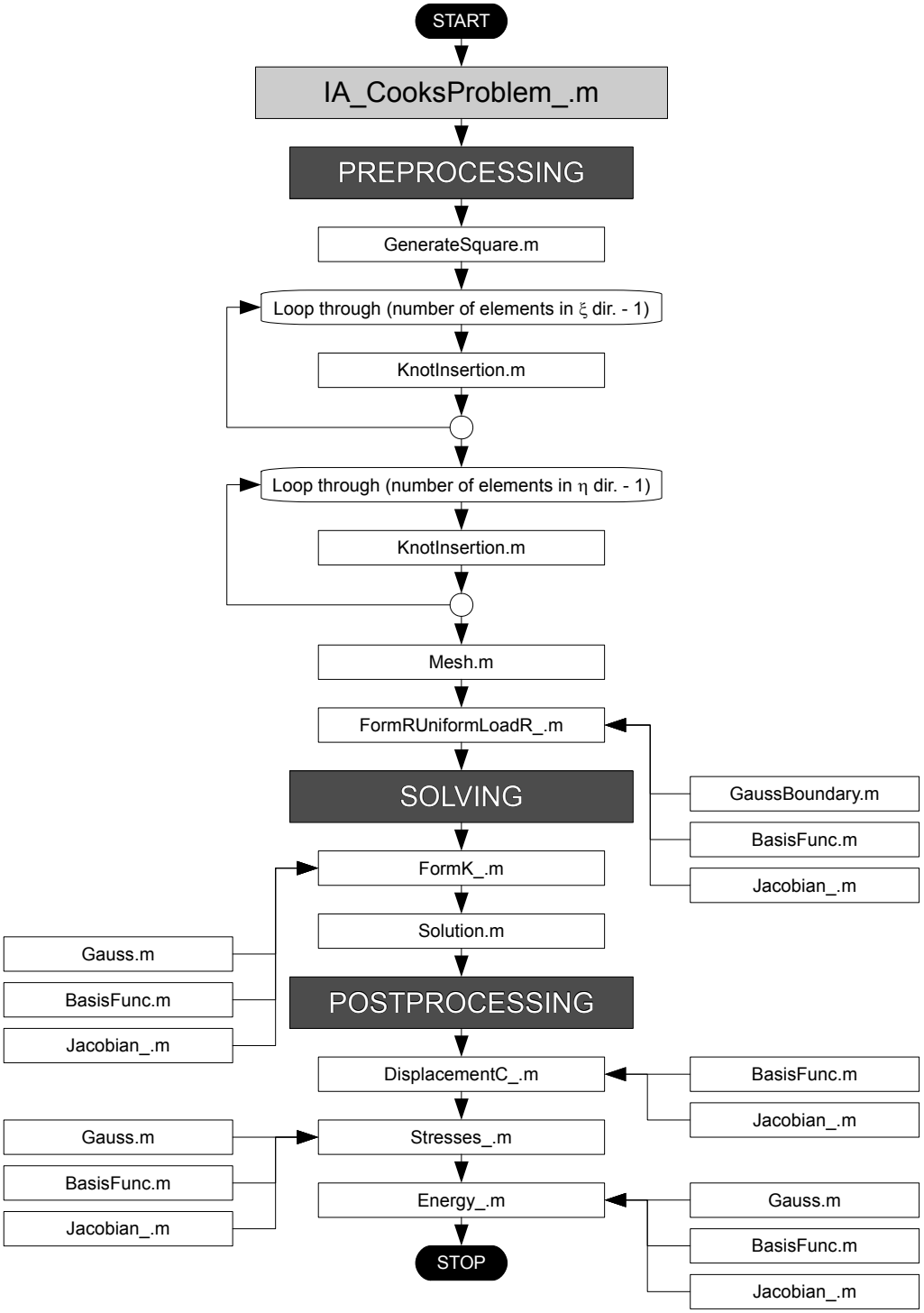


Figure C.1: Cook's problem, flow chart of the subfunctions involved

```
%-----------------ISOGEOMETRIC ANALYSIS - COOK'S PROBLEM-----------------%

close all; clear all
tStart=tic;

%-----------------------------------PRE-----------------------------------%

% MATERIAL PROPERTIES
Emod=1;
nu=1/3;
E=Emod/(1-nu^2)*[1 nu 0;nu 1 0;0 0 (1-nu)/2];   % Plane stress

% GEOMETRY, CONTROL POINTS AND ELEMENT TOPOLOGY
t=1;                              % Thickness
XY=[0 0;48 44;0 44;48 60];        % Coordinates of corners [X1 Y1;...;X4 Y4],
                                  % left to right and bottom to top
nx=15;          % Number of elements, xi direction
ny=15;          % Number of elements, eta direction
poly=2;         % Polynomial order, same in both directions
nel=nx*ny;      % Total number of elements
n=nx+poly;      % Number of basis functions/control points, xi direction
m=ny+poly;      % Number of basis functions/control points, eta direction
ncp=n*m;        % Total number of basis functions/control points
gDof=2*ncp;     % Global degrees of freedom

% The function GenerateSquare.m generates control points and weights
% from the given square
[B,knot]=GenerateSquare(XY,poly);

% The function KnotInsertion.m refines the mesh by knot insertion
for i=1:nx-1
    [B knot.xi]=KnotInsertion(B,knot.xi,i/nx,poly,1);
end
for j=1:ny-1
    [B knot.eta]=KnotInsertion(B,knot.eta,j/ny,poly,2);
end

% Transfer control points from two matrices (one for x values and one for
% y) to one matrix with both x (1st column) and y (2nd)
P=zeros(ncp,2);
k=1;
for j=1:m
  for i=1:n
      P(k,1)=B(i,j,1);
      P(k,2)=B(i,j,2);
      k=k+1;
  end
end

% ELEMENT TOPOLOGY
IEN=Mesh(nx,ny,poly);

% FOR STRUCTURE
K=zeros(gDof);
D=zeros(gDof,1);
R=zeros(gDof,1);
```

```
% LOADS
% Uniformly distributed shear traction at right hand side (sum = 1)
% The function FormRUniformLoadR_.m numerical integrates consistent nodal
% loads and assembles to R
q=1/16;
R=FormRUniformLoadR_(IEN,P,R,nx,ny,ncp,poly,q);

% BOUNDARY CONDITIONS
% Fixed at left end
prDof=zeros(2*m,1);
for j=1:m
    prDof(j)=1+n*(j-1);
    prDof(m+j)=1+m*(j-1)+ncp;
end

%---------------------------------SOLVE---------------------------------%

% STIFFNESS MATRIX
% The function FormK_.m forms k for each element and assembles to K
K=FormK_(IEN,P,K,E,t,nx,ny,ncp,poly);

% SOLVE SYSTEM
% The function Solution.m solves the system in terms of active DOFs
[D,R]=Solution(gDof,prDof,K,D,R);

%---------------------------------POST----------------------------------%

disp(['IGA COOK`S PROBLEM P',num2str(poly),' (',num2str(gDof),' DOF)'])
SolutionTime=toc(tStart)

% Write displacements and reactions
Displacements=[(1:gDof)' D];
Reactions=[prDof R(prDof)];

% Calculate vertical displacement at point C
vC=DisplacementC_(IEN,D,nx,ny,ncp,poly)

% Calculate strains and stresses and write stresses at Gauss points
stress=Stresses_(IEN,P,D,E,nx,ny,ncp,poly);

% Calculate strain energy of the system
U1=D'*R;
[U2,U3]=Energy_(IEN,P,D,E,t,nx,ny,ncp,poly);

toc(tStart)
```

C.1.2 End Loaded Beam

Figure C.2 shows a flow chart of the subfunctions involved in the main code for the end loaded beam, IA_EndLoadedBeam_.m.

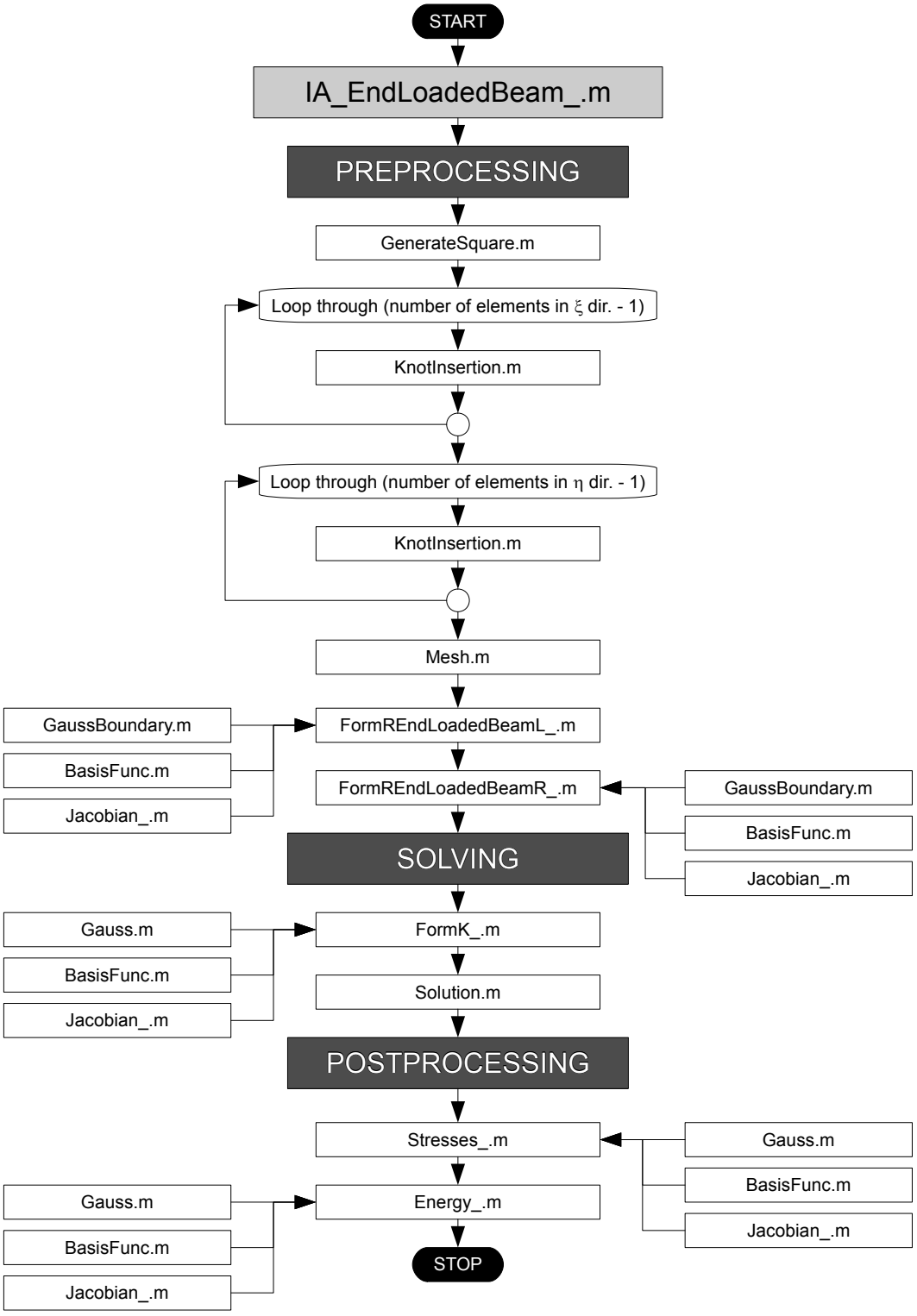


Figure C.2: End loaded beam, flow chart of the subfunctions involved

```
%-----------------ISOGEOMETRIC ANALYSIS - END LOADED BEAM-----------------%

close all; clear all
tStart=tic;

%-----------------------------------PRE-----------------------------------%

% MATERIAL PROPERTIES
Emod=1000;
nu=0.25;
E=Emod/(1-nu^2)*[1 nu 0;nu 1 0;0 0 (1-nu)/2];   % Plane stress

% GEOMETRY AND CONTROL POINTS
t=1;                              % Thickness
XY=[0 -10;100 -10;0 10;100 10]; % Coordinates of corners [X1 Y1;...;X4 Y4],
                                  % left to right and bottom to top
nx=22;          % Number of elements, xi direction
ny=11;          % Number of elements, eta direction
poly=2;         % Polynomial order, same in both directions
nel=nx*ny;      % Total number of elements
n=nx+poly;      % Number of basis functions/control points, xi direction
m=ny+poly;      % Number of basis functions/control points, eta direction
ncp=n*m;        % Total number of basis functions/control points
gDof=2*ncp;     % Global degrees of freedom

% The function GenerateSquare.m generates control points and weights
% from the given square
[B,knot]=GenerateSquare(XY,poly);

% The function KnotInsertion.m refines the mesh by knot insertion
for i=1:nx-1
    [B knot.xi]=KnotInsertion(B,knot.xi,i/nx,poly,1);
end
for j=1:ny-1
    [B knot.eta]=KnotInsertion(B,knot.eta,j/ny,poly,2);
end

% Transfer control points from two matrices (one for x values and one for
% y) to one matrix with both x (1st column) and y (2nd)
P=zeros(ncp,2);
k=1;
for j=1:m
  for i=1:n
      P(k,1)=B(i,j,1);
      P(k,2)=B(i,j,2);
      k=k+1;
  end
end

% ELEMENT TOPOLOGY
IEN=Mesh(nx,ny,poly);

% FOR STRUCTURE
K=zeros(gDof);
D=zeros(gDof,1);
R=zeros(gDof,1);
```

```
% LOADS
% Parabolic shear traction (sum = 80) at both sides (shear force equivalent)
% Normal traction (+- 120 top/bottom) at support side (moment equivalent)
% The function FormREndLoadedBeamR_.m numerical integrates consistent
% nodal loads at right hand side and assembles to R
% The function FormREndLoadedBeamL_.m numerical integrates consistent
% nodal loads at left hand side and assembles to R
R=FormREndLoadedBeamR_(IEN,P,R,nx,ny,ncp,poly);
R=FormREndLoadedBeamL_(IEN,P,R,nx,ny,ncp,poly);

% BOUNDARY CONDITIONS
% u(0,-10) = u(0,0) = u(0,10) = v(0,0) = 0
prDof=[1 1+((m-1)/2)*n 1+(m-1)*n 1+((m-1)/2)*n+ncp]';

%----------------------------------SOLVE---------------------------------%

% STIFFNESS MATRIX
% The function FormK_.m forms k for each element and assembles to K
K=FormK_(IEN,P,K,E,t,nx,ny,ncp,poly);

% SOLVE SYSTEM
% The function Solution.m solves the system in terms of active DOFs
[D,R]=Solution(gDof,prDof,K,D,R);

%----------------------------------POST----------------------------------%

disp(['IGA END LOADED BEAM P2 (',num2str(gDof),' DOF)'])
SolutionTime=toc(tStart)

% Write displacements and reactions
Displacements=[(1:gDof)' D];
Reactions=[prDof R(prDof)];

% Calculate strains and stresses and write stresses at Gauss points
stress=Stresses_(IEN,P,D,E,nx,ny,ncp,poly);

% Calculate strain energy of the system
U1=D'*R
[U2,U3]=Energy_(IEN,P,D,E,t,nx,ny,ncp,poly)

toc(tStart)
```

C.2 Subfunctions

The following subfunctions for conventional isogeometric analysis are shared with the program based on Bézier extraction. These subfunctions may therefore be found in Appendix B.4,

- Gauss
- Gauss Boundary
- Generate Square
- Knot Insertion
- Mesh
- Solution

C.2.1 Basis Func

This subfunction forms B-spline basis functions and derivatives separated in ξ and η direction based on Eqs. (3.2), (3.3) and (3.4) from Section 3.2.2.

