Technische Universität Bergakademie Freiberg

PERSONAL PROGRAMMING PROJECT

Implementation of Iso-geometric Analysis (IGA) for Piezoelectric Material

Vikas Diddige
64041

Supervised by
Dr. Sergii Kozinov

April 5, 2020

Contents

1 Introduction

Among all the numerical methods, Finite Element Methods (FEM) are more popularly used to find approximate solutions of partial differential equations. FEM approximates the Computer Aided Drawing (CAD) geometry by discretizing it into smaller geometries called elements. Such geometrical approximations may create numerical errors and seriously affect the accuracy of the solution. Isogeometric analysis (IGA), on the other hand, is a technique to generate geometry using CAD concept of Non-Uniform Rational B-Splines (NURBS) and analyze using its basis functions [2]. With the use of NURBS basis functions instead of Lagrangian basis functions, the geometry is captured exactly for the analysis and rules out the possibility of the geometrical errors. Moreover, the time from design to analysis phase is greatly reduced, saving the cost and time for the industry. The IGA technique is firstly pioneered by Tom Hughes and his group at The University of Texas at Austin.

In the present programming project, Python code is developed, which can generate any 2D NURBS surface, given the physical and parametric details of the geometry. Few commercial software (for example, Rhino) can be used to get the parametric details of the surface. Later the code is extended to analyze the linear elastic mechanical loading case as a displacement driven algorithm. Further, the electro-mechanical coupling is added. The written code gives accurate results for a 2nd order NURBS basis functions. For higher-order basis functions, a particular treatment is required for defining boundary conditions like least square minimization technique (an exception is when all the control points are on the surface itself). The results have been verified using Abaqus results. Comparing the IGA program generated results with Abaqus elements output is justified because IGA aims at reducing the approximation from traditional FEM procedures.

1.1 Advantages of IGA over FEA

- The exact representation of the geometry for analysis rules out the possibility of geometrical approximations.
- A huge amount of time and effort involved in finite element modelling can be avoided.

2 B-Splines

In this section, a brief description of B-Splines is discussed since NURBS is an extended version of B-Splines. A B-Spline basis function is defined by its order and knot vectors. A B-spline basis function, along with control points, defines a B-Spline curve. A surface or volume can be generated using a curve by tensor product between its basis functions, which will be discussed in detail in the future sections.

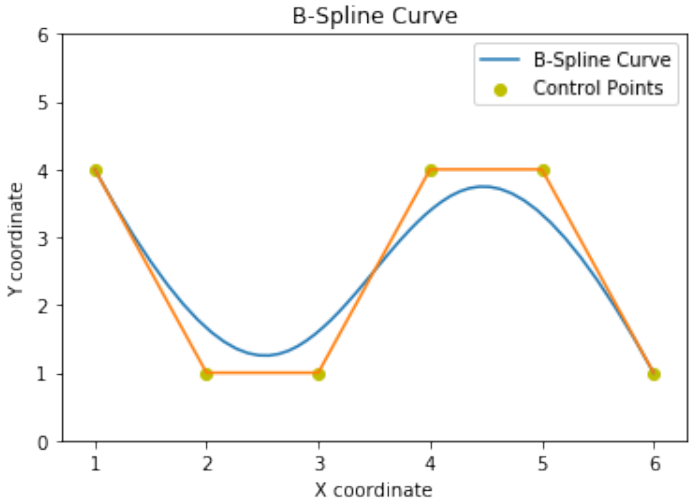

Figure 1:
A B-Spline curve with six control points

2.1 Order

For a point on a B-Spline curve, the order of the basis function speaks about the number of nearby control points that influence the given point. The degree p of the basis function is one less than the order of the curve. The following figure shows a B-Spline curve with the same number and position of control points but with different orders.

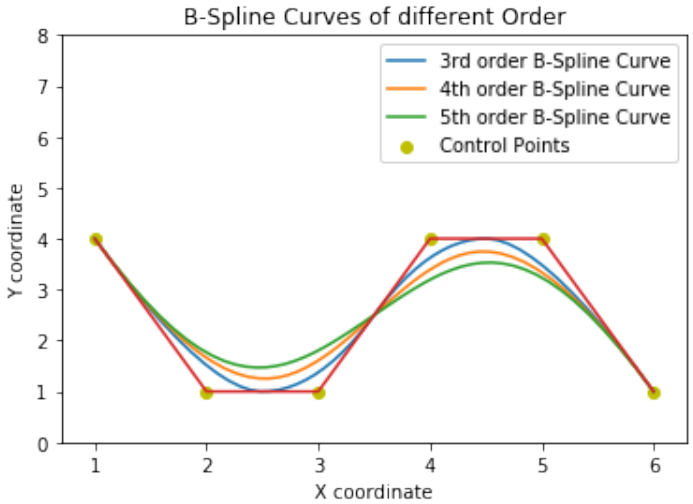

Figure 2:
B-Spline curves with same control points but different order

2.2 Knot vector

A knot vector is an array with an ascending order of parameter values written as $\Xi = \{\xi_0, \xi_1, \xi_2 ....., \xi_{n+p}\}$ (ξ_i is called ith knot and $\{\xi_i, \xi_{i+1}\}$ is called ith Knot span), with $n+1$ basis functions which will be discussed in later sections. The number of knots in a knot vector is equal to the summation of the degree of the curve and total number of control points defining the curve. B-Spline curves are defined in parametric space which is divided by knot spans. Knot vector should be an ascending order of knots. For example $\{0, 0, 1, 1, 2, 3, 3\}$ is valid but not $\{0, 0, 1, 2, 3, 2, 3\}$. There is no difference between $\Xi = \{0, 0, 0, 1, 2, 3, 4, 4, 4\}$ and $\Xi = \{0, 0, 0, 1/4, 2/4, 3/4, 1, 1, 1\}$ which can be seen from Fig.(3) because the latter can be obtained by dividing the former by 4.

