Ε Θ Ν Ι Κ Ο Μ Ε Τ Σ Ο Β Ι Ο Π Ο Λ Υ Τ Ε Χ Ν Ε Ι Ο

Δ.Π.Μ.Σ: ΜΑΘΗΜΑΤΙΚΗ ΠΡΟΤΥΠΟΙΗΣΗ ΣΤΙΣ ΣΥΓΧΡΟΝΕΣ ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΤΗΝ ΟΙΚΟΝΟΜΙΑ

ΕΠΙΛΥΣΗ ΜΕΡΙΚΩΝ ΔΙΑΦΟΡΙΚΩΝ ΕΞΙΣΩΣΕΩΝ ΜΕ ΤΗΝ ΧΡΗΣΗ ΤΗΣ ΜΕΘΟΔΟΥ ΙΣΟΓΕΩΜΕΤΡΙΚΗΣ ΑΝΑΛΥΣΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξης Ε.Παπαγιαννόπουλος

Επιβλέπων: Παναγιώτης Κακλής

Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2011

Ε θ ν ι κ ο Μ ε τ σ ο β ι ο Π ο λ υ τ ε χ ν ε ι ο

Δ.Π.Μ.Σ: ΜΑΘΗΜΑΤΙΚΗ ΠΡΟΤΥΠΟΙΗΣΗ ΣΤΙΣ ΣΥΓΧΡΟΝΕΣ ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΙ ΤΗΝ ΟΙΚΟΝΟΜΙΑ

ΕΠΙΛΥΣΗ ΜΕΡΙΚΩΝ ΔΙΑΦΟΡΙΚΩΝ ΕΞΙΣΩΣΕΩΝ ΜΕ ΤΗΝ ΧΡΗΣΗ ΤΗΣ ΜΕΘΟΔΟΥ ΙΣΟΓΕΩΜΕΤΡΙΚΗΣ ΑΝΑΛΥΣΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξης Ε.Παπαγιαννόπουλος

Επιβλέπων: Παναγιώτης Κακλής

Καθηγητής Ε.Μ.Π

Εγκρίθηκε απο την τριμελή εξεταστική επιτροπή την 18η Ιουλίου 2011

....	....	....
Π.Κακλής	Κ.Πολίτης	Α.Γκίνης
Καθηγητής Ε.Μ.Π	Αν.Καθηγητής ΤΕΙ Αθήνας	Επ.Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2011

....

Αλέξης Ε. Παπαγιαννόπουλος

Διπλωματούχος Μαθηματικός Εφαρμογών

Περίληψη

Η παρούσα εργασία έχει ως στόχο να εξετάσει το θέμα της ισογεωμετρικής ανάλυσης.Η ισογεωμετρική ανάλυση είναι μια πρόσφατη υπολογιστική προσέγγιση που προσφέρει την δυνατότητα της "ενσωμάτωσης" της μεθόδου πεπερασμένων στοιχείων με εργαλεία που προσφέρει το CAD(σχεδιασμός υποβοηθούμενος από υπολογιστή)

Το κεφάλαιο 1 ξεκινάει με τον ορισμό των συναρτήσεων B-spline, απο τις οποίες κατασκευάζονται οι συναρτήσεις NURBS.Παρουσιάζεται μια αναλυτική περιγραφή των γεωμετρικών ιδοτήτων τους μαζί με κάποια παραδείγματα. Στην συνέχεια ορίζουμε τις συναρτήσεις NURBS και εξηγούμε την σύνδεση τους με τις B-spline.Τέλος παρουσίαζονται οι κλασικές στρατηγικές εκλέπτυνσης η h και p , ενώ εξηγείται και η καινούργια k εκλέπτυνση.

Στο κεφάλαιο 2 εξετάζεται η χρήση του CAD σε πλαίσια ανάλυσης.Ξεκινάει με την μελέτη ενός κλασικού προβλήματος συνοριακών τιμών Poisson, το οποίο και λύνεται με την μέθοδο των πεπερασμένων στοιχείων του Galerkin .Ακολούθως παρουσιάζεται η ισοπαραμετρική προσέγγιση για να διαλευκανθούν οι ομοιότητες και οι διαφορές ανάμεσα στα κλασικά πεπερασμένα στοιχεία, όπως τα πολυώνυμα, με τα πεπερασμένα στοιχεία που βασίζονται σε συναρτήσεις NURBS.Τέλος, παρουσιάζονται εκτημήτριες σφάλματος κατά την h στρατηγική εκλέπτυνσης.

Το τελευταίο κεφάλαιο, εξηγεί τις δομές δεδομένων του GeoPDEs , ένα περιβάλλον συμβατό με MATLAB που αναπτύχθηκε στο Πανεπιστήμιο της Πάβια και το πολυτεχνείο του Μιλάνο, για να δοκιμαστεί η ισογεωμετρική προσέγγιση στην επίλυση ελλειπτικών προβλημάτων 2 διαστάσεων .Τα αποτελέσματα εξετάζουν κυρίως τις στρατηγικές εκλέπτυνσης .

Λέξεις κλειδιά: NURBS , Ισογεωμετρική ανάλυση, Ανάλυση πεπερασμένων στοιχείων, hpk-εκλέπτυνση, GEOpdes

Abstract

This thesis aims to examine the subject of Isogeometric analysis . **Isogeometric analysis (IGA** for short) is a recently developed computational approach that offers the possibility of integrating finite element analysis (FEA) into conventional NURBS(Non Rational uniform Basic Splines) based CAD design tools.

Chapter 1 starts with the definition of B-Splines , from which NURBS are built. An analytic description of their geometrical properties along with some examples are given . Next , we give the definition of NURBS and explain their relation with B-Splines. Finally , classic h and p refinement strategies are presented as well as the new k refinement is explained .

Chapter 2 explains how CAD can be used within an analysis framework. It starts by imposing a classic Poisson boundary value problem , which is then solved using the Galerkin finite element method . The isoparametric approach is then presented in order to clearly highlight the differences and similarities between classic finite elements, such as piecewise polynomials, and NURBS based finite elements. Finally we present error estimates during h refinement.

The final chapter, Chapter 3, explains the data structures of GEOpdes , a MATLAB compatible environment developed at IMATI, Università di Pavia and Politecnico di Milano, for testing the isogeometric approach in the context of NURBS-based finite element analysis. Some numerical examples on 2D elliptical problems are presented in order to obtain results mainly concerning refinement strategies.

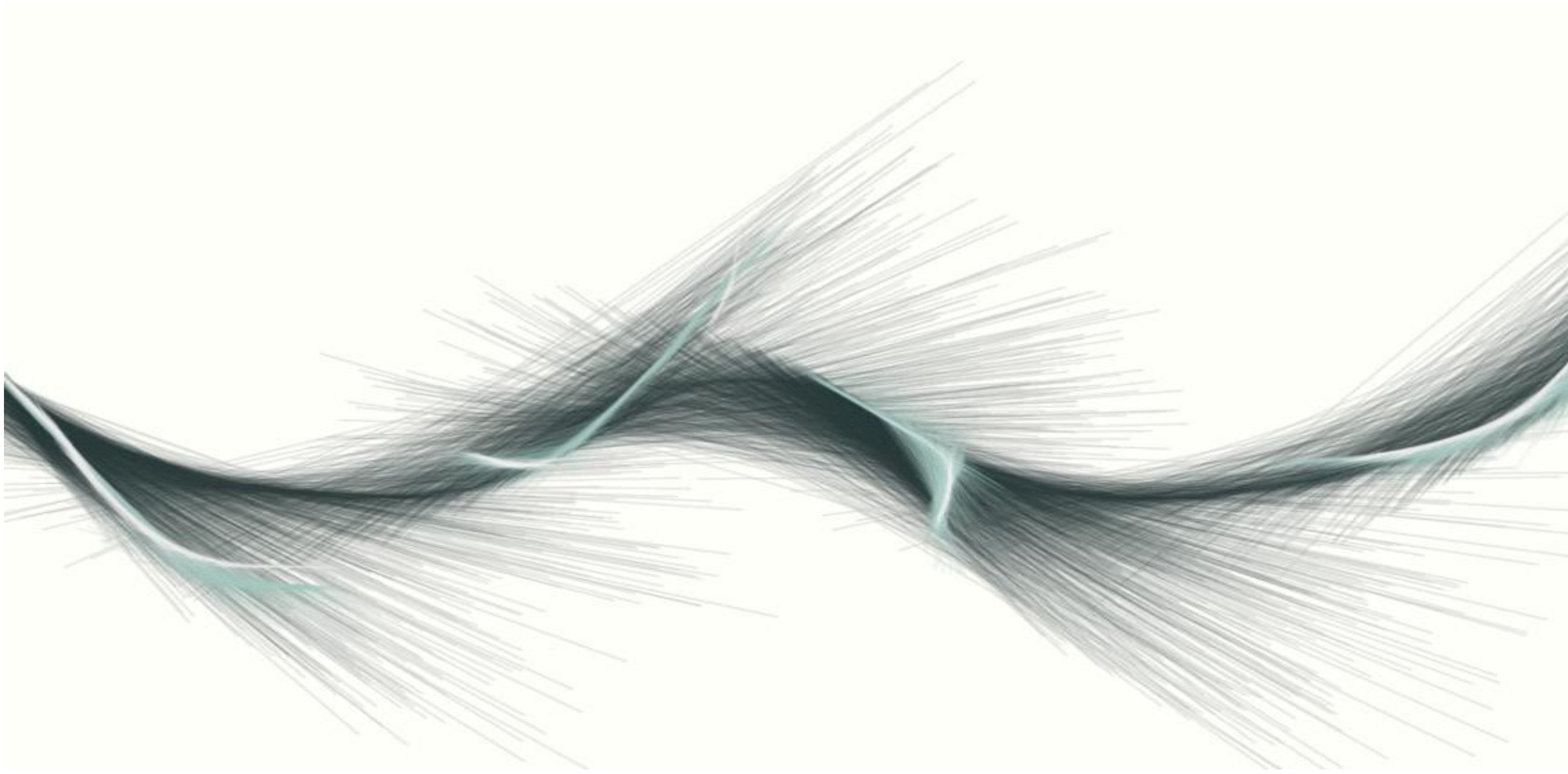
Key words: NURBS , Isogeometric analysis, Finite element analysis, hpk-refinement, GEOpdes

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τον Καθηγητή της σχολής Ναυπήγων Μηχανολόγων μηχανικών Παναγιώτη Κακλή για την ανάθεση της παρούσας διπλωματικής , την υπομονή που επέδειξε και την υποστήριξη κατα την διάρκεια της εκπόνησης της.

Θα ήθελα επίσης να ευχαριστήσω τον Αναπληρωτή καθηγητή Κωνσταντίνο Πολίτη καιθώς και τον Λέκτορα Αλέξανδρο Γκίνη για την πολύτιμη συμβολή τους στην επίλυση τεχνικών προβλημάτων που ανέκειψαν κατα την διάρκεια της υλοποίησης και για την άριστη συνεργασία που είχαμε.

Alexis E. Papagiannopoulos

ISOGEOMETRIC ANALYSIS



Athens,2011

Table of Contents

Chapter 1: NURBS As Tool For Geometric Design

Non-Uniform ***R****ational* ***B-Splines***, commonly referred to as ***NURBS***, have become the de facto industry standard for representation, design and data exchange of geometric information processed by computers. Many national and international standards recognize NURBS as powerful tools for geometric design. Until recently, B-spline curves and surfaces (NURBS) were principally of interest to the computer aided design community, where they have become the standard for curve and surface description. Today we are seeing expanded use of NURBS in modeling objects. The enormous success behind NURBS is largely due to the fact that:

- ✓ NURBS provide a unified mathematical basis for representing both analytic shapes, such as conic sections and quadric surfaces, as well as free-form entities, such as car bodies and ship hulls

- ✓ designing with NURBS is intuitive, almost every tool and algorithm has an easy to understand geometric interpretation

- ✓ NURBS algorithms are fast and numerically stable

- ✓ NURBS curves and surfaces are invariant under common geometric transformations, such as rotation, parallel and perspective projections

- ✓ NURBS are generalizations of non-rational B-splines and rational Bézier curves and surfaces

The excellent mathematical and algorithmic properties, combined with successful industrial applications, have contributed to the enormous popularity of NURBS. This first chapter examines NURBS as a tool for geometric design. A starting point of our examination is at B-splines, since NURBS are built from them.

1.1 Basis functions

Let $\Xi=\{\xi_1,\xi_2,....,\xi_m\}$ be a non decreasing sequence of real numbers, i.e. , $\xi_i\leq\xi_{i+1}$, $i=1,2...,m$. The ξ_i are called ***knots*** and Ξ is the ***knot vector***. If the knot vector is equally spaced then it is **uniform**, otherwise it is **non-uniform.** Knot values can be repeated, meaning that more than one knot can take on the same value. As we will see, this has an enormous impact on the properties of the basis. A knot vector is **open** if its first and last value appear $p+1$ times.

The *i*-th B-spline basis function of p-degree (order p+1), denoted by $N_{i,p}(\xi)$ is defined as :

$$N_{i,0}(\xi) = \begin{cases} 1\ , & if\ \xi_i \le \xi \le \xi_{i+1} \\ 0\ , & otherwise \end{cases} \qquad (1.1.1)$$

For p = 1, 2, 3, . . . they are defined by:

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \qquad (1.1.2)$$

Equation *(1.1.2)* is also known as ***Cox–de Boor recursion formula.*** The results of applying *(1.1.1)* and *(1.1.2)* to a uniform knot vector are presented in **Fig. 1.1** .

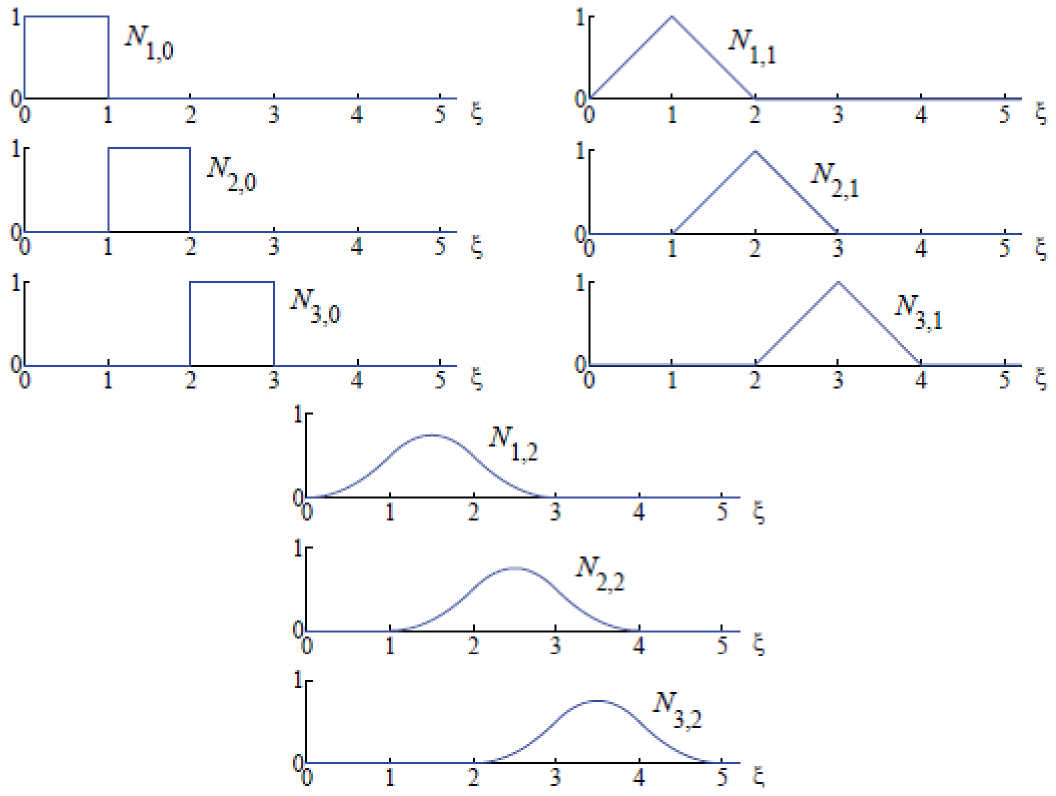


Fig. 1.1

For *p=0* and *p=1*, the B-splines functions are the same with piecewise constants and linear functions in **Finite Elements Analysis** (***FEA***). However, quadratic B-spline functions differ from the quadratic finite elements: even though they are identical, they are shifted relative to each other. The shape of quadratic finite elements exclusively depends on whether it corresponds for an internal node or an end node. This stands for all higher order B-splines.

We now list a number of important properties of the B-spline functions. As we will see, it is these properties which determine the many desirable geometric characteristics in B-spline curves and surfaces:

I. The first is that the basis constitutes a partition of unity, that is, $\forall \xi$,

$$\sum_{i=1}^{n} N_{i,p}(\xi) = 1 \tag{1.1.3}$$

This property also applies for an arbitrary knot span, $[\xi_i,\xi_{i+1})$, since $\sum_{j=i-p}^{i} N_{j,p}(\xi) = 1$ for all $\xi \in [\xi_i,\xi_{i+1})$. To prove this, consider

$$\sum_{j=i-p}^{i} N_{j,p}(\xi) = \sum_{j=i-p}^{i} \frac{\xi-\xi_j}{\xi_{j+p}-\xi_j} N_{j,p-1}(\xi) + \sum_{j=i-p}^{i} \frac{\xi_{j+p+1}-\xi}{\xi_{j+p+1}-\xi_{j+1}} N_{j+1,p-1}(\xi) .$$

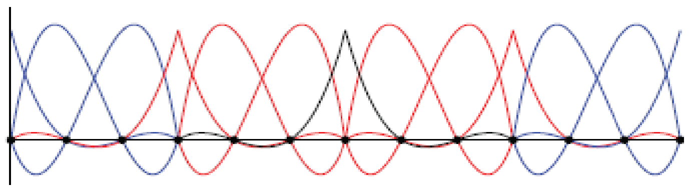
Changing the summation variable in the second sum from *(i-p)* to *(i-p+1)* and considering that $N_{i-p,p-1}(\xi)=N_{i+1,p-1}(\xi)=0$, we have:

$$\sum_{j=i-p}^{i} N_{j,p}(\xi) = \sum_{j=i-p}^{i} \left[\frac{\xi-\xi_j}{\xi_{j+p}-\xi_j} + \frac{\xi_{j+p+1}-\xi}{\xi_{j+p+1}-\xi_{j+1}} \right] N_{j,p-1} = \sum_{j=i-p+1}^{i} N_{j,p-1}(\xi) .$$

Applying the same concept recursively yields

$$\sum_{j=i-p}^{i} N_{j,p}(\xi) = \sum_{j=i-p+1}^{i} N_{j,p-1}(\xi) = ... = \sum_{j=i}^{i} N_{j,0}(\xi) = 1 .$$

II. $N_{i,p}(\xi)=0$, if ξ is outside the interval $[\xi_i,\xi_{i+p+1})$ (**local support property**), meaning that the support of the B-spline functions is *p+1* knot spans. Classical FEA functions have support over much less portions, leading to the misconception that the increasing support of the B-spline functions leads to increased bandwidth in a numerical method. As we see in **Fig. 1.2**, for cubics (*p*=*3*), the total number of functions that any given function shares support with (including itself) is *2p+1*, either we are using a FEA basis or B-splines.


(a) Standard cubic finite element basis functions with equally spaced nodes

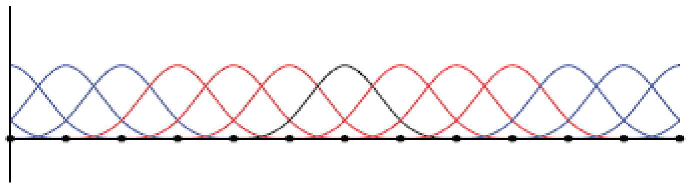

(b) Cubic B-spline basis functions with equally spaced knots

Fig. 1.2

III. $N_{i,p} \geq 0$ for all i, p and ξ (***non negativity***). This is proven by induction on p. It is clearly true for p=0; assume it is true for $p-1$, $p \geq 0$ with i and ξ arbitrary. By property **II.**, $N_{i,p-1}(\xi)=0$ if $\xi \notin [\xi_i, \xi_{i+p+1})$. But $\xi \in [\xi_i, \xi_{i+p+1})$ implies that $\frac{\xi - \xi_i}{\xi_{i+p} - \xi_i}$ is non-negative. By assumption, $N_{i,p}$ is non-negative and thus, the first term of equation *(1.1.2)*. The same are true for the second term and hence the $N_{i,p}(\xi)$ *are non-negative.*

IV. All derivatives of $N_{i,p}(\xi)$ exist on the interior of a knot span. At a knot, $N_{i,p}(\xi)$ is p-k times continuously differentiable, where k is the multiplicity of the knot. Hence increasing degree increases continuity while increasing knot multiplicity decreases continuity.

1.1.2 Derivatives of B-spline basis functions

We can prove by induction on p that for a given polynomial order p and knot vector Ξ, the derivative of the i-th basis function is given, in terms of lower order B-spline functions, by:

$$\frac{d}{d\xi} N_{i,p} = \frac{p}{\xi_{i+1} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \qquad (1.1.4)$$

Now let $N_{i,p}^{(k)}$ denote the k-th derivative of $N_{i,p}(\xi)$. Repeated differentiation of *(1.1.4)* produces the general formula:

$$\frac{d^k}{d\xi^k}N_{i,p}(\xi)=\frac{p}{\xi_{i+1}-\xi_\iota}\left(\frac{d^{k-1}}{d^{k-1}\xi}N_{i,p-1}(\xi)\right)-\frac{p}{\xi_{i+p+1}-\xi_{i+1}}\left(\frac{d^{k-1}}{d^{k-1}\xi}N_{i+1,p-1}(\xi)\right) \qquad (1.1.5)$$

Equation *(1.1.6)* is another generalization of equation *(1.1.5)*. It computes the *k*-th derivative of $N_{i,p}^{(k)}$ in terms of lower order functions $N_{i,p-k}, \ldots, N_{i+k,p-k}$. We have :

$$\frac{d^k}{d\xi^k}N_{i,p}(\xi)=\frac{p!}{(p-k)!}\sum_{j=0}^{k}a_{j.k}N_{i+1,p-k}(\xi) \qquad (1.1.6)$$

with

$$\alpha_{0,0}=1$$

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

$$\alpha_{k,j}=\frac{a_{k-1,j}-a_{k-1,j-1}}{\xi_{i+p+j-k+1}-\xi_{i+j}} \qquad j=1,\ldots,k-1$$

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

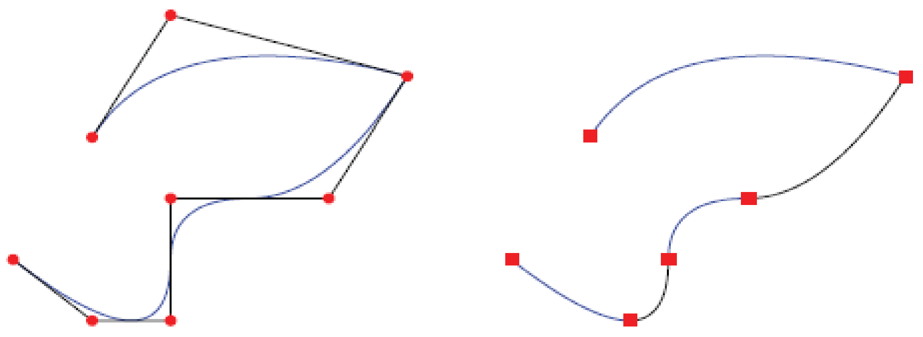
We should remark that *k* should not exceed *p*. Also, the denominators involving knot differences can become zero; the quotient is defined to be zero in this case.

1.2 B-spline curves

A *p*-th degree B-spline curve is defined by :

$$C(\xi)=\sum_{i=1}^{n}N_{i,p}(\xi)B_i \qquad (1.2.1)$$

where $\{B_i\}$ *are the* control points and the $\{N_{i,p}\}$ are the *p*-th degree B-spline basis functions defined on a knot vector Ξ. The control points are analogous to the nodal values in the way that they are coefficients of the basis, however the control points are non-interpolatory. The polygon formed by the $\{B_i\}$ is called ***control polygon***. All three, the degree, the number of control points $(n+1)$ and the number of knots $(m+1)$ are related by $m=n+p+1$. Unless stated otherwise, we assume $\xi_1=0$ and $\xi_{n+p+1}=1$. If the knot vector is open, then the B-spline curve is interpolatory at endpoints, i.e., $C(0)=B_1$ and $C(1)=B_n$.


(a) Curve and control points
(b) Curve and mesh denoted by knot locations

<u>Fig. 1.3</u>

An affine transformation is applied to the B-spline curve when applied to the control points. An affine transformation is a mapping $\Phi: \mathbb{R}^3 \rightarrow \mathbb{R}^3$ such that for any vector **x**:

$$\Phi(\mathbf{x}) = \mathbf{Ax} + \mathbf{v}$$

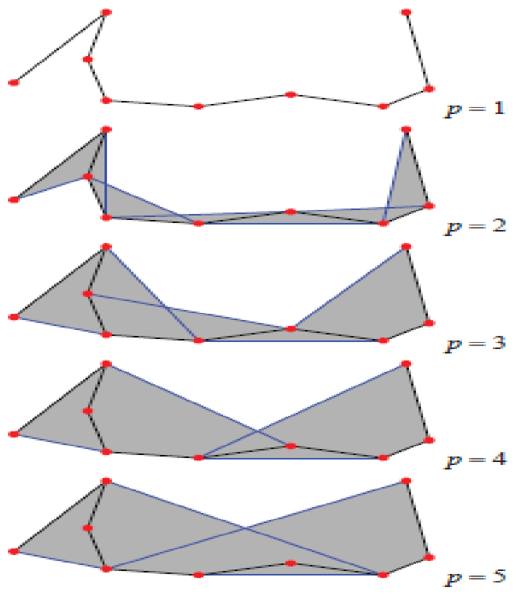
for some matrix $\boldsymbol{A} \in \mathbb{R}^3 \times \mathbb{R}^3$ and vector $\mathbf{v} \in \mathbb{R}^3$. Affine transformations include translations, rotations, scalings, and uniform stretchings and shearings. The affine invariance property, as it is called, follows from the partition of unity of the $N_{i,p}$ .Thus let $x = \sum a_i p_i$, where $p_i \in \mathbb{R}^3$ and $\sum a_i = 1$. Then:

$$\Phi(x) = \Phi(\sum a_i p_i) = A(\sum a_i p_i) + v = \sum a_i A p_i + \sum a_i v = \sum a_i (Ap_i + v) = \sum a_i \Phi(p_i) .$$

The continuity and differentiability of $\boldsymbol{C(\xi)}$ follows from that of the basis $N_{i,p}$(since $\boldsymbol{C(\xi)}$ is just a linear combination of the $N_{i,p}$). Thus, $\boldsymbol{C(\xi)}$ is infinitely differentiable in the interior of the knot intervals and it is at least p-k times continuously differentiable at a knot of multiplicity k.

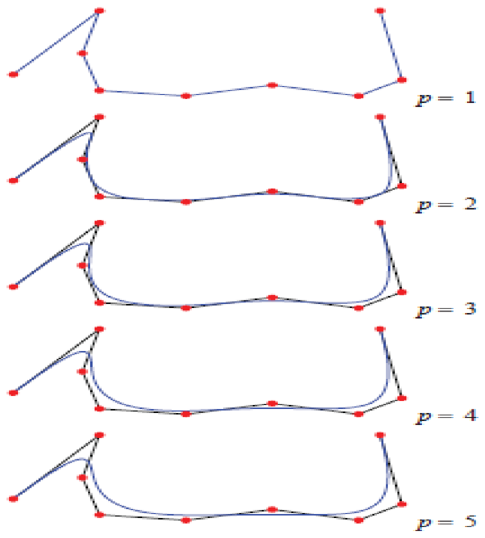
The example shown in **<u>Fig.1.3</u>** is built from the quadratic (p=2) basis functions considered in **<u>Fig.2.5</u>**. The curve is interpolatory at the first and last control points, a general feature of a curve built from an open knot vector. Note that the curve is also interpolatory at the sixth control point. As discussed above, this is due to the fact that the multiplicity of the knot $\xi = 4$ is equal to the polynomial order. Note also that the curve is tangent to the control polygon at the first, last and sixth control points. The curve is C^{p-1}= C^1-continuous everywhere except from the location of the repeated knot, $\xi = 4$, where it is C^{p-2}=C^0-continuous. Note the difference between the control points, shown in **<u>Fig.1.3</u>(a)** and the images of the knots, shown in **<u>Fig.1.3</u>(b).** There, the knots are mapped into the physical space.