Output: `Nxi`, `dxi`, `Neta`, `deta`

Input: `xi`, `eta`, `nx`, `ny`, `poly`

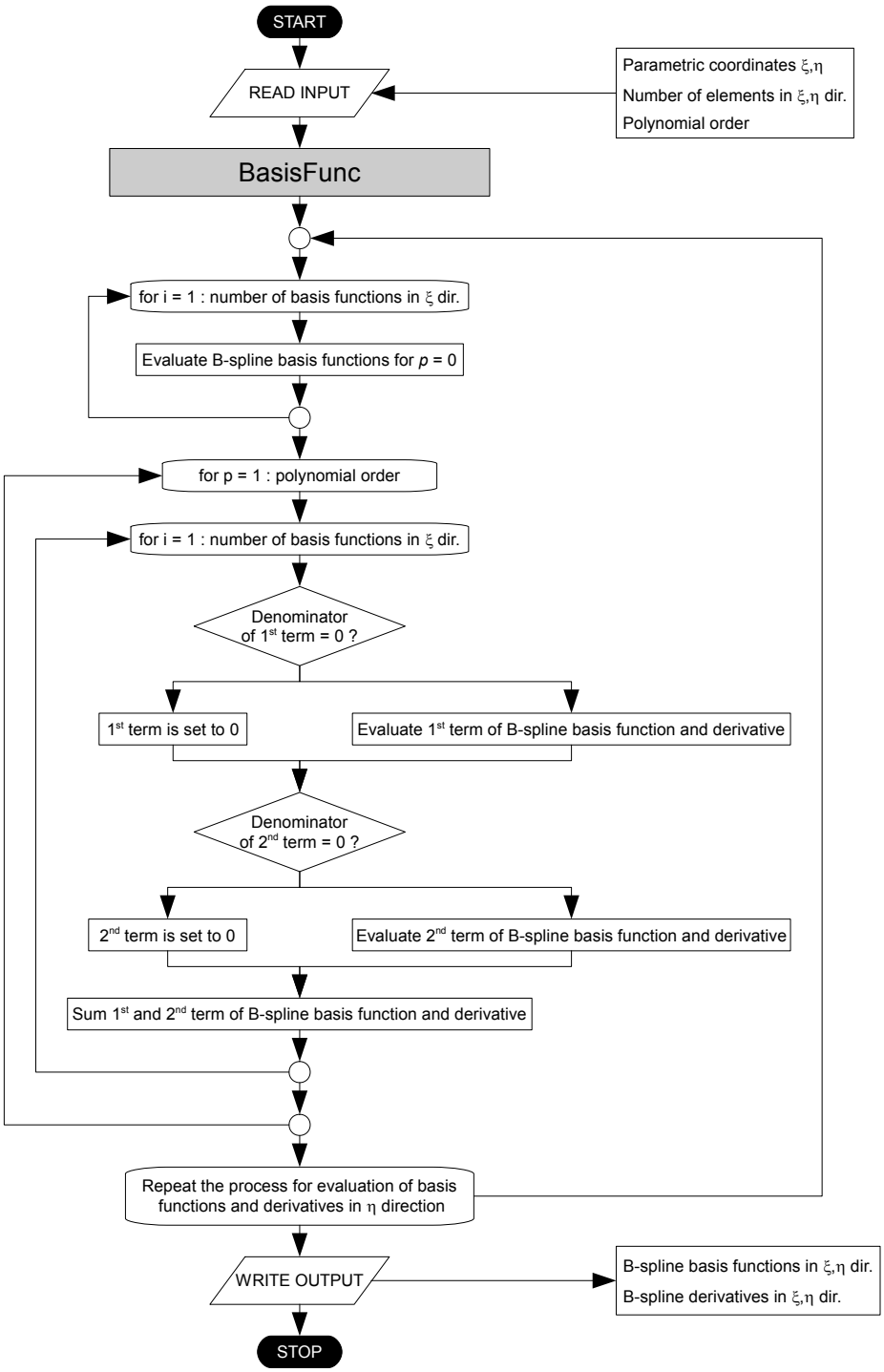


Figure C.3: Flow chart for BasisFunc.m

```
% This function forms basis functions and derivatives
%
% Output:   Nxi - basis functions xi direction
%           dxi - derivatives of basis functions xi direction
%           Neta - basis functions eta direction
%           deta - derivatives of basis functions eta direction
%
% Input:    xi - coord. of 1st parametric direction
%           eta - coord. of 2nd parametric direction
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           poly - polynomial order

function [Nxi,dxi,Neta,deta]=BasisFunc(xi,eta,nx,ny,poly)

% KNOT VECTOR
q=poly+1;               % Assume same polynomial order in both directions

XI(1:q)=0;
XI(q+1:q+nx-1)=(1:nx-1);
XI(q+nx:2*q+nx-1)=nx;

ETA(1:q)=0;
ETA(q+1:q+ny-1)=(1:ny-1);
ETA(q+ny:2*q+ny-1)=ny;

% FOR STRUCTURE
Nxi=zeros(1,1); dxi=zeros(1,1);
Neta=zeros(1,1); deta=zeros(1,1);

%-----------------Basis functions Nxi and derivatives dxi-----------------%

% p=0
p=0;
n=length(XI)-p-1;       % Number of functions Ni
for i=1:n
        if xi>=XI(i) && xi<XI(i+1)
            Nxi(i)=1;
        else
            Nxi(i)=0;
        end
end

% p=1,2,3,...
for p=1:poly
n=length(XI)-p-1;
    for i=1:n
        if (XI(i+p)-XI(i))==0
            A=0;
            a=0;
        else
            A=(xi-XI(i))*Nxi(i)/(XI(i+p)-XI(i));
            a=p*Nxi(i)/(XI(i+p)-XI(i));
        end
        if (XI(i+p+1)-XI(i+1))==0
            B=0;
```

```
            b=0;
        else
            B=(XI(i+p+1)-xi)*Nxi(i+1)/(XI(i+p+1)-XI(i+1));
            b=p*Nxi(i+1)/(XI(i+p+1)-XI(i+1));
        end
        Nxi(i)=A+B;
        dxi(i)=a-b;
    end
end

%----------------Basis functions Neta and derivatives deta----------------%

% p=0
p=0;
m=length(ETA)-p-1;
for i=1:m
        if eta>=ETA(i) && eta<ETA(i+1)
            Neta(i)=1;
        else
            Neta(i)=0;
        end
end

% p=1,2,3,...
for p=1:poly
m=length(ETA)-p-1;
    for i=1:m
        if (ETA(i+p)-ETA(i))==0
            A=0;
            a=0;
        else
            A=(eta-ETA(i))*Neta(i)/(ETA(i+p)-ETA(i));
            a=p*Neta(i)/(ETA(i+p)-ETA(i));
        end
        if (ETA(i+p+1)-ETA(i+1))==0
            B=0;
            b=0;
        else
            B=(ETA(i+p+1)-eta)*Neta(i+1)/(ETA(i+p+1)-ETA(i+1));
            b=p*Neta(i+1)/(ETA(i+p+1)-ETA(i+1));
        end
        Neta(i)=A+B;
        deta(i)=a-b;
    end
end

end
```

C.2.2 Displacement C

This subfunction calculates vertical displacement at point C of Cook's problem by interpolating the basis functions over the middle element at the right hand side boundary. The number of elements in η direction should therefore be odd, `ny` =an odd number. An exception is for polynomial order 1 where the basis functions are exactly equal to 1 at the control points and no interpolation is necessary, but in return there must be a control point at the middle. The number of elements in η direction for `poly` = `1` should therefore be even, `ny` =an even number.

Since the basis functions in the conventional program are related to the parameter space and not the reference element, the ξ and η values are given for the parameter space. Basis functions extracted are the ones in η direction which support the middle element.

Output: `vC`
Input: `IEN`, `D`, `nx`, `ny`, `ncp`, `poly`

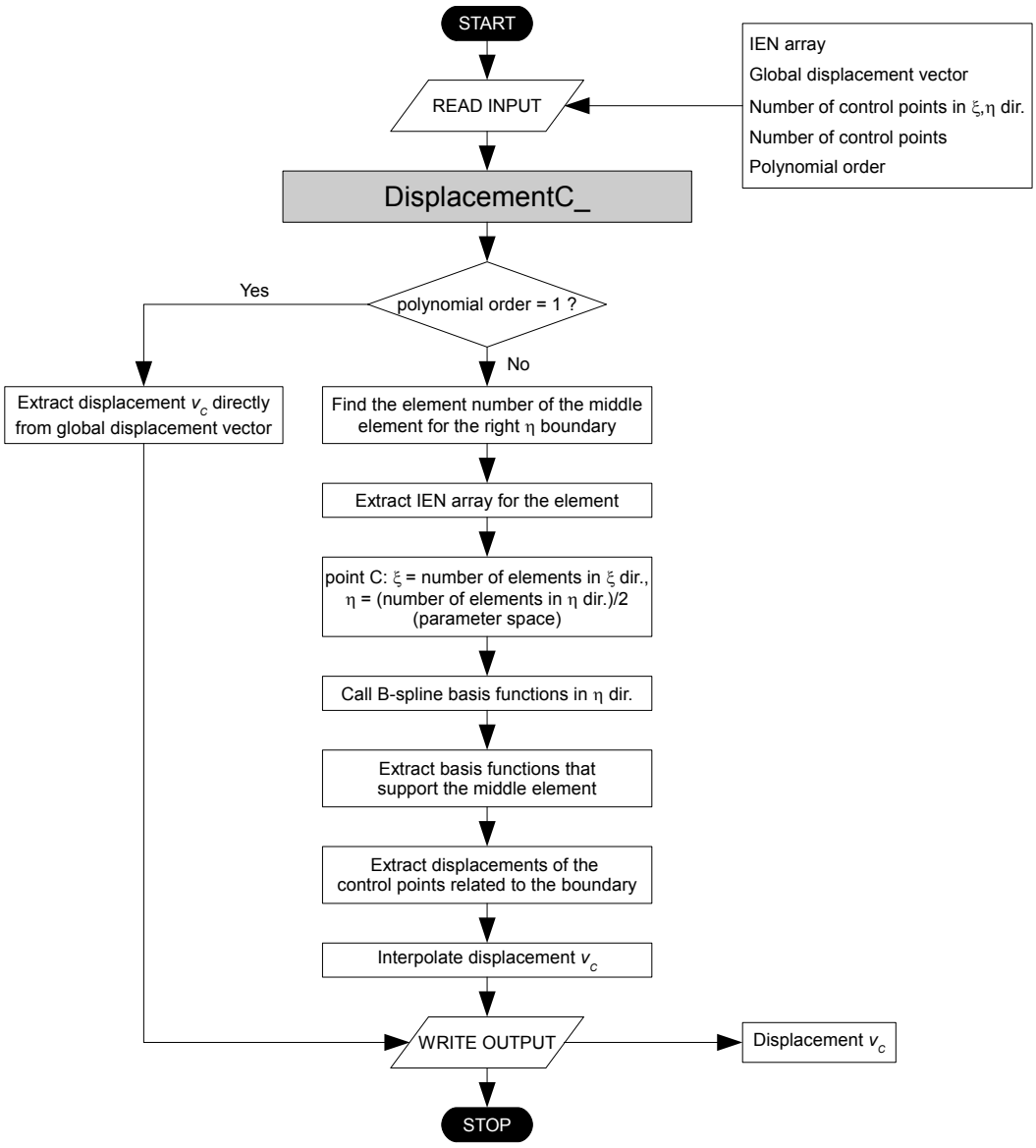


Figure C.4: Flow chart for DisplacementC_.m

```
% This function calculates vertical displ. of point C in Cook's problem
% by interpolating the basis functions over the middle element at the
% right hand side boundary. The *number of elements in eta direction*
% should therefore be *odd*.
%
% An exception is for *IGA P1* where the basis functions are exactly equal
% to 1 at the control points and no interpolation is necessary, but in
% return there must be a control point at the middle. The *number of
% elements in eta direction* for poly=1 should therefore be *even*.
%
% Output:   vC - vertical displacement at point C
%
% Input:    IEN - element topology: numbering of control points
%           D - global displacement vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order

function vC=DisplacementC_(IEN,D,nx,ny,ncp,poly)

if poly==1
    vC=D((ny/2+1)*(nx+poly)+ncp);
else
    s=(ny+1)/2;
    e=nx*s;
    IEN_e=IEN(e,:);

    xi=nx;
    eta=ny/2;
    [~,~,Neta,~]=BasisFunc(xi,eta,nx,ny,poly);
    Neta=Neta(s:s+poly);

    D_ey=D(IEN_e+ncp);
    D_eyR=D_ey((poly+1):(poly+1):(poly+1)^2);
    vC=Neta*D_eyR;
end

end
```

C.2.3 Energy

This subfunction calculates strain energy of the system by applying Eqs. (2.30) and (2.31). The subfunction is more extensive than its counterpart in Appendix C.2.3; the double loop through the elements, the mapping of Gauss points from the reference element to the parameter space, the mapping of derivatives from the parameter space and back to the reference element (where the Jacobian is evaluated) and the extracting of basis functions which support the element.