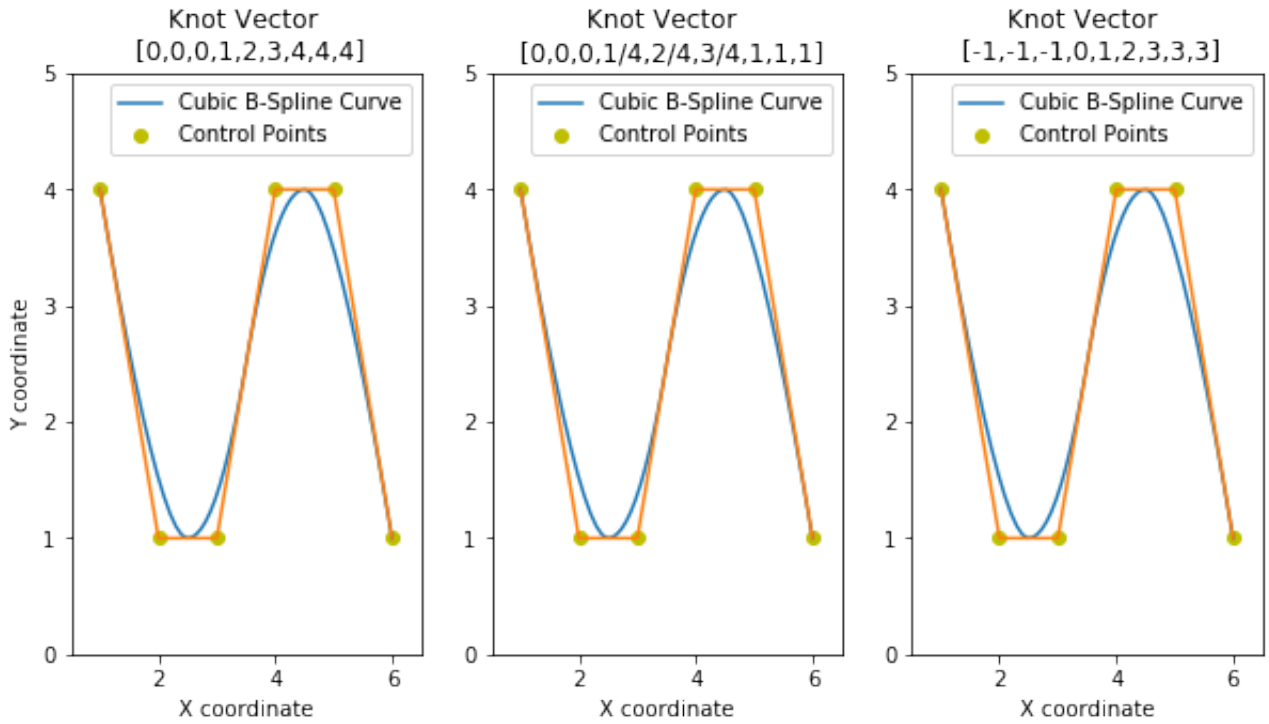

Figure 3:
B-Spline curves showing same trend with different Knot Vectors

The effect of the control points on a B-Spline curve is completely defined by knot vector parameter values as shown in Fig.(4).

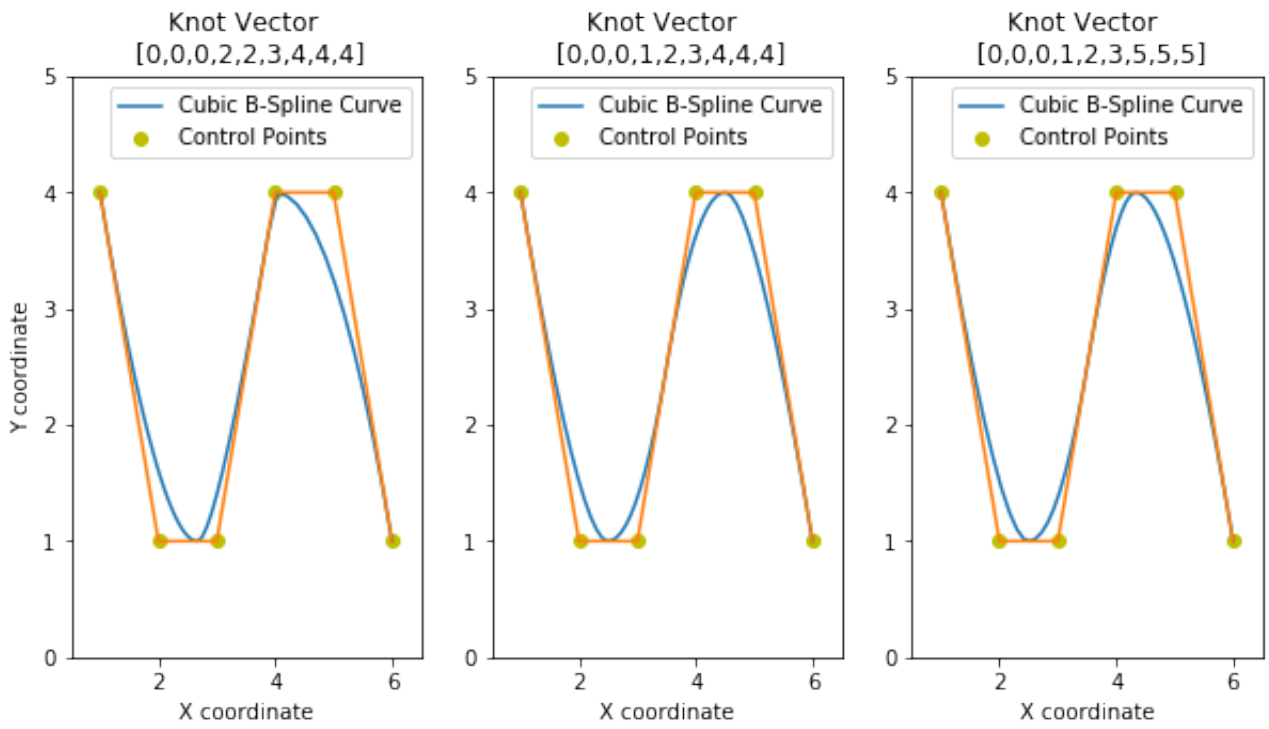

Figure 4:
B-Spline curves with same control points but with different Knot vectors

2.3 Control points

The co-ordinates and number of the control points determine the shape of the curve, and the shape can be varied by altering the knot values in the knot vector, as discussed in section (2.2). A span on the B-Spline curve is controlled by $p+1$ number of points. The total number of control points is given by $n_{cp}(\xi)$ = total number of knots in $[\Xi] - (p+1)$. For a *pth* degree curve, at least $p+1$ control points have to be defined.

2.4 B-Spline basis functions

For a given Knot vector Ξ, the B-spline basis function for polynomial degree ≥ 1 is defined by a recursive function [7]

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

with

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1} \\ 0 & otherwise \end{cases} \tag{2}$$

2.4.1 Properties

1. $N_{i,0}(\xi)$ is a step wise function with a value 1 over the half open interval $\xi \in [\xi_i \leq \xi < \xi_{i+1})$ and zero on the rest.
2. Basis functions sum upto to unity $\sum_{i=0}^{n} N_{i,p}(\xi) = 1$.
3. Basis functions are non-negative $N_{i,p}(\xi) \geq 0$ over the entire domain.

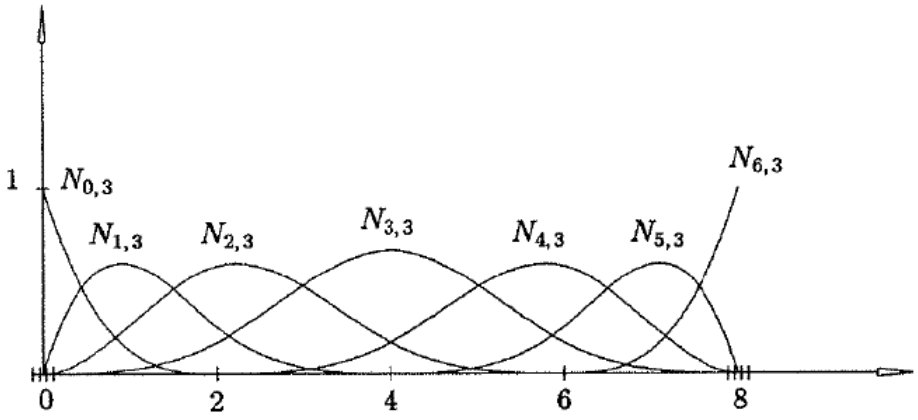

Figure 5:
Cubic B-Spline functions with a uniform knot vector [7]

The following Python function outputs non zero basis functions in a given knot vector span

```
-------------------------------------------------------------------------------
def BasisFuns(i,xi,p,XI):
#----------Inputs---------#
i  - knot span
xi - parametric coordinate
p  - Degree of the curve
XI - knot vector of the curve
#----------Outputs---------#
non zero basis functions for the given parametric coordinate
-------------------------------------------------------------------------------
```