The curve is contained in the convex hull of its control polygon; this follows from the nonnegativity and partion of unity of the $N_{i,p.}$ and the property that $N_{j,p}(\xi)=0$ for j<i-p and j>i when $\xi \in [\xi_i, \xi_{i+1}).$ Fig. 1.4 shows such convex hulls for $p =$ 1 through $p =$ 5 for a given set of control points. Note, in particular, that the convex hull for a piecewise linear curve is just the control polygon itself.



<u>Fig. 1.4</u>

As we can observe in **<u>Fig.1.5</u>**, the smoothness of the curve increases along with the degree *p* while the effect of each control point on the final shape decreases .



<u>Fig. 1.5</u>

B-spline curves also possess a variation diminishing property. No plane (line, in the case of one dimension) has more intersections with the curve than it has with the control polygon. This property is particularly striking when compared with the behavior of standard Lagrange polynomials. An example is illustrated in Fig. 1.6a where Lagrange polynomials of orders three, five, and seven interpolate a discontinuity represented by eight data points in $\mathbb{R}^2$. Note that as the order is increased, the amplitude of the oscillations also increases. B-splines behave very differently when the data are viewed as control points. The variation diminishing property leads the B-spline curves in Fig. 1.6b to be monotone, a property that proves useful in analysis.

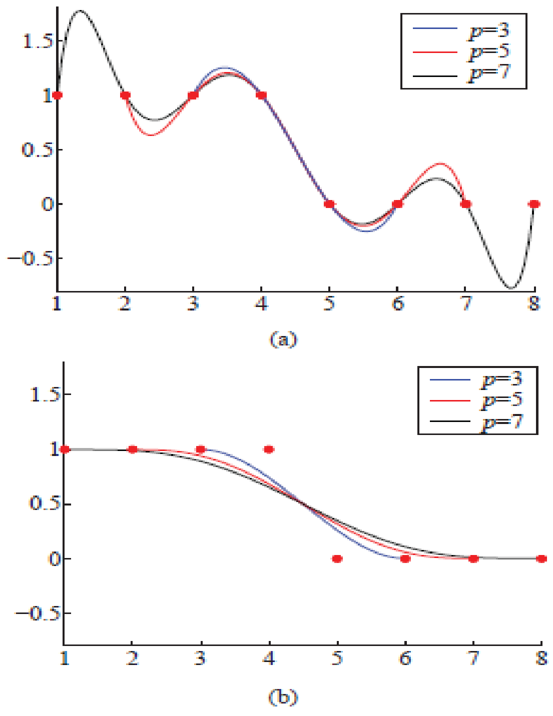


Fig. 1.6

1.3 B-spline surfaces

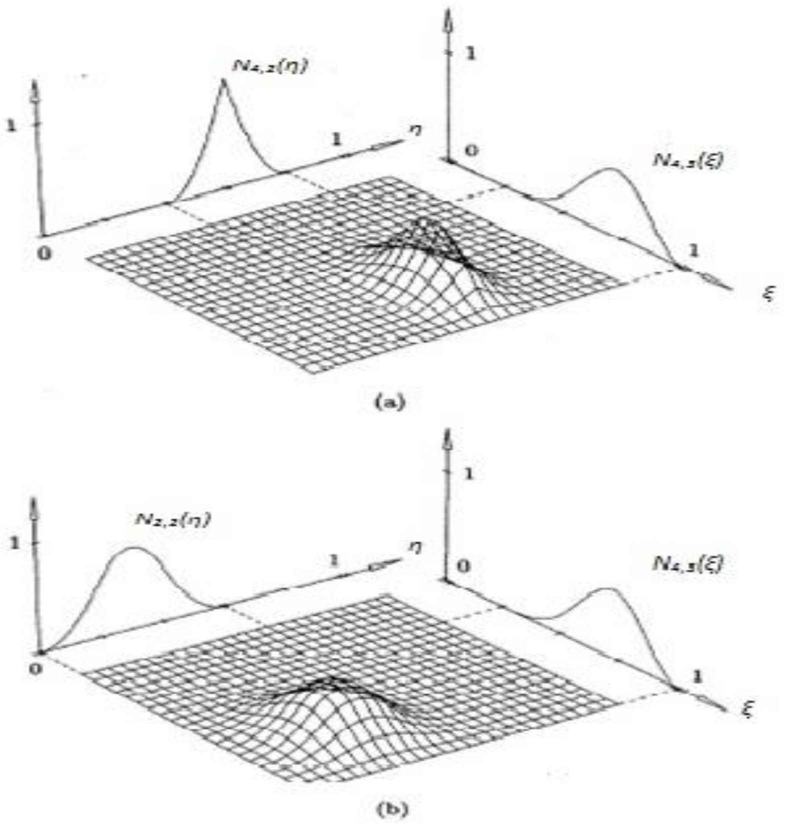
A *B-spline surface* is obtained by taking a bidirectional net of control points, called ***control net***, two knot vectors $\Xi=\{\xi_1,\ldots, \xi_{n+p1}\}$ and $H=\{\eta_1,\ldots,\eta_{m+p+1}\}$, and the products of the univariate B-spline functions:

$$S(\xi,\eta)=\sum_{i=1}^{n}\sum_{j=1}^{m}N_{i,p}(\xi)M_{j,q}(\eta)B_{i,j} \qquad (1.3.1)$$

Many of the properties of a B-spline surface are the result of its tensor product nature. The basis is pointwise non-negative and forms a partition of unity as $\forall(\xi,\eta)\in[\xi_1,\xi_{n+p+1}]\times[\eta_1,\eta_{m+q+1}]$,

$$\sum_{i=1}^{n}\sum_{j=1}^{m}N_{i,p}(\xi)M_{j,q}(\eta)=\left(\sum_{i=1}^{n}N_{i,p}(\xi)\right)\left(\sum_{j=1}^{m}M_{j,q}(\eta)\right)=1 \qquad (1.3.2)$$

Interior to the rectangles formed by ξ and η knot lines, where the function is a bivariate polynomial, all partial derivatives of $N_{i,p}(\xi)$ and $M_{j,q}(\eta)$ exist; at a ξ (or η) knot it is *p-k(q-k)* differentiable in the $\xi(\eta)$ direction, where *k* is the multiplicity of the knot. The properties of convex hull and affine invariance still hold. Fig. 1.7 shows an example of *[cubic$\times$quadratic]* basis functions.



<u>Fig. 1.7</u>
(a) $N_{4,3}(\xi)N_{4,2}(\eta)$
(b) $N_{4,3}(\xi)N_{2,2}(\eta)$
Ξ={0 0 0 0 1/4 1/2 3/4 1 1 1 1 } and H={0 0 0 1/5 2/5 3/5 4/5 1 1 1}

As a consequence of this strong convex hull property, a B-spline surface can contain embedded flat regions and lines of sharp discontinuity. This is a particularly desirable characteristic for many design situations. **Fig.1.8(a)** to **1.8(d)** show a series of open B-spline surfaces and their control points in each parametric direction. Notice that the control points in the η-direction are collinear. The resulting surface is ruled in the η-direction. The B-spline surface shown in **Fig.1.8(a),** defined by four control points in the ξ-direction, is smoothly curved there.

The B-spline surface shown in **Fig.1.8(b)** is defined by five control points in the ξ direction, the three of which are collinear. Notice that the center of the resulting surface is flat. Similarly, five of the seven control points in the ξ-direction, for the surface shown in **Fig.1.8(c),** are collinear. Again, the surface is flat in the central region. The flat area is larger than in **Fig.1.8(b).**

Fig.1.8(d) shows that this very strong convex hull property extends to both the parametric directions. Thus, a flat region can be embedded in the interior of a sculptured surface. This flat region becomes smaller as the order of the surface increases.

Fig.1.9 illustrates the effect of coincident net lines. In **Fig.1.9(a),** three coincident net lines are used to generate a hard line or knuckle in the center of a fourth-order B-spline surface. **Fig.1.9(b)** shows result when three coincident net lines are used in both the parametric directions. Here, the fourth order B-splines surface contains two ridges that rise up to a point in the center of the surface.

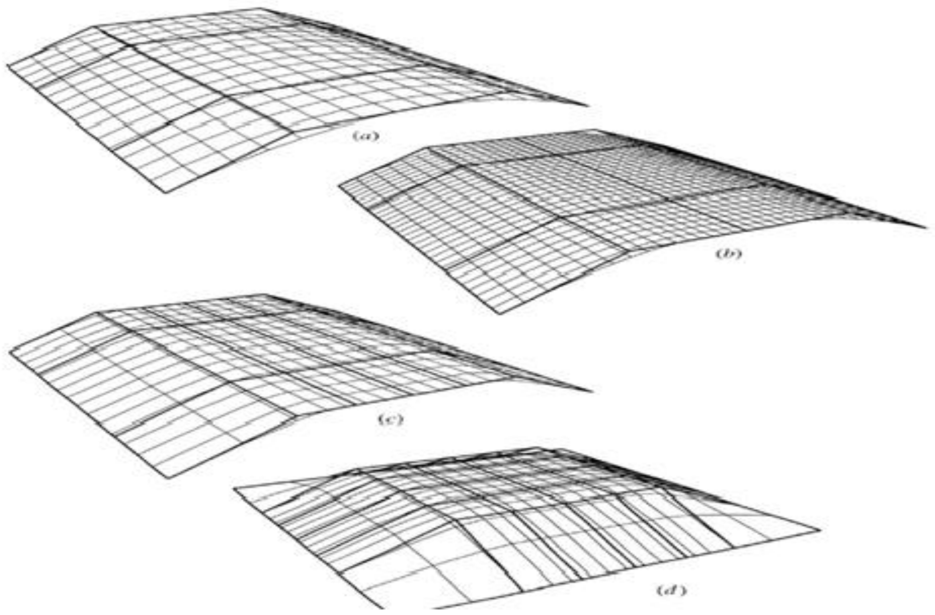


Fig. 1.8: *Third order B-spline surfaces*
(**a**) Smooth ruled surface
(**b**) Small interior flat region caused by three colinear control points in ξ
(**c**) Larger interior flat region caused by
five colinear control points in ξ
(**d**) Flat region embedded within a sculptured surface

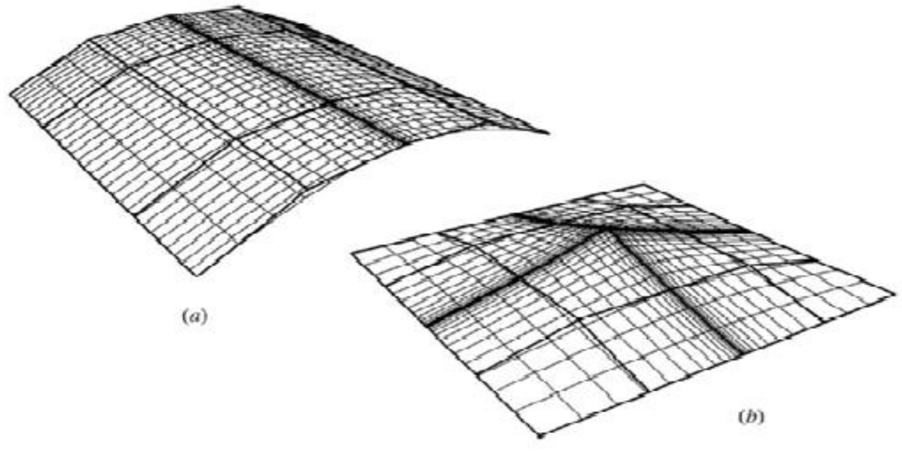


Fig. 1.9

The local support of the basis functions also follows directly from the one-dimensional functions that form them. The support of a given bivariate function $N_{i,j;p,q}(\xi, \eta) = N_{i,p}(\xi) M_{j,q}(\eta)$ is exactly $[\xi_i, \xi_{i+p+1}] \times [\eta_j, \eta_{j+q+1}]$.

1.4 Non-Uniform Rational B-Splines

We start by defining NURBS curve. A rational B-spline curve is the projection of a non rational (polynomial) B-spline curve defined in four-dimensional (4D) homogeneous coordinate space, back into three-dimensional (3D) physical space. (see **Fig.1.10**) Specifically :

$$C^W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) B_i^w (1.4.1)$$

where the B_i^w s are the four-dimensional control polygon vertices for the non rational four-dimensional B-spline curve. $N_{i,p}(\xi)$ is the non rational B-spline basis function previously given in .

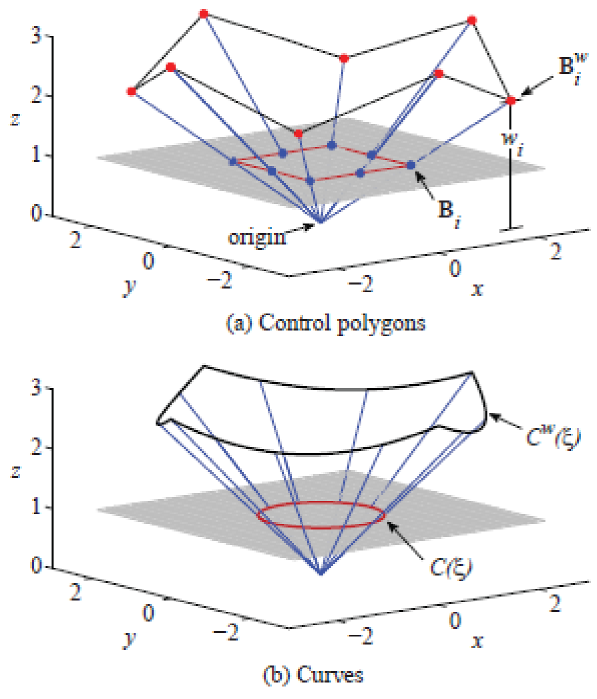


Fig. 1.10

Projecting back into the three-dimensional space by dividing through the homogeneous coordinate yields the rational B-spline curve:

$$C(\xi) = \frac{\sum_{i=1}^{n} N_{i,p}(\xi) w_i B_i}{W(\xi)} = \frac{\sum_{i=1}^{n} N_{i,p}(\xi) w_i B_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i},p}(\xi) w_{\hat{i}}} = \sum_{i=1}^{n} B_i R_{i,p}(\xi) \quad (1.4.2)$$

where the B_i *'s* are the three-dimensional control net vertices for the rational B-spline curve and

$$R_{i,p}(\xi) = \frac{N_{i,p}(\xi) w_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i},p}(\xi) w_{\hat{i}}} \quad (1.4.3)$$

are the rational B-spline basis functions . Here, $w_i \geq 0$ for all values of i. Note that rational B-spline basis functions for $w_i < 0$ are valid but are not convenient in terms of the current discussion.

$$W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) w_i \qquad (1.4.4)$$

Equation *(1.4.4)* is called **weighting function**.

The NURBS basis functions and curves are a generalization of non rational B-spline basis functions and curves. Thus, they carry forward nearly all the analytic and geometric characteristics of their non rational counterparts. In particular, the continuity of the functions, as well as their support, follows directly from the knot vectors exactly as before. The basis still constitutes a partition of unity and it is pointwise non-negative. These properties taken together result in a strong convex hull property for the NURBS functions.

Any *projective* transformation is applied to a rational B-spline curve by applying it to the control points; i.e. , the curve is invariant with respect to a projective transformation. Remark that this is a stronger condition compared to non rational B-splines, which are only invariant to an *affine* transformation.

From equation *(1.4.3),* it is clear that when w_i are all equal to one, then $R_{i,p}=N_{i,p}$. Thus non rational B-spline basis functions are included as a special case of NURBS. Due to the fact that NURBS are a generalization of non rational B-spline algorithms for degree elevation, subdivision and curve fitting are valid by applying them to the four-dimensional control points.

Next, we study the effect of the weights w_i by illustrating an example. Here, an open knot vector $\Xi=[0\ 0\ 0\ 1\ 2\ 3\ 3\ 3]$ and cubic basis functions are used along with a weight vector which is defined as $w_i=1\ , i\neq 3$. Values of w_i range from 0 to 5. The rational B-splines basis functions are shown in **Fig. 1.12(a)** to **1.12(d)**.

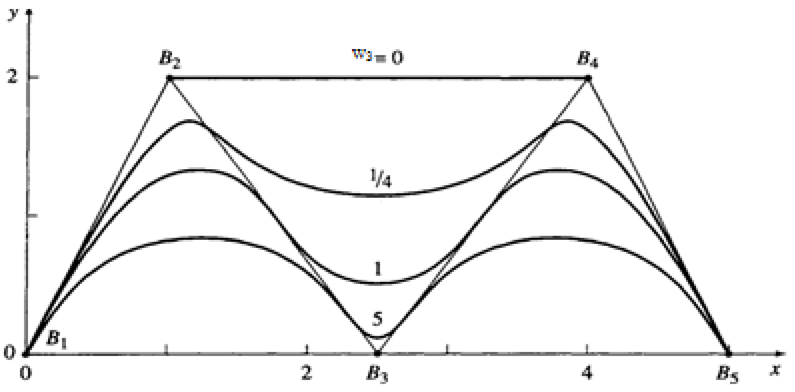


Fig. 1.11

Notice that for $w_3=0$, then $R_{3,3}=0$ everywhere . Thus, the corresponding control point B_3, has no influence on the shape of the B-spline curve. This effect is shown in **Fig.1.11** , where the control points B_2 and B_4 are connected by a straight line. Fig. 1.12 also shows that as w_3 increases, $R_{3,3}$ also increases; but -as a consequence of the partition of unity property- $R_{2,3}$ and $R_{4,3}$ decrease. The effects on the B-spline curve are shown in **Fig.1.11**. In particular, as w_3 increases, the whole curve is pulled closer to B_3. Hence, as mentioned previously, the weight coordinates provide additional blending capability.

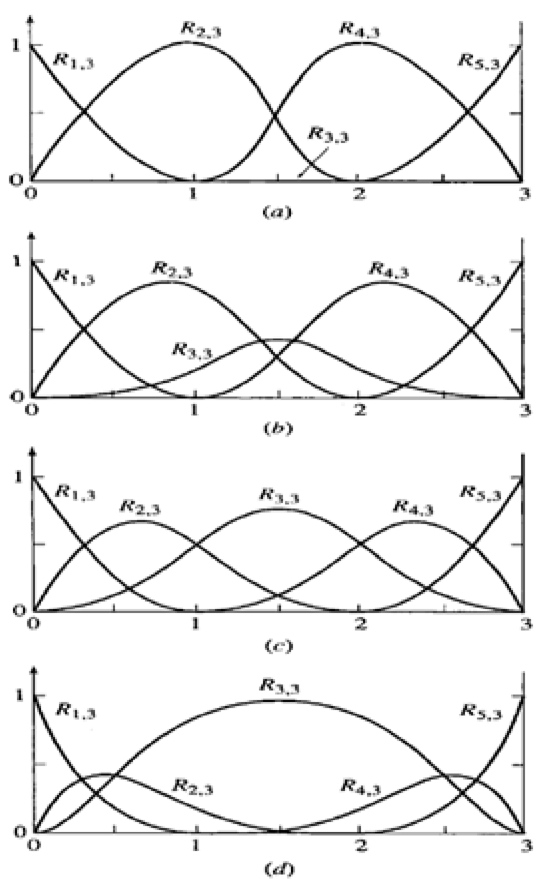


Fig. 1.12

Rational surfaces are defined similarly in terms of the rational basis functions:

$$R_{i,j}^{p,q}(\xi,\eta)=\frac{N_{i,p}(\xi)M_{j,q}(\eta)w_{i,j}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m}N_{\hat{i},p}(\xi)M_{\hat{j},q}(\eta)w_{\hat{i},\hat{j}}}=R_{i,p}S_{j,q} \quad (1.4.5)$$

1.4.1 Derivatives of NURBS basis functions

As the NURBS basis functions are constructed from the B-spline basis functions, the derivatives of rational functions will clearly depend on the derivatives of their non-rational counterparts as well. Simply applying the quotient rule to (1.4.2) yields

$$\frac{d}{d\xi}R_i^p(\xi)=w_i\frac{W(\xi)N_{i,p}^{'}(\xi)-W^{'}(\xi)N_{i,p}(\xi)}{(W(\xi))^2} \quad (1.4.6)$$

where $N^{'}_{i,p}(\xi) \equiv \frac{d}{d\xi} N_{i,p}(\xi)$ and

$$W^{'}(\xi) = \sum_{\hat{i}=1}^{n} N^{'}_{\hat{i},p}(\xi) \quad (1.4.7)$$

An expression is also available for higher-order derivatives of NURBS basis functions. Let us simplify notation by defining:

$$A_i^{(k)}(\xi) = w_i \frac{d^k}{d\xi^\kappa} N_{i,p}(\xi) \text{ (no sum on } i \text{)}$$

where we *do not sum on the repeated index*, and let

$$W^{(k)}(\xi) = \frac{d^k}{d\xi^k} W(\xi) \quad (1.4.8)$$

Higher-order derivatives of these rational functions may be expressed in terms of lower-order derivatives as

$$\frac{d^k}{d\xi^k} R_i^p(\xi) = \frac{A^{(k)}(\xi) - \sum_{j=1}^{k} \binom{k}{j} W^{(j)}(\xi) \frac{d^{(k-j)}}{d\xi^{(k-j)}} R_i^p(\xi)}{W(\xi)} \quad (1.4.9)$$

where

$$\binom{k}{j} = \frac{k!}{j!(k-j)!}.$$

1.5 Mesh generation and refinement strategies

The term " e l e m e n t " in isogeometric analysis simply denotes the mapping of the knot spans from the parametric space to the physical space (a line, a square or a cube, depending on the dimension space ; see Fig. 1.13). Although an open vector can guarantee interpolation at the ends of the interval or at the ends of patches (if we refer to 2 or 3 dimensional spaces) we are to be confused with the term nodes on classic FEA. So a boundary of a B-spline object with d parametric dimensions consists itself a B-spline object of dimension (d-1).

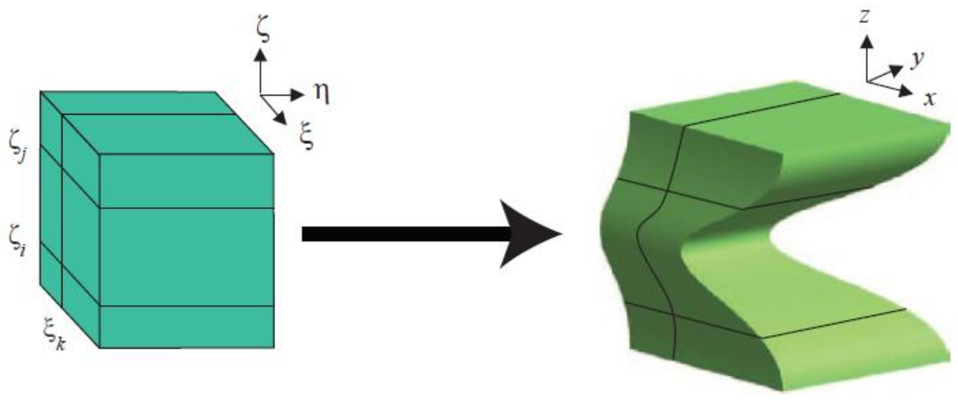


Fig. 1.13

Thus, a mesh is constructed through this mapping. Isogeometric analysis carries over the classical refinement strategies, h and p-refinement, and offers a new way of refinement called k. We will now exam the way we can handle and achieve these ways of refinement with respect to the geometrical properties of NURBS.

1.5.1 Knot insertion (h-refinement)

Let $C(\xi)=\sum_{i=1}^{n} R_{i,p}B_i$ be a NURBS curve defined on $\Xi=\{\xi_1,.....,\xi_{n+p+1}\}$. Let $\bar{\xi}\in[\xi_k,\xi_{k+1})$, and insert $\bar{\xi}$ into Ξ to form the new knot vector $\bar{\Xi}=\{\bar{\xi}_1=\xi_1,.....,\bar{\xi}_k=\xi_k,\bar{\xi}_{k+1}=\bar{\xi}_k,\bar{\xi}_{k+2}=\xi_{k+1},....,\bar{\xi}_{(n+1)+p+1}=\xi_{n+p+1}\}$. If V_{Ξ} and $V_{\bar{\Xi}}$ denote the vector spaces of curves defined on Ξ and $\bar{\Xi}$, respectively, then clearly $V_{\Xi}\subset V_{\bar{\Xi}}$ (and $dim(V_{\bar{\Xi}})=dim(V_{\Xi})+1$); thus C(ξ) has a representation on $\bar{\Xi}$ of the form:

$$C(\xi)=\sum_{i=1}^{n}\bar{R}_{i,p}(\xi)Q_i \tag{1.5.1}$$

where the $\{\bar{R}_{i,p}(\xi)\}$ are the *p*-th degree basis function on $\bar{\Xi}$. The term knot insertion refers to the process of determining the $\{Q_i\}$ in the equation. It is important to note that knot insertion is really just a change of vector space basis; the curve is not changed, neither geometrically nor parametrically.

Although not immediately obvious, **knot insertion** is one of the most important of all B-spline algorithms. Some of its *u s e s* are:

- *e v a l u a t i n g* points and derivatives on curves and surfaces
- *s u b d i v i d i n g* curves and surfaces
- adding control points in order to increase *f l e x i b i l i t y* in shape control ***(interactive design)***

Now, the Q_i in equation can be obtained by setting up and solving a system of linear equations. If we set

$$\sum_{i=1}^{n} R_{i,p}(\xi)B_i = \sum_{i=1}^{n} \bar{R}_{i,p}(\xi)Q_i \tag{1.5.2}$$

then by substituting $n+2$ suitable values of ξ into equation we obtain anon singular, banded system of $n+2$ linear equations in the $n+2$ unknowns , Q_i. However, there is a more efficient solution. The fact that on a given knot span $[\xi_j,\xi_{j+1})$, at most $p+1$ of the $R_{i,p}$ are nonzero, namely the functions $R_{j-p,p},\ldots,R_{j,p}$ and $\bar{\xi} \in [\xi_k,\xi_{k+1})$ imply that:

$$\sum_{i=k-p}^{k} R_{i,p}(\xi)B_i = \sum_{i=k-p}^{k+1} \bar{R}_{i,p}(\xi)Q_i \tag{1.5.3}$$

for all $\xi \in [\xi_k,\xi_{k+1})$,

and

$$\begin{aligned} R_{i,p}(\xi) &= \bar{R}_{i,p}(\xi) \quad i=0,\ldots,k-p-1 \\ R_{i,p}(\xi) &= \bar{R}_{i+1,p}(\xi) \quad i=k+1,\ldots,n \end{aligned} \tag{1.5.4}$$

Equations (1.5.3) and (1.5.4), together with the linear independence of the basis function, imply that

$$\begin{aligned} B_i &= Q_i \quad i=0,\ldots,k-p-1 \\ B_i &= Q_{i+1} \quad i=k+1,\ldots,n \end{aligned} \tag{1.5.5}$$

Now consider the $R_{i,p}(\xi)$ for $i=k-p,\ldots,k$. They can be expressed in terms of the $\bar{R}_{i,p}(\xi)$ when $i=k-p,\ldots,k+1$,by

$$R_{i,p}(\xi) = \frac{\bar{\xi}-\bar{\xi}_i}{\bar{\xi}_{i+p+1}-\bar{\xi}_i}\bar{R}_{i,p}(\xi) + \frac{\bar{\xi}_{i+p+2}-\bar{\xi}}{\bar{\xi}_{i+p+2}-\bar{\xi}_{i+1}}\bar{R}_{i|+1,p}(\xi) \tag{1.5.6}$$

Equation *(1.5.6)* is proven by induction on p.