Output:	`U2`, `U3`
Input:	`IEN`, `P`, `D`, `E`, `t`, `nx`, `ny`, `ncp`, `poly`
Subfunctions:	Gauss.m, BasisFunc.m, Jacobian_.m

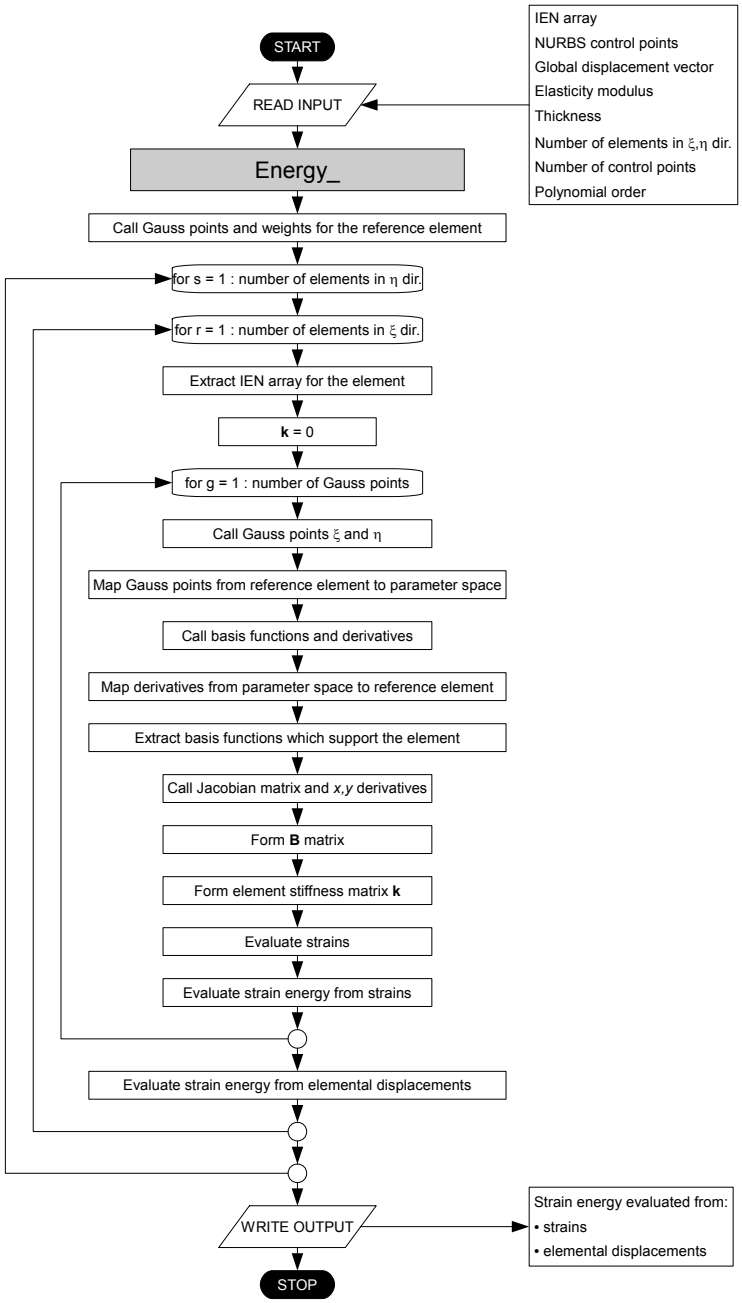


Figure C.5: Flow chart for Energy_.m

```
% This function calculates strain energy of the system
%
% Output:   U2 - strain energy calculated from element stiffness matrices
%           U3 - strain energy calculated from strains
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of control points
%           D - global displacement vector
%           E - constitutive matrix
%           t - thickness
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order


function [U2,U3]=Energy_(IEN,P,D,E,t,nx,ny,ncp,poly)

[G,W]=Gauss(poly);                      % Call Gauss points and weights

U2=0;
U3=0;

e=1;
xi_v=(0:nx);                            % Vectors defining parameter space
eta_v=(0:ny);

for s=1:ny
    for r=1:nx
        IEN_e=nonzeros(IEN(e,:))';      % Element topology of current el.
        eDof=[IEN_e IEN_e+ncp];         % eDof: first x, then y

        k=0;

        for g=1:size(G,1)               % For each Gauss point
            xi_n=G(g,1);                % Gauss coord. reference element
            eta_n=G(g,2);               % Gauss coord. reference element

            % Gauss coordinates in parameter space
            xi=xi_v(r)+(xi_n+1)*(xi_v(r+1)-xi_v(r))/2;
            eta=eta_v(s)+(eta_n+1)*(eta_v(s+1)-eta_v(s))/2;

            [Nxi,dxi,Neta,deta]=BasisFunc(xi,eta,nx,ny,poly);

            dxi=dxi./2;                 % Mapping of derivatives from
            deta=deta./2;               % parameter space to reference el.

            Nxi=Nxi(r:r+poly);          % Collect basis functions and
            dxi=dxi(r:r+poly);          % derivatives which support the el.
            Neta=Neta(s:s+poly);
            deta=deta(s:s+poly);

            [J,dxy]=Jacobian_(Nxi,dxi,Neta,deta,P,IEN_e,poly);

            B=zeros(3,2*length(IEN_e));     % B matrix
            B(1,1:length(IEN_e))=dxy(1,:);
```

```
            B(2,length(IEN_e)+1:2*length(IEN_e))=dxy(2,:);
            B(3,1:length(IEN_e))=dxy(2,:);
            B(3,length(IEN_e)+1:2*length(IEN_e))=dxy(1,:);

            k=k+B'*E*B*t*det(J)*W(g);    % Numerical integration of k
            strains=B*D(eDof);           % Strains (3x1) in each Gauss point
            U3=U3+t*strains'*E*strains*det(J)*W(g);
        end

    U2=U2+D(eDof)'*k*D(eDof);
    e=e+1;
    end
end

end
```

C.2.4 Form K

This subfunction forms the global stiffness matrix for plane stress or plane strain by employing Eq. (2.21). The changes compared to its counterpart in Appendix B.4.7 are as mentioned in Appendix C.2.3.

Output:	`K`
Input:	`IEN`, `P`, `K`, `E`, `t`, `nx`, `ny`, `ncp`, `poly`
Subfunctions:	Gauss.m, BasisFunc.m, Jacobian_.m

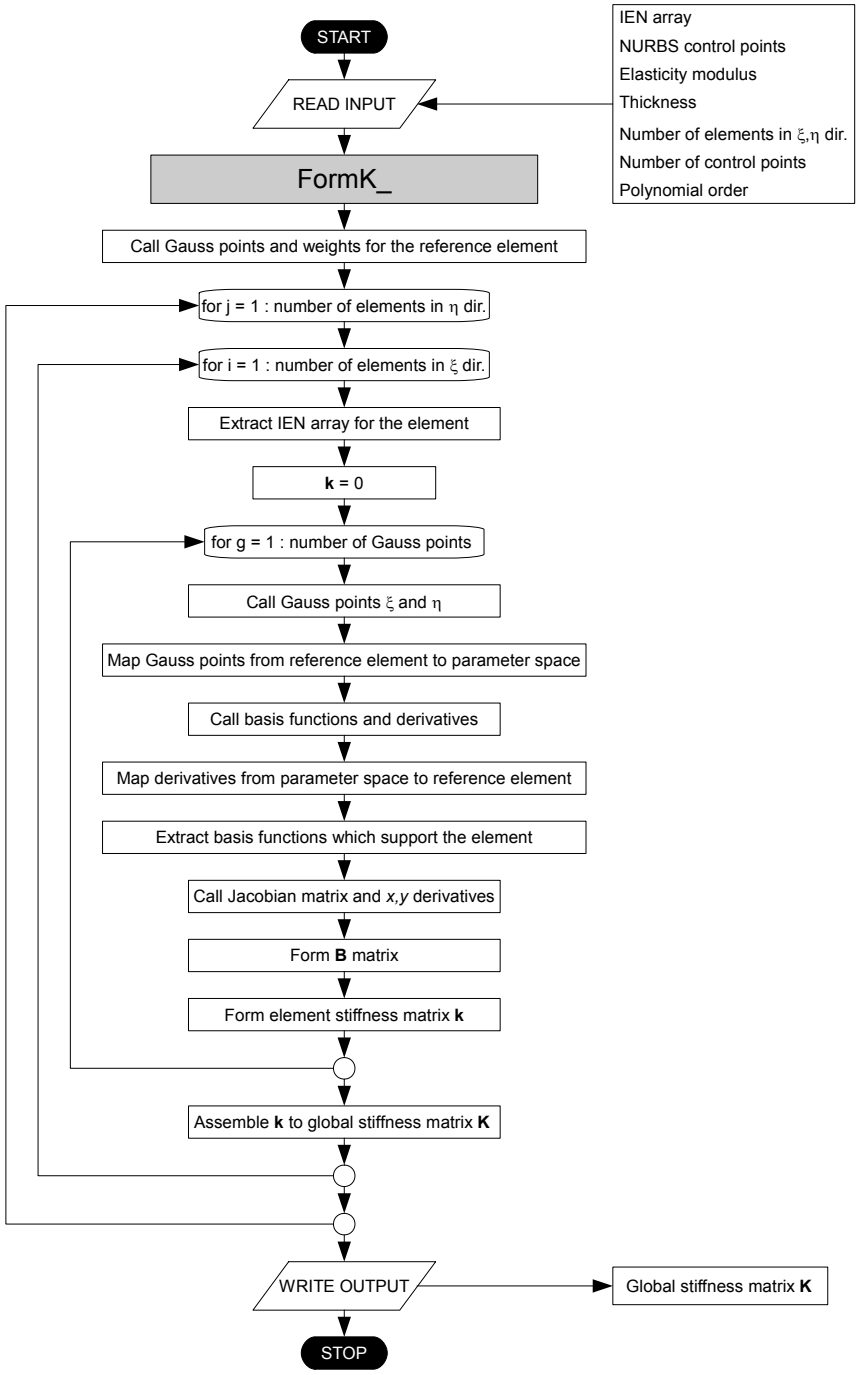


Figure C.6: Flow chart for FormK_.m

```
% This function forms global stiffness matrix (plane stress/strain)
%
% Output:   K - global stiffness matrix
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of control points
%           K - empty global stiffness matrix
%           E - constitutive matrix
%           t - thickness
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order

function K=FormK_(IEN,P,K,E,t,nx,ny,ncp,poly)

[G,W]=Gauss(poly);                      % Call Gauss points and weights

e=1;
xi_v=(0:nx);                            % Vectors defining parameter space
eta_v=(0:ny);

for s=1:ny
    for r=1:nx
        IEN_e=nonzeros(IEN(e,:))';      % Element topology of current el.
        eDof=[IEN_e IEN_e+ncp];         % eDof: first x, then y

        k=0;

        for g=1:size(G,1)               % For each Gauss point
            xi_n=G(g,1);                % Gauss coord. reference element
            eta_n=G(g,2);               % Gauss coord. reference element

            % Gauss coordinates in parameter space
            xi=xi_v(r)+(xi_n+1)*(xi_v(r+1)-xi_v(r))/2;
            eta=eta_v(s)+(eta_n+1)*(eta_v(s+1)-eta_v(s))/2;

            [Nxi,dxi,Neta,deta]=BasisFunc(xi,eta,nx,ny,poly);

            dxi=dxi./2;                 % Mapping of derivatives from
            deta=deta./2;               % parameter space to reference el.

            Nxi=Nxi(r:r+poly);          % Collect basis functions and
            dxi=dxi(r:r+poly);          % derivatives which support the el.
            Neta=Neta(s:s+poly);
            deta=deta(s:s+poly);

            [J,dxy]=Jacobian_(Nxi,dxi,Neta,deta,P,IEN_e,poly);

            B=zeros(3,2*length(IEN_e));     % B matrix
            B(1,1:length(IEN_e))=dxy(1,:);
            B(2,length(IEN_e)+1:2*length(IEN_e))=dxy(2,:);
            B(3,1:length(IEN_e))=dxy(2,:);
            B(3,length(IEN_e)+1:2*length(IEN_e))=dxy(1,:);

            k=k+B'*E*B*t*det(J)*W(g);   % Numerical integration of k
```

```
        end

        K(eDof,eDof)=K(eDof,eDof)+k;
        e=e+1;
    end
end

end
```

C.2.5 Form R End Loaded Beam L and Form End Loaded Beam R

These subfunctions form global load vector for the end loaded beam for the left and right hand side as described in Appendix A.2. The additional processes compared to its counterpart in Appendix B.4.8 are the mapping of Gauss points, the mapping of derivatives and the extracting of basis functions which support the element.

Output:	`R`
Input:	`IEN`, `P`, `R`, `nx`, `ny`, `ncp`, `poly`
Subfunctions:	GaussBoundary.m, BasisFunc.m, Jacobian_.m