2.4.2 Derivatives

The first derivative of a B-Spline basis function [5] with its variable ξ is given by

$$\frac{d}{d\xi}N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}N_{i+1,p-1}(\xi) \tag{3}$$

Higher-order derivatives are not necessary for IGA formulation.
The following Python function outputs derivatives of non zero basis functions in a given knot vector span.

```
-------------------------------------------------------------------------------
def DersBasisFuns(i,xi,p,g,XI):
#----------Inputs---------#
i  - knot span
xi - parametric coordinate
p  - Degree of the curve
XI - knot vector of the curve
g  - derivatives upto and including g(th)
#----------Outputs---------#
The function returns non zero basis functions and their derivatives
upto and including gth derivative for the given parametric coordinate
-------------------------------------------------------------------------------
```

2.5 B-Spline curves

A $pth - degree$ B-Spline curve with a set of control points P_i is given by [7]

$$C(\xi) = \sum_{i=0}^{n} N_{i,p}(\xi) P_i \qquad \xi_0 \leq \xi \leq \xi_{n+p} \tag{4}$$

defined on the knot vector $\Xi = \{\xi_0, \xi_1, \xi_2 ....., \xi_{n+p}\}$

2.6 B-Spline surfaces

A B-Spline surface $S(\xi, \eta)$ is built by tensor product between B-Spline curves along each parametric direction (ξ, η). It requires knot vectors ($\Xi = \{\xi_0, \xi_1, \xi_2 ....., \xi_{n+p}\}$,$H = \{\eta_0, \eta_1, \eta_2 ....., \eta_{m+q}\}$) along each parametric direction and control net $P_{i,j}$

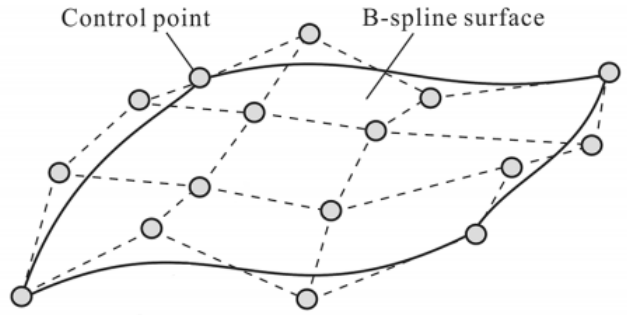

Figure 6:
A B-Spline surface with control points net [9]

$$S(\xi, \eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(\xi) N_{j,q}(\eta) P_{i,j} \tag{5}$$

where p,q are the degrees of the B-Spline basis functions along ξ, η directions respectively, and n,m are the number of control points along ξ, η directions respectively. Eq. (6) can be compactly written as

$$S(\xi, \eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,j}^{p,q}(\xi, \eta) P_{i,j} \tag{6}$$

2.6.1 Derivatives

The partial derivatives of bivariate B-Spline basis functions w.r.t parametric coordinates are given as [5]

$$\frac{\partial N_{i,j}^{p,q}(\xi, \eta)}{\partial \xi} = \frac{d}{d\xi}\Big(N_{i,p}(\xi)\Big) N_{j,q}(\eta) \qquad \frac{\partial N_{i,j}^{p,q}(\xi, \eta)}{\partial \eta} = \frac{d}{d\eta}\Big(N_{j,q}(\eta)\Big) N_{i,p}(\xi) \tag{7}$$

3 Non Uniform Rational B-Splines

NURBS are very often used in computer-aided design(CAD), manufacturing (CAM) and engineering (CAE) due to its flexibility to represent complex geometries. NURBS curves and surfaces are considered as the generalization of B-Spline and Bezier curves and surfaces. A NURBS basis function is defined by its order and knot vector.

3.1 NURBS basis functions

NURBS basis functions $R_{i,p}(\xi)$ are defined as [7]

$$R_{i,p}(\xi) = \frac{N_{i,p}(\xi)w_i}{\sum_{i=0}^{n} N_{i,p}(\xi)w_i} \tag{8}$$

where $N_{i,p}(\xi)$ is the ith B-Spline basis function with degree p and w_i denotes weight of the ith control point (P_i). When $w_i = constant \quad \forall i$ the NURBS basis function reduces to B-Spline basis function.
The usage of weights can be illustrated in Fig.(7). The same circle can be drawn with different number of control points by altering their weights.

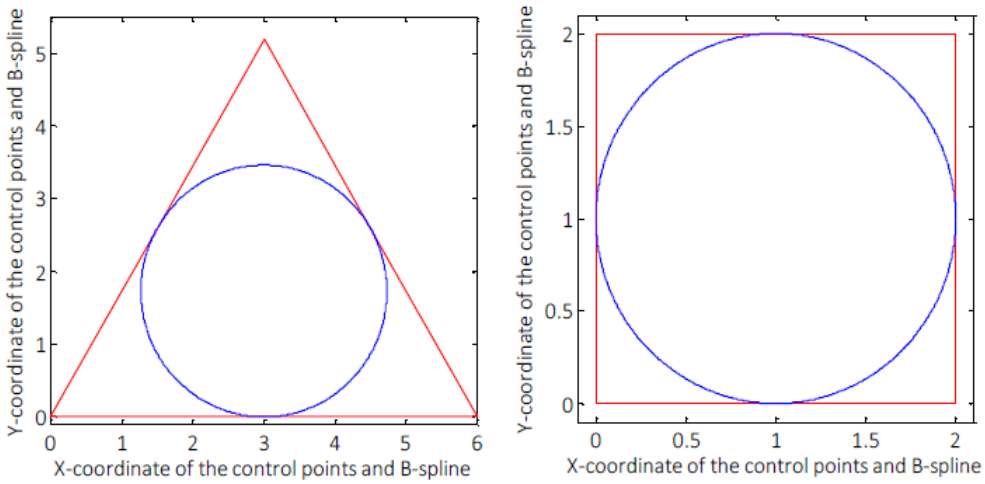

Figure 7:
NURBS circle with different control points and weights [8]

3.1.1 Derivatives

The first derivative of a NURBS basis function with its variable ξ is given by [5]

$$\frac{d}{d\xi}R_{i,p}(\xi) = \frac{N'_{i,p}(\xi)W(\xi) - N_{i,p}(\xi)W'(\xi)}{W^2(\xi)}w_i \tag{9}$$

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

and $W'(\xi) = \sum_{i=0}^{n} N'_{i,p}(\xi)w_i$

3.2 NURBS Curves

The p^{th} degree NURBS curve is given by [7]

$$C(\xi) = \frac{\sum_{i=0}^{n} N_{i,p}(\xi)w_iP_i}{\sum_{i=0}^{n} N_{i,p}(\xi)w_i} \qquad \xi_0 \leq \xi \leq \xi_{n+p} \tag{10}$$

in short form

$$C(\xi) = \sum_{i=0}^{n} R_{i,p}(\xi) P_i \tag{11}$$

A NURBS curve with different weights on control points along with its basis function is shown in Fig. (8)