For brevity we now write $\bar{R}_i$ for $\bar{R}_{i,p}(\xi)$. Substituting equation *(1.5.6)* into equation *(1.5.3)* yields:

$$\left(\frac{\bar{\xi}-\bar{\xi}_{k-p}}{\bar{\xi}_{k+1}-\bar{\xi}_{k-p}}\bar{R}_{k-p}+\frac{\bar{\xi}_{k+2}-\bar{\xi}}{\bar{\xi}_{k+2}-\bar{\xi}_{k-p+1}}\bar{R}_{k-p+1}\right)B_{k-p}$$
$$+\left(\frac{\bar{\xi}-\bar{\xi}_{k-p+1}}{\bar{\xi}_{k+2}-\bar{\xi}_{k-p+1}}\bar{R}_{k-p+1}+\frac{\bar{\xi}_{k+3}-\bar{\xi}}{\bar{\xi}_{k+3}-\bar{\xi}_{k-p+2}}\bar{R}_{k-p+2}\right)B_{k-p+1}$$
$$\vdots$$
$$+\left(\frac{\bar{\xi}-\bar{\xi}_{k}}{\bar{\xi}_{k+p+1}-\bar{\xi}_{k}}\bar{R}_{k}+\frac{\bar{\xi}_{k+p+2}-\bar{\xi}}{\bar{\xi}_{k+p+2}-\bar{\xi}_{k+1}}\bar{R}_{k+1}\right)B_{k}$$
$$=\bar{R}_{k-p}Q_{k-p}+\dots.+\bar{R}_{k+1}Q_{k+1}$$

By equating coefficients and using the knot vector Ξ in place of $\bar{\Xi}$ we obtain:

$$0=\bar{R}_{k-p}(Q_{k-p}-B_{k-p})$$
$$+\bar{R}_{k-p+1}(Q_{k-p+1}-\frac{\bar{\xi}-\bar{\xi}_{k+p+1}}{\bar{\xi}_{k+1}-\bar{\xi}_{k-p+1}}B_{k-p+1}-\frac{\bar{\xi}_{k+1}-\bar{\xi}}{\bar{\xi}_{k+1}-\bar{\xi}_{k-p+1}}B_{k-p})$$
$$+\bar{R}_{k}(Q_{k}-\frac{\bar{\xi}-\bar{\xi}_{k}}{\bar{\xi}_{k+p}-\bar{\xi}_{k}}B_{k}-\frac{\bar{\xi}_{k+p}-\bar{\xi}}{\bar{\xi}_{k+p}-\bar{\xi}_{k}}B_{k-1})+\bar{R}_{k+1}(Q_{k+1}-B_{k})\quad(1.5.7)$$

for $i=k-p+1,\dots.,k$ we set

$$a_i=\frac{\bar{\xi}-\bar{\xi}_i}{\bar{\xi}_{i+p}-\bar{\xi}_i}\quad(1.5.8)$$

And note that

$$1-a_i=\frac{\bar{\xi}_{i+p}-\bar{\xi}}{\bar{\xi}_{i+p}-\bar{\xi}_i}\quad(1.5.9)$$

Using the linear independence of the basis functions, and substituting equations *(1.5.8)* and *(1.5.9)* into equation *(1.5.7)* yields:

$$Q_{k-p}=B_{k-p}$$
$$Q_i=a_iB_i+(1-a_i)B_{i-1}\ \ k-p\le i\le k\quad(1.5.10)$$
$$Q_{k+1}=B_k$$

Finally, by combining equations (1.5.5) and (1.5.10), we obtain the formula for computing all the new control points Q_i of equation , that is :

$$Q_i=a_iB_i+(1-a_i)B_{i-1}\ (1.5.11)$$

Where

$$a_i = \begin{cases} 1, & i \le k-p \\ a_i = \dfrac{\bar{\xi} - \bar{\xi}_i}{\bar{\xi}_{i+p} - \bar{\xi}_i}, & k-p+1 \le i \le k \\ 0, & i \ge k+1 \end{cases}$$

equation (1.5.11) says that only *p* new control points must be computed.

For example, let *p=3* and *Ξ={0,0,0,0,1,2,3,4,5,5,5,5}*. the control points are $B_1,B_2,.....,B_7$. We insert $\bar{\xi}=5/2$. Then $\bar{\xi}\in[\xi_5,\xi_6)$ and *k=5*. Thus $Q_1=B_1, ..,Q_3=B_3$ and $Q_7=B_6,...,Q_9=B_8$. Applying the equation, we find that :

$$a_4 = \frac{\frac{5}{2}-0}{3-0} \Rightarrow Q_4 = \frac{5}{6}B_4 + \frac{1}{6}B_3$$
$$a_5 = \frac{\frac{5}{2}-1}{4-1} \Rightarrow Q_5 = \frac{5}{6}B_5 + \frac{1}{6}B_4$$
$$a_6 = \frac{\frac{5}{2}-2}{5-2} \Rightarrow Q_6 = \frac{1}{6}B_6 + \frac{1}{6}B_5$$

Fig. 1.15(a) shows the control polygon before and after the insertion and Fig. 1.15(b) shows the basis functions before and after the insertion. The bottom part of Fig. 1.15(a) shows the ratios to subdivide the polygon legs.

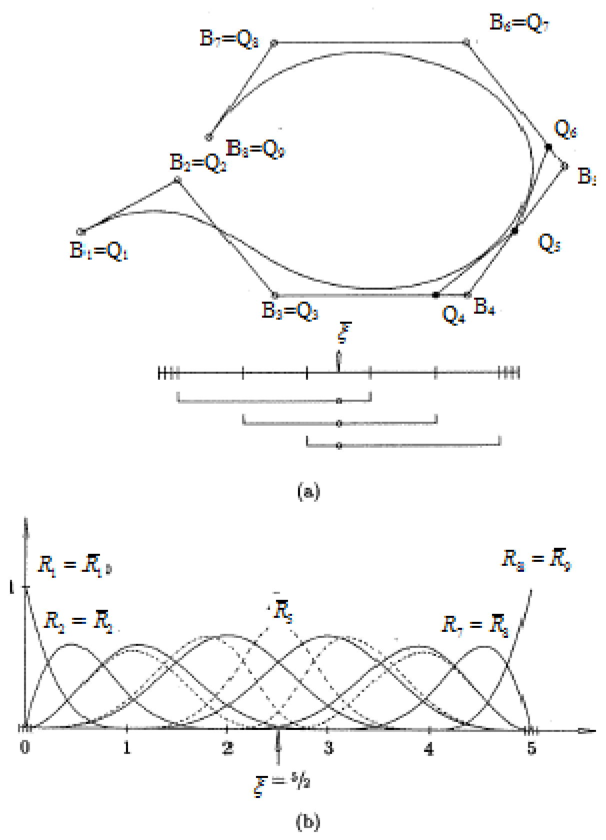
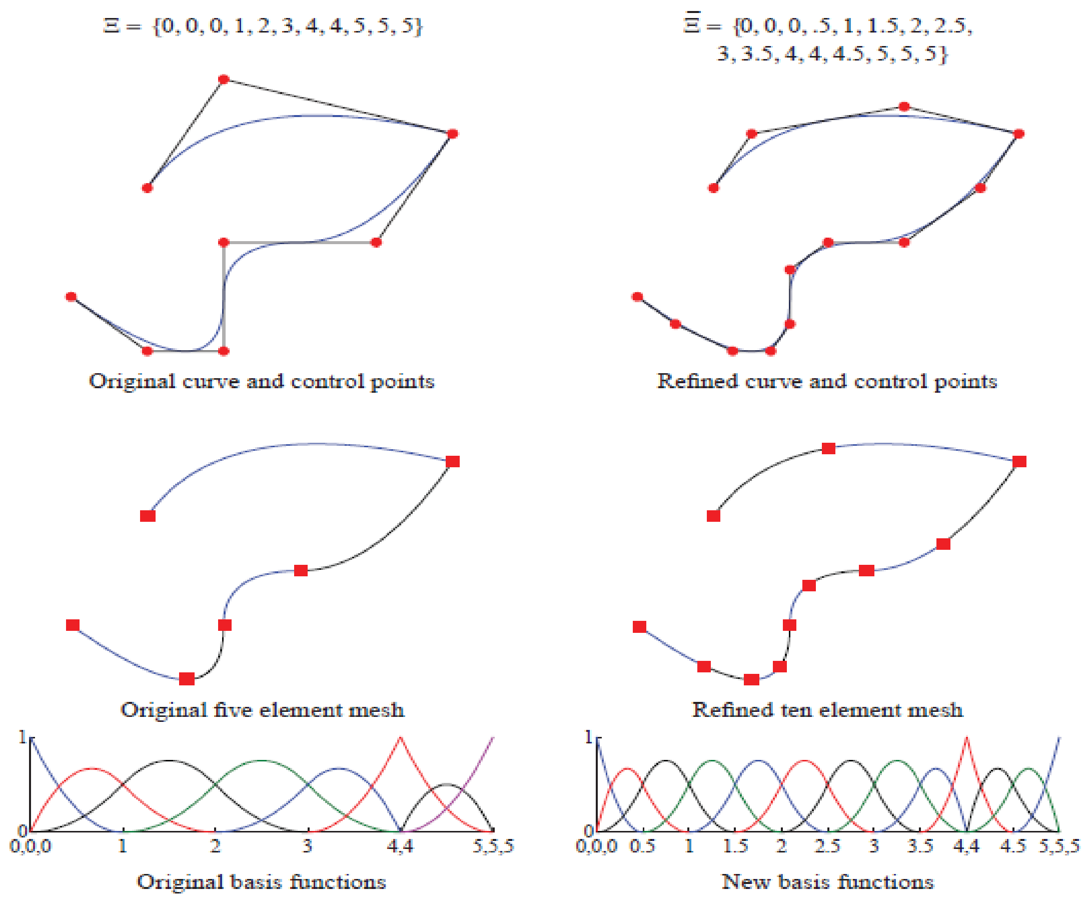


Fig. 1.15

The h-refinement strategy is introduced in isogeometric analysis through the knot insertion process. This way the geometry and the parameterization stay intact, but the solution is enriched, since we add basis functions of the same order.

As we mentioned, the basis functions are C^{p-1} -continuous across the knot spans. In order to perfectly replicate h-refinement, we have to add new knots with *p* multicity. This way the functions will be C^0 -continuous across the elements. However, increasing the multiplicity of the existing knots to decrease the continuity of the basis is not similar to h-refinement strategy in classic finite element analysis, since FEA meshes have C^0 element boundaries to begin with. Fig. 1.16 depicts what happen to the curve and its basis functions when 5 new knots are inserted. As we observe, the addition of every knot causes the elements to split, creating new elements.



Ξ = {0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5}
Ξ̄ = {0, 0, 0, .5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4, 4.5, 5, 5, 5}
Original curve and control points
Refined curve and control points
Original five element mesh
Refined ten element mesh
Original basis functions
New basis functions

Fig. 1.16

1.5.2 Order elevation (p-refinement)

Let $C_p = \sum_{i=0}^{n} R_{i,p} B_i$ be a *p*-th degree NURBS curve on the knot vector ξ. Since C_p is a piecewise polynomial curve, it should be possible to elevate its degree to *p+1*, that is there must exist control point Q_i and a knot vector $\hat{\Xi}$ such that

$$C_p = C_{p+1} = \sum_{i=0}^{\hat{n}} R_{i,p} Q_i \, .$$

C_p and C_{p+1} are the same curve , both geometrically and parametrically. C_{p+1} is simply C_p embedded in a higher dimensional space. Degree elevation refers to the process (the algorithm) for computing the unknown Q_i and $\hat{\Xi}$.

As usual, degree elevation algorithms are applied to C_p in four-dimensional space.

There are three unknowns in equation ,n̂ , $\hat{\Xi}$, and the {Q_i}. To determine n̂ and $\hat{\Xi}$, assume that Ξ has the form :

$$\Xi = \{\xi_1, \ldots.., \xi_{n+p+1}\} = \{\underbrace{a, \ldots, a}_{p+1}, \underbrace{\xi_2, \ldots, \xi_2}_{m_1}, \ldots\ldots, \underbrace{\xi_{n+p}, \ldots, \xi_{n+p}}_{m_s}, \underbrace{b, \ldots, b}_{p+1}\}$$

where $m_1, \ldots., m_s$ denote multiplicities of the interior knots. Now C_p is a polynomial curve on each non degenerate knot span, hence its degree can be elevated to $p+1$ on each knot span. At a knot of multiplicity m_i, C_p is C^{p-mi} continuous. Since the degree elevated curve $C_{p+1}(\xi)$ must have the same continuity, follows that the same knot must have multiplicity m_i+1 for C_{p+1}. This yields

$$\hat{n}=n+s+1$$

and

$$\hat{\Xi} = \{\xi_1, \ldots.., \xi_{\hat{m}}\} = \{\underbrace{a, \ldots, a}_{p+2}, \underbrace{\xi_2, \ldots, \xi_2}_{m_1+1}, \ldots\ldots, \underbrace{\xi_{n+p}, \ldots, \xi_{n+p}}_{m_s+1}, \underbrace{b, \ldots, b}_{p+2}\}$$

$$\hat{m}=m+s+2.$$

The only remaining problem is to compute the $\{Q_i\}$. An obvious but very inefficient method to do this is to solve a system of linear equations.

Setting

$$\sum_{i=0}^{\hat{n}} R_{i,p}(\xi) Q_i = \sum_{i=0}^{n} R_{i,p}(\xi) B_i$$

and evaluating the $R_{i,p}(\xi)$ and $R_{i,p+1}(\xi)$ at appropriate $\hat{n}+1$, yields a banded system of $\hat{n}+1$ linear equations in the unknowns ,Q_i .Instead we apply an algorithm which involves the extraction of Bézier segments of the curve (for more information see [2])

It follows that

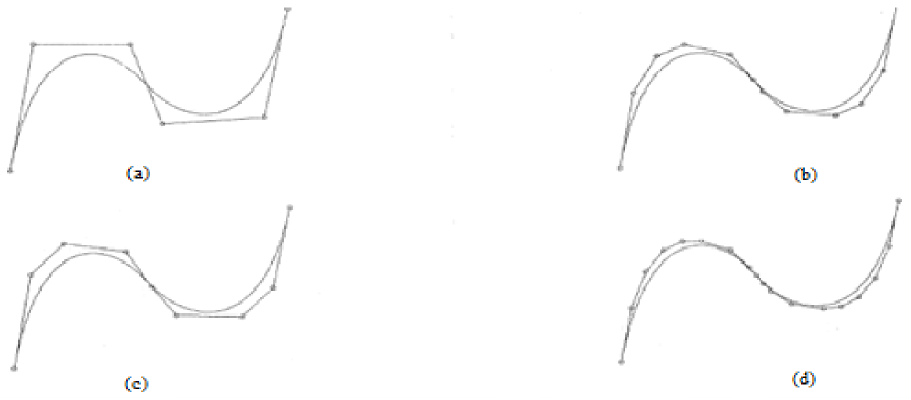
$$Q_i=(1-a_i)B_i+a_iB_{i-1}$$

where

$$a_i = \frac{i}{p+1} \quad i = 0, \ldots, p+1$$

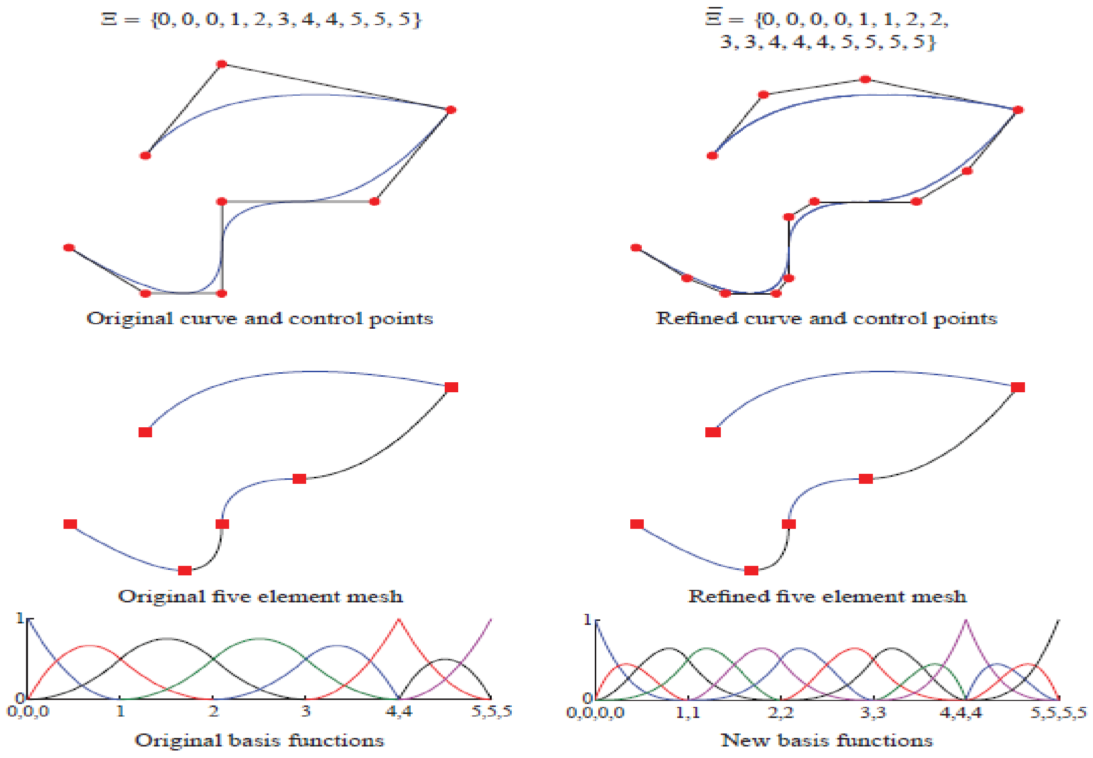
Several curve degree elevation examples are shown in **Fig.1.17(a)-(d)**.The original third degree curve is raised to the fourth, fifth, and seventh degree in **Fig.1.17(b), 1.17(c),**

<u>1.17(d)</u> , respectively. Note that the control polygon converges to the curve as the degree is raised.



<u>Fig. 1.17</u>

P-refinement strategy has much in common with order elevation, as it increases the polynomial degree of the basis. The main difference is that order elevation, unlike classsic p-refinement, does not require to begin with a basis that is C^o everywhere but can be applied with any type of continuity that exists in the unrefined mesh.


$\Xi = \{0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5\}$
$\bar{\Xi} = \{0, 0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5\}$
Original curve and control points
Refined curve and control points
Original five element mesh
Refined five element mesh
1
0
0,0,0
1
2
3
4,4
5,5,5
Original basis functions
1
0
0,0,0,0
1,1
2,2
3,3
4,4,4
5,5,5,5
New basis functions

<u>Fig. 1.18</u>

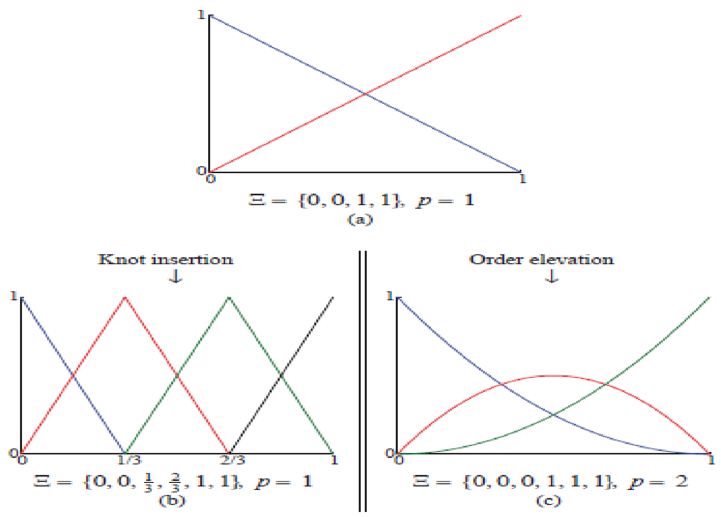
1.5.3 k-refinement

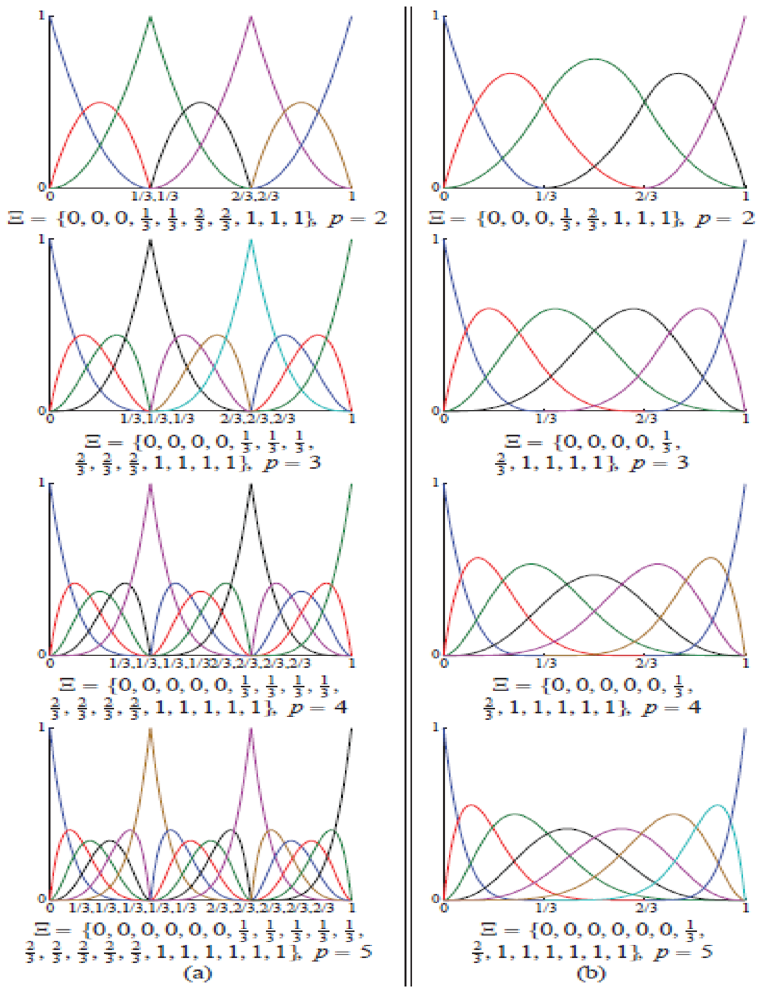
Isogeometric analysis offers a new strategy of refinement, referred as k-refinement. The non commutation of knot insertion and order elevation leads to a mixed strategy which leads to a much more non restrictive way of manipulating a mesh. As we mentioned, both h and p-refinement require the C^0 continuity of the basis. In k-refinement that is not obligatory. This way we can avoid the proliferation of control variables during refinement, often caused by the mandatory condition of C^0 maintenance (see **Fig.1.20**), and achieve a higher order refinement.

K-refinement is achieved through order elevation followed by knot insertion. This way we elevate the coarsest mesh form a degree p to q and then insert the new knot value ξ , the basis will have q-1 continuous derivatives at ξ. If we inverse the process that would lead to a basis which would preserve p-1 continuity, while possessing a polynomial degree q.

Fig.1.19(b) and **1.19(c)** depict a classic p-refinement and k-refinement approach respectively. We may assume that the coarsest mesh consists of one element and $p+1$ basis functions. We begin by inserting new knot values until we have n-p and n basis functions. We then perform order elevation while we maintain continuity at the level of p-1. This causes the replication of each knot value and the addition of a basis function in each element leading to an increase of $2n$-p basis functions. If we continue to elevate the order to r then we would have $[(r+1)n\text{-}rp]$, a rather large number of functions.

On the other hand, if we begin the refinement process by elevating r times and insert knots until we have $(n\text{-}p)$ elements then the final number of basis functions would be $(n+r)$. This is quite smaller than $[(r+1)n\text{-}rp]$.





s

<u>Fig. 1.19</u>

In order to perform k-refinement, the coarsest mesh needs to be consisted of one element. This way we avoid the constraints on the continuity across the element boundaries, which would have been carried out otherwise by the refinement process. Also the continuity level must be preserved to *p-1* at the element boundaries to get significant results from the k-refinement strategy.

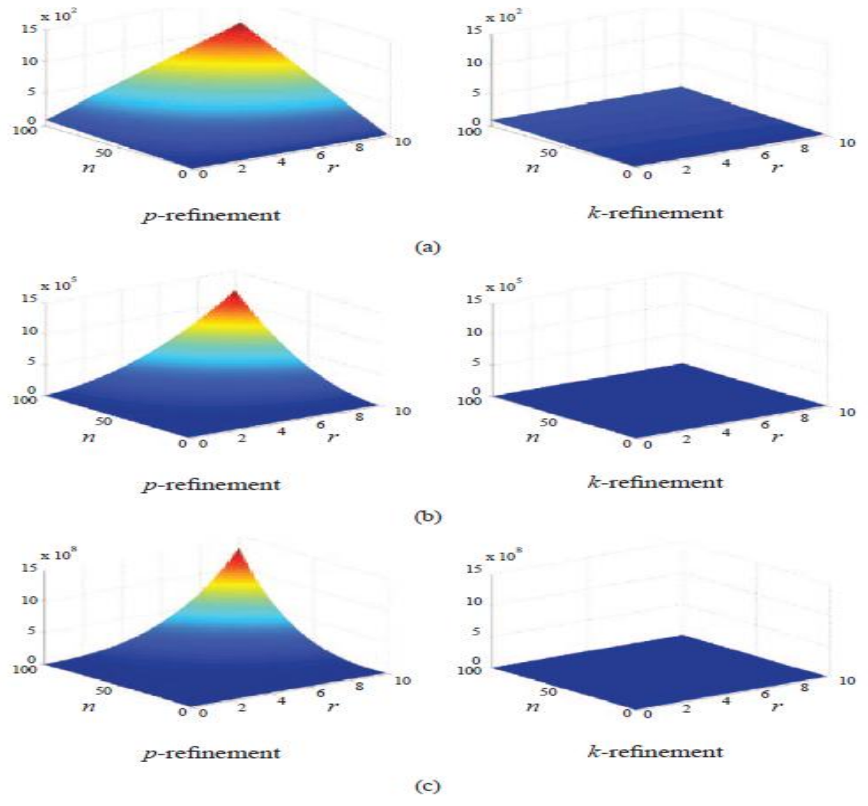


Fig. 1.20

Having in mind that B-splines can have no more than *(p-1)* continuous derivatives across element boundaries, **Fig.1.21** depicts the set of allowable refinements (*highlighted region*).

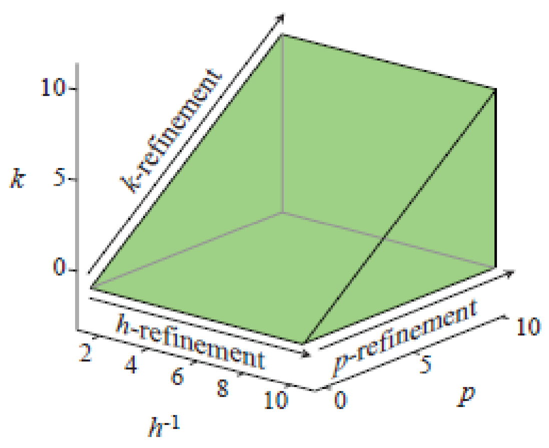
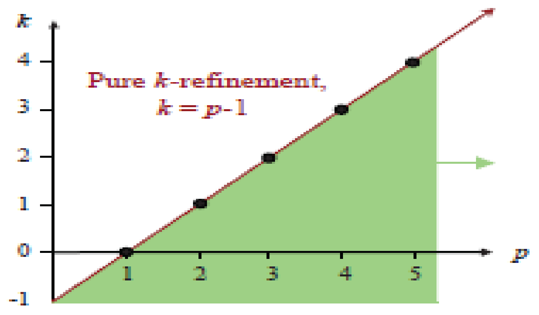
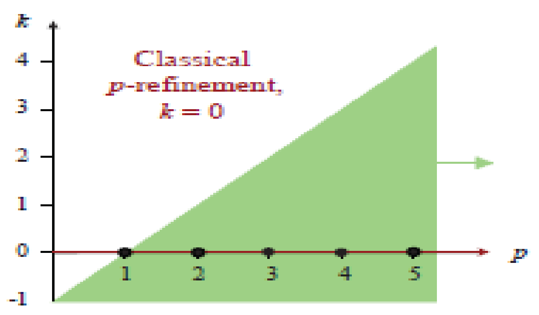


Fig. 1.21

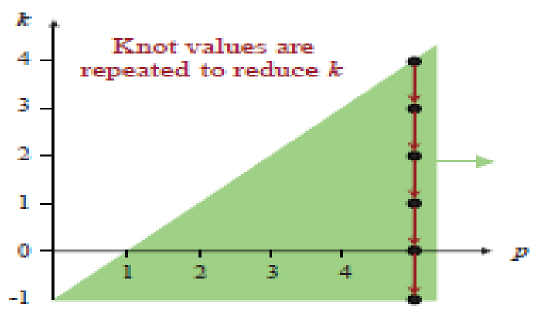
In **Fig.1.22(a)** we can see that during k-refinement, the element size h stays fixed and that as the polynomial order p increases, the continuity of the functions across the element boundaries is also increased, so that C^{p-1} is maintained at all levels of refinement. **Fig.1.22(b)** shows that in pure p-refinement, the continuity k is fixed at $k=0$, while the polynomial order p increases without affecting the element size h .The repetition of existing knot values decreases the continuity across element boundaries, without creating new elements or affecting the polynomial order (**Fig.1.22(c)**). Pure h-refinement (**Fig.1.22(d))** creates new elements that have C^0 boundaries resulting in the decrease of the element size h. Inserting new knot values with multiplicity of 1 (**Fig.1.22(e))** also creates new elements, but now the basis have p-1 continuous derivatives across element boundaries. Finally, in **Fig.1.22(f)** combined refinement strategies are presented. This permits us to tranverse the allowable refinement space.