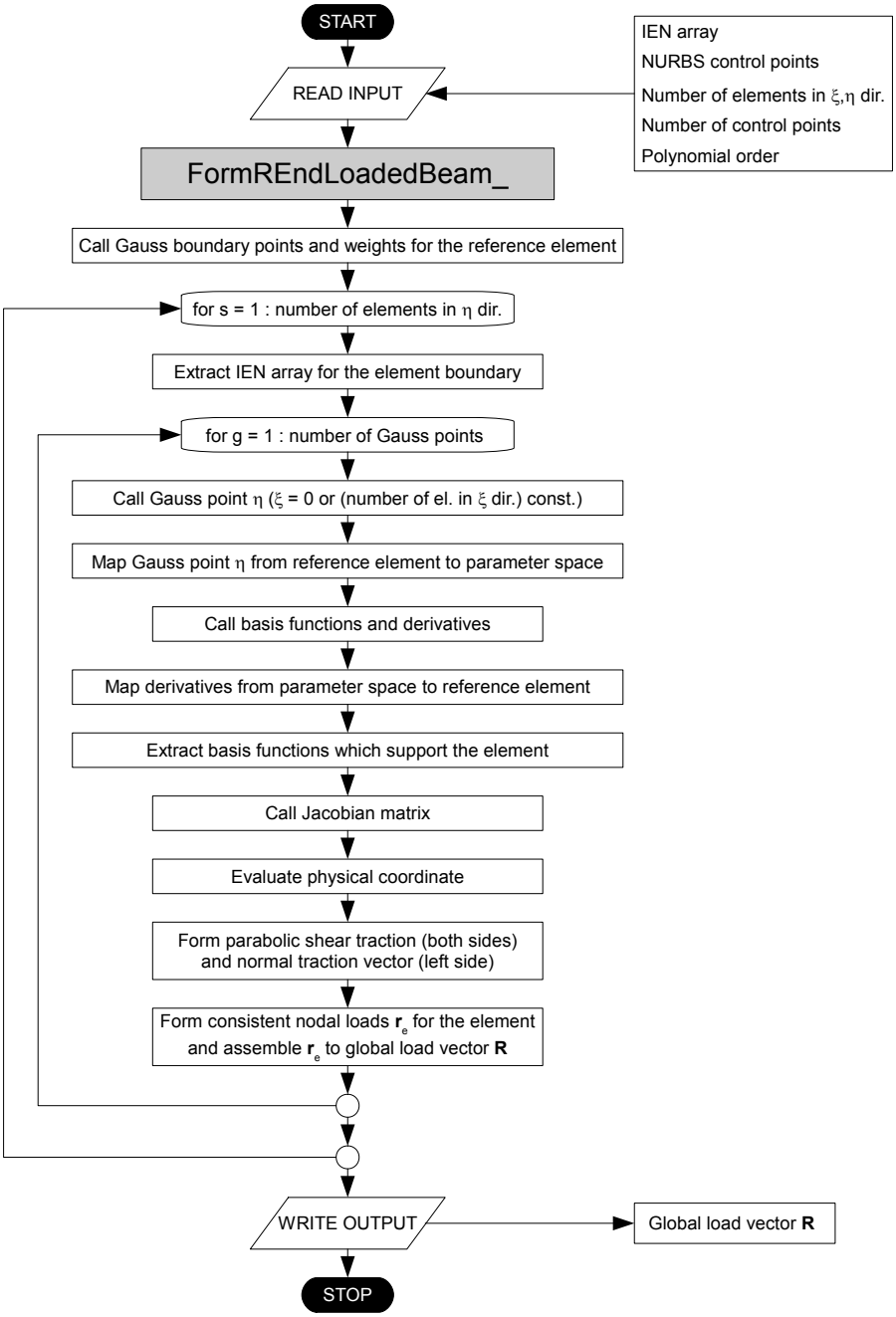


Figure C.7: Flow chart for FormREndLoadedBeamL_.m and FormREndLoadedBeamR_.m

```
% This function forms global load vector for a parabolic shear traction
% (sum = 80) and a normal traction (+- 120 top/bottom) at left hand side
%
% Output:   R - global load vector
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           R - empty global load vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order

function R=FormREndLoadedBeamL_(IEN,P,R,nx,ny,ncp,poly)

[G,W]=GaussBoundary(poly);      % Call Gauss boundary points and weights

xi=0;
r=1;
eta_v=(0:ny);                   % Vector defining parameter space

for s=1:ny
    e=1+nx*(s-1);
    IEN_e=IEN(e,:);             % Element topology of current element
    IEN_ey=IEN_e+ncp;
    % Control points x direction at left hand side for each element
    edgeDofxL=IEN_e(1:(poly+1):(poly+1)^2-poly);
    % Control points y direction at left hand side for each element
    edgeDofyL=IEN_ey(1:(poly+1):(poly+1)^2-poly);

    for g=1:size(G,1)           % For each Gauss boundary point
        eta_n=G(g);             % Gauss coord. reference element

        % Gauss coordinates in parameter space
        eta=eta_v(s)+(eta_n+1)*(eta_v(s+1)-eta_v(s))/2;

        [Nxi,dxi,Neta,deta]=BasisFunc(xi,eta,nx,ny,poly);

        dxi=dxi./2;             % Mapping of derivatives from
        deta=deta./2;           % parameter space to reference el.

        Nxi=Nxi(r:r+poly);      % Collect basis functions and
        dxi=dxi(r:r+poly);      % derivatives which support the el.
        Neta=Neta(s:s+poly);
        deta=deta(s:s+poly);

        [J,~]=Jacobian_(Nxi,dxi,Neta,deta,P,IEN_e,poly);

        % Physical coordinate of Gauss point
        y=0; k=1;
        for j=1:poly+1
            for i=1:poly+1
                y=y+Nxi(i)*Neta(j)*P(IEN_e(k),2);
                k=k+1;
            end
        end
```

```
        p=12*y; pv=[p p p p p]';              % Form normal traction vector
        q=6-(3/50)*y^2; qv=[-q -q -q -q -q]';% Form parabolic shear traction

        R(edgeDofxL)=R(edgeDofxL)+Neta'*Neta*pv(1:poly+1)*...
            sqrt(J(2,1)^2+J(2,2)^2)*W(g);
        R(edgeDofyL)=R(edgeDofyL)+Neta'*Neta*qv(1:poly+1)*...
            sqrt(J(2,1)^2+J(2,2)^2)*W(g);
    end
end

end
```

```
% This function forms global load vector for a parabolic shear traction
% (sum = 80) at right hand side
%
% Output:   R - global load vector
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of NURBS control points
%           R - empty global load vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order

function R=FormREndLoadedBeamR_(IEN,P,R,nx,ny,ncp,poly)

[G,W]=GaussBoundary(poly);      % Call Gauss boundary points and weights

xi=nx-1/1e12;                   % Basis functions are defined for values
                                % up to nx, xi=nx -> singular K
r=nx;
eta_v=(0:ny);                   % Vector defining parameter space

for s=1:ny
    e=nx*s;
    IEN_e=IEN(e,:);             % Element topology of current element
    IEN_ey=IEN_e+ncp;
    % Control points y direction at right hand side for each element
    edgeDofyR=IEN_ey((poly+1):(poly+1):(poly+1)^2);

    for g=1:size(G,1)           % For each Gauss boundary point
        eta_n=G(g);             % Gauss coord. reference element

        % Gauss coordinates in parameter space
        eta=eta_v(s)+(eta_n+1)*(eta_v(s+1)-eta_v(s))/2;

        [Nxi,dxi,Neta,deta]=BasisFunc(xi,eta,nx,ny,poly);

        dxi=dxi./2;             % Mapping of derivatives from
        deta=deta./2;           % parameter space to reference el.

        Nxi=Nxi(r:r+poly);      % Collect basis functions and
        dxi=dxi(r:r+poly);      % derivatives which support the el.
        Neta=Neta(s:s+poly);
        deta=deta(s:s+poly);

        [J,~]=Jacobian_(Nxi,dxi,Neta,deta,P,IEN_e,poly);

        % Physical coordinate of Gauss point
        y=0; k=1;
        for j=1:poly+1
            for i=1:poly+1
                y=y+Nxi(i)*Neta(j)*P(IEN_e(k),2);
                k=k+1;
            end
        end
```

```
        q=6-(3/50)*y^2; qv=[q q q q q]';     % Form parabolic shear traction

        R(edgeDofyR)=R(edgeDofyR)+Neta'*Neta*qv(1:poly+1)*...
            sqrt(J(2,1)^2+J(2,2)^2)*W(g);
    end

end
```

C.2.6 Form R Uniform Load R

This subfunction forms global load vector for a uniform load at the η boundary (on the right hand side), described in Section 5.1.1. The changes compared to its counterpart in Appendix B.4.7 are as mentioned in Appendix C.2.5.

Output:	`R`
Input:	`IEN`, `P`, `R`, `nx`, `ny`, `ncp`, `poly`, `q`
Subfunctions:	GaussBoundary.m, BasisFunc.m, Jacobian_.m

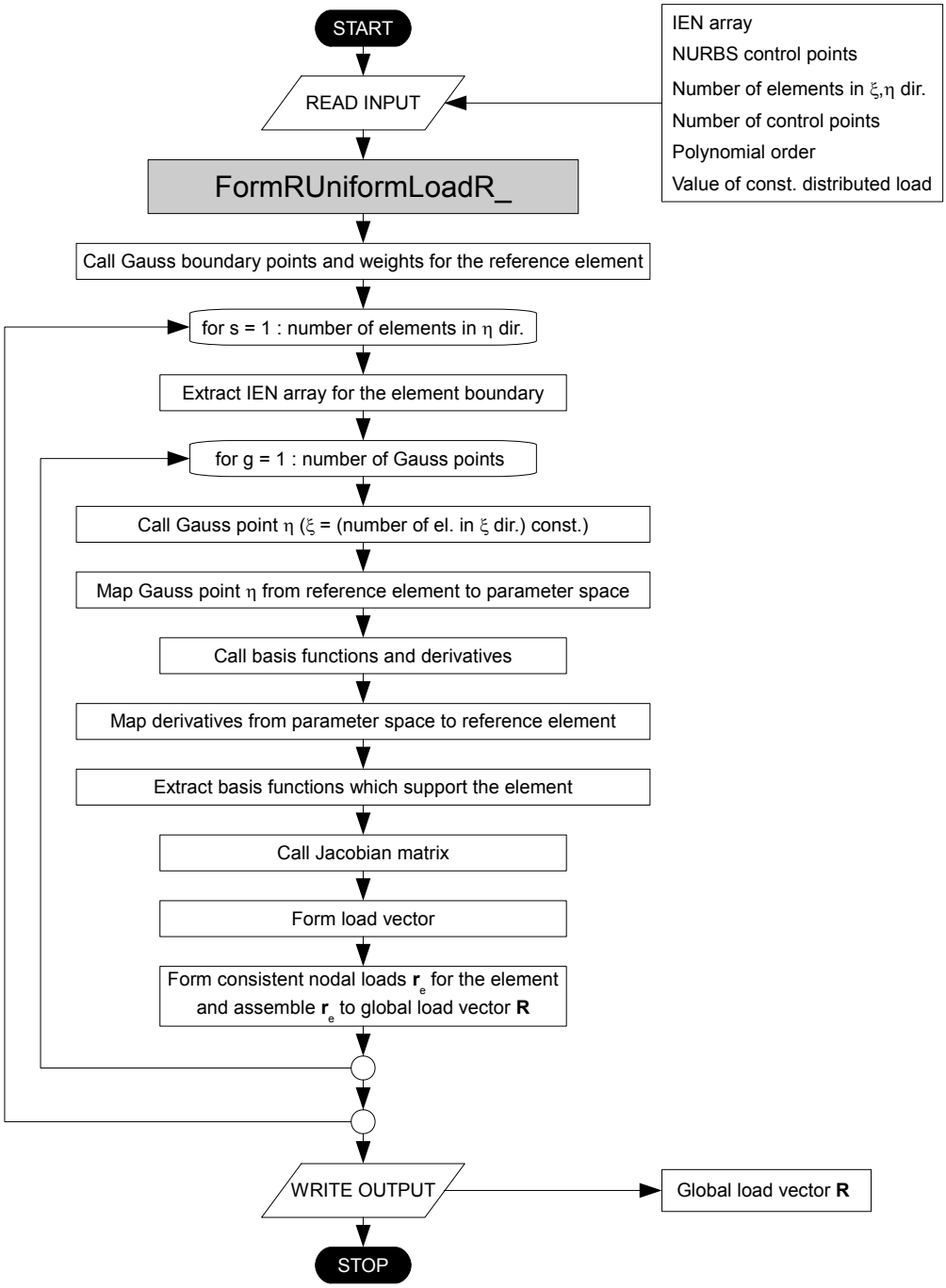


Figure C.8: Flow chart for FormRUniformLoadR_.m

```
% This function forms global load vector for a uniformly distributed shear
% traction at right hand side
%
% Output:   R - global load vector
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of control points
%           R - empty global load vector
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order
%           q - constant value of distributed load

function R=FormRUniformLoadR_(IEN,P,R,nx,ny,ncp,poly,q)

[G,W]=GaussBoundary(poly);      % Call Gauss boundary points and weights

xi=nx-1/1e12;                   % Basis functions are defined for values
                                % up to nx, xi=nx -> singular K
r=nx;
eta_v=(0:ny);                   % Vector defining parameter space

for s=1:ny
    e=nx*s;
    IEN_e=IEN(e,:);             % Element topology of current element
    IEN_ey=IEN_e+ncp;
    % Control points y direction at right hand side for each element
    edgeDofyR=IEN_ey((poly+1):(poly+1):(poly+1)^2);

    for g=1:size(G,1)           % For each Gauss boundary point
        eta_n=G(g);             % Gauss coord. reference element

        % Gauss coordinates in parameter space
        eta=eta_v(s)+(eta_n+1)*(eta_v(s+1)-eta_v(s))/2;

        [Nxi,dxi,Neta,deta]=BasisFunc(xi,eta,nx,ny,poly);

        dxi=dxi./2;             % Mapping of derivatives from
        deta=deta./2;           % parameter space to reference el.

        Nxi=Nxi(r:r+poly);      % Collect basis functions and
        dxi=dxi(r:r+poly);      % derivatives which support the el.
        Neta=Neta(s:s+poly);
        deta=deta(s:s+poly);

        [J,~]=Jacobian_(Nxi,dxi,Neta,deta,P,IEN_e,poly);

        qv=[q q q q q]';         % Form load vector, (poly+1)x1

        R(edgeDofyR)=R(edgeDofyR)+Neta'*Neta*qv(1:poly+1)*...
            sqrt(J(2,1)^2+J(2,2)^2)*W(g);
    end
end
end
```

C.2.7 Jacobian

This subfunction calculates the Jacobian matrix and physical derivatives on a general basis as described in Section 5.1.2. The differences compared to the Jacobian subfunction presented in Appendix B.4.15 is the double loop instead of the single loop, in addition to the evaluation of bivariate derivatives. The changes are due to that the B-spline basis functions in this program are given as univariate basis functions.