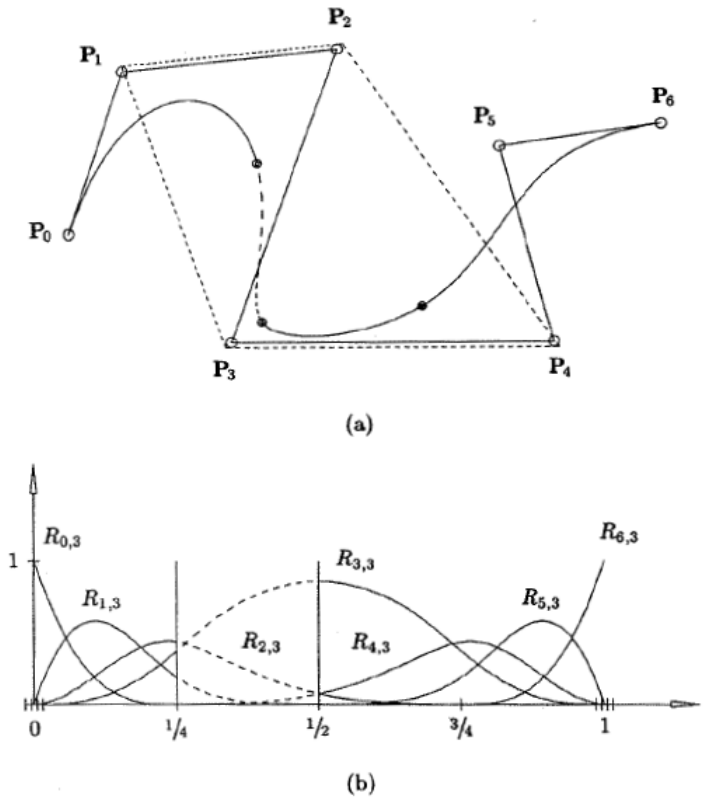

Figure 8:
$\Xi = \{0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4\}, w = \{1, 1, 1, 3, 1, 1, 1\}$
(a) A third degree NURBS curve; (b) Associated NURBS basis functions [7]

3.2.1 Properties

1. NURBS basis functions sum upto to unity $\sum_{i=0}^{n} R_{i,p}(\xi) = 1$
2. NURBS basis functions are non-negative $R_{i,p}(\xi) \geq 0$ over the entire domain
3. $R_{0,p}(0) = R_{n,p}(1) = 1$
4. For $w_i = 1$ for all i, NURBS basis functions $R_i(\xi)$ reduces B-Spline basis functions $N_i(\xi)$

3.3 NURBS Surfaces and solids

NURBS Surfaces and solids are generated by the tensor product between NURBS curve basis functions.

1. NURBS Surfaces:
   A NURBS surface with degree p in ξ direction and degree q in η direction is defined as [2]

$$S(\xi, \eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}^{p,q}(\xi, \eta) P_{i,j} \tag{12}$$

where the bivariate NURBS basis functions are given by

$$R_{i,j}^{p,q}(\xi,\eta) = \frac{N_{i,p}(\xi)N_{j,q}(\eta)w_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(\xi)N_{j,q}(\eta)w_{i,j}} \tag{13}$$

2. NURBS Solids:
   A NURBS solid with degree p, q, r in ξ, η, ζ directions respectively is defined as [2]

$$S(\xi,\eta,\zeta) = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l}R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta)P_{i,j,k} \tag{14}$$

   where the $R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta)$ is given by

$$R_{i,j,k}^{p,q,r}(\xi,\eta,\zeta) = \frac{N_{i,p}(\xi)N_{j,q}(\eta)N_{k,r}(\zeta)w_{i,j,k}}{\sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l}N_{i,p}(\xi)N_{j,q}(\eta)N_{k,r}(\zeta)w_{i,j,k}} \tag{15}$$

3.4 Derivatives of NURBS bivariate Basis Functions

The first partial derivatives of the NURBS bivariate basis function are given by [5]

$$\frac{\partial R_{i,j}^{p,q}}{\partial\xi} = \frac{N_{i,p}^{'}N_{j,q}W - N_{i,p}N_{j,q}W_{\xi}^{'}}{W^2}w_{i,j} \tag{16}$$

$$\frac{\partial R_{i,j}^{p,q}}{\partial\eta} = \frac{N_{i,p}N_{j,q}^{'}W - N_{i,p}N_{j,q}W_{\eta}^{'}}{W^2}w_{i,j} \tag{17}$$

where

$$W = \sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}N_{j,q}w_{i,j} \tag{18}$$

$$W_{\xi}^{'} = \sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}^{'}N_{j,q}w_{i,j} \tag{19}$$

$$W_{\eta}^{'} = \sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}N_{j,q}^{'}w_{i,j} \tag{20}$$

The first partial derivatives of the trivariate basis functions can be computed similarly, like the bivariate basis functions using a chain rule.

4 Implementation Procedure for IGA

This section describes the step-by-step implementation of the Isogeometric analysis. A modified FEM code structure can be used to implement IGA. Similar to FEM, IGA can be divided into pre-processing, processing and post-processing stages. The flow in an IGA analysis in each stage is shown with the help of a flow chart in respective sections.

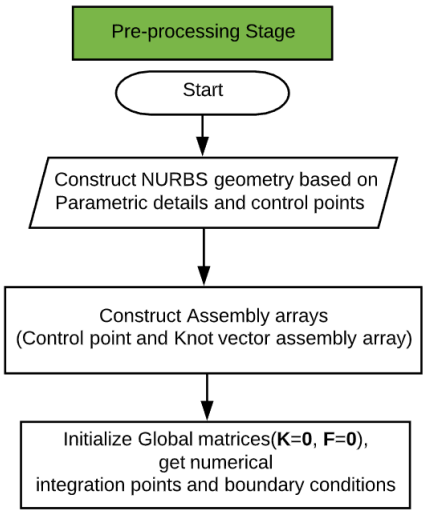

Figure 9:
Flow chart describing Pre-processing Stage of IGA

4.1 Pre-processing Stage of the Analysis

This subsection mainly deals with NURBS based geometry creation, types of assembly arrays needed to assemble discretized geometries and how to deal with homogeneous and non-homogeneous boundary conditions on boundary defining control points.

4.1.1 Geometry Creation

As mentioned before in the section 3, apart from the physical details of the geometry like length, width and thickness etc. the construction of NURBS discretized geometry requires parametric details such as control points, knot vectors and the order of the NURBS curve. Commercial software like "Rhino" can be used to extract parametric details of the complex NURBS geometry.
The following function in Python is used for finding a point on the NURBS surface.

```
def NURBS_Surface_Point(n,p,XI,m,q,ETA,Pw,xi,eta):
```

Where, p and q are the degrees of the NURBS curve in ξ and η directions respectively, Pw control points vector, XI and ETA are knot vectors in ξ and η directions respectively. n and m are calculated using below code

```
n=(np.size(XI)-1) -p-1
m=(np.size(ETA)-1)-q-1
```

xi and eta are any points in knot span, for example if $XI = \{0, 0, 1, 2, 3, 4, 4\}$, xi can be 2.5 which belongs to knot span $\{2, 3\}$ in XI.
The entire surface can be generated by looping over knot span points and by extracting surface

points by using the above function and plotting the output. An example output of the geometry generated with the written code is shown below