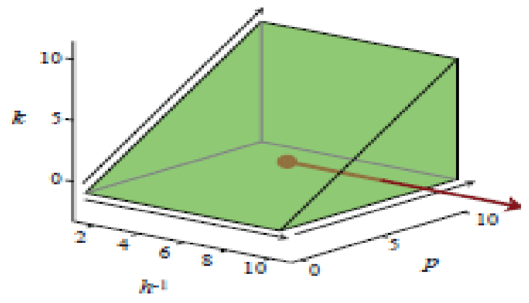
(a) Pure k-refinement



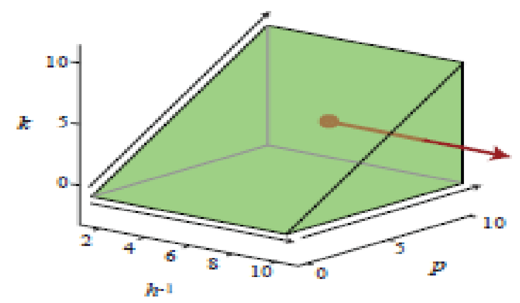
(b) Pure p-refinement



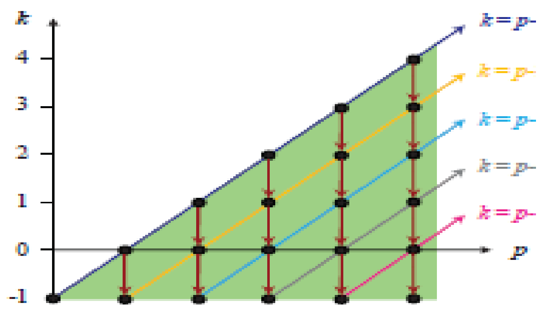
(c) Knot repetition



(d) Pure h-refinement



(e) Insertion of new knot values



(f) Combined refinement strategies

Fig. 1.22

Chapter 2: NURBS As A Tool For Analysis

This chapter is going to examine the use of NURBS functions in the field of finite element analysis. As will we see the use of isogeometric analysis bridges the gap between the solution space and geometry involved in the approximation using finite elements.

2.1 Boundary value problems

To start off, consider Laplace's equation (2.1.1). The goal here is to find a u so that it satisfies conditions (2.1.2) and (2.1.3), also known as Dirichlet and Neumman conditions, respectively.

$$\Delta u + f = 0 \tag{2.1.1}$$

$$u = g \text{ on } \Gamma_D \tag{2.1.2}$$

$$\Delta u \cdot \mathbf{n} = h \text{ on } \Gamma_N \tag{2.1.3}$$

Let us also consider a closed domain $\bar{\Omega}=\Omega \cup \partial\Omega$,where $\partial\Omega$ denotes the boundary. Γ_N and Γ_D are the Neumman and Dirichlet boundary sides, where the conditions are implemented with the property $\overline{\Gamma_D \bigcup \Gamma_N} = \partial\Omega$ and **n** is the unit outward normal vector on $\partial\Omega$. The functions $f : \Omega \to \mathbb{R}$, $g : \Gamma_D \to \mathbb{R}$, $h : \Gamma_N \to \mathbb{R}$ are all given. *Problem 2* constitutes the strong form of the BVPS, which is not our main concern. We are mainly interested in developing schemes for approximating solutions. If Ω is sufficiently smooth and under certain restrictions of g and h the Lax-Milgram theorem guarantees a solution, but in most often cases its analytical form is either hard or impossible to obtain.

The techniques of approximating the analytical solution are referred as ***numerical methods***. Different numerical methods are simply different techniques for finding an approximate solution, such as $u^h \approx u$. This chapter focuses on the application of Galerkin's approximation technique and its implementation on isogeometric analysis.

2.2 Introducing the Galerkin Method

The Galerkin method is firmly attached to finite element analysis, even though there have been numerous and various approaches, regarding the best way to achieve a numerical solution for BVPs. We will introduce the **Bubnov-Galerkin method** which is the most common method of modern finite element analysis. In order to understand this method, several steps are needed to be applied. We start by defining the weak, or variational, formulation of *Problem 2*. A new class of functions is defined, known as ***trial solutions***. This class contains functions that satisfy the *Dirichlet condition (2.1.1)*.

To define the trial spaces, let us first define the space of square integrable functions on Ω . This space, called $L^2(\Omega)$, is defined as the collection of all functions $u : \Omega \to \mathbb{R}$ such that

$$\int_{\Omega} u^2 < +\infty \tag{2.2.1}$$

Let us consider a multi-index $\alpha \in \mathbb{N}^d$ where d is the number of spatial dimensions in the space. For $\alpha = \{\alpha_1, . . . , \alpha_d\}$, we define | $|a| = \sum_{i=1}^{d} a_i$ .We now have a concise way to represent derivative operators. Let $D^a = D^{a_1} D^{a2} .....D^{a_D}$, where $D_i^j = \frac{\partial}{\partial x_i^j}$. The variational formulation needs to be well defined, which (as we will clearly see) leads to the demand that the derivatives of our trial solutions be square integrable . Specifically, if $u : \Omega \to \mathbb{R}$ is a trial solution, then we must insist that

$$\int_{\Omega} \nabla u \nabla u < +\infty \tag{2.2.2}$$

Such a function is said to be in the *Sobolev space* $H^1(\Omega)$,which is defined as:

$$H^1(\Omega) = \{u | D^a u \in L^2(\Omega), |\alpha| \le 1\}. \tag{2.2.3}$$

The set of ***trial solutions***, denoted by *S*, contains the functions which have square-integrable derivatives and satisfy the *Dirichlet condition*:

$$u|\Gamma_D = g. \tag{2.2.4}$$

This is written as

$$S = \{u | u \in H^1(\Omega), u|_{\Gamma_D} = g\}. \tag{2.3.5}$$

The second collection of functions in which we are interested is called the ***weighting functions.*** The weighting functions satisfy the homogeneous Dirichlet condition, i.e. g=0. So the set, denoted by V, can be written as:

$$V = \{w | w \in H^1(\Omega), w|_{\Gamma_D} = 0\}. \tag{2.2.6}$$

Multiplying *(2.1.1)* by an arbitrary test function $w \in V$ and integrating by parts and applying the divergence theorem, results in the weak formulation of *Problem 2*. The initial BVP can now be expressed as such:

Given f, g, and h, find $u \in S$ such that for all $w \in V$

$$\int_{\Omega} \nabla w \cdot \nabla u d\Omega = \int_{\Gamma} wh d\Gamma + \int_{\Omega} wf d\Omega. \tag{2.2.7}$$

As we can see, equation *(2.2.7)* gathers all the unknown information, namely *u* to the left-side, while the given data are contained in the right side.
The necessity of working in $H^1(\Omega)$ spaces is quite obvious, since the weak formulation of the original BVP requires that *u* is square intergrable, while the strong form requires *u* to have well defined second derivatives.

This weak form may be rewritten as

$$a(w, u) = L(w) \tag{2.2.8}$$

where

$$a(w,u) = \int_{\Omega} \nabla u \nabla w d\Omega \tag{2.2.9}$$

and

$$L(w) = \int_{\Omega} wfd\Omega + \int_{\Gamma_N} whd\Gamma \tag{2.2.10}$$

A few properties of $a(\cdot\,,\ \cdot)$ and $L(\cdot)$ are worth noticing. The first is the <u>symmetry</u> of $a(\cdot\,,\ \cdot)$. It follows directly from its definition that $a(w, u) = a(u,w)$. Also, $a(\cdot\,,\ \cdot)$ is <u>bilinear</u> and $L(\cdot)$ is <u>linear</u>. That is, for all constants C_1 and C_2,

$$a(C_1u + C_2v,w) = C_1a(u,w) + C_2a(v,w), \tag{2.2.11}$$
$$L(C_1u + C_2v) = C_1L(u) + C_2L(v). \tag{2.2.12}$$

The details may vary, but Laplace's equation can represent the general idea behind the process of setting up the weak formulation of BVPs as well as how to implement the boundary conditions .

The solution to *(2.2.7)*, or equivalently to *(2.2.8)*, is called a ***weak solution***. The ***strong solution*** always satisfies *(2.2.7)* and the weak solution, under appropriate assumptions, can become a strong one.

<u>2.2.1 The method itself</u>

The first step in developing the method is to construct finite-dimensional approximations of *S* and *V*, denoted S^h and V^h, respectively. The superscript refers to the association of S^h and V^h with a discretization of the domain Ω, which is parameterized by characteristic length scale h. The approximation sets imply that:

$$S^h \subset S, \tag{2.2.13}$$
$$V^h \subset V. \tag{2.2.14}$$

If c_1 and c_2 are constants and $v,w \in V$, then $(c_1v + c_2w) \in V$. Thus, both V and V^h posses the property of a linear space. However, this property is not shared by S and S^h due to the *inhomogeneous boundary condition* *(2.1.2)*, since if u_1 and u_2 are members of S, then $(u_1+u_2=g+g=2g) \notin S$.

We can further characterize S^h by recognizing that if we have a *given* function $g^h \in S^h$ such that $g^h|\Gamma_D = g$, then, for every $u^h \in S^h$ there, exists a unique $v^h \in V^h$ such that

$$u^h = v^h + g^h \tag{2.2.15}$$

This clearly will not be possible for an arbitrary function g, but for now let us assume that such a g^h exists.

We can now write a variation equation of the form of *(2.2.8)*. The Galerkin form of the problem is:

Given g^h, h, and r , find $u^h = v^h + g^h$, where $v^h \in V^h$, such that for all $w^h \in V^h$

$$a(w^h, u^h) = L(w^h). \tag{2.2.16}$$

Recalling *(2.2.15)* and the <u>bilinearity</u> of $a(\cdot\,,\,\cdot)$, we can rewrite *(2.2.16)* as

$$a(w^h, v^h) = L(w^h) - a(w^h, g^h). \tag{2.2.17}$$

In this latter form, the unknown information is on the left-hand side, while everything on the right-hand side is given, as before. Equations *(2.2.16)* and *(2.2.17)* are sometimes referred to as <u>**the Bubnov-Galerkin method**</u>.

<u>2.3 Isoparametric Concept</u>

The following questions arise: ***How can we ensure that the solution space used to approximate the Galerkin solution converges to the exact solution, when refinement strategies are applied? What conditions must be satisfied in order to achieve convergence?*** A lot of shape functions -the functions consisting the basis of our approximate finite solution space- are used in order to achieve better accuracy in FEA .
This section remarks some conditions to ensure convergence, although there are shape functions that don't follow them but converge nonetheless to an exact solution. These conditions, however may be considered basic by *providing the criteria* for using the right shape functions.

<u>The basic convergence requirements are that the shape functions should be :</u>

- ✓ ***C_1* :** ***smooth*** (i.e., at least ***C***) on each element interior domain, Ω^e

- ✓ ***C_2* :** ***continuous*** across each element boundary Γ^e
- ✓ ***C_3* :** ***completeness***

The following remarks should be mentioned:

1. Conditions ***C_1*** and ***C_2*** guarantee that the first derivatives of the shape functions have, at worst, finite jumps across the element boundaries (see **Fig.2.1**). This way, all the integrals appearing in the weak formulation are well defined, since first derivatives appear in the integrands.

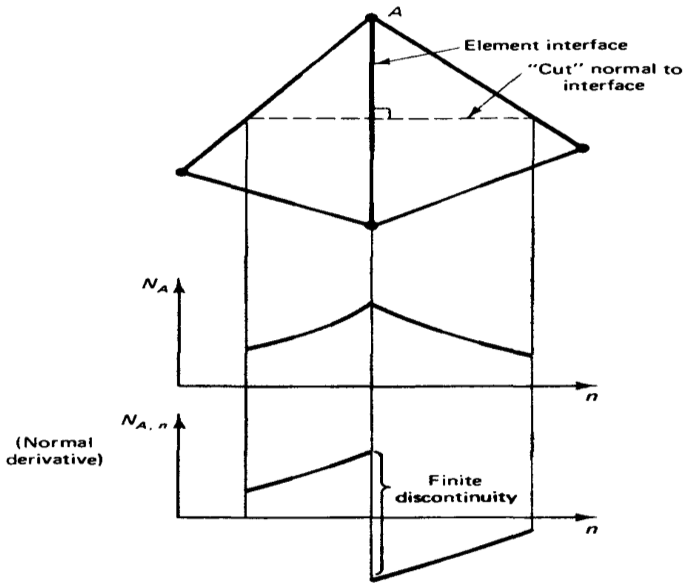


Fig. 2.1

2. Shape functions that satisfy ***C_1*** and ***C_2*** are of class $C^\circ(\bar{\Omega})$. Finite elements constructed from $C^\circ(\bar{\Omega})$ shape functions are often referred to as *C°-elements* .

3. If the integrands involve derivatives of order m, Condition ***C_1*** should be strengthened to C^m-continuity on Ω^e and Condition ***C_2*** should be strengthened to C^m-continuity across Γ^e. Finite elements that satisfy this property are called *conforming* or *compatible*.

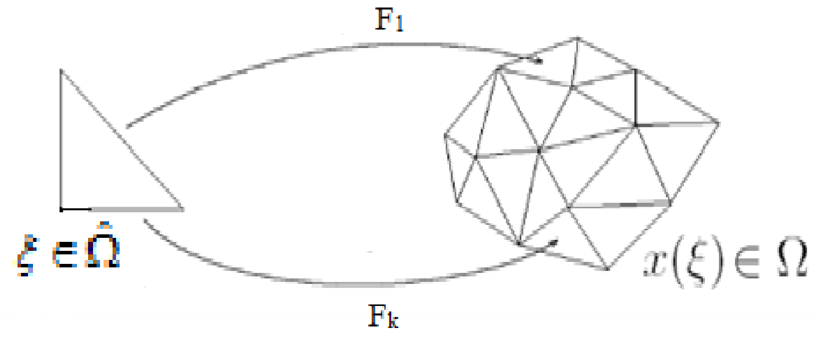
4. Condition ***C_2*** can be ensured by requiring that each function $u^h \in S^h$ is continuous across Γ^e.

In order to understand ***completeness***, let us first introduce the isoparametric approach in order to find an approximate solution. The dimension of the domain Ω is considered to be a subset of $\mathbb{R}^2$, unless otherwise stated. In standard FEM the basis functions are chosen as piecewise polynomials and the concept of isoparametric elements is invoked to approximate curved boundaries. Assume we have n_{en} shape functions N_j , e.g. polynomials, defined over a standard geometry $\hat{\Omega} \subset \mathbb{R}^2$ like a triangle or a square (in d =

3 dimensions a tetrahedron or a hexahedron). We call $\hat{\Omega}$ the parameter domain or parameter space. The computational domain is partitioned into a mesh of elements Ω^e that are sub-domains of the same shape as $\hat{\Omega}$. On each element there are n_{en} specific grid points x^e_i , e.g., the corners, that can be used to define the geometry functions $F^e : \hat{\Omega} \rightarrow \Omega^e$

$$F^e(\xi) = \sum_{i=1}^{n_{en}} N_i(\xi) x_i^e$$

(see also **<u>Fig.2.2</u>***)*



<u>Fig. 2.2</u>

The basis functions N_i for the Galerkin projection are compositions of the shape functions with the inverse of the geometry function. So for $x \in \Omega^e$ we get the local representation of the approximate solution:

$$u^h = \sum_{i=1}^{n_{en}} N_i(\xi) \circ F^{-1} d_i^e$$

where d_i^e stands for the unknown coefficients or nodal values. This local representation offers a local evaluation of element stiffness matrices and force vectors, and the linear system is then assembled from these element contributions.

So if $F: \hat{\Omega} \rightarrow \Omega$ is of the form

$$F(\xi) = \sum_{i=1}^{n_{en}} N_i(\xi) x_i^e \qquad (2.3.1)$$

and the element interpolation function u^h can be written as

$$u^h = \sum_{i=1}^{n_{en}} N_i(\xi) \circ F^{-1} x_i^e \qquad (2.3.2)$$

The element is said to be ***isoparametric***.

The key point to observe in the definition is that the shape functions which define *(2.3.1)* also serve to define *(2.3.2)*.
In this case, the shape functions are said to be complete if

$$d_i^e = c_0 + c_1 x_i^e + c_2 y_i^e \qquad (2.3.3)$$

implies that

$$u^h(x) = c_0 + c_1 x + c_2 y \qquad (2.3.4)$$

where $c_0,..,c_2$ are arbitrary constants.

In words, *c o m p l e t e n e s s* requires that the element interpolation function is capable of exactly representing an arbitrary linear polynomial when the nodal degrees of freedom are assigned values in accordance with it. Completeness is a plausible requirement as the following argument indicates: As the finite element mesh is further and further refined, the exact solution and its derivatives approach constant values over each element domain. To ensure that these constant values are representable, the shape functions must contain all constant and linear monomials. This argument was originally given in and has been proved to be the key mathematical idea for proving convergence theorems for finite element approximations.

R e f i n e m e n t in isoparametric FEM is either performed by splitting the element into smaller ones (h-refinement) or by using higher order polynomials as shape functions in each element (p-refinement). Well-established a posteriori error estimators as well as mesh-refinement algorithms are available. Moreover, polynomials as local basis functions can be easily evaluated and integrated. Note that global smoothness is C^0 in general.

The most obvious drawback of isoparametric FEM is the lack of an exact geometry representation for complex engineering shapes. In this case the boundary must be approximated and also the boundary conditions, which may lead to additional errors or even wrong boundary layers.

We proceed in checking whether ***isoparametric elements ensure convergence to the exact solution or not***.

Convergence Condition C_1 → If F:$\hat{\Omega}$ →Ω is :

i. one-to-one;
ii. onto;
iii. C^k ,k≥ 1; and if
iv. the jacobian determinant j(ξ) > 0 for all ξ∈ $\hat{\Omega}$

then the inverse mapping $\xi = F^{-1} : \overline{\Omega}^e \rightarrow \Omega$ exists and is C^k.

PROPOSITION 1: Let the mapping defined by *(2.3.1)* satisfy (i) through (iv). Then the smoothness requirement (***C_I***) is satisfied.

Proof: By virtue of the hypotheses, $N_i = N_i(\xi)$ is also a C^1 function. Since *F* satisfies (i) through (iv) , $\xi = \xi(x)$ is also C^1. Thus $N_i(x) = N_i(\xi(x))$ is also a C^1 function of *x*.

(This last fact may be proved with the aid of the chain rule).

In practice, the mappings $F : \Omega \rightarrow \overline{\Omega}^e$ usually satisfy (i) through (iv). However, there is one exception of practical importance. It is concerned with the technique of element "degeneration," in which nodes are coalesced. The simplest example of this procedure, in which two nodes of the standard bilinear quadilateral element are coalesced to form a triangle
When degeneration is performed, the Jacobian determinant vanishes at certain nodal points within the element. Away from these points it is positive, and the mapping ξ=ξ(x) remains smooth (i.e., Cl is satisfied). For reasons that will be apparent later on, it is not usually required to calculate derivatives at these points.

Convergence Condition C_3 [*completeness*] → If $\sum_{i=1}^{n_{en}} N_i = 1$, then completeness condition C3 is satisfied for isoparametric elements.

Proof:

(We shall prove the assertion for the two-dimensional case)

$$u^h = \sum_{a=1}^{n_{en}} N_i d_i^e (2.3.5)$$

$$= \sum_{a=1}^{n_{en}} N_i (c_0 + c_1 x_i^e + c_2 y_i^e)$$

$$= c_0 (\sum_{i=1}^{n_{en}} N_i) + c_1 (\sum_{i=1}^{n_{en}} N_i x_i^e) + c_2 (\sum_{a=1}^{n_{en}} N_i y_i^e)$$

$$= c_0 (\sum_{i=1}^{n_{en}} N_i) + c_1 x + c_2 y (2.3.6)$$

The only remaining convergence condition is $\boldsymbol{C_2}$, the continuity requirement on Ω^e. This condition can be verified once the construction of the global shape functions from the element shape functions is explicated. It happens that if this procedure is done in the "obvious" way, continuity is achieved. In the sequel we shall consider this issue on a case-by-case basis.

The importance of the isoparametric concept is that the three basic convergence conditions are applied. In addition, isoparametric elements may be designed to take on convenient shapes, including curved boundaries, and lend themselves to concise computer implementation. This notion of using the same basis for geometry and analysis is called the ***isoparametric concept***, and it is quite common in classical finite element analysis.

2.4 The isogeometric concept

Isogeometric analysis follows the isoparametric approach meaning that the basis used to define the geometry under study is the same for the approximation of the solution. However, this time, the main criteria for choosing the basis is to exactly replicate geometry and use them as a solution space as well. This fact should not considered a drawback since the NURBS basis consist a reliable and desirable solution field. In a sense, we are reversing the isoparametric arrow such that it points from the geometry toward the solution space *(see* **Fig.2.2***)*.

The use of polynomials in classic FEA is quite common due to their simplicity. They are simple to understand and prove theorems regarding their convergence. They are also convenient during calculation process. This is not to say that proving theorems about other bases is impossible. Though precise results for non-polynomial bases do exist most basic convergence requirements in many numerical methods are achieved by *any* reasonably smooth isoparametric basis that is also a partition of unity.

As seen sufficient conditions for a basic convergence proof for a wide class of problems are satisfied by a basis that indulges conditions $\boldsymbol{C_1 - C_3}$.

The requirements of C^1-continuity on the element interiors and C^0-continuity on the element boundaries are not at all restrictive. Most bases that we might consider are C^∞ on the element interiors and have at least C^0-continuity on the element boundaries. The third condition, ***completeness***, requires that, on any given element Ω^e, the basis be capable of representing all linear functions.

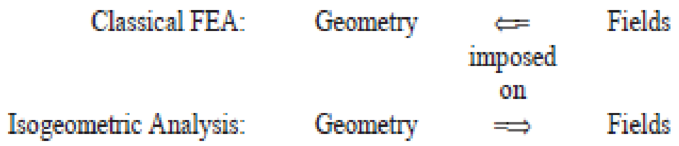


Fig. 2.3

Isogeometric analysis based on the NURBS as basis functions allows to exactly map the unit square in the parameter space $\mathbb{R}^2$ to an arbitrary domain that was designed in a (NURBS-based) CAD-program. The global geometry function F :$\hat{\Omega}$:= $[0,1]^2 \rightarrow \Omega$ is element of a NURBS the knot span space R_p . With the control points of the NURBS $B_i \in \mathbb{R}^2$ in linear ordering (i=1,..., n). The geometry function is defined for all $\xi \in \hat{\Omega}$ by

$$F(\xi) \equiv \begin{Bmatrix} x(\xi) \\ y(\xi) \end{Bmatrix} = \sum_{i=1}^{n_{en}} R_i(\xi) B_i = \sum_{i=1}^{n_{en}} R_i(\xi) \begin{Bmatrix} x_i^e \\ y_i^e \end{Bmatrix} \quad (2.4.1)$$

(see also **Fig.2.3***)*

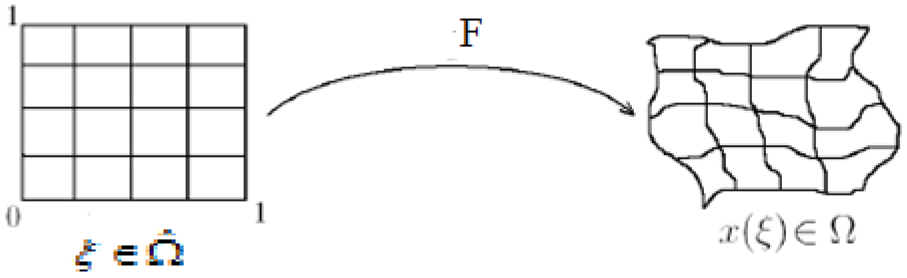


Fig. 2.4

The representation of the approximate solution u^h stays the same as in the isoparametric case, except we are now already in a global setting. So we have for all x $\in \Omega$

$$u^h(x) = \sum_{i=1}^{n_{en}} R_i \circ F^{-1} d_i \quad (2.4.2)$$

It is important to understand that this method does not use elements in the classical sense but patches instead. The knot vectors defining a NURBS create two-dimensional boxes in the preimage of F, their image under F is called a patch.

So ,given the basis $\{R_i\}_{i=1}^{n_{en}}$, completeness demands that there are coefficients d_i such that for arbitrary constants,c_o,c_1 and c_2 :

$$u^h|_{\Omega^e} = \sum_{i=1}^{n_{en}} R_i \circ F^{-1} d_i = c_0 + c_1 x + c_2 y \quad (2.4.3)$$

As the basis is a partition of unity, at that same point ξ we have

$$\sum_{i=1}^{n_{en}} R_i(\xi) = 1. \qquad (2.4.4)$$

Inserting *(2.4.1)* and *(2.4.4)* into *(2.4.3)* and solving for d_A yields

$$d_i^e = c_0 + c_1 x_i^e + c_2 y_i^e \, .$$

Thus, the isoparametric concept and the partition of unity are enough to ensure completeness. Moreover, they are vital to ensuring that isogeometric analysis will result in convergent methods for many different choices. Compared to the piecewise polynomials in the classical FEM, the basis functions are now globally defined and have a larger support. Global smoothness can be easily increased to C^1 or even higher.

2.5 Assembly routine

The use of NURBS basis for the approximation of the solution leads to a linear algebraic system, when the Galerkin method is applied. This section examines thoroughly with use of index notations a way of assembling in an efficient way this system. Suppose our solution space consists of n_{np} basis $R_i{:}\hat{\Omega} \to \mathbb{R}$ .Most of the functions are zero on the boundary of the domain due to the fact that their support is highly localized. Those functions will be differed from the ones that are non-zero, in order to organize in a more efficient way the algebraic system.

So, we may assume that there are n_{eq} such that

$$R_i\big|_{\Gamma_D} = 0 \text{, i=1,….,n}_{\text{eq}} \qquad (2.5.1)$$

Thus for the weighting functions $w^h \subset V^h$, there exist constants c_i, $i = 1, \ldots , n_{eq}$ such that

$$w^h = \sum_{i=1}^{n_{eq}} R_i c_i \qquad (2.5.2)$$

The function g^h , also referred as lifting function ,can be approximated using functions from the NURBS space. This time in order in order to distinguish between the Dirichlet counterparts and the ones that are calculated, we will choose g^h such that $g_1=\ldots=g_{neq}=0$.