Output: `J`, `dxy`

Input: `Nxi`, `dxi`, `Neta`, `deta`, `P`, `IEN_e`, `poly`

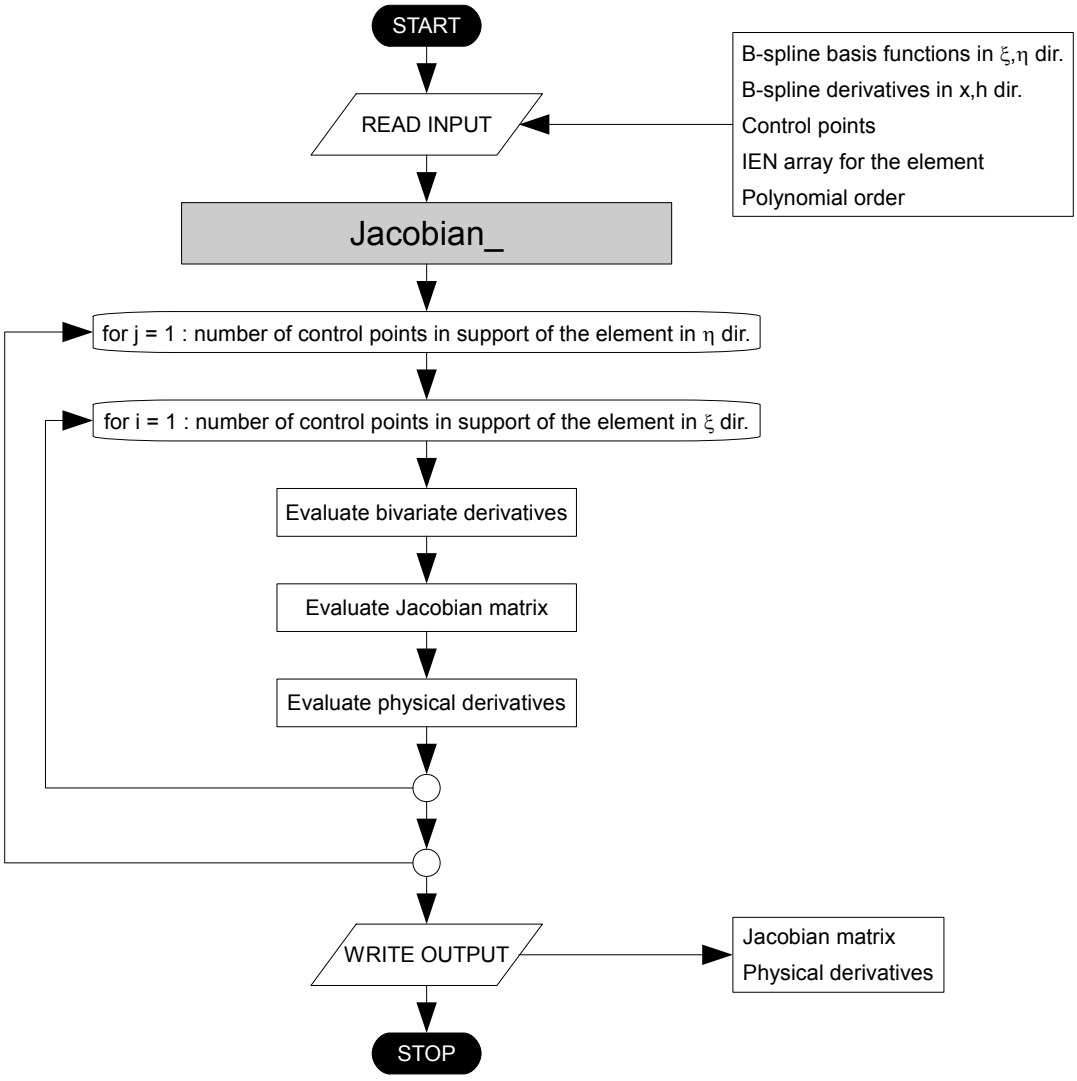


Figure C.9: Flow chart for Jacobian_.m

```
% This function calculates the Jacobian matrix and x,y derivatives
%
% Output:   J - Jacobian matrix
%           dxy - x,y derivatives
%
% Input:    Nxi - basis functions xi direction
%           dxi - derivatives of basis functions xi direction
%           Neta - basis functions eta direction
%           deta - derivatives of basis functions eta direction
%           P - coordinates of control points
%           IEN_e - element topology of the current element
%           poly - polynomial order

function [J,dxy]=Jacobian_(Nxi,dxi,Neta,deta,P,IEN_e,poly)

J=zeros(2,2);
dR=zeros(2,(poly+1)^2);

k=1;
for j=1:poly+1
    for i=1:poly+1
        dR(1,k)=dxi(i)*Neta(j);
        dR(2,k)=Nxi(i)*deta(j);

        % P(IEN_e(a),1) = x value of control points which support the el.,
        % P(IEN_e(a),2) = y value
        J(1,1)=J(1,1)+dR(1,k)*P(IEN_e(k),1);
        J(1,2)=J(1,2)+dR(1,k)*P(IEN_e(k),2);
        J(2,1)=J(2,1)+dR(2,k)*P(IEN_e(k),1);
        J(2,2)=J(2,2)+dR(2,k)*P(IEN_e(k),2);

        k=k+1;
    end
end

dxy=J\dR;

end
```

C.2.8 Stresses

This subfunction calculates strains according to Eq. (2.7) and stresses according to Eqs. (2.1) and (2.2) and prints out stresses σ_x, σ_y and τ_{xy} as described in Section 2.2.3. The changes compared to its counterpart in Appendix B.4.28 are as mentioned in Appendix C.2.3.

Output:	`stress`
Input:	`IEN`, `P`, `D`, `E`, `nx`, `ny`, `ncp`, `poly`
Subfunctions:	Gauss.m, BasisFunc.m, Jacobian_.m

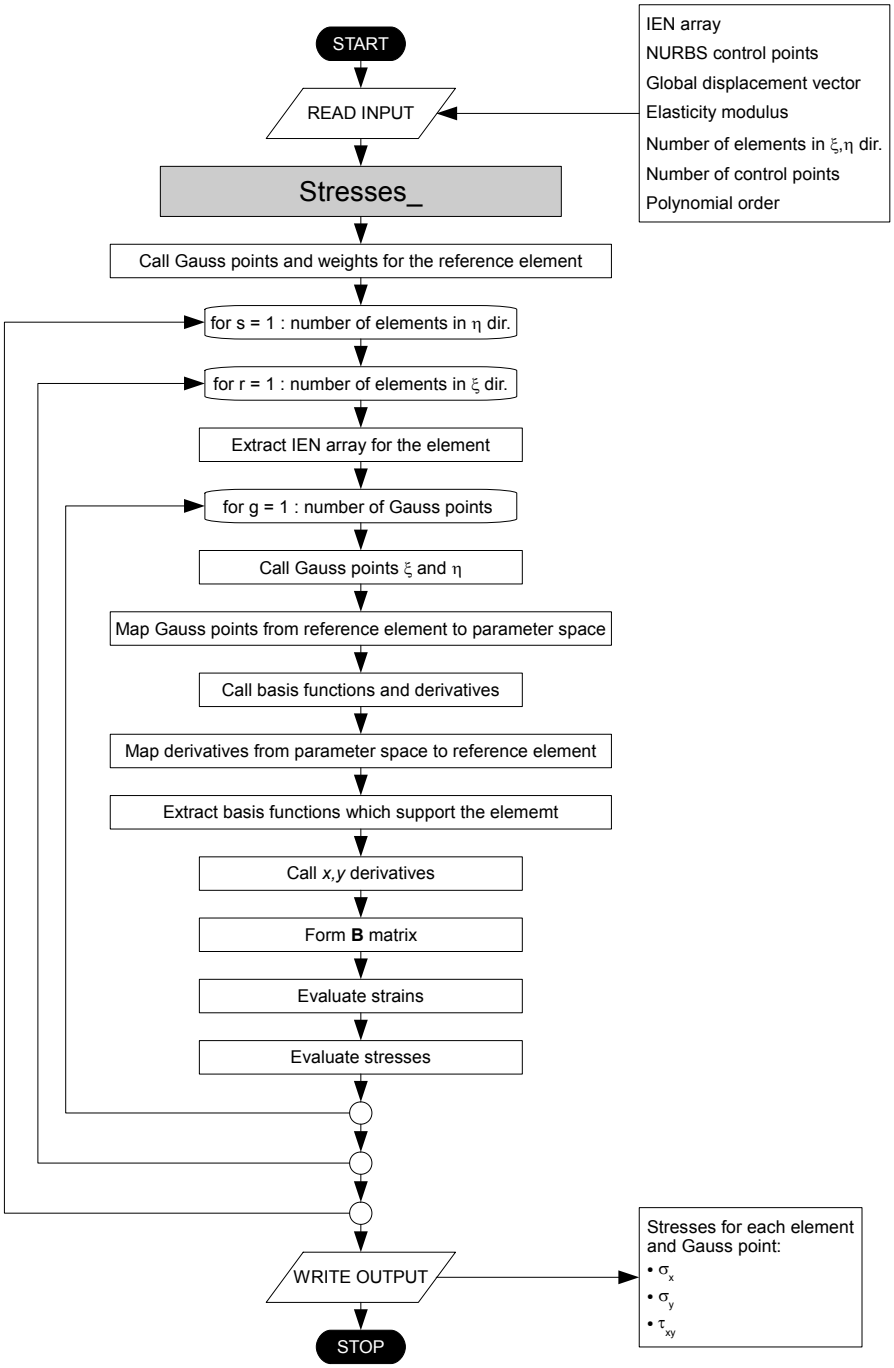


Figure C.10: Flow chart for Stresses_.m

```
% This function calculates strains and stresses (plane stress/strain)
%
% Output:   stress - given as three matrices, one for each stress component
%           stress(e,g,1)=sigma x
%           stress(e,g,2)=sigma y
%           stress(e,g,3)=tau xy
%           rows represent elements (e), columns represent Gauss points (g)
%
% Input:    IEN - element topology: numbering of control points
%           P - coordinates of control points
%           D - global displacement vector
%           E - constitutive matrix
%           nx - number of elements in xi direction
%           ny - number of elements in eta direction
%           ncp - number of control points
%           poly - polynomial order

function stress=Stresses_(IEN,P,D,E,nx,ny,ncp,poly)

[G,~]=Gauss(poly);                      % Call Gauss points

stress=zeros(nx*ny,size(G,1),3);

e=1;
xi_v=(0:nx);
eta_v=(0:ny);

for s=1:ny
    for r=1:nx
        IEN_e=nonzeros(IEN(e,:))';      % Element topology of current el.
        eDof=[IEN_e IEN_e+ncp];         % eDof: first x, then y

        for g=1:size(G,1)               % For each Gauss point
            xi_n=G(g,1);                % Gauss coord. reference element
            eta_n=G(g,2);               % Gauss coord. reference element

            % Gauss coordinates in parameter space
            xi=xi_v(r)+(xi_n+1)*(xi_v(r+1)-xi_v(r))/2;
            eta=eta_v(s)+(eta_n+1)*(eta_v(s+1)-eta_v(s))/2;

            [Nxi,dxi,Neta,deta]=BasisFunc(xi,eta,nx,ny,poly);

            dxi=dxi./2;                 % Mapping of derivatives from
            deta=deta./2;               % parameter space to reference el.

            Nxi=Nxi(r:r+poly);          % Collect basis functions and
            dxi=dxi(r:r+poly);          % derivatives which support the el.
            Neta=Neta(s:s+poly);
            deta=deta(s:s+poly);

            [~,dxy]=Jacobian_(Nxi,dxi,Neta,deta,P,IEN_e,poly);

            B=zeros(3,2*length(IEN_e));     % B matrix
            B(1,1:length(IEN_e))=dxy(1,:);
            B(2,length(IEN_e)+1:2*length(IEN_e))=dxy(2,:);
            B(3,1:length(IEN_e))=dxy(2,:);
```

```
            B(3,length(IEN_e)+1:2*length(IEN_e))=dxy(1,:);

            strains=B*D(eDof);          % Strains (3x1) in each Gauss point
            stress(e,g,:)=E*strains;
        end

        e=e+1;
    end
end

end
```

Appendix D

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

This article is written for MekIT'11 Sixth National Conference on Computational Mechanics, held in Trondheim May 23-24, 2011. Co-author is master student Ole Jørgen Fredheim, and advisors in the master thesis, Kjell Magne Mathisen and Kjetil André Johannessen, have also given their contribution. The article is also published in Nguyen et al. [18].

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 has 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

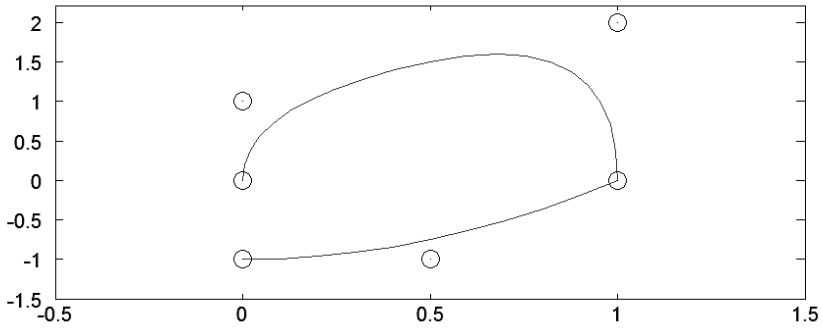


Figure 1: B-spline curve constructed from quadratic basis functions, and knot vector $\Xi = \{0, 0, 0, 1, 2, 2, 3, 3, 3\}$. The control points are marked as circles.

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.

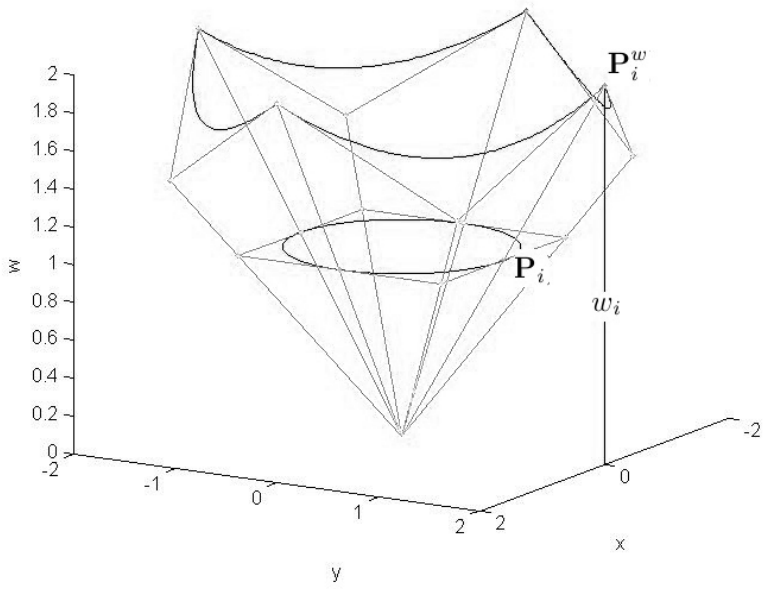


Figure 2: B-spline curve projected onto the plane $z = 1$ to create the NURBS representation of a circle.

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

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

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

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

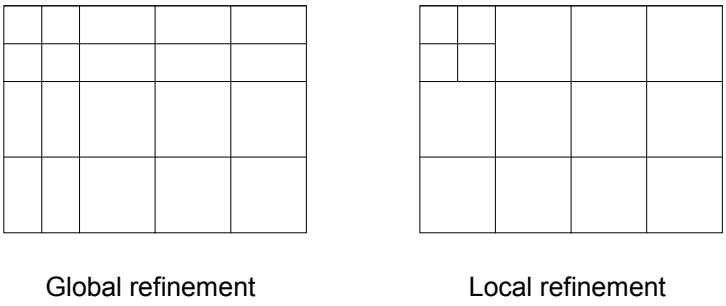


Figure 3: Global and local refinement.

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 (12).