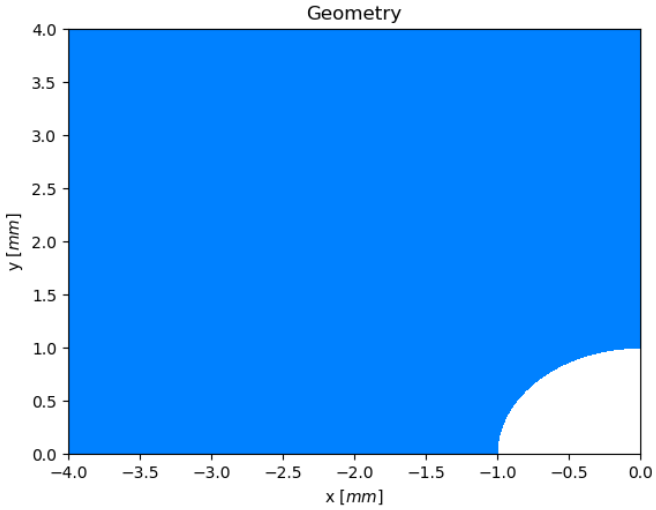

Figure 10:
NURBS Geometry

Parametric details of the geometry [2]:

```
Pw=
[[-1.,0.,0.,1],[-0.85,0.35,0.,0.85],[-0.35,0.85,0.,0.85],[0.,1.,0.,1]],
[[-2.5,0.,0.,1.],[-2.5,0.75,0.,1.],[-0.75,2.5,0.,1.],[0.,2.5,0.,1.]],
[[-4,0.,0.,1.],[-4,4.,0.,1.],[-4.,4.,0.,1.],[0.,4.,0.,1.]]
p=2 and q=2 # Degree of the curve in xi and eta directions
XI  = [0., 0., 0., 1., 2., 2., 2.]
ETA = [0., 0., 0., 1., 1., 1.]
```

4.1.2 Assembly arrays

Assembly arrays are required to assemble local discretized geometries to global geometry. For IGA, as knot vector and control points define the geometry, two assembly arrays (1) Control point assembly array and (2) Knot vector connectivity array are required.

1. Control point assembly array:
   The degree of the NURBS curve determines the number of control points (n^e_{cp}) present in an IGA element. Considering a two-dimensional element (Ω^e)
   $n^e_{cp} = (p+1)(q+1)$
   The details of the control points are stored row wise in assembly array. The following function generates a control point assembly array

```
def ControlPointAssembly(n,p,m,q,ele_no):

#-----------------Inputs----------------------#
# n-no.of control points along xi direction
# p-Degree of basis function along xi direction
# m-no.of control points along eta direction
# q-Degree  of basis function along eta direction
```

```
#-----------------Output----------------------#
CP - Control point assembly array
```

A control point assembly contains a two-dimensional array with control point numbers of each element, row-wise. An assembly array with 2 elements, as in Fig. (11) is shown below

$$\mathbf{CP} = \begin{bmatrix} 1 & 2 & 4 & 5 \\ 2 & 3 & 5 & 6 \end{bmatrix} \tag{21}$$

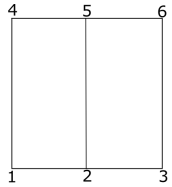

Figure 11:
A simple 2D square geometry with two elements

2. Knot vector connectivity array:
The knot vector connectivity matrix **CP** is a row-wise matrix with each row corresponds to the respective element. The columns correspond to span ranges (Span_XI and Span_ETA) along ξ and η directions.
Knot vector array along ξ and η direction for geometry, Fig.(11):
$XI = [0, 0, 1, 2, 2]$
$ETA = [0, 0, 1, 1]$
Span_XI is a 2D array which has rows equal to number of elements in ξ direction.

$$\mathbf{Span\_XI} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \tag{22}$$

Similarly with only one element along η direction

$$\mathbf{Span\_ETA} = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{23}$$

Knot connectivity for above geometry is given by,

$$\mathbf{CP} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \tag{24}$$

The following lines of code executes Knot Connectivity array

```
uniqueXI  = np.unique(XI)
uniqueETA = np.unique(ETA)
nelXI  = ncpxi-p
nelETA = ncpeta-q
knotConnectivity = np.zeros((nel,2))
Span_XI  = np.zeros((nelXI,2))
#Span_XI have rows equal to number of elements in xi direction
Span_ETA = np.zeros((nelETA,2))
#Span_ETA have rows equal to number of elements in eta direction
count=0
```

```
for i in range(0,nelETA):
for j in range(0,nelXI):
knotConnectivity[count,:] = [j+1,i+1]
# First and second coloumns of 'knotConnectivity' store the global number
of knot span ranges
# or row number of Span_XI and Span_ETA arrays
Span_XI[j,:]= [uniqueXI[j],uniqueXI[j+1]]
Span_ETA[i,:]= [uniqueETA[i],uniqueETA[i+1]]
count=count+1
knotConnectivity = knotConnectivity -1
knotConnectivity = knotConnectivity.astype(int)
```

4.1.3 Boundary Conditions

A brief description of how to define boundary conditions (BCS) is described in this section. When the order of the NURBS curve is two in each direction, a traditional way of defining homogeneous and inhomogeneous boundary conditions as in FEM can be followed. In any other case, a special treatment for defining BCS have to be followed, which is not in the scope of this project. Procedures like the least square minimization method are usually adopted for this purpose. As discussed before, an exception is when all the control points are on the surface itself, we can use common way of defining BCS.

4.2 Processing Stage of the Analysis

In the processing stage of analysis, it is required to compute the global elemental stiffness matrix and global force vector and solve these for the solution field. To formulate the matrices, it requires NURBS basis functions and their derivatives evaluation. A numerical integration scheme, like the Gauss-Legendre rule, is employed to solve the volume and area integrals involved in forming the stiffness matrix and internal force vector. Numerical integration involves mapping elements from physical space to master space (which is also called the unit domain). As NURBS basis functions are defined in parametric space as given in Eq. (8)it requires an additional mapping from physical space to parametric space ($\Omega_e \rightarrow \widetilde{\Omega_e}$). Later parametric space can be mapped on to master space ($(\widetilde{\Omega_e} \rightarrow \overline{\Omega_e})$). This procedure is illustrated in Fig.(12).

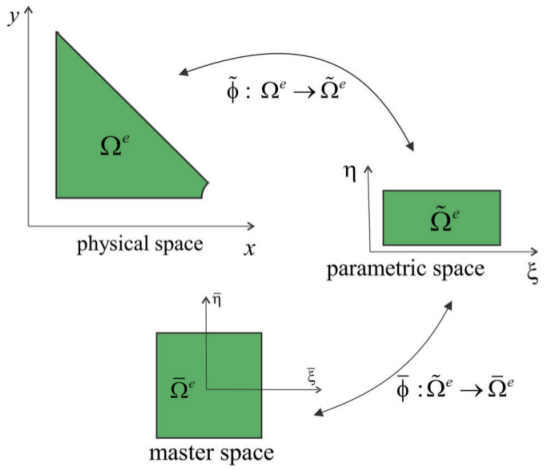

Figure 12:
Mapping IGA Physical element to Master element [2]

1. Mapping from master space to parametric space [2]:
   Consider a discretized IGA surface which is defined in parametric space $\widetilde{\Omega_e} = [\xi_i, \xi_{i+1}] \otimes [\eta_i, \eta_{i+1}]$, Refer Fig.(12). The NURBS basis functions and their derivatives are evaluated at ξ, η of the element $\widetilde{\Omega_e}$. These ξ, η co-ordinate values are calculated by a linear mapping as shown below

$$\xi = \frac{1}{2}[(\xi_{i+1} - \xi_i)\overline{\xi} + (\xi_{i+1} + \xi_i)] \tag{25}$$

$$\eta = \frac{1}{2}[(\eta_{i+1} - \eta_i)\overline{\eta} + (\eta_{i+1} + \eta_i)] \tag{26}$$

   where $\overline{\xi}, \overline{\eta}$ are the integration points defined in master space
   $\mathbf{J}_2$ matrix is defined as

$$\mathbf{J}_2 = \frac{\partial \xi}{\partial \overline{\xi}} \frac{\partial \eta}{\partial \overline{\eta}} \tag{27}$$