So

$$g^h = \sum_{i=n_{eq}+1}^{n_{np}} R_i g_i \qquad (2.5.3)$$

Since $u^h=v^h+g^h$ the approximate solution can be written as

$$u^h = \sum_{i=1}^{n_{eq}} R_i d_i + \sum_{j=n_{eq}+1}^{n_{np}} R_j g_j = \sum_{i=1}^{n_{eq}} R_i d_i + g^h \qquad (2.5.4)$$

Substituting in *(2.2.16)* with *(2.5.1)* and *(2.5.4)* and having in mind the ***linearity*** of both a(.,.) and L(.) we get:

$$\sum_{i=1}^{n_{eq}} c_i \left(\sum_{j=1}^{n_{eq}} a(R_i, R_j) d_j - L(R_i) + a(R_{j,} g^h) \right) = 0 \qquad (2.5.5)$$

Since the c_j's are arbitrary it follows that the term in parentheses must vanish. Thus, for $A = 1, \ldots, n_{eq}$,

$$\sum_{j=1}^{n_{eq}} a(R_i, R_j) = L(R_i) - a(R_{i,} g^h) \qquad (2.5.6)$$

Proceeding to define

$$K_{ij} = a(R_{\mathrm{i}}, R_j), \qquad (2.5.7)$$
$$F_i = L(R_i) - a(R_{\mathrm{i}}, g^h), \qquad (2.5.8)$$

and

$$\mathbf{K} = [K_{ij}], \qquad (2.5.9)$$
$$\mathbf{F} = \{F_i\}, \qquad (2.5.9)$$
$$\mathbf{d} = \{d_i\} \qquad (2.5.11)$$

for $i, j = 1 \ldots, n_{eq}$, we can rewrite *(2.5.6)*as the matrix problem: $\boxed{\mathbf{Kd} = \mathbf{F}}$ *(2.5.12).*

K is known as the ***stiffness matrix*** and vectors *d, F* as *displacement* and *force vector* due to the implication of FEA on structural analysis.

Solving *(2.5.12)* for the d_i's for $i = 1, \ldots, neq$ as

$$\mathbf{d} = \mathbf{K}^{-1}\mathbf{F} \qquad (2.5.13)$$

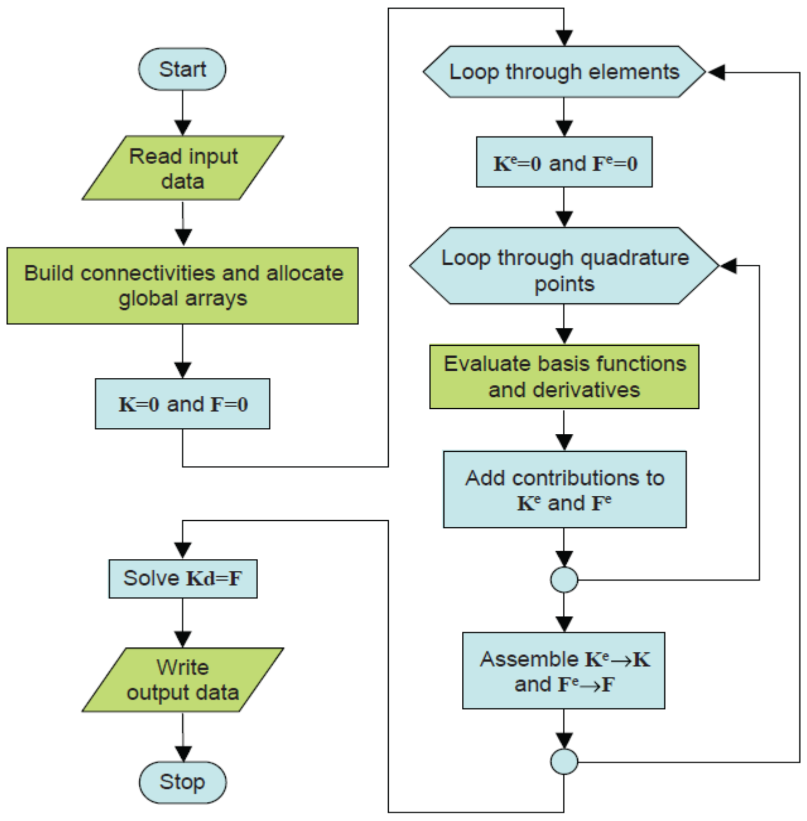
and inserting them back into *(2.5.4),* we can finally obtain our approximate solution.

2.5.1 Assembling the system

So far we have viewed the finite element method simply as a particular Galerkin approximation procedure applied to the weak statement of the problem in question. What makes what we have done a finite element procedure is the character of the selected basis functions; particularly their piecewise smoothness and local support. This is the mathematical point of view; it is a global point of view in that the basis functions are considered to be defined everywhere on the domain of the BVP. The global viewpoint is useful in establishing the mathematical properties of the finite element method.

Now we wish to discuss another point of view called the local, or element, point of view.. This viewpoint is the traditional one in engineering and is useful in the computer implementation of the finite element method and in the development of finite elements. The stiffness matrix K is a sparse matrix due to the local support of the NURBS basis over the elements. Thus, for many combinations of i and j $K_{ij}(R_i,R_j)=0$. This way we can take advantage of that fact in order to reduce the amount of work needed to solve the algebraic system.

The process of building the global stiffness matrix and force vector is called ***assembly***. Instead of looping through all of the global shape functions, taking global integrals to build **K** one entry at a time, we will loop through the elements, building element stiffness ($\mathbf{K}^e$) matrices and force vertices($\mathbf{F}^e$) as we go *(see* **Fig.2.4***)*. Every entry of each of these dense element stiffness matrices will then be added to the appropriate spot in the global stiffness matrix. In this way, we need not expend effort integrating functions over regions in which we know *a priori* that they are zero.



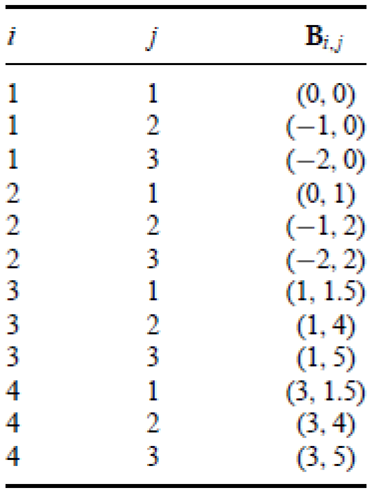
Start
Read input data
Build connectivities and allocate global arrays
K=0 and F=0
Solve Kd=F
Write output data
Stop
Loop through elements
K^e=0 and F^e=0
Loop through quadrature points
Evaluate basis functions and derivatives
Add contributions to K^e and F^e
Assemble K^e→K and F^e→F

Fig. 2.5

At next, we will introduce the ***connectivity arrays*** through an example, in order to obtain a clear perception of their use during the assembly process. The connectivity arrays are used to link every local shape function number to a global shape number.

2.5.2 Connectivity Arrays

Let us consider a specific example of a biquadratic ($p = q = 2$) surface formed from knot vectors $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$ and $H = \{0, 0, 0, 1, 1, 1\}$, with control points listed in **TABLE 1**, resulting in the control net and mesh shown in **Fig. 2.6**.



i	j	$\mathbf{B}_{i,j}$
1	1	(0, 0)
1	2	(−1, 0)
1	3	(−2, 0)
2	1	(0, 1)
2	2	(−1, 2)
2	3	(−2, 2)
3	1	(1, 1.5)
3	2	(1, 4)
3	3	(1, 5)
4	1	(3, 1.5)
4	2	(3, 4)
4	3	(3, 5)

TABLE 1

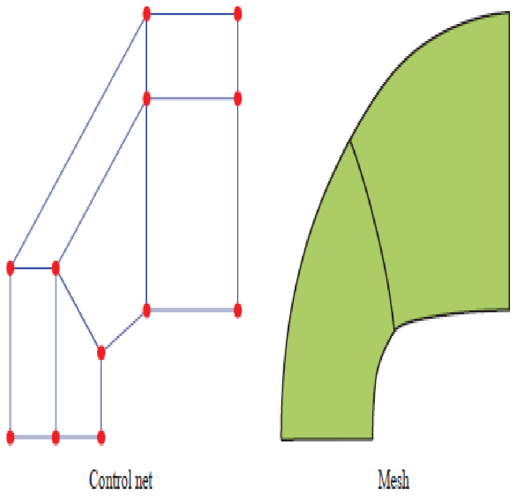


Fig. 2.6

The INC array

For higher-dimensional NURBS objects, it is very convenient to introduce the concept of ***NURBS coordinates***. Examining the index space, which uniquely identifies each knot and discriminates among knots having multiplicity more than one, in **Fig. 2.7** we can see that the NURBS coordinates of any vertex in the mesh are simply the *indices* of the knots that define it. For example, the vertex created by the intersection of the knot lines corresponding to ξ_3 and η_2 has NURBS coordinates (3, 2). Note that this is the vertex at which the support of the blue function begins. In fact, this is how we will most frequently use NURBS coordinates: to identify the knots at which the support of a function begins.

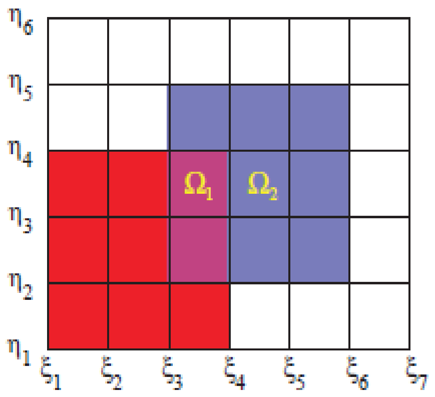


Fig. 2.7

This leads us to a natural scheme for the global numbering of basis functions. If there are n functions in the ξ -direction and m functions in the η-direction, then define

$$I = n(j-1) + i \qquad (A.1)$$

such that the global bivariate function $\tilde{R}_I(\xi,\eta)$ is the tensor product of univariate functions R_i (ξ) and S_j (η). We define the INC ("NURBS coordinates") array such that given a global basis function number and a parametric direction, it returns the index of the one-dimensional basis function in the specified direction that was used to build the global function. Because the support of any one-dimensional NURBS function R_i (ξ) is $[\xi_i, \xi i_{+p+1}]$, we can also interpret the INC array as relating the global basis function number and the specified parametric direction with the index of the knot in the appropriate knot vector at which the support of the function begins. Thus, with $\tilde{R}_I(\xi,\eta) = R_i(\xi)S_j(\eta)$ we have

$$i = \text{INC}(I, 1) \text{ and } j = \text{INC}(I, 2). \qquad (A.2)$$

Turning our attention to Fig. 2.7 and noting that $n = 4$, $p = 2$, $m = 3$, and $q = 2$, we have the INC array given in Table 2. Thus we see that the red function is $\tilde{R}_1(\xi,\eta)$ and has NURBS coordinates (1, 1), while the blue function is $\tilde{R}_7(\xi,\eta)$ with NURBS coordinates (3, 2). The NURBS coordinates are required by many routines, such as basis function evaluation, that explicitly utilize the knot vectors.

I*(global function number)*

INC	1	2	3	4	5	6	7	8	9	10	11	12
1 *(ξ -coordinate)*	1	2	3	4	1	2	3	4	1	2	3	4
2 *(η- coordinate)*	1	1	1	1	2	2	2	2	3	3	3	3

TABLE 2

The IEN array

Let us call the connectivity array IEN. For each element number e from $1, \ldots, n_{el}$, and local function number $\tilde{\imath}$ from $1, \ldots, n_{en}$, there is a global function number I from $1, \ldots, n_{eq}$ such that $\text{IEN}(\tilde{\imath}, e) = \text{I}$. That is, local function $\tilde{R}_{\tilde{\imath}}$ of element e and global function $\tilde{R}_I$ are exactly the same. This allows us to build the global stiffness matrix from a sequence of local ones. Similarly, the global force vector **F** is assembled from the local force vectors $\mathbf{F}^e$. Along the way, we are only performing integration on functions that are non-zero.

The concept of NURBS coordinates provides us with an easy way to determine which functions have support in a given element. First, let us assign element numbers. Knowing that we are using open knot vectors, the number of elements in the ξ-direction is $n - p$; similarly, in the η-direction we have $m - q$ elements (note that due to the possibility of repeated *internal* knots, some of these elements may have zero measure in the parametric domain; this scheme does not, however, apply element numbers to the knot spans that are known *a priori* to have zero measure due to the use of open knot vectors). Consider an element $e = [\xi_i, \xi i_{+1}] \times [\eta_j, \eta_{j+1}]$, where $p + 1 \leq i \leq n$ and $q + 1 \leq j \leq m$. A natural numbering scheme is to assign the element number

$$e = (j - q - 1)(n - p) + (i - p). \qquad (A.3)$$

Thus, the "lower, left-hand corner" of element e has NURBS coordinates (i, j). See Fig. 2.8 . Any function of the form $R_\alpha(\xi)S_\beta(\eta)$ for integers α and β such that $i - p \leq \alpha \leq i$ and $j - q \leq \beta \leq j$ has its support on element e . Thus, the total number of local basis functions is $n_{en} = (p + 1)(q + 1)$. Let us assign local function number 1 to the function with NURBS coordinates (i, j). We then assign the remaining local numbers, working backwards in ξ first, followed by η. Thus, with I as in *(A.1)*, the global numbers of the first $p + 1$ local functions are $I, I - 1, \ldots, I- p$.

The function $\tilde{R}_{I-p-1}$ does not have support in the element, so we move a row in the η-direction and continue numbering with $I - n, I - n - 1, \ldots, I - n - p$. Again, we must move to the next row and continue with $I - 2n, \ldots, I - 2n - p$. This continues until we reach our last set of function numbers, $I - qn, \ldots, I - qn - p$, at which point we are finished.

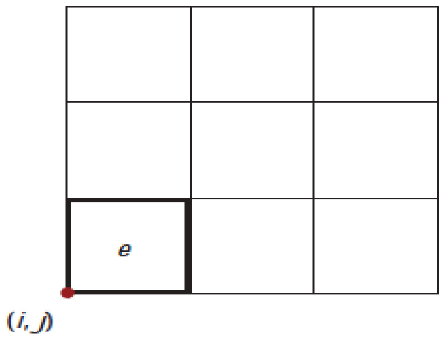


Fig. 2.8

The IEN ("***element nodes***") array connects these global function numbers to their local ordering on the element. In finite elements, global basis function numbers are identified with global node numbers, and local basis function numbers are identified with local node numbers. It is for this reason that the IEN array is referred to as the "element nodes" array. Even though this designation no longer applies in the present case, we retain the name. Given the element number, *e*, and the local basis function number, *b*, the corresponding global basis function number, *I*, is given by

$$I = \mathrm{IEN}(\tilde{\imath}\ , e). \qquad (A.4)$$

Thus, if $I = \mathrm{IEN}(1, e)$ as in the previous paragraph, then we have, for example, $I - 1 = \mathrm{IEN}(2, e)$, $I - n = \mathrm{IEN}(p + 2, e)$, and $I - qn - p = \mathrm{IEN}((p + 1)(q + 1), e)$.The IEN array corresponding to the mesh in **Fig.s 2.7** and **2.9** is shown in **TABLE 3**.

Observe that the blue function, $\tilde{R}_7$, which has support in both elements, has local number $\tilde{\imath} = 4$ on element $e = 1$ and also local number $\tilde{\imath} = 5$ on element $e = 2$. That is, $\mathrm{IEN}(4, 1) = \mathrm{IEN}(5, 2) = 7$. The red function, $\tilde{R}_1$, has support in only the first element. The only entry corresponding to it is $\mathrm{IEN}(9, 1) = 1$.

I (local basis function number)

IEN	1	2	3	4	5	6	7	8	9
e=1	11	10	9	7	6	5	3	2	1
e=2	12	11	10	8	7	6	4	3	2

TABLE 3

The ID array

Now that we have numbered all the basis functions used to construct our geometry, established a local numbering convention, and collected the connectivity information relating the two points of view, we need to turn our attention to the specific requirements of analysis. Recall that in the beginning of the chapter we assumed a numbering of the global functions such that each function with support on the Dirichlet boundary had a higher index than any without support on that boundary. This was convenient for the exposition of finite element concepts, but we have no reason to expect it to be compatible with the numbering system proposed in the previous section. In general, we have one equation corresponding to each function that does not have support on the Dirichlet boundary. This assumes Dirichlet boundary conditions are satisfied *strongly*). We must construct a mapping between the global index of those functions, and an equation number between 1 and n_{eq} , the total number of equations (which, in the scalar case, is less than or equal to the total number of functions). This information is stored in the ID "***destination***" array.

The ID array itself will depend on the specifics of the boundary conditions. Referring to **Fig. 2.9**, assume that we have Dirichlet data prescribed along the edge from (3, 1.5) to (3, 5) in the physical space. We can tell from **Fig. 2.7** that any function $\tilde{R}_I$ such that $INC(I, 1) = 4$ is going to have support on that edge, and thus will *not* have an equation number corresponding to it. Though there are many conventions we might adopt, we simply assign equation numbers in ascending order, assigning 0 to any function with support on the Dirichlet boundary. Thus, we arrive at the ID array shown in **TABLE 4**.

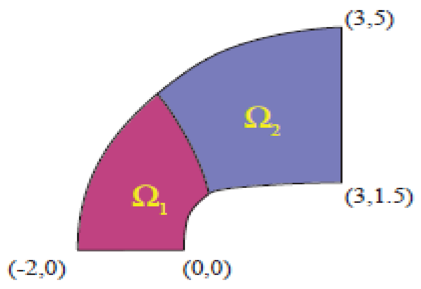


Fig. 2.9

I*(global function number)*

ID	1	2	3	4	5	6	7	8	9	10	11	12
P*(equation number)*	1	2	3	0	4	5	6	0	7	8	9	0

TABLE 4

The LM array

The final connectivity array that we will consider is just a composition of the previous two. The most common form of the **LM** (*Location Matrix*). The LM array may then be constructed from the relation

$$LM(\tilde{\imath}, e) = ID(IEN(\tilde{\imath}, e))$$

Thus, we obtain **TABLE 5:**

I *(local basis function number)*

LM	1	2	3	4	5	6	7	8	9
e=1	1	2	3	4	5	6	7	8	9
e=1	9	8	7	6	5	4	3	2	1

TABLE 5

2.5.3 Evaluation of gradients and Jacobian determinant

After introducing the connectivity arrays we are ready to assemble the matrix equations recall that:

$$K_{ij} = a(R_i, R_j) = \int_{\Omega} (\nabla R_i^T) \nabla R_j d\Omega$$

and

$$F_{\ i} = L(R_i) - a(R_i, g^h) = \int_{\Omega} R_i f d\Omega + \int_{\Gamma_N} R_i h d\Gamma - \int_{\Omega} (\nabla R_i)^T (\nabla g^h) d\Omega$$

The stiffness and force matrices are formulated with respect to the local indices:

$$K = \sum_{e=1}^{n_{el}} K^e \text{ and } F = \sum_{e=1}^{n_{el}} F^e$$

where

$$\mathrm{K}^{\mathrm{e}}_{\tilde{\imath}j} = a(R_{i,} R_j) = \int_{\Omega^e} (\nabla R_i)^{\mathrm{T}} \nabla R_j d\Omega (2.5.13) \qquad (2.5.13)$$

$$F_a^e = \int_{\Omega^e} R_i f d\Omega + \int_{\Gamma_N^e} R_i h d\Gamma - \int_{\Omega^e} \nabla R_i \nabla g^h d\Omega (2.5.14) \qquad (2.5.14)$$

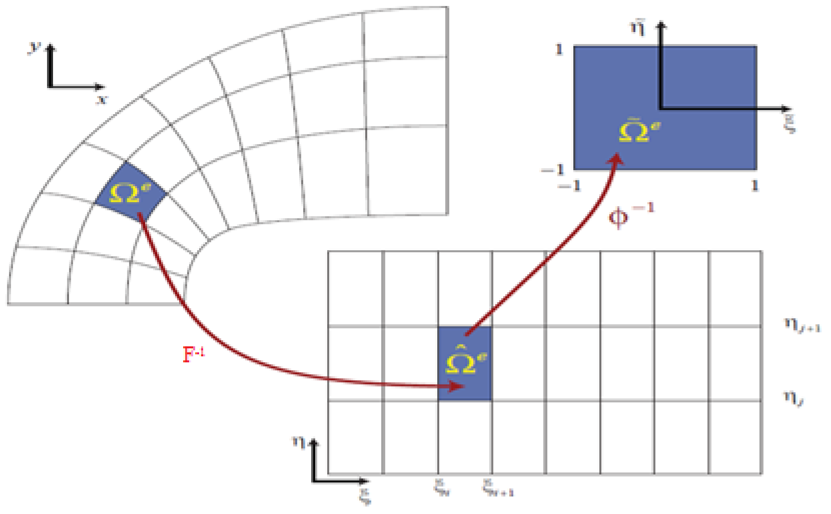


Fig. 2.10

Now that have assembled the system in a way that the calculation takes place in the domain of the element, it's time to approximate the integrals using the Gaussian quadrature rule. Let us consider the affine mapping φ:$\hat{\Omega}^e$ →.$\tilde{\Omega}^e$. Each element in the parameter space is pulled back by the inverse of the affine function to a bi-unit square where the actual integration takes part. The domain $\hat{\Omega}^e$ is called parent element.

In each quadrature point we must estimate the gradients of the basis:

$$\nabla R_i = [\frac{dR_i(\xi,\eta)}{dx} \quad \frac{dR_i(\xi,\eta)}{dy}]$$

The derivatives of R_i with respect to x and y may be evaluated with the aid of the chain rule:

$$\frac{dR_i}{dx} = \frac{dR_i(\xi,\eta)}{d\xi}\frac{d\xi}{dx} + \frac{dR_i(\xi,\eta)}{d\eta}\frac{d\eta}{dx}$$
$$\frac{dR_i}{dy} = \frac{dR_i(\xi,\eta)}{d\xi}\frac{d\xi}{dy} + \frac{dR_i(\xi,\eta)}{d\eta}\frac{d\eta}{dy}$$

It is worthwhile to recast the above relations in the following matrix form:

$$[\frac{dR_i(\xi,\eta)}{dx} \quad \frac{dR_i(\xi,\eta)}{dy}] = [\frac{dR_i(\xi,\eta)}{d\xi} \quad \frac{dR_i(\xi,\eta)}{d\eta}]\begin{bmatrix} \frac{d\xi}{dx} & \frac{d\xi}{dy} \\ \frac{d\eta}{dx} & \frac{d\eta}{dy} \end{bmatrix}$$

The derivatives $\dfrac{dR_i(\xi,\eta)}{d\xi}$ and $\dfrac{dR_i(\xi,\eta)}{d\eta}$ may be explicitly computed. However , the terms in the matrix cannot be directly computed since we do not have explicit expressions *ξ=ξ(x,y)* and *η=η(x,y)* . On the other hand, we do have the inverse relations:

$$x(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,p}(\xi) S_{\xi,q}(\eta) x_{ij}$$

$$y(\xi,\eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,p}(\eta) S_{j,q}(\eta) y_{ij}$$

Here, $x_{i,j}$ and $y_{i,j}$ are the x and y coordinates of the control points of the solution surface. Relations and enable us to calculate the matrix:

$$x_\xi = \begin{bmatrix} \dfrac{dx}{d\xi} & \dfrac{dx}{d\eta} \\ \dfrac{dy}{d\xi} & \dfrac{dy}{d\eta} \end{bmatrix}$$

The matrix is the inverse of the matrix ,i.e.,

$$\begin{bmatrix} \dfrac{d\xi}{dx} & \dfrac{d\xi}{dy} \\ \dfrac{d\eta}{dx} & \dfrac{d\eta}{dy} \end{bmatrix} = (x_\xi)^{-1} = \frac{1}{J} \begin{bmatrix} \dfrac{dy}{d\eta} & -\dfrac{dx}{d\eta} \\ -\dfrac{dy}{d\xi} & \dfrac{dx}{d\xi} \end{bmatrix}$$

where

$$J = \det(x_\xi) = \frac{dx}{d\xi}\frac{dy}{d\eta} - \frac{dx}{d\eta}\frac{dy}{d\xi} \quad .$$

So the following equation is used to estimate the gradients

$$\nabla R_i = [\frac{dR_i(\xi,\eta)}{d\xi} \quad \frac{dR_i(\xi,\eta)}{d\eta}] \frac{1}{J} \begin{bmatrix} \dfrac{dy}{d\eta} & -\dfrac{dx}{d\eta} \\ -\dfrac{dy}{d\xi} & \dfrac{dx}{d\xi} \end{bmatrix}$$

Since we are integrating over the parent element, we should also compute the Jacobian determinant:

$$\tilde{J} = \det\begin{bmatrix} \frac{dx}{d\tilde{\xi}} & \frac{dx}{d\tilde{\eta}} \\ \frac{dy}{d\tilde{\xi}} & \frac{dy}{d\tilde{\eta}} \end{bmatrix} = \det\begin{bmatrix} \frac{dx}{d\xi}\frac{d\xi}{d\tilde{\xi}} & \frac{dx}{d\eta}\frac{d\eta}{d\tilde{\eta}} \\ \frac{dx}{d\xi}\frac{d\xi}{d\tilde{\xi}} & \frac{dx}{d\eta}\frac{d\eta}{d\tilde{\eta}} \end{bmatrix}$$

For an arbitrary element domain $\hat{\Omega}^e = [\xi_{i,}\xi_{i+1}]\times[\eta_j, \eta_{j+1}]$ and thus NURBS coordinates *(i,j)* we calculate $(\xi,\eta) \in \hat{\Omega}^e$ from $(\tilde{\xi},\tilde{\eta}) \in \tilde{\Omega}^e$

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

$$\eta = \eta_i + (\tilde{\eta}+1)\frac{(\eta_{i+1}-\eta_i)}{2} = \frac{((\eta_{\imath+1}-\eta_i)\,\tilde{\eta} + (\eta_{\imath+1}+\eta_i))}{2}$$

The Gaussian quadrature rule seems to be effective even though NURBS are no necessary polynomials.

2.6 Boundary Conditions

This chapter examines the implementation of the boundary conditions of BVPS using the isogeometric approach. The most significant boundary conditions are those presented in *(2.1.2)* and *(2.1.3)*, that is the *Dirichlet* & *Neumann* conditions, respectively.

Dirichlet conditions are often referred to as "essential boundary conditions", due to the fact that during the variational formulation of the problem they are directly built into the solution space since the Galerkin formulation doesn't offer the chance to impose them. The lifting function presented in *chapter 2.5* in most cases is just an approximation of *g*. This way of imposing the Dirichlet conditions is known as strong imposition of boundary condition.

If *g=0* the we have "*homogeneous Dirichlet conditions*". In this case, the lifting function g^h can be built to the solution space by setting $g_i=0$ for i=$n_{eq}+1$, ..., n_{np} , since $g_{-1}, \ldots, g_{neq}=0$. If g is a constant then we set g_i to that constant since the partition of unity of the NURBS basis can guarantee the safe implementation of the boundary condition. Other functions, such as linear functions, that exist in the NURBS space can be set by setting appropriately the control points.

If *g* doesn't exist in the NURBS space then the lifting function becomes an approximation such that $g|_{\Gamma_D} \approx g$. In classical finite element analysis the elements interpolate *g* at the nodes. However in isogeometric analysis we can interpolate g using the appropriate control points but as the basis itself is non interpolatory this results in a slightly deformed g^h . Frequently this yields better results than FEA *(see* **Fig. 1.6(b)**).

An alternative approach is done via the weak imposition method. Here the weighting function *w* doesn't satisfy the *homogeneous Dirichlet condition*, i.e. we don't enforce $w|_{\Gamma_D} = 0$. This results to the following variational form of the BVP problem:

$$-\int_{\Omega} \nabla w \nabla u d\Omega + \int_{\Gamma} w \nabla u \cdot n d\Gamma + \int_{\Omega} w f d\Omega = 0 \tag{2.6.1}$$

To ensure the convergence of the method and to restrict the error we add two terms resulting in the following formulation:

$$\begin{aligned} &-\int_{\Omega} \nabla w \nabla u d\Omega + \int_{\Gamma} w \nabla u \cdot n d\Gamma + \int_{\Omega} w f d\Omega \\ &+ \int_{\Gamma_D} \gamma (\nabla w \cdot n)(u-g) d\Gamma + \int_{\Gamma_D} \frac{C}{h} (\nabla w \cdot n)(u-g) d\Gamma = 0 \end{aligned} \tag{2.6.2}$$

where h is the element length scale (for a full definition see *chapter 2.7*) , C is a constant and $\gamma = \pm 1$. Notice that the exact solution for *(2.6.1)* is also a solution for *(2.6.2)*. For more information on the weakly imposition of the Dirichlet boundary condition see [18]
In addition, the imposition of the *Neumann conditions* is straight forward. Integration by parts introduces in a "natural" way a boundary integral over Γ_N. Using condition *(2.1.3)* we replace $\nabla u \cdot n$ with *h* resulting in equation *(2.2.7). Neumann conditions* are also known as "*natural boundary conditions*".