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]

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

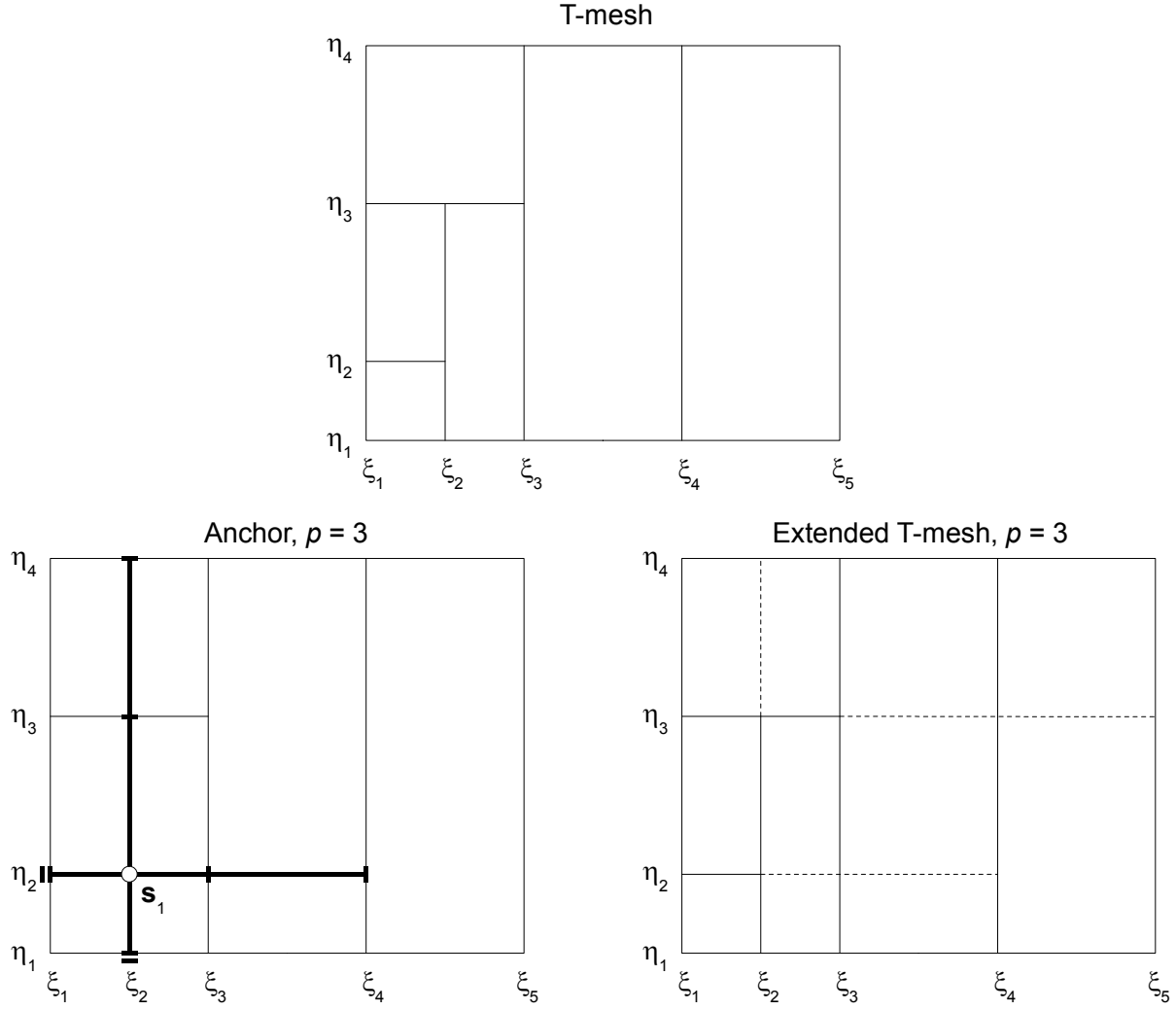


Figure 4: T-mesh, anchors of odd polynomial degrees ($p = q = 3$) and extended T-mesh, $p = 3$.

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

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

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

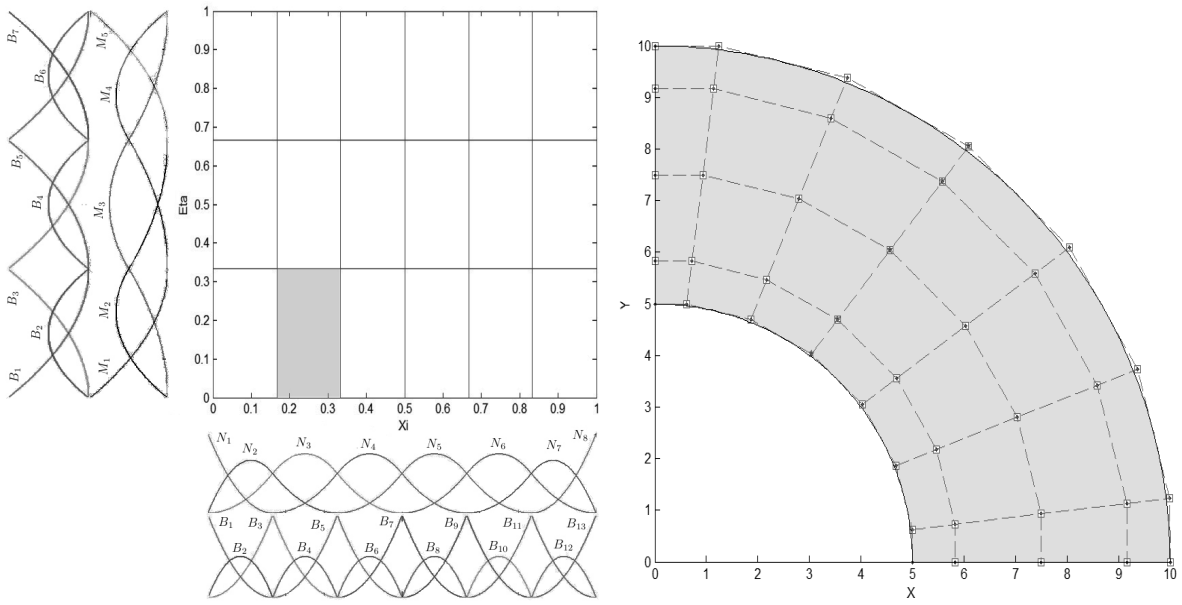
In order to increase our understanding of the Bézier decomposition we will take a closer look at a circular beam that is to be analysed. In the analysis we want to use quadratic NURBS basis functions, and we want an element mesh consisting of 6 elements in the tangential direction, and 3 elements in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

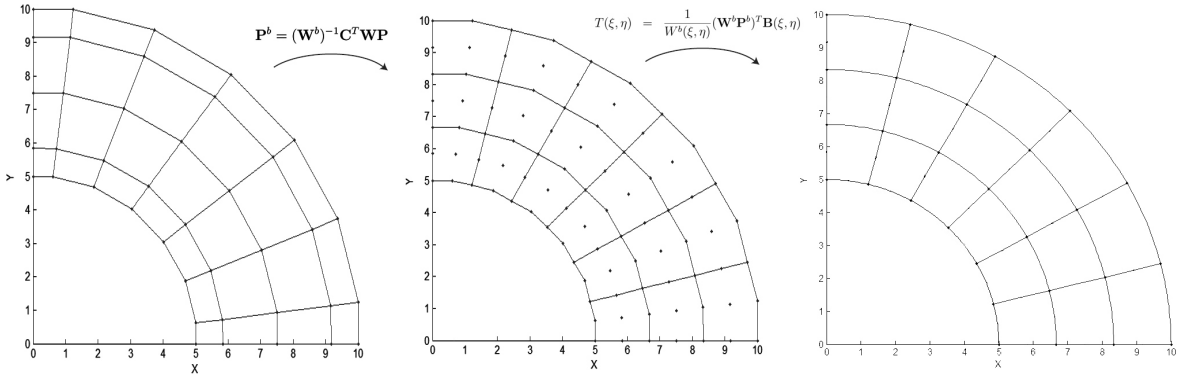


Figure 6: From control points to Bézier control points to Bézier physical mesh.

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 are 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 $\Xi = \{0, 0, 1, 2, 3\}$.

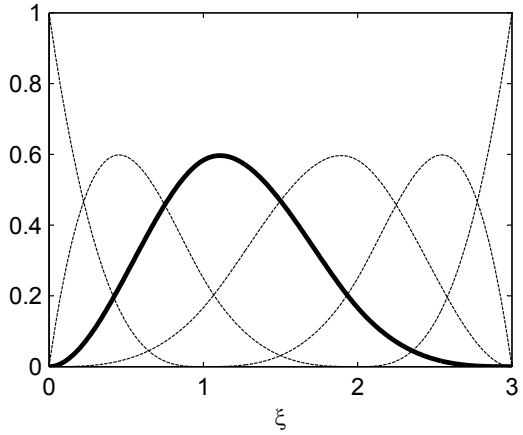


Figure 7: Basis function N_3 (thick solid line) to the local knot vector $\Xi = \{0, 0, 1, 2, 3\}$ and the additional basis functions (thin dotted lines) to the extended knot vector $\bar{\Xi} = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$.

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.

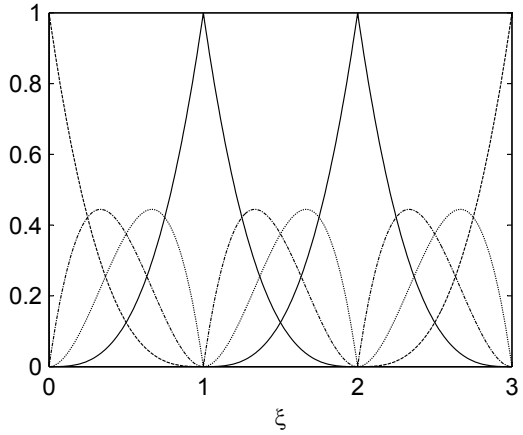


Figure 8: Basis functions of the Bézier elements after Bézier decomposition of $\bar{\Xi} = \{0, 0, 0, 0, 1, 2, 3, 3, 3, 3\}$.

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 circular 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 program 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 [10] with a traditional FEA. The error in the strain energy is defined as

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

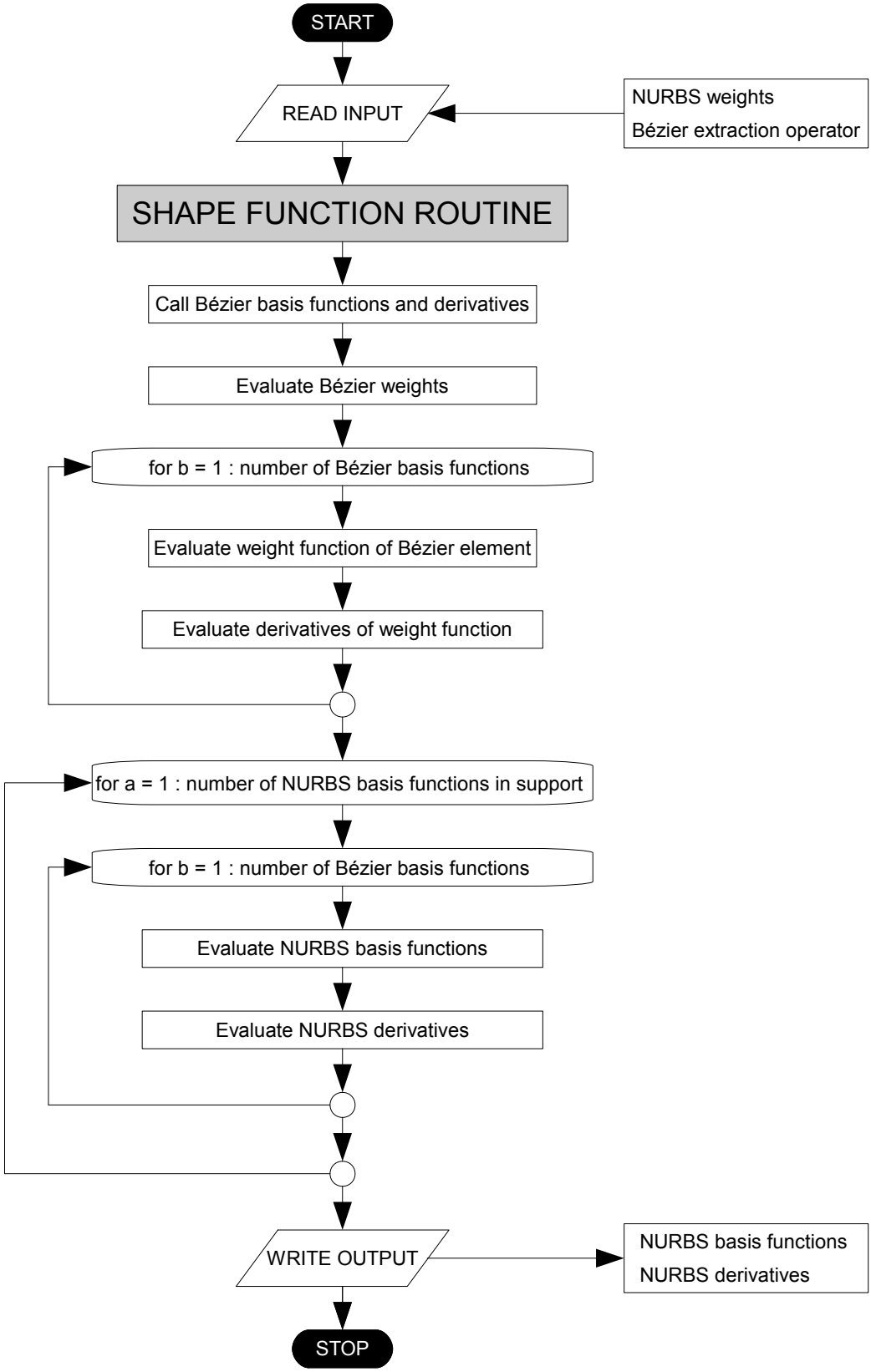


Figure 9: Flow chart of shape function routine for NURBS using Bernstein polynomials.

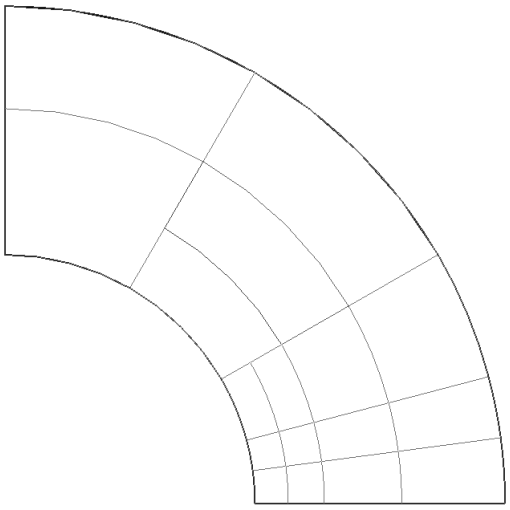


Figure 10: The extended T-mesh imported into the FE solver based on Bézier extraction of NURBS.

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 [9]

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

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

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

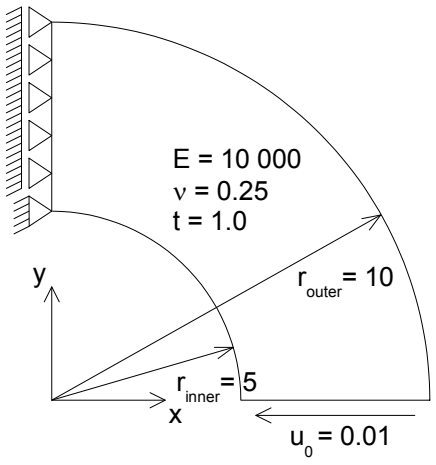
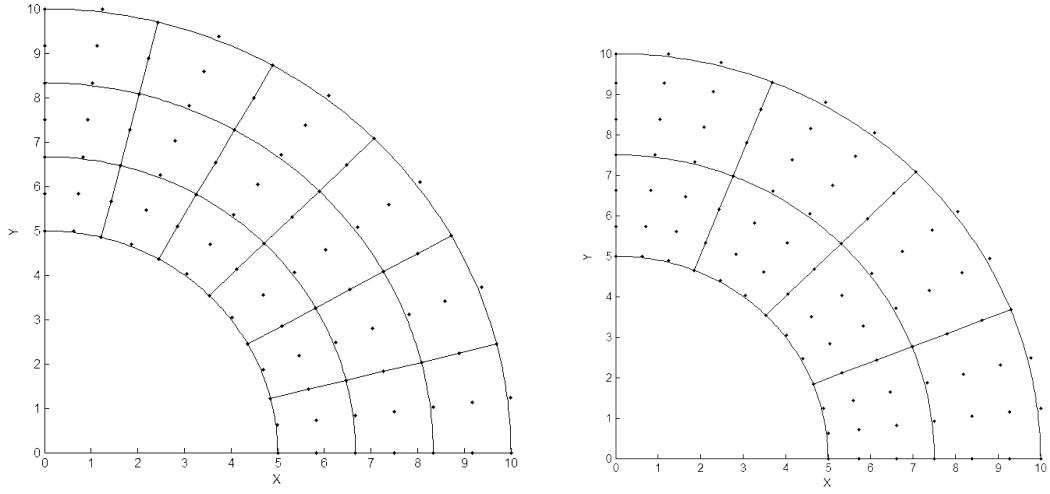


Figure 11: The geometry of the circular beam with end shear, with material properties, boundary conditions and prescribed displacements.