   Determinant of $\mathbf{J}_2$ matrix is required in numerical integration scheme for linear mapping

2. Mapping from physical space to parametric space [2]:
   The Jacobian matrix $\mathbf{J}_1$ used to map from physical space to parametric space ($\Omega_e \rightarrow \widetilde{\Omega_e}$) is computed as:

$$\mathbf{J}_1 = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \tag{28}$$

   The components of the $\mathbf{J}_1$ matrix are calculated using Eq. (36).

$$\frac{\partial x}{\partial \xi} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \xi} x_i \qquad \frac{\partial x}{\partial \eta} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \eta} x_i \tag{29}$$

$$\frac{\partial y}{\partial \xi} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \xi} y_i \qquad \frac{\partial y}{\partial \eta} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \eta} y_i \tag{30}$$

The two different mappings described above can be illustrated with an example considering $\mathbf{F}(x, y)$ integrated over the physical space Ω

$$\begin{aligned}
\int_{\Omega} \mathbf{F}(x,y)d\Omega &= \sum_{e=1}^{nel} \int_{\Omega_e} \mathbf{F}(x,y)d\Omega \\
&= \sum_{e=1}^{nel} \int_{\widetilde{\Omega_e}} \mathbf{F}(\xi,\eta)|\mathbf{J}_1|d\xi d\eta \\
&= \sum_{e=1}^{nel} \int_{\overline{\Omega_e}} \mathbf{F}(\overline{\xi},\overline{\eta})|\mathbf{J}_1||\mathbf{J}_2|d\overline{\xi}d\overline{\eta} \\
&= \sum_{e=1}^{nel} \int_{-1}^{1}\int_{-1}^{1} \mathbf{F}(\overline{\xi},\overline{\eta})|\mathbf{J}_1||\mathbf{J}_2|d\overline{\xi}d\overline{\eta} \\
&= \sum_{e=1}^{nel} \left[\sum_{i=1}^{n_{gp}^{e}} \mathbf{F}(\overline{\xi}_i,\overline{\eta}_i) g w_i |\mathbf{J}_1||\mathbf{J}_2|d\overline{\xi}d\overline{\eta} \right]
\end{aligned}$$

where nel is the total number of elements and n_{gp}^{e}, gw_i denotes the number of Gauss points and their respective Gauss weights.
The following function is used to get the J1 and J2 matrix along with their determinants.

```
-------------------------------------------------------------
def Jacobian12(xi,eta,elXI,elETA):
#-----------------Input-----------------#
Parametric Co-ordinates (xi,eta) of Gauss Points
Knot span range (elXI,elETA) in xi and eta direction
#-----------------Output----------------#
J1 and J2 matrices along with their determinants
-------------------------------------------------------------
```

The formulated global stiffness matrix and force vector are solved using a numerical scheme like the Newton-Raphson method for the solution field.
A flow chart describing processing stage flow is shown in Fig.(13)

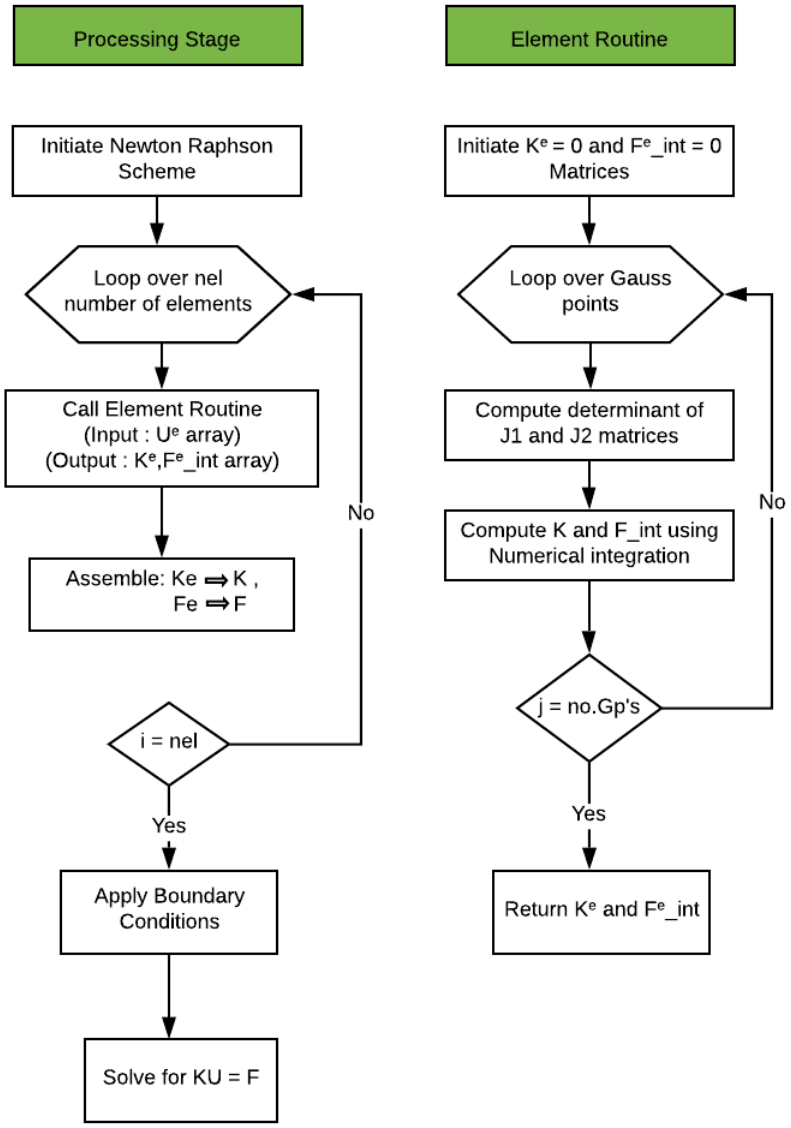

Figure 13:
Flow chart describing Processing Stage of IGA

4.3 Post-processing Stage of the Analysis

This section deals with the visualization of the deformed geometry and how a displacement and solution dependent variables are plotted.

1. Visualization of the deformed NURBS geometry:

   Visualization of the deformed geometry can be done in the same manner as the visualization of the initial geometry. After determining the displacement field at the control points, they are added to the initial control points coordinates [**P**].

   $$[\mathbf{P}_{new}] = [\mathbf{P}] + [\mathbf{U}]$$

   $[\mathbf{P}_{new}]$ control points array is used to plot the deformed geometry. As discussed in sec-

tion (4.1.1), instead of initial control point matrix of the geometry $\mathbf{P}$, P_{new} is given as input to the function as a transpose

```
def NURBS_Surface_Point(n,p,XI,m,q,ETA,Pw_new,xi,eta):
```

and the deformed geometry can be plotted.

2. Plotting of displacements and solution dependent variables:

   A contour plot can be made using the displacement values (U) on deformed or undeformed geometry. Due to the higher continuity of the NURBS basis functions, stress recovery techniques are not necessary to extract the solution field on the deformed geometry.

   plt.contourf a matplotlib function is used to plot displacements, reaction forces, and electrical potentials, so that the comparison with Abaqus post-processing plots would be easier. The latter plots the solution fields as a contour plot.