2.7 Error estimates

Recall from above that a Sobolev space of order *r* is defined by

$$H^r(\Omega) = \{u | D^a u \in L^2(\Omega), |a| \le r\} \tag{2.7.1}$$

The norm associated with H^r (*Ω*) is given by

$$\|u\|_r^2 = \sum_{|a| \le r} (D^a u) \cdot (D^a u) dx \tag{2.7.2}$$

The finite element function spaces are endowed with an approximation property that may be stated as follows. Given a function u∈H^r, then there exists a function $u^h \in S^h$ (sometimes called the ***interpolate***) such that

$$\|u - u^h\|_m \le C h^{\beta} \|u\|_r \tag{2.7.3}$$

where $\|\cdot\|_m$ and $\|\cdot\|_r$ are the norms corresponding to Sobolev spaces $H^m(\Omega)$ and $H^r(\Omega)$, C is a constant independent of u and h, $\beta = min(p + 1 - m, r - m)$, p is the polynomial degree appearing in the element shape functions, and h is the mesh parameter, a scalar characterizing the refinement of the finite element mesh. The mesh parameter may be taken to be the diameter of the largest element in the mesh *(see* **Fig. 2.11** *).* A collection of finite element spaces $\{\mathcal{Q}_h\}$ (i.e.. meshes parameterized by h) possessing the approximation property .is called k, m-regular.

The ***order of convergence***, β, expresses how the error changes under refinement of the mesh. In particular, if we use h-refinement to bisect each of the elements in the mesh (*i.e.*, h is replaced with $h/2$), we would expect the error to decrease by a factor of $(1/2)^\beta$. As long as p+ 1 and r are greater than m, we have optimal convergence in the H^m norm.

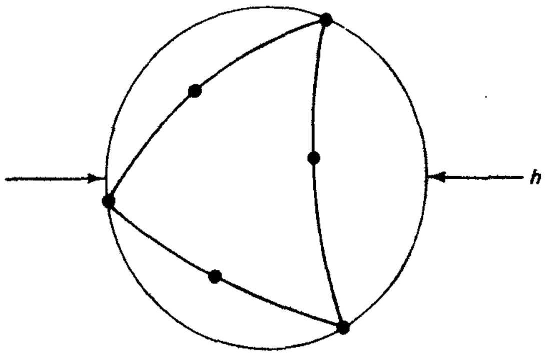


Fig. 2.11

Let Π_m *be a projector* from the Sobolev space H^m to the solution space spanned from the basis used, then if :

$$\eta^h = \Pi_m u$$

such that

$$\left\|u-\eta^h\right\|_m \le \left\|u-u^h\right\|_m \quad \forall u^h \in S^h,$$

then η^h is called **optimal interpolate**.

In order to conclude to inequality we first have to bound the term $\left\|u-\eta^h\right\|_m$ over each element . To obtain the global result we sum over all elements. Once this is done we compare the approximate solution u^h ,which is obtained after applying the Galerkin method with the optimal interpolate. The two results yield inequality. This inequaliity ensures us that ,at least up to a constant , the Galerkin method gives the optimal result. In the case of NURBS though it is not that simple to obtain such a result since we are facing several difficulties.

The approximation properties of the rational basis are more complex than those of standard polynomials. The weights depend only by the geometry, so the approximation over a field over that geometry is not easily achieved and cannot be adjusted to improve the result.

Assume that d knot vectors Ξ_α with $1 \leq \alpha \leq d$, are given. Let $(0, 1)^d \subset \mathbb{R}^d$ be an open parametric domain, referred to as a patch. Associated with the knot vectors Ξ_α there is a mesh $\mathcal{Q}$, that is, a partition of $(0, 1)^d$ into d-dimensional open knot spans, or elements,

$$\mathcal{Q} \equiv \mathcal{Q}(\Xi_1,...,\Xi_\mathrm{d}) := \{Q = \otimes_{a=1}^{d}(\xi_{i_a,a}, \xi_{i_a+1,a}) \big| Q \neq \emptyset, p_a + 1 \leq i_a \leq n_a - 1\}$$

The tensor product B-spline basis functions are defined as:

$$\mathcal{S}=\mathcal{S}(\Xi_1,....,\Xi_d; p_1,..., p_d) := span\{N_{i,a}\}_{i=1,...,n_a}^{n_1,...,n_d}$$

To a (non empty) element $Q = \otimes_{a=1}^{d}(\xi_{i_a-m_a+1,a}, \xi_{i_a+m_a,a}) \in \mathcal{Q}$ we associate $\tilde{Q} \subset (0,1)^d$ defined as

$$\tilde{Q} = \otimes_{a=1}^{d}(\xi_{i_a-m_a+1,a}, \xi_{i_a+m_a,a})$$

The set $\tilde{Q}$ will be referred to as the support extension of Q, since it is the union of the supports of basis functions whose support intersects Q.

The NURBS space on the patch, denoted by, is

$$\mathcal{N}=\mathcal{N}(\Xi_1,....,\Xi_d; p_1,..., p_d; w) := span\{R_{i_1....i_d}\}_{i=1,...,n_a}^{n_1,...,n_d}$$

The NURBS geometrical map F is given by equation *(2.4.1)*. F is invertible, with smooth inverse. Each element $Q \in \mathcal{Q}$ is mapped into an element

$$K = F(Q) = \{F(\xi) \,|\, \xi \in Q\})$$

and analogously $\tilde{Q}$, the support extension of Q, is mapped into

$$\tilde{K} = F(\tilde{Q}) .$$

We then introduce the mesh $\mathcal{K}$ in the physical domain

$$\mathcal{K} := \{\mathrm{K} = F(Q) \,|\, Q \in \mathcal{Q}\}$$

and the space $\mathcal{V}$ of NURBS on (which is the push-forward of the space $\mathcal{N}$ of NURBS on the patch)

$$\mathcal{V} \equiv \mathcal{V}(p_1,....,p_a) := span\{R_{i_1...i_d} \circ F^{-1}\}_{i_1=1,....,i_d=1}^{n_1,...,n_d}$$

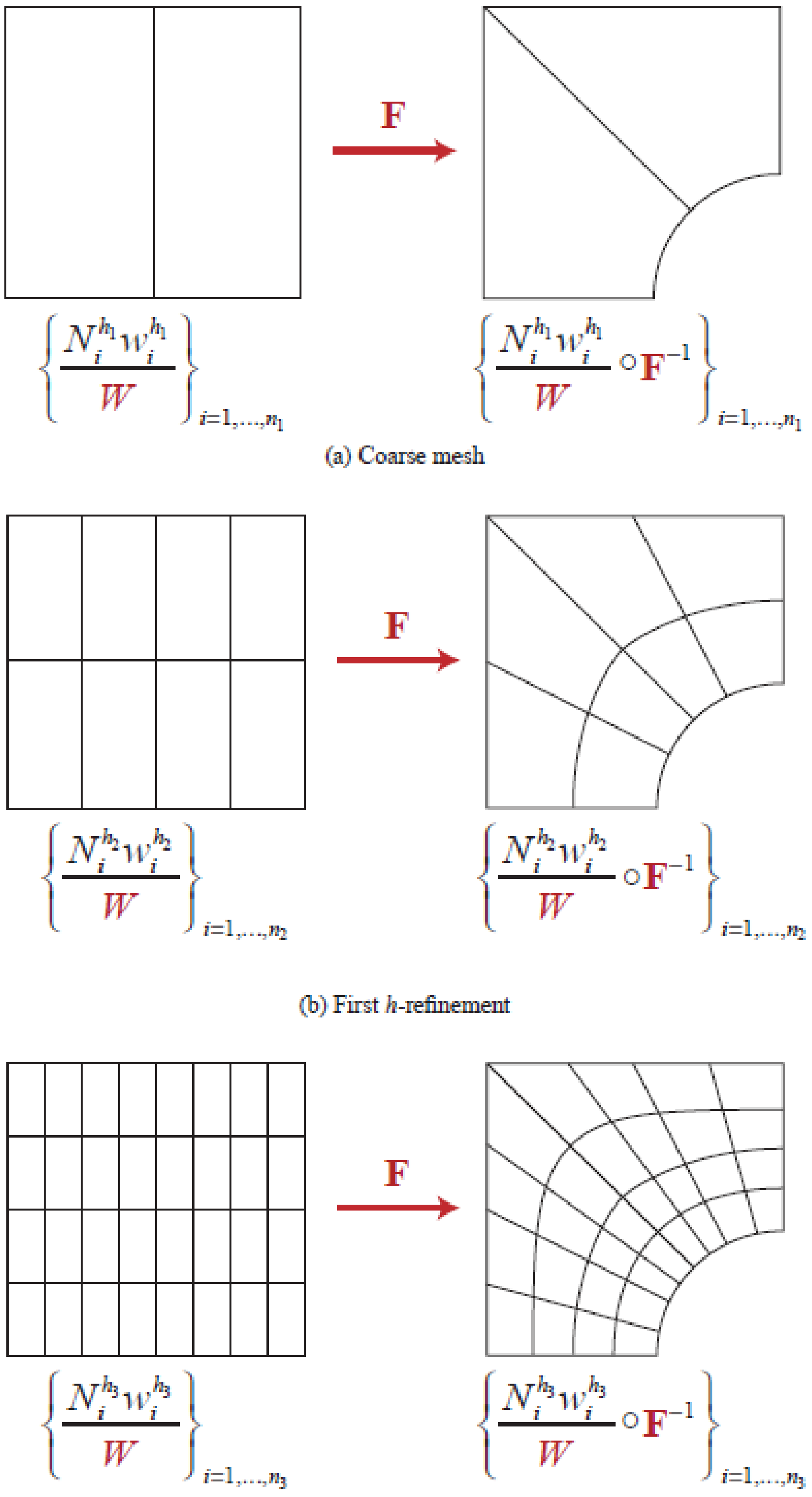
We consider now a family of meshes $\{\mathcal{Q}_h\}_h$ on $(0, 1)^d$, where h denotes the family index, representing the global mesh size:

$$h = \max\{h_Q | \; Q \in \mathcal{Q}_h\}.$$

The family of meshes is assumed to be shape regular, that is, the ratio between the smallest edge of $Q \in \mathcal{Q}_h$ and its diameter h_Q is bounded, uniformly with respect to Q and h. This implies that the mesh is locally quasi-uniform—the ratio of the sizes of two neighboring elements is uniformly bounded. Following the construction in the previous section, associated with the family of meshes $\{\mathcal{Q}_h\}_h$ we introduce the families of meshes on the physical domain $\{\mathcal{K}_h\}_h$, and the spaces $\{\mathcal{S}_h\}_h$,$\{\mathcal{N}_h\}_h$, $\{\mathcal{V}_h\}_h$.

In practical applications, the geometry of the physical domain *Ω* is frequently described on a mesh of relatively few elements, while the computation of an approximate solution to the problem is performed on a refined mesh (fine enough to achieve desired accuracy). Therefore, we assume that there is a coarsest mesh $\mathcal{Q}_{ho}$ in the family $\{\mathcal{Q}_h\}_h$, of which all the other meshes are a refinement, and that the description of the geometry is fixed at the level of $\mathcal{Q}_{ho}$. This means that the weighting function *W*(see equation *(1.4.4)*) and the geometrical map *F* are assigned in $\mathcal{S}_{ho}$ and $(\mathcal{N}_{ho})_d$, respectively, and are the same for every h. When the mesh and the spaces are refined , the weights $w_{i1...id}$ are selected so that w stays fixed , in a similar way, the control points $B_{i1...id}$ are adjusted such that *F* remains unchanged. Thus the geometry and its parameterization are held fixed in the refinement process. See **Fig.2.12** for an illustration of this idea.

This fact leads to the decision to pull back our solution we wish approximate to the parametric domain $\widehat{\Omega}$ via the inverse mapping of F, defining this way $\hat{u}=u\circ F^{-1}$: $\widehat{\Omega} \rightarrow \mathbb{R}^d$. Then recalling the fact that the rational basis in $\mathbb{R}^d$ is the projective transformation of a B-spline basis in $\mathbb{R}^{d+1}$ we define the lifting function $\tilde{u} : \{W\hat{u}, W\} : \hat{\Omega} \rightarrow \mathbb{R}^{d+1}$.Thus the examination of the rational basis is now "moved" to the unit cube , rendering their manipulation much less complex than in the case of the physical domain


F
$\left\{ \frac{N_i^{h_1} w_i^{h_1}}{W} \right\}_{i=1,\dots,n_1}$
$\left\{ \frac{N_i^{h_1} w_i^{h_1}}{W} \circ \mathbf{F}^{-1} \right\}_{i=1,\dots,n_1}$
(a) Coarse mesh
F
$\left\{ \frac{N_i^{h_2} w_i^{h_2}}{W} \right\}_{i=1,\dots,n_2}$
$\left\{ \frac{N_i^{h_2} w_i^{h_2}}{W} \circ \mathbf{F}^{-1} \right\}_{i=1,\dots,n_2}$
(b) First *h*-refinement
F
$\left\{ \frac{N_i^{h_3} w_i^{h_3}}{W} \right\}_{i=1,\dots,n_3}$
$\left\{ \frac{N_i^{h_3} w_i^{h_3}}{W} \circ \mathbf{F}^{-1} \right\}_{i=1,\dots,n_3}$

(c) Second *h*-refinement

<u>Fig. 2.12</u>

The functions in $\mathcal{S}$ are piecewise polynomials of degree p_a in the a coordinate. The regularity of each d-dimensional basis function $N_{i1\ldots id}$ across the element boundaries depends on the regularity of the one-dimensional basis functions $N_{i_a,a}$ for $1 \le a \le d$, at the corresponding knots. This fact constitutes the second difficulty when studying error approximation, since each function has support over many elements and the continuity can vary from one boundary to the other. Given two adjacent elements Q_1 and Q_2 we denote by m_{Q_1,Q_2} the number of continuous derivatives across their common (d−1)-dimensional face $\partial Q_1 \cap \partial Q_2$; $m_{Q_1,Q_2} = -1$ is associated with a discontinuity. For the subsequent analysis, we introduce the following "bent" Sobolev space of order $m \in \mathbb{N}$:

$$\mathcal{H}^m := \left\{ \begin{array}{l} u \in L^2((0,1)^d) : 1) u\big|_Q \in H^m(Q), \forall Q \in \mathbb{Q} \\ 2) \nabla^k (u\big|_{Q_1}) = \nabla^k (u\big|_{Q_2}) \quad on \quad \partial Q_1 \cap \partial Q_2, \forall k \in \mathbb{N} \quad with \quad 0 \le k \le \min\{m_{Q_1} m_{Q_2}, m-1\}, \forall Q_1, Q_2 \quad \partial Q_1 \cap \partial Q_2 \neq \emptyset \end{array} \right\}$$

where $\nabla^k u$ denotes the (k-linear) *k*-th order partial derivative operator, while $\nabla^0 u=u$. This is a well-defined Hilbert space, endowed with the seminorms:

$$|u|^2_{\mathcal{H}^i} := \sum_{Q \in Q} |u|^2_{H^i(Q),} \quad 0 \le i \le m$$

and norm:

$$\|u\|^2_{\mathcal{H}^m} := \sum_{i=0}^{m} |u|^2_{\mathcal{H}^i} \, .$$

We also need the restriction of Hm to a given support extension $\widetilde{Q}$, which is denoted by $\mathcal{H}^m(Q) := \{ u_{|\tilde{Q}} \, | u \in \mathcal{H}^m \}$ and endowed with the seminorm:

$$\|u\|^2_{\mathcal{H}^i(\tilde{Q})} := \sum_{\substack{Q' \in Q \\ Q' \cap \tilde{Q} \neq \emptyset}} |u|^2_{H^i(Q')}$$

and norm:

$$\|u\|^2_{\mathcal{H}^m(\tilde{Q})} := \sum_{i=0}^{m} |u|^2_{\mathcal{H}^i(\tilde{Q})} \, .$$

The bent Sobolev spaces are intermediate in continuity between standard Sobolev spaces and so-called ***"broken" Sobolev spaces*** utilized in the analysis of discontinuous Galerkin methods.

In what follows, we will denote by C a positive, dimensionless constant, possibly different at each occurrence, which depends only on the space dimension d, on the polynomial degrees p_a, $a = 1, 2, ..., d$, and on the shape regularity of the mesh family $\{\mathcal{Q}_h\}_h$. Observe that the p_a are considered fixed, since we only address h-refinement in this chapter . We will denote by C_{shape} another positive, dimensionless constant, possibly different at each occurrence, which may also depend on the geometry of but still not on h. Specifically, C_{shape} depends on the shape of Ω , but not on its size; therefore C_{shape} is by assumption homogeneous of order 0 with respect to W and ∇F, where ∇F is the matrix of partial derivatives of the coordinate components of F, that is, C_{shape} is invariant if W and ∇F are scaled by a multiplicative factor. Actually, C_{shape} only depends on the dimensionless functions $W / \|W\|_{L^\infty(\Omega)}$ and $\nabla F / \|\nabla F\|_{L^\infty(\Omega)}$ .Furthermore, if C_{shape} appears in a local estimate, then it depends only on the local values of W and ∇F.

Let p be defined as $p := \min_{1 \le a \le d}\{p_a\}$ .The following lemma is shown in

Lemma 1: Let k and l be integer indices with $0 \le k \le l \le p + 1$. Given $Q \in \mathcal{Q}_h$, $u \in \mathcal{H}_h^l$ there exists an $s \in S_h$ such that :

$$|u - s|_{\mathcal{H}_h^k(\tilde{Q})} \le C h^{l-k} |u|_{\mathcal{H}_h^l(\tilde{Q})} \quad (2.7.4)\ .$$

Now we will introduce a projector on the spline space $\mathbb{S}_h$, defined as:

$$\Pi_{\mathcal{S}^h} u := \sum_{i_1=1,\dots,i_d=1}^{n_1,\dots,n_d} (\lambda_{i_1 \dots i_d} u) N_{i_1 \dots i_d}, \quad \forall u \in L^2((0,1)^d)$$

where the $\lambda_{i_1 \dots i_d}$ are dual functions:

$$\lambda_{j_1 \dots j_d} N_{i_1 \dots i_d} = 1, \quad if \quad j_a = i_a, \forall 1 \le a \le d$$
$$\lambda_{j_1 \dots j_d} N_{i_1 \dots i_d} = 0, \quad if \quad j_a \ne i_a\ .$$

This projector has the following *properties*:

i. $\Pi_{\mathcal{S}^h} s = s, \forall s \in \mathcal{S}^h (spline \quad reserving)$

ii. $\|\Pi_{\mathcal{S}^h} u\|_{L^2(Q)} \le C \|u\|_{L^2(\tilde{Q}),} \forall u \in L^2((0,1)^d), \forall Q \in \mathcal{Q}_h (stability)$

It can be shown that if $\Pi_{\mathcal{S}^h} : L^2((0,1)^d) \to \mathcal{S}_h$ is a projector with the above properties then for all $Q \in \mathcal{Q}_h$ the following inequality is preserved:

$$\left| u - \Pi_{\mathcal{S}_h} u \right|_{H^k(Q)} \le C h^{l-k} \left| u \right|_{\mathcal{H}^l(\tilde{Q})}, \forall u \in \mathcal{H}_h^l(\tilde{Q}) \cap L^2((0,1)^d) (2.7.5)$$

Using this inequality and along with the definition of the NURBS projector we can derive the approximation properties of the NURBS space on the patch $(0,1)^d$. The NURBS projector is defined as:

$$\Pi_{\mathcal{N}_h} u := \frac{\Pi_{\mathcal{S}_h}(Wu)}{W}.$$

With inequalities *(2.7.4)* and *(2.7.5)* in hand we find the approximation of NURBS in the parametric domain:

$$\left| u - \Pi_{\mathcal{N}_h} u \right|_{H^k(Q)} \le C_{shape} h_Q^{l-k} \left| u \right|_{\mathcal{H}^l(\tilde{Q})}, \forall u \in \mathcal{H}_h^l(\tilde{Q}), Q \in \mathcal{Q}_h (2.7.6), 0 \le k \le l \le p+1$$

in the physical domain

$$\left| u - \Pi_{\mathcal{V}_h} u \right|_{H^k(Q)} \le C_{shape} h_K^{l-k} \sum_{i=0}^{l} \left\| \nabla F \right\|_{L\infty(\tilde{Q})}^{i-l} \left| u \right|_{H^i(\tilde{K})}, \forall u \in H^l(\tilde{K}) \cap L^2(\Omega), 0 \le k \le l \le p+1 \quad (2.7.7)$$

where h_K is the element size on the physical domain defined as:

$$h_K = \left\| \nabla F \right\|_{L\infty(Q)} h_Q$$

Thus, by *(2.7.7)* we have the global error estimate:

$$\sum_{K \in \mathcal{K}_h} \left| u - \Pi_{\mathcal{V}_h} u \right|_{\mathcal{K}_h^k(K)}^2 \le C_{shape} \sum_{K \in \mathcal{K}_h} h_K^{2(l-k)} \sum_{i=0}^{l} \left\| \nabla F \right\|_{L\infty(F^{-1}(K))}^{2(i-l)} \left| u \right|_{H^i(K)}^2, \forall u \in H^l(\Omega), 0 \le k \le l \le p+1 (2.7.8)$$

With inequalities *(2.7.7)* and *(2.7.8)* we can understand that that the NURBS space $\mathcal{V}_h$ on the physical domain delivers the optimal rate of convergence, as for the classical finite element spaces of degree p. This result is independent of the order of continuity the mesh possesses. The bisection of NURBS element (cutting the mesh parameter from *h* to *h/2*) requires much less degrees of freedom while maintaining *p-1* continuity than bisecting the elements of a FEA mesh. This means that NURBS converge at the same rate with FEA polynomials, while remaining much more efficient.

Chapter 3: GeoPDEs

Throughout this chapter we will study the convergence rate of isogeometric analysis. To do this a recently developed tool was used, by the name of **GeoPDEs**. GeoPDEs is fully compatible with MATLAB. The various shapes were designed using the NURBS toolbox.

The general idea is to find an approximate solution for the general boundary value problem:

$$k(x)\Delta u + f = 0$$

$$u = g \text{ on } \Gamma_D$$

$$\nabla u \cdot \mathbf{n} = h \text{ on } \Gamma_N$$

The BVP will be solved using the Galerkin procedure, through the isoparametric and non isoparametric concept. The NURBS toolbox allows us to construct the geometry under study and geopde defines the solution space used .The data structures used by GeoPDEs are mainly ***mesh, geometry*** and ***space*** along with some ***operators***, which we will be discussed later on.

Let us recall the variational formulation of the above problem is:

$$\int_\Omega k(x)\nabla w\cdot\nabla u dx = \int_\Gamma w h d\Gamma + \int_\Omega w f dx \quad \forall w \in H^1_{0,\Gamma_D}$$

The variational formulation of the discrete problem is:

$$\int_\Omega k(x)\nabla w_h\cdot\nabla u_h dx = \int_\Gamma w_h h d\Gamma + \int_\Omega w_h f dx \quad \forall w \in V^h (3.1.1)$$

where V^h is the discrete space, formed by NURBS functions, defined as:

$$V^h = \{u_h \in H^1_{0,\Gamma_D} : u_h = \hat{u}_h \circ F^{-1}, \hat{u}_h \in \hat{V}^h\}$$

where F^{-1}:$\Omega \rightarrow \hat{\Omega}$ is the proper pullback function mapping the physical domain to the parametric one, and $\hat{V}^h$ is the discrete space of the parametric domain .Let us assume that N_h= dim(V^h)=dim($\hat{V}^h$) is the dimension of our finite dimensional and $\{\hat{u}_i\}_{i=1}^{N_h}$ is a basis for $\hat{V}^h$. Due to the parameterization of F, the basis of V^h is defined as:

$$\{u_i = \hat{u}_i \circ F^{-1}\}_{i=1}^{N_h}$$

This way the weighting and trial solutions can be written in the form $w_h = \sum_{j=1}^{N_h} d_j u_j = \sum_{j=1}^{N_h} d_j \hat{u}_j \circ F^{-1}$ and $u_h = \sum_{i=1}^{N_h} d_i u_i = \sum_{i=1}^{N_h} d_i \hat{u}_i \circ F^{-1}$ respectively.

Substituting in equation *(3.1.1)* yields:

$$\sum_{j=1}^{N_h} K_{ij} d_j = \int_{\Omega} k(x) \sum_{j=1}^{N_h} d_j \nabla u_i \nabla w_j dx = \int_{\Omega} f w_i dx + \int_{\Omega} h w_i dx = f_i + h_i \quad (3.1.2)$$

for i=1,…,N^h where K_{ij} are the coefficients of the stiffness matrix, and f_i and h_i are the coefficients of the force and boundary terms respectively.

As mentioned the integrals are approximated by a suitable quadrature rule *(usually the Gaussian)*.The main difference here is that the quadrature rule is performed in the parametric domain $\hat{\Omega}$ and not in the parent domain mentioned earlier. The parametric domain is partinioned into n_{el} elements. The quadrature rule is defined in the domain of each element e determined by n_{int} quadrature points and their corresponding weights $q_{l,e}$, $l=1,\ldots.,n_{int}$. The intergral $f \in L^1(\Omega_e)$ after a change of variables is computed as :

$$\int_{\Omega_e} f dx = \int_{\hat{\Omega}_e} f(F(\xi)) \left|\det(DF(\xi))\right| \delta\xi \simeq \sum_{l=1}^{n_{\text{int}}} q_{l,e} f(x_{l,e}) \left|\det(DF(\xi_{l,e}))\right| \quad (3.1.3)$$

where $x_{l,e} := F(\xi_{l,e})$ are the images of the quadrature nodes in the physical domain.. Applying the above formula the coefficients $K_{i,j}$ of the stiffness matrix are numerically computed as

$$K_{i,j} \simeq \sum_{e=1}^{n_{el}} \sum_{l=1}^{n_{\text{int}}} k(x_{l,e}) q_{l,e} \nabla u_j(x_{l,e}) \nabla w_i(x_{l,e}) \left|\det(DF(\xi_{l,e})\right| \quad (3.1.4)$$

while the coefficients f_i of the righ-hand side vector are approximated as

$$f_i \simeq \sum_{e=1}^{n_{el}} \sum_{l=1}^{n_{\text{int}}} f(x_{l,e}) q_{l,e} w_i(x_{l,e}) \left|\det(DF(\xi_{l,e})\right| \quad (3.1.5)$$

To numerically compute the boundary term a quadrature rule is defined on the boundaries, inherited from the one defined on the whole domain. If $t_{l,e}$ are the parametric coordinates of quadrature nodes and $F_b{:}[0,1]\rightarrow\Gamma_N$ is the restriction of F to the boundary(assuming that each side of the parametric domain is completely mapped into Γ_N or Γ_D) then the boundary line intergrals are approximated as follows:

$$h_i \simeq \sum_{e=1}^{n_{el}} \sum_{l=1}^{n_{\text{int}}} h(x^b_{l,e}) q^b_{l,e} w_i(x^b_{l,e}) \left| F^{'}(t_{l,e}) \right| .$$

3.1 Data Structures

Geometry

We start by defining our geometry ,which is the physical domain of our problem .the construction of the geometry is accomplished via the NURBS toolbox. The toolbox was mainly developed based on algorithms of [2].

Remember that the parametric domain $\hat{\Omega}$ consists of a unit square (or cube) mad the mapping to the physical domain Ω F:Ω→ $\hat{\Omega}$ is defined as

$$F(\xi) = \sum_{i \in I} N_i(\xi) B_i$$

with B_i being the control point and N_i the NURBS basis functions defined as :

$$R_i = \frac{w_i N_i}{\sum_{j \in J} w_j N_j}$$ (N_i denotes the B-spline functions).