(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.

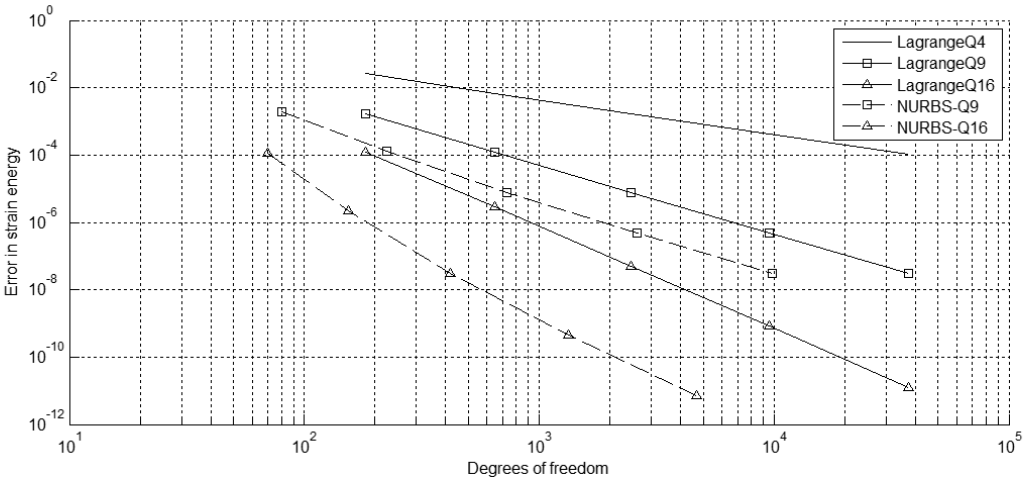
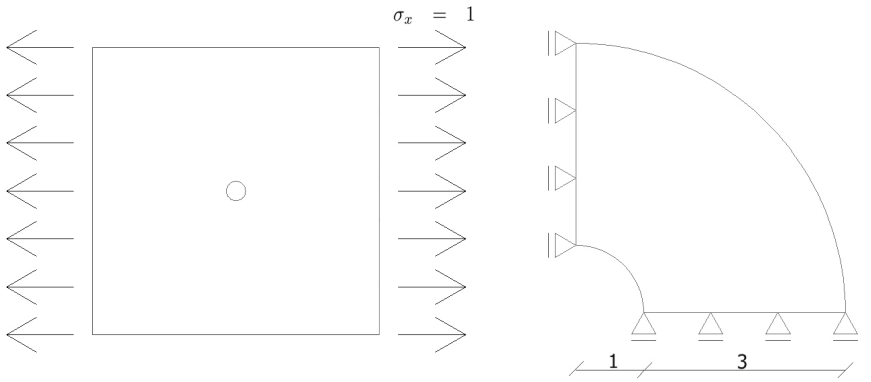


Figure 13: Error in strain energy vs. number of degrees of freedom for the circular beam.



(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.

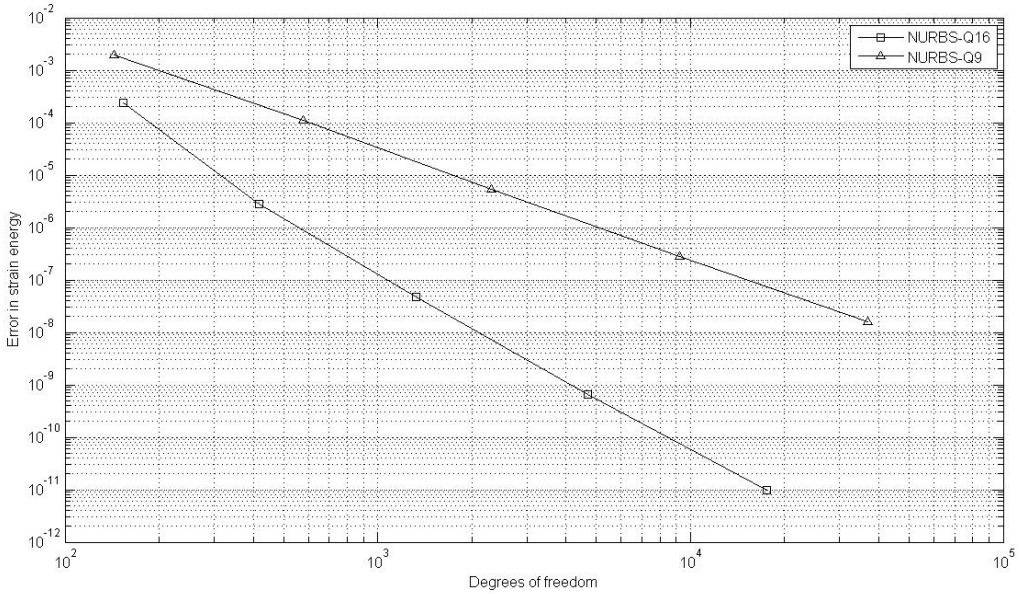


Figure 15: Error in strain energy vs. number of degrees of freedom for the infinite plate.

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

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. M. Calo, J. A. Cottrell, J. A. Evans, T. J. R. Hughes, S.Lipton, M. A. Scott and T. Sederberg Isogeometric Analysis using T-splines *Computer Methods in Applied Mechanics and Engineering*, **vol.199**, 229 – 263, 2010.

[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*, **vol.86**, 2011.

[3] M. G. Cox *The Numerical Evaluation of B-splines* Technical Report, National Physics Laboratory DNAC 4, 1971.

[4] C. deBoor On Calculation with B-splines *Journal of Approximation Theory*, **vol.6**, 50–62, 1972.

[5] T. J. R. Hughes, J. A. Cottrell and Y. Bazilevs Isogeometric Analysis: CAD, Finite Elements, NURBS, Exact Geometry and Mesh Refinement *Computer Methods in Applied Mechanics and Engineering*, **vol.194**, 4135–4195, 2005.

[6] J. A. Cottrell, T. J. R. Hughes and Y. Bazilevs *Isogeometric Analysis: Toward Integration of CAD and FEA* John Wiley & Sons, Chichester, England, 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*, **vol.86**, 2011.

[8] S. P. Timoshenko and J. N. Goodier Theory of Elasticity 3rd edition, McGraw-Hill, New York, 1970.

[9] 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**, 1331–1364, 1992.

[10] O. C. Zienkiewicz, R. L. Taylor and J. Z. Zhu *The Finite Element Method: Its Basis and Fundamentals* Elsevier Butterworth and Heinemann, Oxford, England, 6th edition, 2005.

Appendix E

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

This is a presentation of the article in Appendix D, prepared for MekIT'11 Sixth National Conference on Computational Mechanics, held in Trondheim May 23-24, 2011.

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

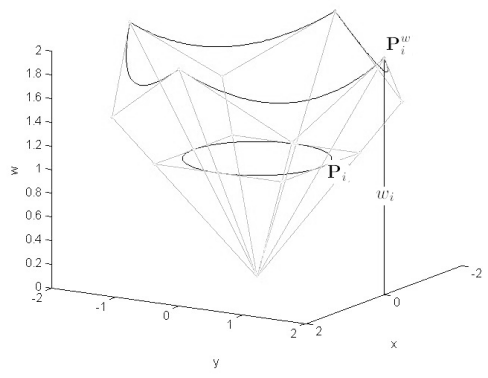
Motivation

- 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

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

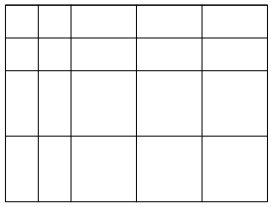
NURBS curve defined equivalently as B-spline curve

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

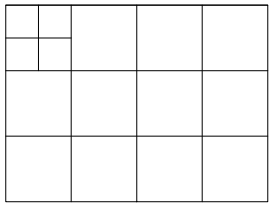
T-spline fundamentals 1

T-spline fundamentals:

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



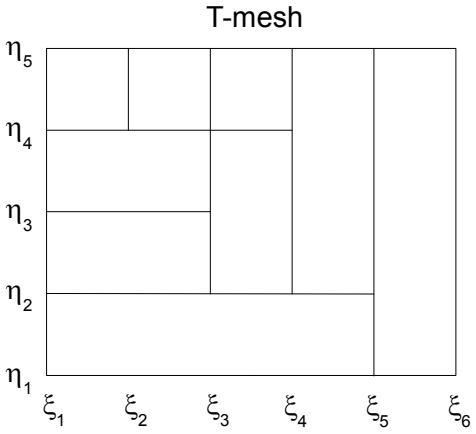
Global refinement



Local refinement

Example: A simple T-mesh

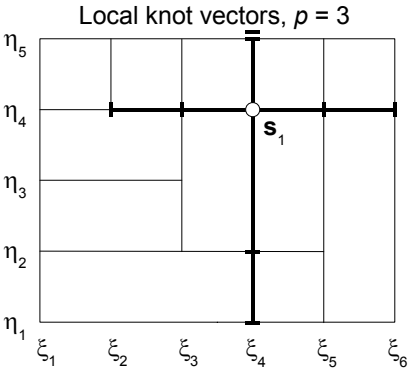
- Each knot line represents a knot value
- Incomplete knot lines terminates in *T-junctions*



T-spline fundamentals 2

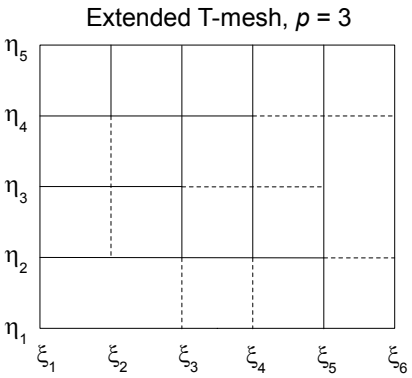
Local knot vectors:

- 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:

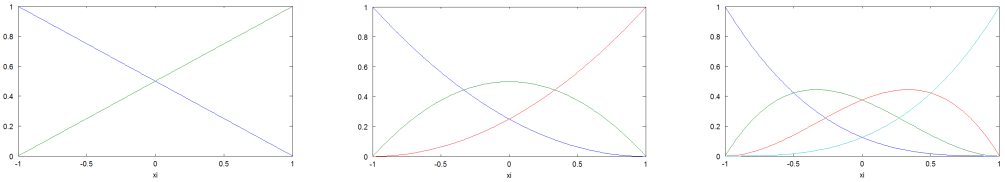
- *Lines of reduced continuity* define T-spline elements
- T-spline elements: regions over which T-spline basis functions are C^∞ continuous



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

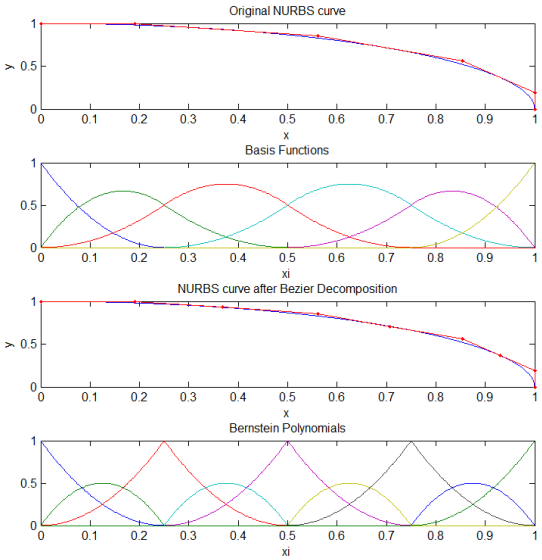
Bézier curves

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

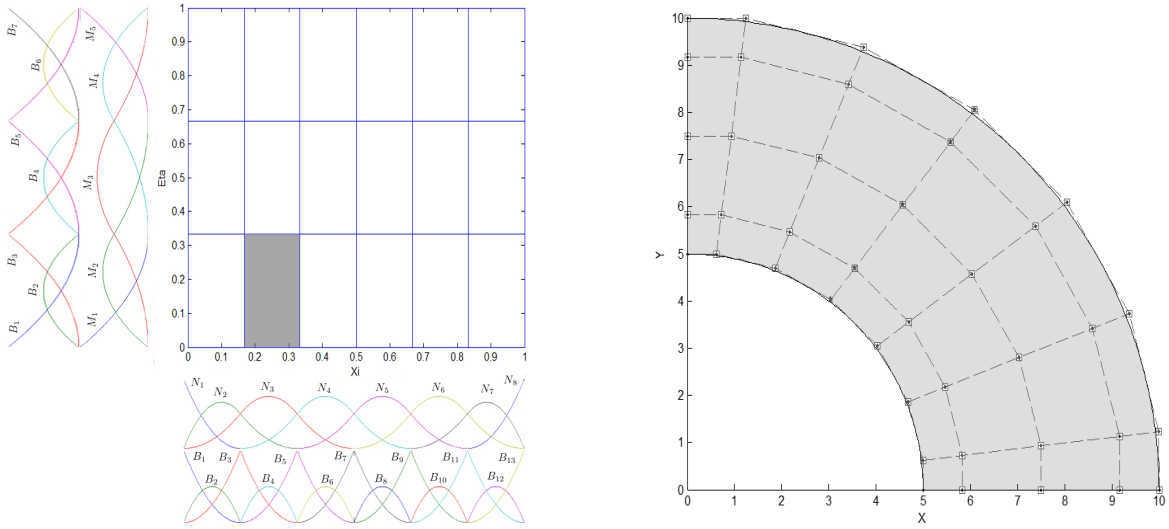
- $\mathbf{P}^b = (\mathbf{W}^b)^{-1}\mathbf{C}^T\mathbf{W}\mathbf{P}$
- $\mathbf{R}(\xi) = \mathbf{W}\mathbf{C}\frac{\mathbf{B}(\xi)}{W^b(\xi)}$