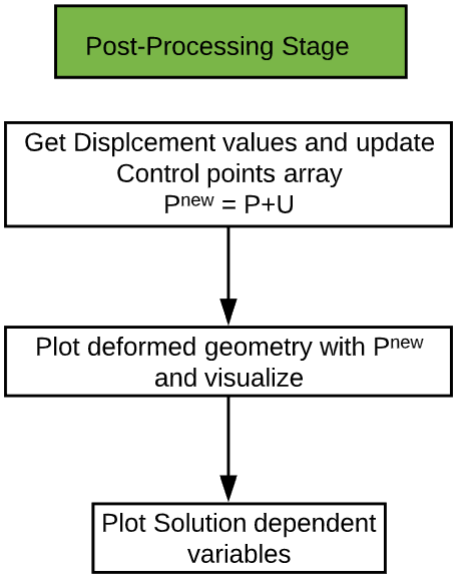

Figure 14:
Flow chart describing Post-processing Stage of IGA

5 Mechanical Case

5.1 Governing Equations

The governing equation for mechanical deformation is based on conservation of linear momentum which can be written as [3]

$$\sigma_{ij,i} + b_j = 0 \tag{31}$$

where σ_{ij} and b_i is the Cauchy stress tensor and the body force. Due to the static nature of the analysis, the inertial term is not included in the eq(31).
Stress and strain are related by following constitutive equation

$$\sigma_{ij} = c_{ijkl}\epsilon_{kl} \tag{32}$$

The infinitesimal strain theory is adopted for the analysis in which displacements of the material particles are considered to be very small compared to the dimensions of the body under loading. Strain in a small strain setting can be written as

$$\epsilon_{ij} = \frac{1}{2}[u_{i,j} + u_{j,i}] \tag{33}$$

where u_i are the displacements in the body

5.2 Weak Formulation

Consider a domain Ω with Γ_u as prescribed displacements and Γ_t as traction boundary conditions. The domain boundary can be represented as $\Gamma = \Gamma_u \cup \Gamma_t$ and $\Gamma_u \cap \Gamma_t = \Phi$ By using the principle of virtual work the eq(31) can be written as

$$\delta W = \int_\Omega (\sigma_{ij,i} + b_j)\delta u_j dV = 0 \tag{34}$$

with, $u = u_o$ on Γ_u (essential boundary condition) and $\sigma_{ij}n_j = \bar{t}_j$ on Γ_t (natural boundary condition)
where δu_j is the virtual displacement field, n_j is unit normal to the surface
Applying integration by parts to the stress term under integral and by making using of conservation of angular momentum ($\sigma_{ij} = \sigma_{ji}$) and Gauss divergence theorem (converting volume integral to surface integral) we approach at the following equation

$$\delta W = \int_\Omega \sigma_{ij}\epsilon_{ij} d\Omega - \left[\int_\Gamma \bar{t}_j \delta u_j d\Gamma_t + \int_\Omega b_j \delta u_j d\Omega\right] \tag{35}$$

as an additional requirement δu_j must be zero at the essential boundary conditions (Γ_u) for a unique solution.

5.3 IGA Formulation

The advantage of IGA over FEM formulation lies in its basis functions incorporation and its ability to capture the exact geometry. While the FEM uses lagrangian basis functions, IGA uses NURBS basis functions, which are used to generate the geometry itself. As discussed in the previous sections, a multidimensional NURBS basis function is represented by $R^{p,q,r}_{i,j,k}(\xi,\eta,\zeta) \rightarrow R_i$. The isogeometric element is represented by basis function R_i and control points P_i as [2]

$$\mathbf{x}^e = \sum_{i=1}^{n^e_{cp}} R_i P_i \tag{36}$$

By Galerkin approach, the displacements and virtual displacements are given by

$$\mathbf{u}^e = \sum_{i=1}^{n_{cp}^e} R_i \mathbf{u}_i \qquad \delta\mathbf{u}^e = \sum_{i=1}^{n_{cp}^e} R_i \delta\mathbf{u}_i \tag{37}$$

where $\mathbf{u}_i$ and $\delta\mathbf{u}_i$ are values at ith control point. The strain displacement matrix $\mathbf{B}$ is given by

$$\mathbf{B} = \begin{bmatrix} R_{1,x} & 0 & 0 & R_{2,x} & 0 & 0 & .... & R_{n_{cp}^e,x} & 0 & 0 \\ 0 & R_{1,y} & 0 & 0 & R_{2,y} & 0 & .... & 0 & R_{n_{cp}^e,y} & 0 \\ 0 & 0 & R_{1,z} & 0 & 0 & R_{2,z} & .... & 0 & 0 & R_{n_{cp}^e,z} \\ R_{1,y} & R_{1,x} & 0 & R_{2,y} & R_{2,x} & 0 & .... & R_{n_{cp}^e,y} & R_{n_{cp}^e,x} & 0 \\ 0 & R_{1,z} & R_{1,y} & 0 & R_{2,z} & R_{2,y} & .... & 0 & R_{n_{cp}^e,z} & R_{n_{cp}^e,y} \\ R_{1,z} & 0 & R_{1,x} & R_{2,z} & 0 & R_{2,x} & .... & R_{n_{cp}^e,z} & 0 & R_{n_{cp}^e,x} \end{bmatrix} \tag{38}$$

By substituting Eqs, (37) and (38) in Eq.(35), the weak form in matrix terms can be written as

$$\sum_{e=1}^{nel} \left[\left(\int_{\Omega_e} \mathbf{B}^T \mathbf{C} \mathbf{B} d\Omega \right) \right] \mathbf{u} = \int_{\Gamma_t^e} \mathbf{R}^T . \mathbf{t} d\Gamma + \int_{\Omega_t^e} \mathbf{R}^T . \mathbf{f} d\Omega \tag{39}$$

where R is defined as
for the boundary Γ^e

$$\mathbf{R} = \begin{bmatrix} R_1(\xi,\eta) & 0 & R_2(\xi,\eta) & 0 & .... & R_{n_{cp}^e}(\xi,\eta) & 0 \\ 0 & R_1(\xi,\eta) & 0 & R_2(\xi,\eta) & .... & 0 & R_{n_{cp}^e}(\xi,\eta) \end{bmatrix} \tag{40}$$

for the domain Ω^e

$$\mathbf{R} = \begin{bmatrix} R_1(\xi,\eta,\zeta) & 0 & R_2(\xi,\eta,\zeta) & 0 & .... & R_{n_{cp}^e}(\xi,\eta,\zeta) & 0 \\ 0 & R_1(\xi,\eta,\zeta) & 0 & R_2(\xi,\eta,\zeta) & .... & 0 & R_{n_{cp}^e}(\xi,\eta,\zeta) \end{bmatrix} \tag{41}$$

Eq.(39) can be rewritten in a standard matrix form as

$$\sum_{e=1}^{nel} [\mathbf{K}^e \mathbf{U}^e = \mathbf{F}^e] \tag{42}$$

where $\mathbf{K}^e$ is isogeometric element's stiffness matrix, $\mathbf{U}^e$ is displacement vector and $\mathbf{F}^e$ force vector

6 Piezoelectric Case

6.1 Governing Equations for Piezoelectric Materials

The coupled electro-mechanical interactions are governed by conservation of momentum and Gauss's law as [6]:

$$\sigma_{ij,j} + f_i = 0 \tag{43}$$

$$D_{i,i} - q = 0 \tag{44}$$

Where, f_i is body force, q is electrical charge, σ_{ij} is Cauchy stress tensor and D_i is electrical displacement vector. The constitutive equations for electromechanical coupling are defined as