The main fields of NURBS toolbox are:

- ✔ **Order**: a vector with the order in each direction .We should remind that B-splines of degree p have order of p+1
- ✔ **Knots**: knot vectors Ξ, stored as a cell array
- ✔ **Number:** number of basis functions along each direction
- ✔ **Coefs:** Control points ,along with their weights ,are stored in the foe form of a cell array of size (4,n_1) for a curve ,(4,n1,n2) for a surface and (4,n1,n2,n3) for a

volume. The first three rows contain the control points B_i multiplied with the according weight q_i and the fourth row contains contains the weight q_j . In the case of B-splines the weights are equal to one.

The desired geometry is then constructed through the function *nrbmak(coefs,knots).*Other significant functions of this toolbox are *nrbkntins* and *nrbdegelev*, which perform knot insertion and degree elevation, respectively. For example, consider a NURBS surface, then the following commands :

```
1.nurbs=nrbmak(coefs,knots);

2.nurbs=nrbdegelev(NURBS,[1 0]);

3.new_knots=linspace(0,1,10);

4.nurbs=nrbkntins(NURBS,{new_knots(2:end-1) new_knots(2:end-1);
```

would create a surface (line1), line 2 would raise the degree of the surface by one in the first parametric direction and line 3 would insert new knots uniformly in both directions.

Once our geometry is defined we proceed by calling the function *geo_load* which is prepared to created the structures for geometries defined in the following ways:

- As a structure of NURBS toolbox
- As an affine transformation, defined by a 4×4 matrix
- As a function defined by the user

For the moment the first case is invoked by the following command:

```
geometry=geo_load('nurbs.mat')
```

the output geometry contains information to compute the geometry parameterization and its derivatives. The main fields of the structure are:

- **map**: function handle to compute the parameterization F at some points in $\hat{\Omega}$
- **map_der** : function handle to compute the jacobian of the parameterzation DF

Let us clarify that the function handles to compute F or DF ,not the values of the map.

Mesh

The second step is to define the domain partition and set a quadrature rule in order to compute the matrices by numerical integration. The defined tensor product partition includes quadrature elements which coincide with knot spans in the geometry.The knot vectors of the geometry structure are used to set the quadrature nodes and weights for a gaussian quadrature rule through the function *msh_gauss_nodes*. The number of quadrature nodes in each direction are equal to the degree of NURBS plus one. All this is accomplished with the following commands:

```
1.knots=geometry.nurbs.knots;

2.[qn,qw]=msh_set_quad_nodes(knots,msh_gauss_nodes(geometry.nurbs
.order);
```

The information for the quadrature rule is stored in the structure. First, the msh structure is computed in the parametric domain $\hat{\Omega}$:

```
msh=msh_2d_tensor_product(knots,qn,qw)
```

Afterwards the structure is mapped to the physical domain Ω using the function msh_push_forward_2d:

```
msh=msh_push_forward_2d(msh.geometry)
```

The main fields of the msh structure are:

- **nel**: n_{el}, the number of elements of the partition.
- **nqn**: n_{int}, the number of quadrature points per element.
- **quad _nodes**: coordinates of the quadrature nodes $\xi_{l,e}$ in the parametric domain Ω.
- **quad _weights**: weights $q_{l,e}$ associated to the nodes.
- **geo_ map**: $x_{l,e}=F(\xi_{l,e})$, coordinates of the quadrature nodes in the physical domain.

• **geo _map_ jac**: Jacobian matrix of the parameterization F evaluated at the quadrature points, i.e., $DF(\xi_{l,e})$

• **jacdet**: absolute value of the Jacobian matrix determinant, evaluated at the quadrature points, i.e., $| DF(\xi_{l,e}))|$.

The values of the last three fields are computed using information from the geometry structure .

The space structure

The space structure contains the information regarding the basis functions of the dicrete space V^h , their evalution at the quadrature nodes. All this is needed to numerically compute the integrals of the problem.

As we mentioned in the beginning, the BVP(Boundary Value Problem) is numerically approximated through the isoparametric concept and thus the geometry and discrete solution space coincide. The NURBS structure already contains the information for the discrete space. the new structure uses this information as well as the information from the msh structure:

```
space=sp_nurbs_2d_phys(geometry.nurbs,msh);
```

The fields concerning this structure assign indices to the basis functions, both local and global, in a similar way described in chapter 2 .They also contain information regarding the support of the functions, which are locally supported. This way the integrals in each element are computed for a reduced number of basis functions. The fields of the space structure are:

• **ndof**: N_h, total number of degrees of freedom, which is equal to the dimension of the space V^h, that is the number of basis functions being used .

• **nsh**: N_s, indices of non-vanishing basis functions in each element.

• **connectivity**: indices the basis functions that do not vanish on each element. It has size $Ns \cdot n_{el}$

• **shape functions**: evaluation of the basis functions at the quadrature points, that is, the quantities $w_i(x_{l,e})$ in equation (3.1.5).Its size is $n_{int} \cdot Ns \cdot n_{el}$,

• **spfun**: function handle to evaluate the fields above at the points given in a msh structure, this is used when evaluating at points different from the quadrature points is required, e.g. for visualization.

Assembly routine

The next step is to assemble both the stiffness and right hand matrix.The following command assemles the coordinates $x_{l,e}$ and $y_{l,e}$:

```
[x,y]=deal(squeeze(msh.geo_map(1,:,:)),squeeze(msh.geo_map(2,:,:)));
```

the calculation is done via function *op_gradu_gradv*:

```
mat=op_gradu_gradv(space,space,msh,k(x,y));
```

GeoPDEs allows to use separate spaces for the trial and weighting solutions, but this time since both solutions use the same space ,the space structure is passed twice. The function constists of a cycle over the elements, two cycles over the elements and a a final cycle over the quadrature points of each element as we can observe below:

```
 function mat = op_gradu_gradv (spu, spv, msh, coeff)

  mat = spalloc (spv.ndof, spu.ndof, 1);

  gradu = reshape (spu.shape_function_gradients, spu.ncomp, [],
msh.nqn, spu.nsh_max, msh.nel);
  gradv = reshape (spv.shape_function_gradients, spv.ncomp, [],
msh.nqn, spv.nsh_max, msh.nel);

  ndir = size (gradu, 2);

  for iel = 1:msh.nel
    if (all (msh.jacdet(:,iel)))
      mat_loc = zeros (spv.nsh(iel), spu.nsh(iel));
      for idof = 1:spv.nsh(iel)
        ishg = reshape(gradv(:,:,:,idof,iel),spv.ncomp * ndir, []);
        for jdof = 1:spu.nsh(iel)
          jshg = reshape(gradu(:,:,:,jdof,iel),spu.ncomp * ndir, []);
          %for inode = 1:msh.nqn
          mat_loc(idof, jdof) = mat_loc(idof, jdof) + ...
              sum (msh.jacdet(:,iel) .* msh.quad_weights(:, iel) .* ...
                     sum (ishg .* jshg, 1).' .* coeff(:,iel));
          %end
        end
      end
      mat(spv.connectivity(:, iel), spu.connectivity(:, iel)) = ...
        mat(spv.connectivity(:, iel), spu.connectivity(:, iel)) +
mat_loc;
    else
      warning ('GeoPDEs:jacdet_zero_at_quad_node', 'op_gradu_gradv:
singular map in element number %d', iel)
    end
  end

end
```

function to compute the stiffness matrix

Next the right hand matrix is assembled with the help of the function op_f_v

```
rhs=op_f_v(space,msh,f(x,y))
```

If g=0 in (2), i.e. for homogeneous Dirichlet conditions, we proceed to the separation of the boundary degrees of freedom ,meaning the functions that do not vanish from the internal degrees of freedom. This is done with the commands:

```
drchlt_dofs=unique([space.boundary(:).dofs]);

int_dofs=setdiff(1:space.ndof,drchlt_dofs);
```

Finally, the coefficients d_j are computed for the internal degrees of freedom and set to zero for boundary ones:

```
u=zeros(space.ndof,1);

u(int_dofs)=mat(int_dofs,int_dofs)\rhs(int_dofs);
```

Treatment of boundary conditions

For the implementation of boundary conditions the msh and space structure contain a field called boundary. The boundary of the parametric domain $\hat{\Omega}$ is divided into a certain number of sides .Then, the boundary field is defined as an array containing for each side the following structures:

- **msh.boundary**: Numerical integration is performed as this structure contains a partition of each boundary side. The field jacdet is no longer the determinant of the jacobian. Instead, it calculates the term $\left|F^{'}(t_{l,e})\right|$.The structure also contains the field normal, with the value of unit exterior vector at the quadrature points.

- **space.boundary**: Contains information of the basis functions and their values at the quadrature points provided by *msh.boundary* .The *ndof* and *connectivity* fields refer to a local numbering of the basis functions whereas the field *dofs* refer to a global numbering .

Suppose g=g(x,y) and h=h(x,y) using the above structures the Neumman condition is computed like below:

```
for iside = nmnn _sides
 x = squeeze (msh. boundary( i _ side ) . geo_ map ( 1 , : , : ) ) ;
y = squeeze (msh. boundary( i _side ) . geo_ map ( 2 , : , : ) ) ;
 hval =h(x,y) ;
  rhs_ loc = op_ f_ v (space . boundary( iside ) , msh.boundary(
iside ) , hval ) ;

 rhs(space . boundary( iside ) . dofs) = rhs(space . boundary( iside
) . dofs) + rhs side ;

end
```

Notice that first we evaluate the h(x,y) function for each Neumman side and then the boundary term is computed using the function *op_f_v*, which is the same as the source term. The assembling of the global right hand side is achieved using the field *dofs*.

The implementation of the Dirichlet boundary condition is done in a similar way:

```
drchlt _dofs = unique ( [ space.boundary( drchlt_ side ) . dofs ] )
;
int dof s =setdiff (1: space.ndof, drchlt_ dofs ) ;
M_drchlt = spalloc ( space.ndof , space.ndof , space.ndof) ;
rhs_drchlt = zeros ( space.ndof, 1);

for iside = drchlt_ sides
sp_bnd = space . boundary( iside ) ;
msh_bnd = msh. boundary( iside ) ;
x = squeeze (msh bnd. geo map ( 1 , : , : ) ) ;
y = squeeze (msh bnd. geo map ( 2 , : , : ) ) ;
gval = g(x,y) ;
M _side = op _u _v (sp bnd , sp bnd , msh bnd, ones ( size (x) ) ;
.M _drchlt (sp _bnd . dofs , sp_ bnd.dofs) = M_drchlt (sp_ bnd .
dofs , sp _bnd . dofs)
     +M side ;
r hs_ side = op_f_v (sp bnd , msh bnd, hval);
r hs _drchlt (sp_ bnd.dofs) = rhs_drchlt (sp_bnd.dofs) + rhs _side;
end

u = zeros ( space . ndof, 1);
u( drchlt_dofs ) = M_drchlt ( drchlt_dofs , drchlt_ dofs ) \
rhs_drchlt( drchlt_dofs ) ;
rhs (int_ dofs ) = rhs ( int_ dofs ) – mat ( int_ dofs , drchlt
_dofs ) * u( drchlt _dofs ) ;
```

The first four lines identify the degrees of freedom on the Dirichlet boundary and set the initializations. Then for each side the term g(x,y) is computed on the quadrature points by using the information on the boundary fields. From line 11 to 14 the matrix and the right hand side are assembled to compute L^2 projection, which is done in line 17.The right hand side of the problem is corrects on line 18.

Postprocessing

The visualization of the computed solution is done via the *Paraview* software. The following command evaluates the solution of the problem at the points given by a (20×20) grid. The results are then saved in a vtk data file format :

```
sp _to _vtk _2d (u , space , geometry, [20 20], 'laplace_ solution.vts
' , 'u ' )
```

h,p,k-refinement

The NURBS toolbox offers the choice of refining our solution, but in contrast with standard FEM piecewise polynomials, isogeometric analysis can do that without affecting our geometry. This paragraph shows how h,p-refinement and isogeometric newly achieved k-refinement, are treated in GeoPDEs. The functions used for the refinement strategy are all contained in the NURBS toolbox.

We start off with P-refinement, which is as explained in chapter 1 involves applying degree elevation to the NURBS basis functions. The NURBS tool box offers the opportunity to do this since it contains the already mentioned *nrbdegelev* function.

Suppose we wanted to solve with cubic NURBS then we would apply the code below:

```
1. nurbs=geometry.NURBS

2. degelev=max([3  3]-(nurbs.order-1) ,0);

3. nurbs=nrbdegelev(nurbs,degelev);

4. geometry=geo_load(nurbs);
```

Line 1 call upon the initial geometry's NURBS structure in order to be refined. Next in line 2, the max function is involved in order to avoid degree elevation if the desired degree is lower than the actual one. In line 4 we replace the old geometry with the refined one.

h-refinement is also achievable since the NURBS toolbox contains the function *nrbkntins* which is responsible for knot insertion as well as the *kntrefine* function which can add new knots uniformly . For example we wanted to add knot 1 knot in each subinterval we would apply the following command:

Listing 1

```
[rknts,zeta,nknts]=kntrefine(nurbs.knots,[2  2],NURBS.order-1,[0 0]);

nurbs=nrbkntins(nurbs,nknts);
```

The last argument of *kntrefine* ensures that the addition of the new knots is done with the right multiplicity so that the discrete space is C^0 - continuous.

Finally, isogeometric analysis offers the new k-refinement which as already mentioned mix the previous refinement strategies. Order elevation is followed by knot insertion global C^{p-1} continuation across element boundaries is constrained.

```
1. nurbs=geometry.nurbs

2. degelev=max([3 3]-(nurbs.oredr-1),0);

3. nurbs = nrbdegelev (nurbs, degelev);

4. [rknots , zeta , nknots] = kntrefine (nurbs . knots , [1 1] ,…

5. nurbs.order-1, nurbs . order-2) ;

6. nurbs = nrbkntins (nurbs, nknots);

7. geometry = geo load (nurbs) ;
```

Here, in Lines 2-3 , we first perform order elevation and then one knot is added in each subinterval uniformly with the right multiplicity, which yields a C^2 -continuity at these knots.

Some m-files were created in order to apply the refinement strategies in a more direct way. *h-refinement (geometry, knts)* is used as a shortcut of Listing 1. The first input is geometry, in which we wish to apply the refinement and *knts* is number of knots we wish to insert in each sub interval . Same for *p-refinement (geometry, degr)* except now *degr* is the degr we wish to elevate our basis in each direction. *k-refinement (geometry,degr,knts)* applies the k-refinement strategy to the geometry by elevating the degree to *degr* in each direction and inserting *knts* knot in each subinterval with the right multiplicity so that C^{degr-1} continuity is preserved (see below):

```
function geo = krefinement(geo,degr,knts)
```

```
nurbs=geo.nurbs;
degelev=max(degr-(nurbs.order-1),0);
NURBS=nrbdegelev(nurbs,degelev);
[rknots,zeta,nknots]=kntrefine(nurbs.knots,knts,nurbs.order-
1,nurbs.order-2);
nurbs=nrbkntins(nurbs,nknots);
geo=geo_load(nurbs);
end
```

GeoPDEs also allows using the non isoparametric approach for the solution of BVPs, meaning that we can use B-spline spaces as the solution space and maintain the parameterization for our geometry using NURBS .This is achievable since GeoPDEs the geometry and the solution space are treated independently. In order to implement the use of bsplines (for the 2 dimensional case) the following command is applied:

```
        space=sp_bspline_2d_phys(knots, degree, msh)
```

As now the geometry and solution space do not coincide the knot vectors are different and handled by the user. The second input determines the degree of the B-splines being used.

3.2 Numerical examples

We proceed to the implementation of GeoPDEs on 2D elliptical problems. For a full description of the geometrical descriptions of the shapes as well as the files used see the end of the chapter.

The first example involves the computation of the boundary problem below:

$$\Delta u = -f \quad (3.2.1)$$

$$u = 0 \text{ on } \Gamma_D , \ (3.2.2)$$

Here the coefficient k(x)=1, and homogeneous Dirichlet conditions are imposed on the whole boundary , i.e $\Gamma_D \equiv \partial\Omega$. Furthermore

$$f = \frac{(8-9\sqrt{x^2+y^2})\sin(2\arctan(\frac{y}{x}))}{x^2+y^2} \quad (3.2.3)$$

and the exact solution is given by :

$$u = (x^2 + y^2 - 3\sqrt{x^2 + y^2} + 2)(3.2.4)$$

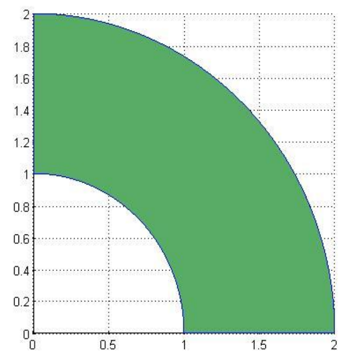
A function file was constructed, *homogeneous_poisson .m*, which takes as input the geometry under study and implements the above boundary value problem above using NURBS discretization(i.e isoparametric approach) :

```
function error=homogeneous_poisson(geometry)

geometry = geo_load (nurbs);
knots    = geometry.nurbs.knots;

%Construction of msh structure
[qn, qw] = msh_set_quad_nodes (knots, msh_gauss_nodes
(geometry.nurbs.order));
msh = msh_2d_tensor_product (knots, qn, qw);
msh = msh_push_forward_2d (msh, geometry);

%construction of space structure
space  = sp_NURBS_2d_phys (geometry.nurbs, msh);


%Assemble the matrices
[x, y] = deal (squeeze (msh.geo_map(1,:,:), squeeze
(msh.geo_map(2,:,:));
mat = op_gradu_gradv (space, space, msh, ones (size (x)));%k(x,y)=1
rhs = op_f_v (space, msh, (8-
9*sqrt(x.^2+y.^2)).*sin(2*atan(y./x))./(x.^2+y.^2));%f(x,y)

%Separate degrees of freedom
drchlt_dofs = unique ([space.boundary( :.dofs]);
int_dofs = setdiff (1:space.ndof, drchlt_dofs);

u = zeros (space.ndof, 1);
u(int_dofs) = mat(int_dofs, int_dofs) \ rhs(int_dofs);

%Postprocessing
sp_to_vtk_2d (u, space, geometry, [20 20], 'laplace_solution.vts', 'u')
err = sp_l2_error (space, msh, u, @(x,y)(x.^2+y.^2-
3*sqrt(x.^2+y.^2)+2).*sin(2.*atan(y./x)));%analytical solution u
error=err;
end
```

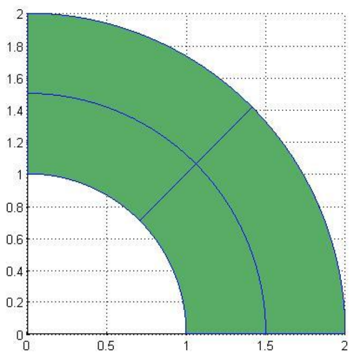
Listing 2

The first domain Ω consists of the intersection of the first quadrant of the Cartesian plane with a circular annulus of internal radius r = 1 and external radius R= 2. The geometrical description is given below. Note that the fourth coordinate denotes the weight of the control point.

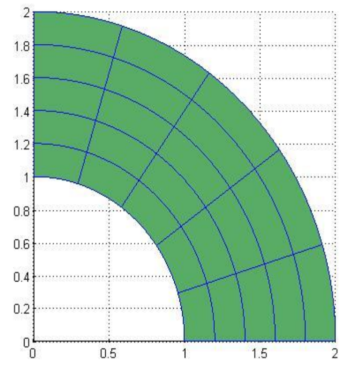
<u>**Fig. 3.1**</u> depicts the meshes created during h- refinement:



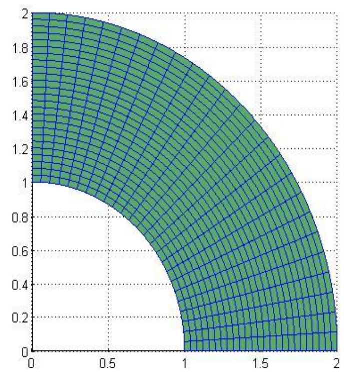
Coarse Mesh



Mesh 2



Mesh 3



Mesh 4

Fig. 3.1

The solution for mesh 4 is viewed via paraview:

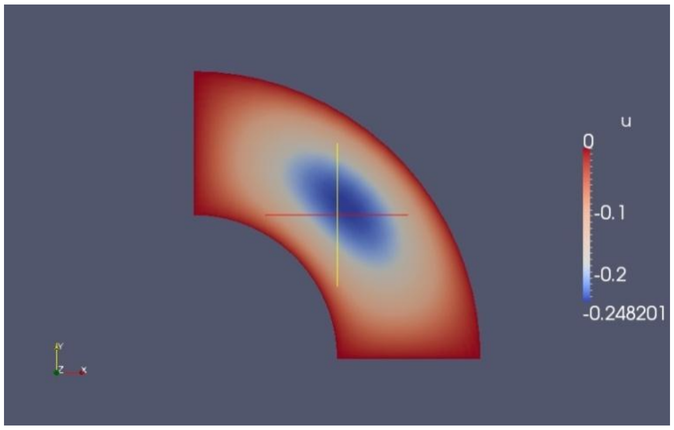


Fig. 3.2:solution for mesh 4

Since the coarse mesh consists of one element we can apply k-refinement to the geometry. The homo_dofs_error.m was created(see Listing 2) in order to compare h-refinement and k-refinement strategies . During k-refinement this time each loop increases the degree of the NURBS used and inserts a new knot, while maintaining C^{p-1} continuity .As we observe k-refinement yields better convergence rate and with much less degrees of freedom involved each time.

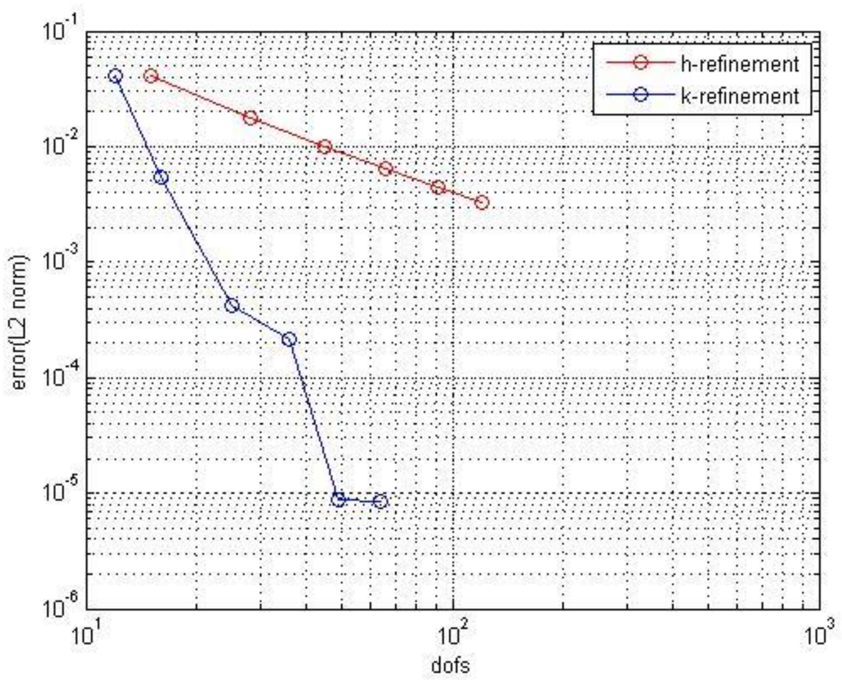
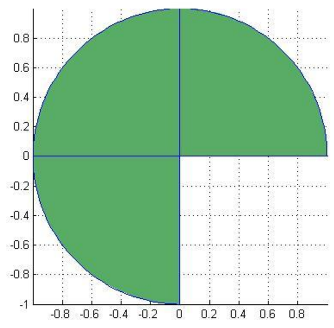
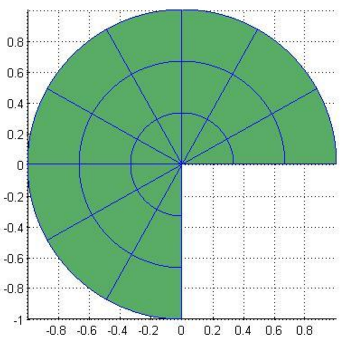


Fig 3.3: h-k refinement for circular annulus

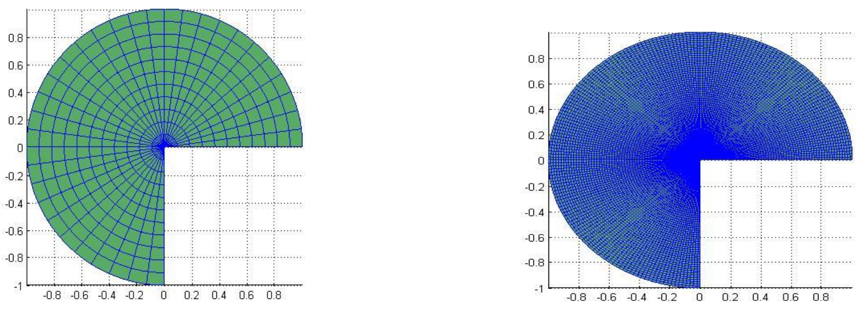
Next we examine the problem, while the domain Ω consists of a circle with a quadrant cut off resulting in a *pacman-like* shape. **Fig. 3.4** show various meshes of the shape



Coarse mesh



mesh 1



mesh 3 **mesh 4**

Fig. 3.4

The solution for mesh 4 is presented in **Fig. 3.5**

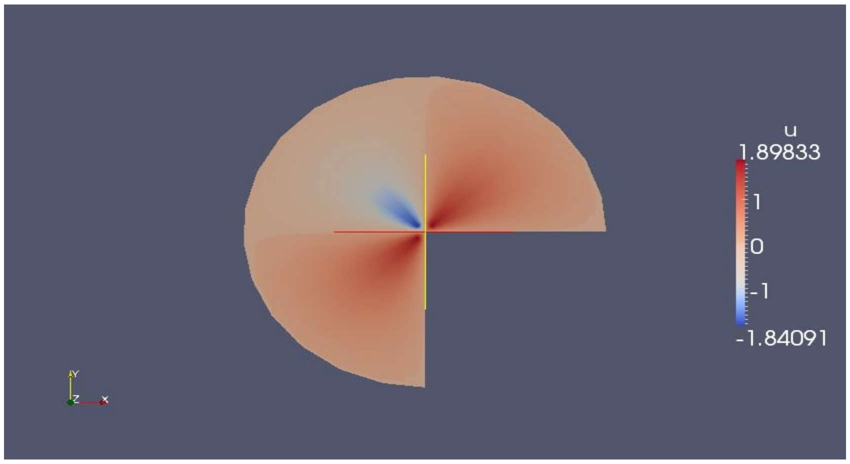


Fig. 3.5:solution of the circle without quadrant for mesh 4

Applying h and p-refinement via homo_dofs_perror.m (see ***Listing 3***) to the coarse mesh yields the following results :

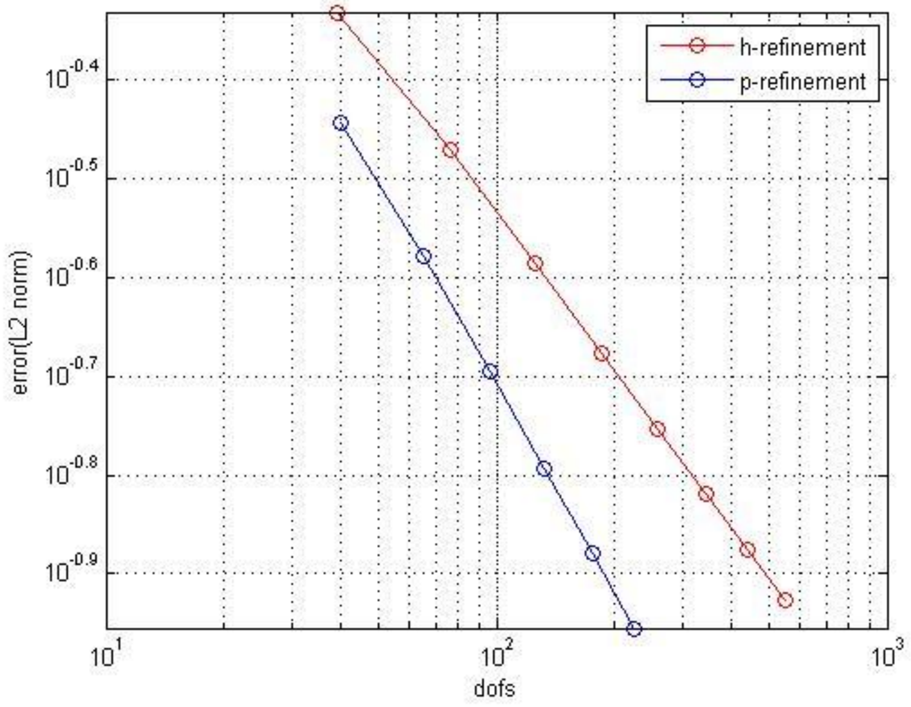


Fig. 3.6:h,p-refinement for circle without quadrant

This time p-refinement results in a better convergence rate than h-refinement for similar degrees of freedom.