Example of Bézier decomposition 1

A circular beam

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

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

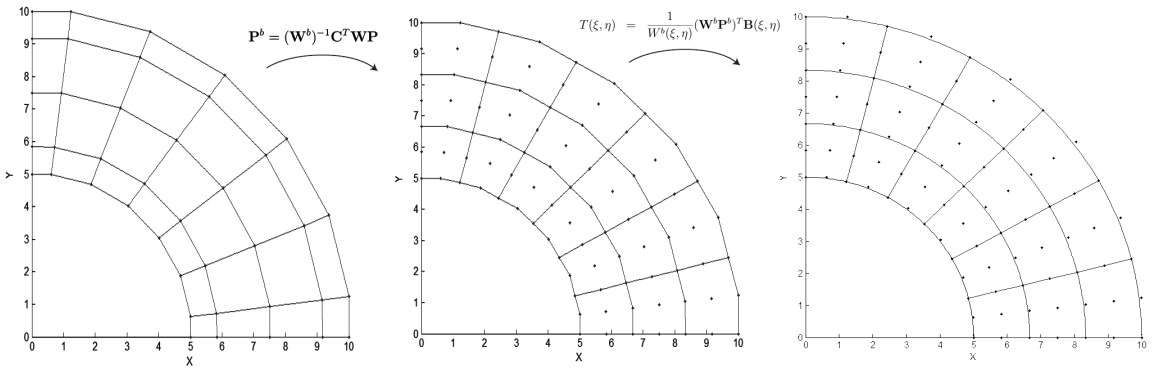


Example of Bézier decomposition 2

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

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

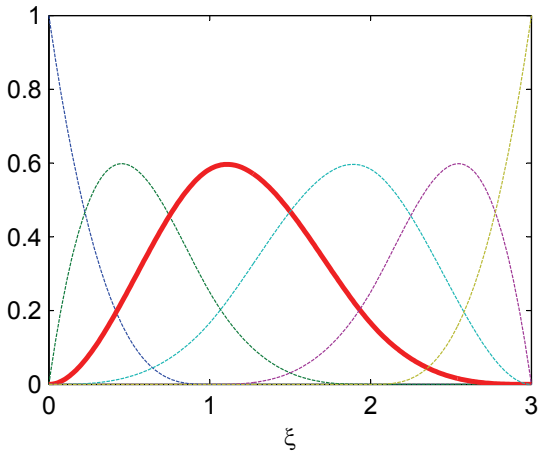
$$\begin{Bmatrix} M_1^1 \\ M_2^1 \\ M_3^1 \end{Bmatrix} = \begin{Bmatrix} M_1 \\ M_2 \\ M_3 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \end{Bmatrix} = \begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 1.0 & 0.5 \\ 0 & 0 & 0.5 \end{bmatrix}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



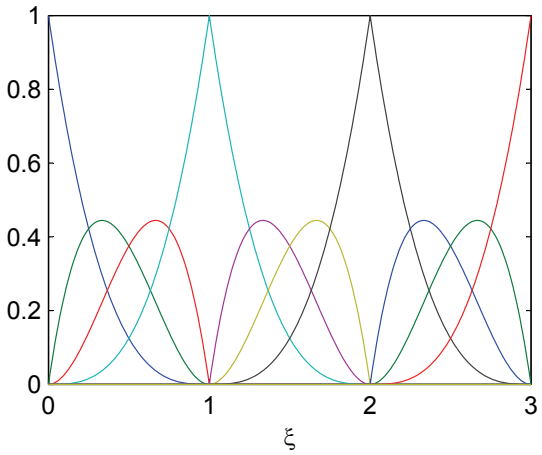
Bézier extraction of T-splines 1

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),
$\Xi = \{0, 0, 1, 2, 3\}$



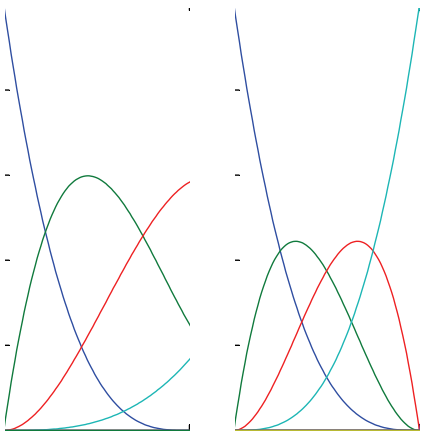
Bernstein polynomials

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

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

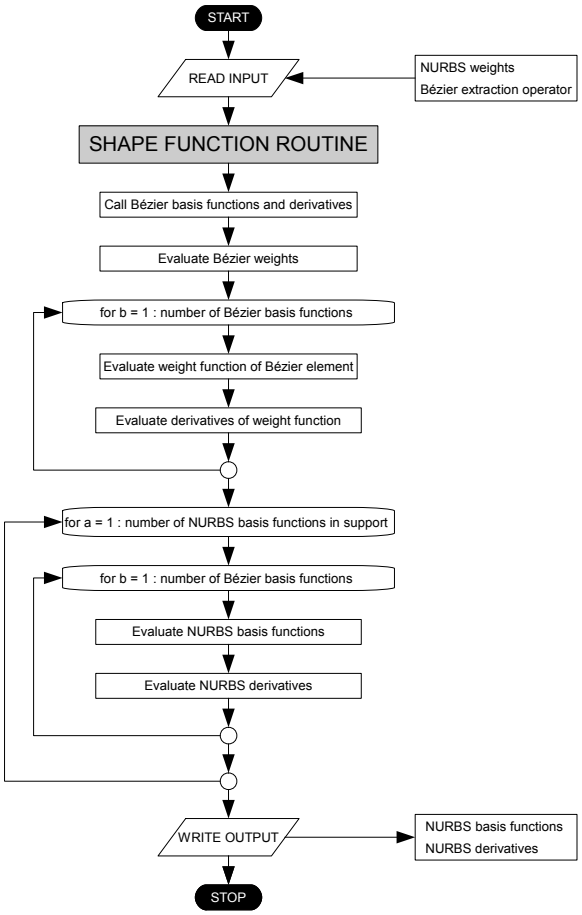


N_1, N_2, N_3, N_4 B_1, B_2, B_3, B_4

Implementation in a finite element solver 1

Isogeometric analysis based on Bézier extraction of NURBS:

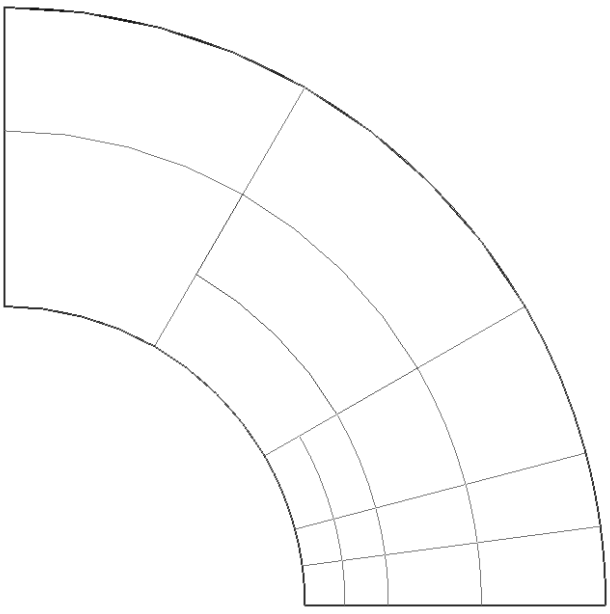
- Changes confined to shape function routine
- Extraction operators and Bézier basis functions are pre-calculated
- Output: NURBS basis functions and derivatives



Implementation in a finite element solver 2

Isogeometric analysis based on Bézier extraction of T-splines:

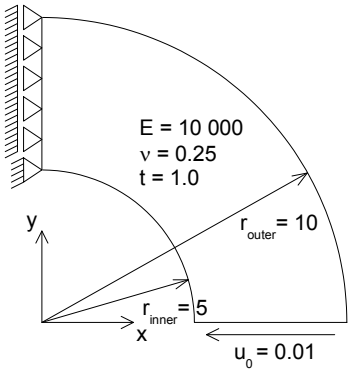
- Performed by importing a T-mesh from Rhinoceros into the FE solver
- Input: Control points and extraction operators
- A parsing script creates the IEN array for the extended T-mesh



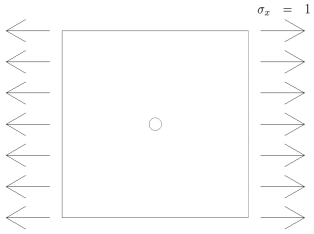
Numerical studies

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

- Circular beam with end shear

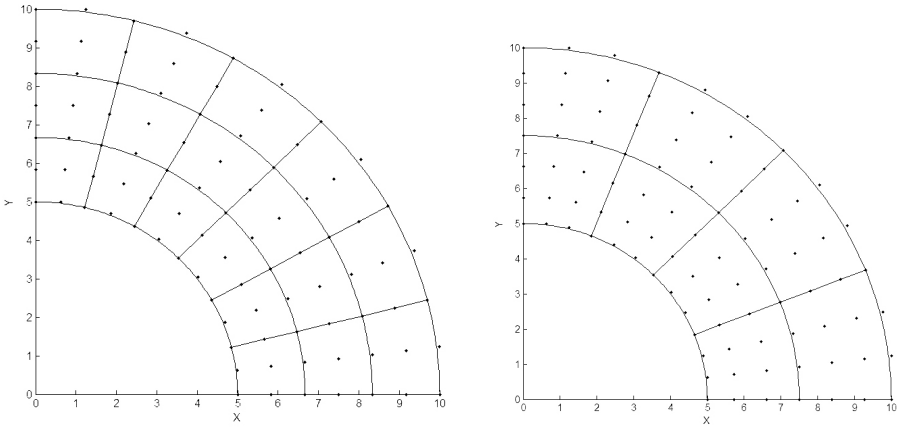


- Infinite plate with a circular hole under far-field uniaxial tension



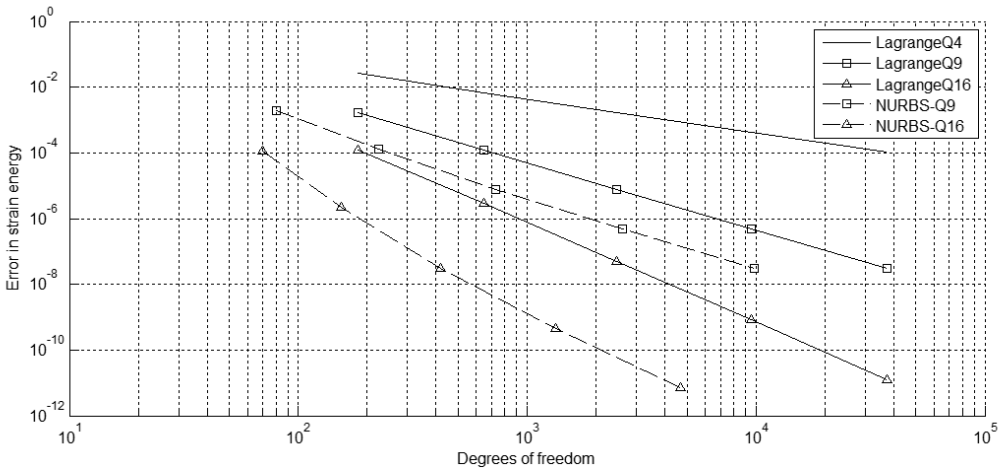
Circular beam with end shear 1

- 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

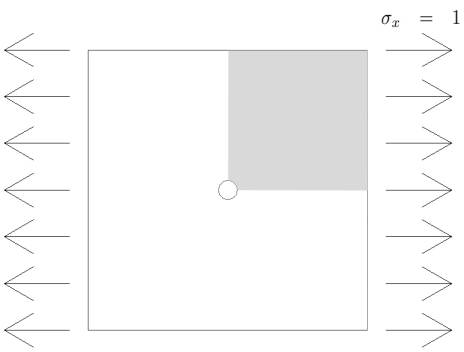
Circular beam with end shear 2

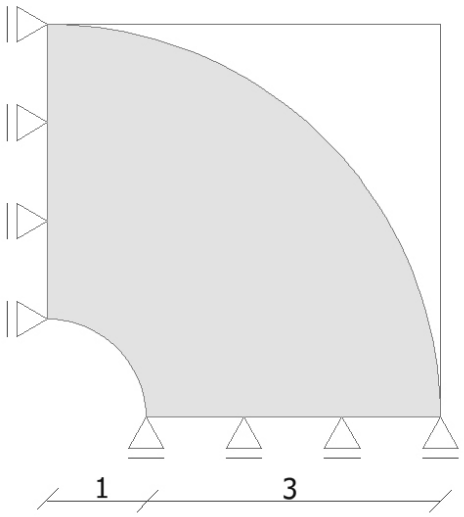


NURBS perform better than Lagrange

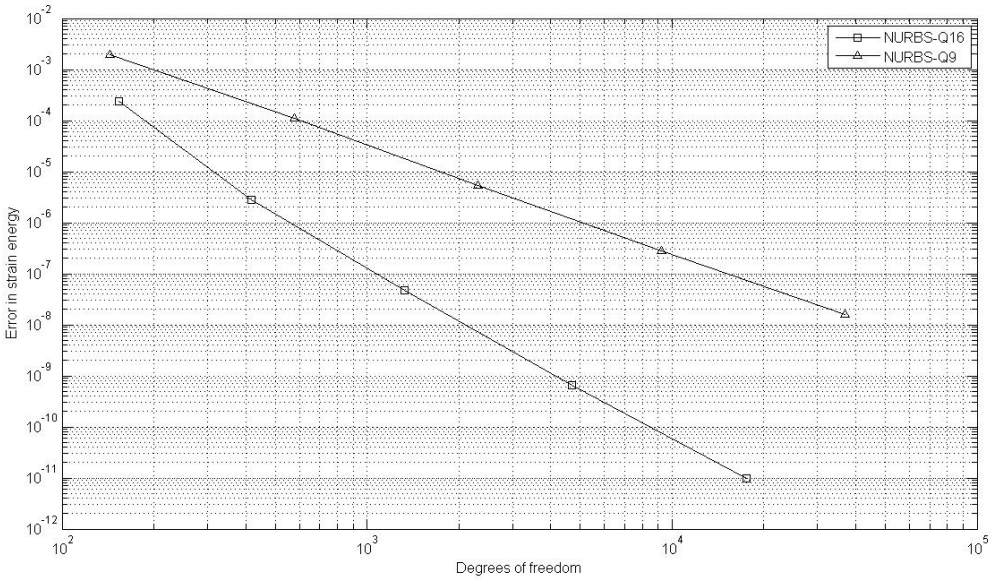
Infinite plate with a circular hole under far-field uniaxial tension 1

- 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