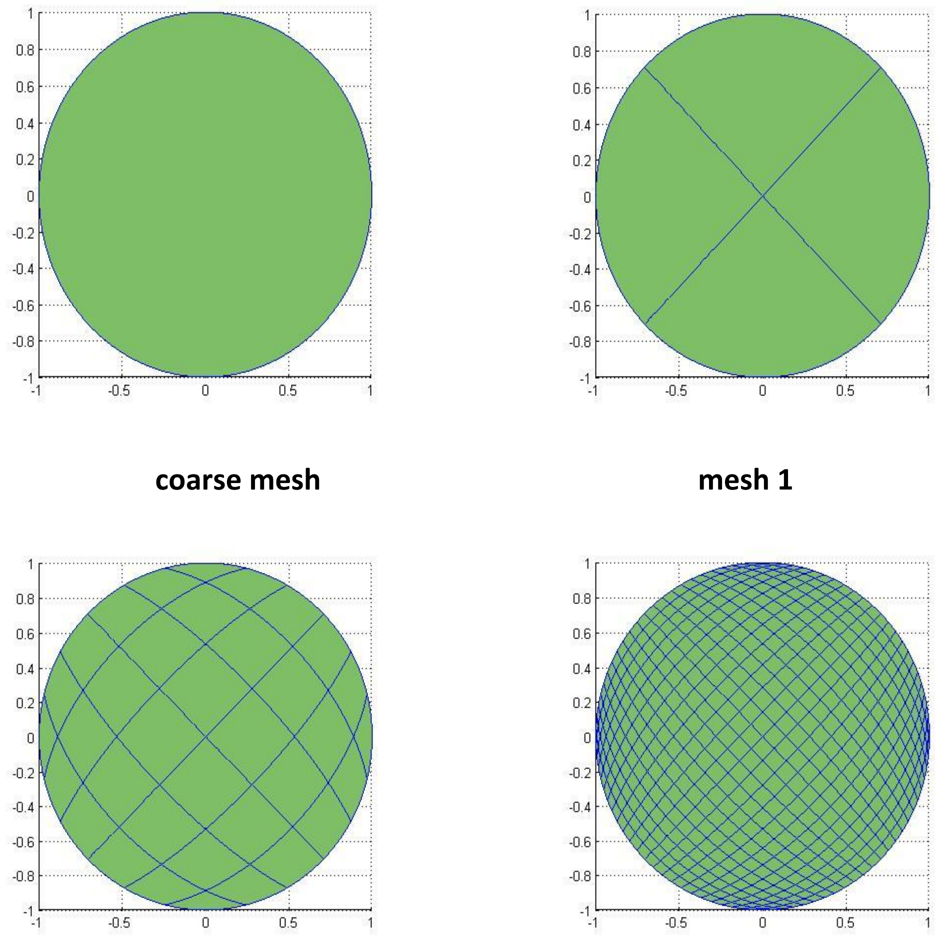
The next boundary value problem involves the implementation of non homogeneous Dirichlet conditions as well as Neumman conditions :

$$\Delta u = 0$$
$$u = x^2 - y^2, \Gamma_D \quad (3.2.5)$$
$$\frac{du}{dn} = 2x\hat{x} - 2y\hat{y}$$

$$u = x^2 - y^2 \text{ on } \Gamma_D \text{ , } (3.2.6)$$
$$\frac{du}{dn} = 2x\hat{x} - 2y\hat{y} \text{ } (3.2.7)$$

Where $\hat{x}$ and $\hat{y}$ denote the coordinates of normal outward vector. This time in addition to the isoparametric approach, the problem was solved using also B-spline (of the same degree) basis imposed on the NURBS geometry to compare results.

The following domain Ω consists of a circle surface . The surface was created using nrbcoons.m command from the NURBS toolbox. The circle_mixed_bc.m (**Listing 4**) file used to solve the problem isoparametrically while circle_mixed_bc_b_spline.m (**Listing 5**) was used to solved with B-splines while test_circle_mixed_bc_g_nmnn.m (**Listing 6**) for the implementation of Neumman condition .



coarse mesh **mesh 1**

mesh 3 **mesh 4**

<u>Fig 3.7:</u>Various meshes for circle

by imposing the Dirichlet condition on all sides the solution for mesh 4 is depicted in **<u>Fig.3.8</u>** :

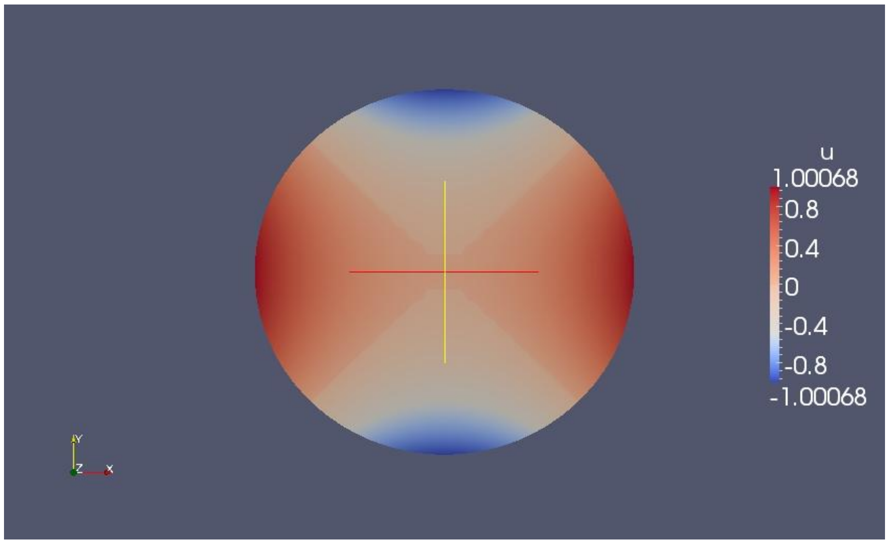


Fig. 3.8:isoparametric solution of circle for mesh 4

Applying h-refinement (**Listing 7**) to both isoparametric and non isoparametric approach yields:

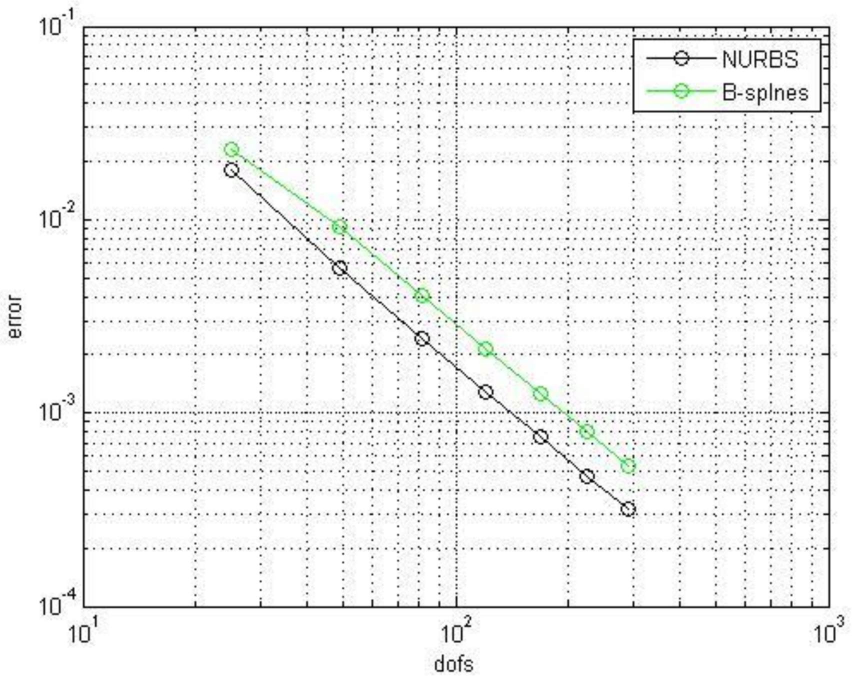
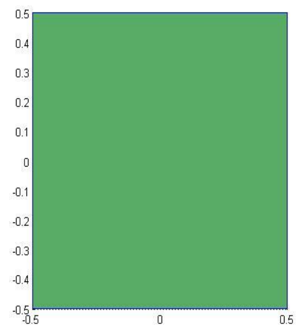


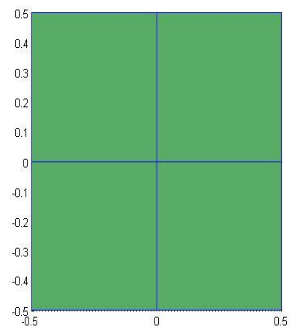
Fig. 3.9: Isoparametric and non-isoparametric results of circle

Next the domain Ω consists of a bi unit square .We impose Dirichlet conditions on sides 1,2 and Neumman on 3,4(that is the left and right side).Ther m- files of the isoparametric and non isoparametric soltuion are similar to these of the circle. **Listing 8** shows the imposition of the Neumman conditions on the square.

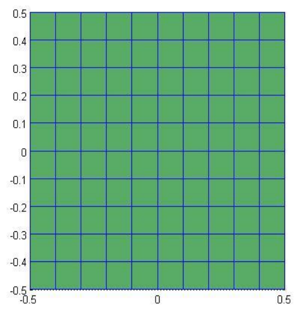
Note that the meshes shown on **<u>Fig.3.10</u>** below could be considered as the parametric domain $\widehat{\Omega}$.



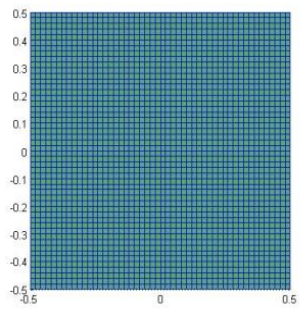
coarse mesh



mesh 1



mesh 3



mesh 4

<u>Fig. 3.10</u>

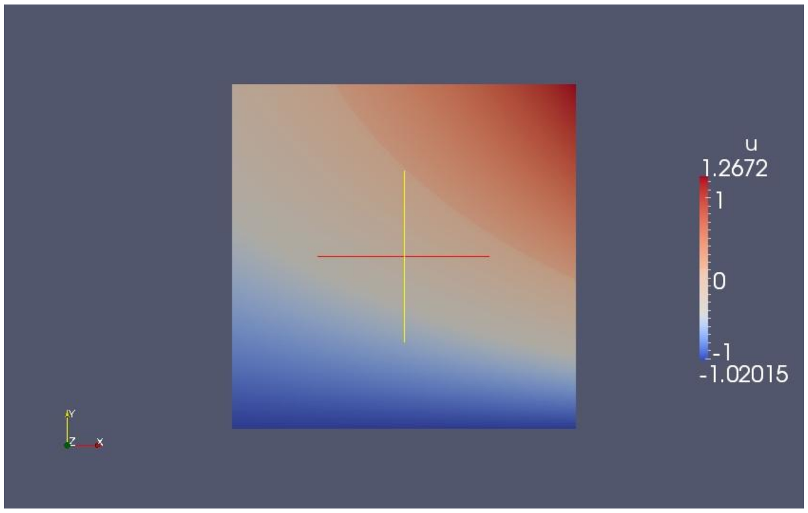


Fig. 3.11:isoparametric solution of square for mesh 4

Once again, as seen in the **Fig.3.12**. below, the isoparametric approach yields better convergence rate :

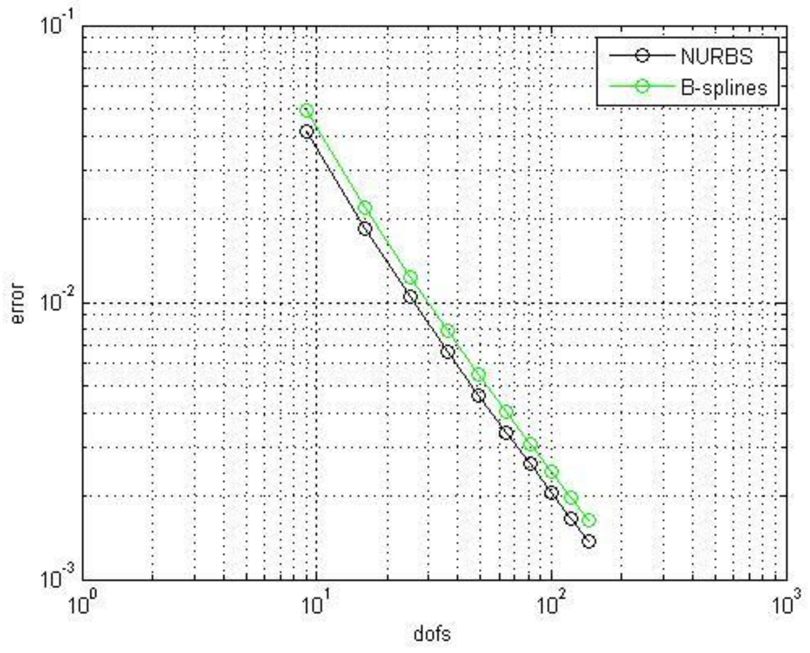


Fig. 3.12

APPENDIX A: Description of the m-files

Listing 2(homo_dofs_error.m)

```
function [errors,dofs] = homo_dofs_error( geometry,I )
dofs=[];
errors=[];
 for j=1:1:i
     refined_geo(j)=hrefinement(geometry,[j j]);

dofs(j)=refined_geo(j).NURBS.number(1)*refined_geo(j).NURBS.number(2);
     errors(j)=homogeneous_poisson(refined_geo(j));


 end
 for j=1:1:i
  refined_geo(j)=krefinement(geometry,[j j],[1 1]);

dofs2(j)=refined_geo(j).NURBS.number(1)*refined_geo(j).NURBS.number(2);
   errors2(j)=homogeneous_poisson(refined_geo(j));


 end


 loglog(dofs,errors,'-ro',dofs2,errors2,'-bo');
 h=legend('h-refinement','k-refinement');
 xlabel('dofs');
 ylabel('error(L2 norm)');
 grid on
end
```

Listing 3(homo_dofs_perror.m)

```
function [errors,dofs] = homo_dofs_perror( geometry,I )
dofs=[];
errors=[];
 for j=1:1:i
     refined_geo(j)=hrefinement(geometry,[j j]);

dofs(j)=refined_geo(j).NURBS.number(1)*refined_geo(j).NURBS.number(2);
     errors(j)=homogeneous_poisson(refined_geo(j));
      end
 for j=2:1:i

  refined_geo(j)=prefinement(geometry,[j j]);

dofs2(j)=refined_geo(j).NURBS.number(1)*refined_geo(j).NURBS.number(2);
   errors2(j)=homogeneous_poisson(refined_geo(j));
```

```
end


loglog(dofs,errors,'-ro',dofs2,errors2,'-blo');
h=legend('h-refinement','p-refinement');
xlabel('dofs');
ylabel('error(L2 norm)');
grid on
```

Listing 4(circle_mixed_bc.m)

```
function [dofs,error_l2] = circle_mixed_bc(i)

% Type of boundary conditions
nmnn_sides   = [  ];
drchlt_sides = [1 2 3 4];

% NURBS map from text file
geo_name = 'circle.mat';

% Physical parameters
c_diff  = @(x, y) ones(size(x));

% Source and boundary terms
f = @(x, y) zeros (size (x));
g = @test_circle_mixed_bc_g_nmnn;
h = @(x, y, ind) x.^2-y.^2;

% Exact solution
uex     = @(x, y)  (x).^2 -(y).^2;
graduex = @(x, y) cat (1, ...
                       reshape (2*x, [1, size(x)]), ...
                       reshape (-2*y, [1, size(x)]));

% Output file for Paraview
output_file = 'circle';

% Points for post-processing
vtk_pts = {linspace(0, 1, 20)', linspace(0, 1, 20)'};


geometry  = geo_load (geo_name);
geometry=hrefinement(geometry,[i i]);
knots=geometry.NURBS.knots;
dofs=geometry.NURBS.number(1)*geometry.NURBS.number(2);

% Construct msh structure
[qn, qw] = msh_set_quad_nodes (knots,
msh_gauss_nodes(geometry.NURBS.order));
msh      = msh_2d_tensor_product (knots,qn, qw);
msh       = msh_push_forward_2d (msh, geometry);

% Construct space structure
sp        = sp_NURBS_2d_phys (geometry.NURBS, msh);

% Precompute the coefficients
x = squeeze (msh.geo_map(1,:,:));
y = squeeze (msh.geo_map(2,:,:));

epsilon = reshape (c_diff (x, y), msh.nqn, msh.nel);
fval    = reshape (f (x, y), msh.nqn, msh.nel) ;

% Assemble the matrices
stiff_mat = op_gradu_gradv (sp, sp, msh, epsilon);
rhs       = op_f_v (sp, msh, fval);

% Apply Neumann boundary conditions
for iside = nmnn_sides
  x = squeeze (msh.boundary(iside).geo_map(1,:,:));
  y = squeeze (msh.boundary(iside).geo_map(2,:,:));
  gval = reshape (g (x, y, iside), msh.boundary(iside).nqn,
msh.boundary(iside).nel);

  rhs(sp.boundary(iside).dofs) = rhs(sp.boundary(iside).dofs) + ...`
      op_f_v (sp.boundary(iside), msh.boundary(iside), gval);
end

% Apply Dirichlet  boundary conditions
u = zeros (sp.ndof, 1);
[u_drchlt, drchlt_dofs] = sp_drchlt_l2_proj(sp, msh, h, drchlt_sides);
u(drchlt_dofs) = u_drchlt;

int_dofs = setdiff (1:sp.ndof, drchlt_dofs);
rhs(int_dofs) = rhs(int_dofs) - stiff_mat(int_dofs,
drchlt_dofs)*u_drchlt;

% Solve the linear system
u(int_dofs) = stiff_mat(int_dofs, int_dofs) \ rhs(int_dofs);




if (exist ('uex', 'var'))
  error_l2 = sp_l2_error (sp, msh, u, uex);
  if (exist ('graduex', 'var'))
    error_h1 = sp_h1_error (sp, msh, u, uex, graduex);
  end

end
end
```

Listing 5(circle_mixed_bc_b_splines.m)

```
function [dofs,error_l2] = circle_mixed_bc_b_splines(i)
degree     = [2 2];      % Degree of the bsplines
regularity = [2 2];      % Regularity of the splines
n_sub      = [9 9];      % Number of subdivisions
nquad      = [4 4];      % Points for the Gaussian quadrature rule

% Type of boundary conditions
nmnn_sides   = [    ];
drchlt_sides = [1 2 3 4];

% NURBS map from text file
geo_name = 'circle.mat';

% Physical parameters
c_diff  = @(x, y) ones(size(x));

% Source and boundary terms
f = @(x, y) zeros (size (x));
g = @test_circle_mixed_bc_g_nmnn;
h = @(x, y, ind) x.^2-y.^2;

% Exact solution
uex     = @(x, y)  (x).^2 -(y).^2;
graduex = @(x, y) cat (1, ...
                       reshape (2*x, [1, size(x)]), ...
                       reshape (-2*y, [1, size(x)]));

% Output file for Paraview
output_file = 'circle';

% Points for post-processing
vtk_pts = {linspace(0, 1, 20)', linspace(0, 1, 20)'};


geometry  = geo_load (geo_name);
geometry=hrefinement(geometry,[i i]);
dofs=geometry.NURBS.number(1)*geometry.NURBS.number(2);
[knots, zeta] = kntrefine (geometry.NURBS.knots, n_sub, degree,
regularity);

% Construct msh structure
rule     = msh_gauss_nodes (nquad);
[qn, qw] = msh_set_quad_nodes (zeta, rule);
msh      = msh_2d_tensor_product (zeta, qn, qw);
msh      = msh_push_forward_2d (msh, geometry);

% Construct space structure
sp       = sp_bspline_2d_phys (knots, degree, msh);

% Precompute the coefficients
x = squeeze (msh.geo_map(1,:,:));
```

```
y = squeeze (msh.geo_map(2,:,:));

epsilon = reshape (c_diff (x, y), msh.nqn, msh.nel);
fval    = reshape (f (x, y), msh.nqn, msh.nel) ;

% Assemble the matrices
stiff_mat = op_gradu_gradv (sp, sp, msh, epsilon);
rhs       = op_f_v (sp, msh, fval);

% Apply Neumann boundary conditions
for iside = nmnn_sides
  x = squeeze (msh.boundary(iside).geo_map(1,:,:));
  y = squeeze (msh.boundary(iside).geo_map(2,:,:));
  gval = reshape (g (x, y, iside), msh.boundary(iside).nqn,
msh.boundary(iside).nel);

  rhs(sp.boundary(iside).dofs) = rhs(sp.boundary(iside).dofs) + ...`
      op_f_v (sp.boundary(iside), msh.boundary(iside), gval);
end

% Apply Dirichlet  boundary conditions
u = zeros (sp.ndof, 1);
[u_drchlt, drchlt_dofs] = sp_drchlt_l2_proj(sp, msh, h, drchlt_sides);
u(drchlt_dofs) = u_drchlt;

int_dofs = setdiff (1:sp.ndof, drchlt_dofs);
rhs(int_dofs) = rhs(int_dofs) - stiff_mat(int_dofs,
drchlt_dofs)*u_drchlt;

% Solve the linear system
u(int_dofs) = stiff_mat(int_dofs, int_dofs) \ rhs(int_dofs);




if (exist ('uex', 'var'))
  error_l2 = sp_l2_error (sp, msh, u, uex);
  if (exist ('graduex', 'var'))
    error_h1 = sp_h1_error (sp, msh, u, uex, graduex);
  end

end
end
```

Listing 6(test_circle_mixed_bc_g_nmnn.m)

```
function g = test_circle_mixed_bc_g_nmnn (x, y, ind)
  [theta, r] = cart2pol (x,y);
  switch (ind)
    case 1
```

```
      g = cos(theta).*2.*x - sin(theta).*2.*y;
    case 2
      g =-cos(theta).*2.*x + sin(theta).*2.*y;
    case 3
      g = cos(theta).*2.*x + sin(theta).*2.*y;
    case 4
      g =-cos(theta).*2.*x - sin(theta).*2.*y;
    otherwise
      error ('g_nmnn: unknown reference number')
  end

end
```

Listing 7(dofs_error_circle.m)

```
function [errors,dofs] = dofs_error_circle( i )
dofs=[];
errors=[];
 for j=1:1:i

  [dofs(j)  errors(j)]=circle_mixed_bc(j);
  [dofs2(j)  errors2(j)]=circle_mixed_bc_b_splines(j);

 end
 loglog(dofs,errors,'-ko',dofs2,errors2,'-go')
 h=legend('NURBS','B-splnes');
 xlabel('dofs');
 ylabel('error');
 grid on
end
```

Listing 8(test_square_g_test.m)

```
function g = test_square_g_test (x, y, ind)
```

```
  switch ind
    case 1
      g = -2*(x);
    case 2
      g = 2*(x);
    case 3
      g = 2*(y);
    case 4
      g = -2*(y);
    otherwise
      error ('g_nmnn: unknown reference number');
  end
end
```

APPENDIX B: Geometrical description

CIRCULAR ANNULUS

Knot vectors:

Ξ={0 0 1 1}

H={ 0 0 0 1 1 1}

Control points:

$B_{1,1}=(1,0,0,1)$ $B_{1,2}=(2,0,0,1)$

$B_{2,1}=(1,1,0,1/\sqrt{2})$ $B_{2,2}=(2,2,0,1/\sqrt{2})$

$B_{3,1}=(0,1,0,1)$ $B_{3,2}=(0,2,0,1)$

CIRCLE WITHOUT QUADRANT

Knot vectors:

Ξ={ 0 0 1/3 1/3 2/3 2/3 1 1}

H={0 0 1 1}

Control points:

$B_{1,1}$=(1,0,0,1) $B_{1,2}$=(1,1,0,$1/\sqrt{2}$) $B_{1,3}$=(0,1,0,0,1)

$B_{1,4}$=(-1,1,0, $1/\sqrt{2}$) $B_{1,5}$=(-1,0,0,1) $B_{1,6}$=(-1,-1,0,$1/\sqrt{2}$) $B_{1,7}$=(-1,0,0,1)

$B_{2,1}$=(0,0,0,1) $B_{2,2}$=(0,0,0, $1/\sqrt{2}$) $B_{2,3}$=(0,0,0,1) $B_{2,4}$=(0,0,0, $1/\sqrt{2}$)

$B_{2,5}$=(0,0,0, 1) $B_{2,6}$=(0,0,0, $1/\sqrt{2}$) $B_{2,7}$=(0,0,0, 1)

CIRCLE

Knot vectors:

Ξ={ 0 0 0 1 1 1}

H={ 0 0 0 1 1 1}

Control points:

$B_{1,1}$=(1,0,0,1) $B_{1,2}$=(1,-1,0,$1/\sqrt{2}$) $B_{1,3}$=(0,-1,0,1)

$B_{2,1}$=(1,1,0, $1/\sqrt{2}$) $B_{2,2}$=(0,0,0, $1-\sqrt{2}$) $B_{2,3}$=(-1,-1,0$1/\sqrt{2}$)

$B_{3,1}$=(0,1,0,1) $B_{3,2}$=(-1,1,0, $1/\sqrt{2}$) $B_{3,3}$=(-1,0,0,1)

SQUARE

Knot vectors:

Ξ={0 0 1 1}

H={0 0 1 1}

Control points:

$B_{1,1}$=(-1,-1,0,1) $B_{1,2}$=(1,-1,0,1)

$B_{2,1}$=(-1,-1,0,1) $B_{2,2}$=(1,1,0,1)

BIBLIOGRAPHY

1. **J.Austin Contrell,Thomas J.R Hughes,Yuri bazilevs**, Isogeometric Analysis: Toward an integration of CAD and FEA, John Wiley & Sons 2009
2. **L. Piegl,W.Tiller**, The NURBS book, Springer 1997
3. **T. J.R Hughes**,The Fnite Element Method,Prentice-Hall 1987
4. **M.R. Dorfel , B. Juttler ,B.Simeon** . Adaptive Isogeometric Analysis by Local h-Refinement with T-Splines,
5. **R.Duvigneau**, An introduction to Isogeometric Analysis
6. **Y.Bazilevs,L.B Veiga, J.A Cottrell. T..J.R hughes, G.SAngalli** ,Isogeometric Analysis: Approximation,stability and error estimates for h-refined meshes
7. **Seung-Hyun Ha,** Isogeometric Shape Design Optimization Using NURBS Basis Functions ,2010
8. **B. Simeon, Anh-Vu Vuong**, Isogeometric Analysis Primer,
9. **C.de Falco ,A.Reali , R.Vasquez**, GeoPDEs: a research tool for IsoGeometric Analysis of PDEs
10. **G. Xu, B. Mourrain, R. Duvigneau , A.Galligo** , Optimal analysis-aware parameterization of computational domain in isogeometric analysis
11. **B.Simeon, A.V .Vuong** , Identification and specification of benchmark problems with typical geometries , computation of reference solutions
12. **B.Simeon, A.V. Vuong** , Procedures for a posteriori error analysis
13. **B.simeon,A.V.Vuong** , isogeometric structural prototype solver

14. **M. A. Scott, M. J. Borden, C.V. Verhoosel, T. W. Sederberg, and T. J. R. Hughes**, Isogeometric finite element data structures based on Bezier extraction of T-splines
15. **P. Frey, P.L. George,** Mesh generation application to finite elements, HERMES Science Europe Ltd, 2000
16. **R. Sevilla** , NURBS Enhanced Finite Element Method
17. **K. Hollig** ,Finite Element Method with B-splines, Society for industrial and applied mathematics ,2003

18. **Y.Bazilevs,C.Michler,V.M Calo, T.J.R Hughes,** Weak Dirichlet for wall bounded turbulant flows.