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl} - e_{kij}E_k \tag{45}$$

$$D_i = e_{ikl}\varepsilon_{kl} + \kappa_{ik}E_k \tag{46}$$

Where, C , e and κ are elastic, piezoelectric and dielectric material constants respectively. The Cauchy strain tensor is defined as:

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \tag{47}$$

and electric field vector as:

$$E_i = -\phi_{,i} \tag{48}$$

6.2 Weak Formulation

Applying the principle of virtual work to the Eq.(43) and Eq. (44) we can write [6]

$$\int_\Omega (\sigma_{ij,j} + f_i)\delta u_i d\Omega = 0 \tag{49}$$

$$\int_\Omega (D_{i,i} - q)\delta\phi d\Omega = 0 \tag{50}$$

with,
essential boundary conditions: $u = u_o$ on Γ_u and $\Phi = \Phi_0$ on Γ_Φ
natural boundary condition: $\sigma_{ij}n_j = \bar{t}_j$ on Γ_t and $D_i n_i = q_0$ on Γ_q
where δu_i and $\delta\phi$ are virtual or arbitrary displacement and potential fields.
Integrating Eq. (49) and Eq. (50) by parts and later applying Gauss divergence theorem and boundary conditions we approach at weak form

$$\int_\Omega \sigma_{ij}\delta\epsilon_{ij}d\Omega - \left[\int_\Gamma \bar{t}_i\delta u_i d\Gamma + \int_\Omega f_i\delta u_i d\Omega\right] = 0 \tag{51}$$

$$\int_\Omega D_i\delta E_i d\Omega - \left[\int_\Gamma Q\delta\phi d\Gamma + \int_\Omega q\delta\phi d\Omega\right] = 0 \tag{52}$$

6.3 IGA Formulation

By Galerkin approach, displacements, potentials and their virtual values are given by below equations [2]

$$\mathbf{u}^e = \sum_{i=1}^{n_{cp}^e} R_i\mathbf{u}_i \qquad \delta\mathbf{u}^e = \sum_{i=1}^{n_{cp}^e} R_i\delta\mathbf{u}_i \tag{53}$$

$$\Phi^e = \sum_{i=1}^{n_{cp}^e} R_i \Phi_i \qquad \delta\Phi^e = \sum_{i=1}^{n_{cp}^e} R_i \delta\Phi_i \tag{54}$$

where $\mathbf{u}_i$, $\delta\mathbf{u}_i$, $\delta\Phi_i$ and Φ_i are values at ith control point.
By substituting Eqs, (53) and (54) in Eq.(51) and Eq.(52) the weak form in matrix notation can be written as

$$\begin{bmatrix} K_{MM} & K_{ME} \\ K_{EM} & K_{EE} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \phi \end{bmatrix} = \begin{bmatrix} F_M \\ F_E \end{bmatrix} \tag{55}$$

Where,

$$K_{MM} = \int_\Omega \mathbf{B}_u^T \mathbf{C} \mathbf{B}_u d\Omega \tag{56}$$

$$K_{ME} = \int_\Omega \mathbf{B}_u^T \mathbf{e} \mathbf{B}_e d\Omega \tag{57}$$

$$K_{EM} = \int_\Omega \mathbf{B}_e^T \mathbf{e}^T \mathbf{B}_u d\Omega \tag{58}$$

$$K_{EE} = -\int_\Omega \mathbf{B}_e^T \kappa \mathbf{B}_e d\Omega \tag{59}$$

$$F_M = \int_\Omega \mathbf{R}_u^T \mathbf{f} d\Omega + \int_\Gamma \mathbf{R}_u^T \mathbf{t} d\Gamma \tag{60}$$

$$F_E = \int_\Omega \mathbf{R}_e^T q d\Omega + \int_\Gamma \mathbf{R}_e^T Q d\Gamma \tag{61}$$

where R is defined as
for the boundary Γ

$$\mathbf{R}_u = \begin{bmatrix} R_1(\xi,\eta) & 0 & R_2(\xi,\eta) & 0 & .... & R_{n_{cp}^e}(\xi,\eta) & 0 \\ 0 & R_1(\xi,\eta) & 0 & R_2(\xi,\eta) & .... & 0 & R_{n_{cp}^e}(\xi,\eta) \end{bmatrix} \tag{62}$$

$$\mathbf{R}_e = \begin{bmatrix} R_1(\xi,\eta) & R_2(\xi,\eta) & .... & R_{n_{cp}^e}(\xi,\eta) \end{bmatrix} \tag{63}$$

for the domain Ω

$$\mathbf{R}_u = \begin{bmatrix} R_1(\xi,\eta,\zeta) & 0 & R_2(\xi,\eta,\zeta) & 0 & .... & R_{n_{cp}^e}(\xi,\eta,\zeta) & 0 \\ 0 & R_1(\xi,\eta,\zeta) & 0 & R_2(\xi,\eta,\zeta) & .... & 0 & R_{n_{cp}^e}(\xi,\eta,\zeta) \end{bmatrix} \tag{64}$$

$$\mathbf{R}_e = \begin{bmatrix} R_1(\xi,\eta,\zeta) & R_2(\xi,\eta,\zeta) & .... & R_{n_{cp}^e}(\xi,\eta,\zeta) \end{bmatrix} \tag{65}$$

B matix is given as

$$\mathbf{B}_u = \begin{bmatrix} R_{1,x} & 0 & 0 & R_{2,x} & 0 & 0 & .... & R_{n_{cp}^e,x} & 0 & 0 \\ 0 & R_{1,y} & 0 & 0 & R_{2,y} & 0 & .... & 0 & R_{n_{cp}^e,y} & 0 \\ 0 & 0 & R_{1,z} & 0 & 0 & R_{2,z} & .... & 0 & 0 & R_{n_{cp}^e,z} \\ R_{1,y} & R_{1,x} & 0 & R_{2,y} & R_{2,x} & 0 & .... & R_{n_{cp}^e,y} & R_{n_{cp}^e,x} & 0 \\ 0 & R_{1,z} & R_{1,y} & 0 & R_{2,z} & R_{2,y} & .... & 0 & R_{n_{cp}^e,z} & R_{n_{cp}^e,y} \\ R_{1,z} & 0 & R_{1,x} & R_{2,z} & 0 & R_{2,x} & .... & R_{n_{cp}^e,z} & 0 & R_{n_{cp}^e,x} \end{bmatrix} \tag{66}$$

$$\mathbf{B}_e = \begin{bmatrix} R_{1,x} & R_{2,x} & .... & R_{n_{cp}^e,x} \\ R_{1,y} & R_{2,y} & .... & R_{n_{cp}^e,y} \\ R_{1,z} & R_{2,z} & .... & R_{n_{cp}^e,z} \end{bmatrix} \tag{67}$$

$$\varepsilon = \mathbf{B}_u.\mathbf{u} \tag{68}$$

$$\mathbf{E} = -\mathbf{B}_e.\Phi \tag{69}$$

Eq. (55) can be solved using numerical methods like Newton-Raphson method for displacements and potential solution fields.

7 Modelling and Results

7.1 2D Plate under linear elastic loading

7.1.1 Problem description

A 2D plate is subjected to mechanical loading as shown in Figure(16). The material used is PZT-PIC151 ceramics [4], and the elastic constants of the material are given in the appendix (13.1). The movement of bottom edge AB is fixed in y-direction and left edge AC in x-direction. A displacement of 100 nm (1e-4 mm) is given on the right edge BD.

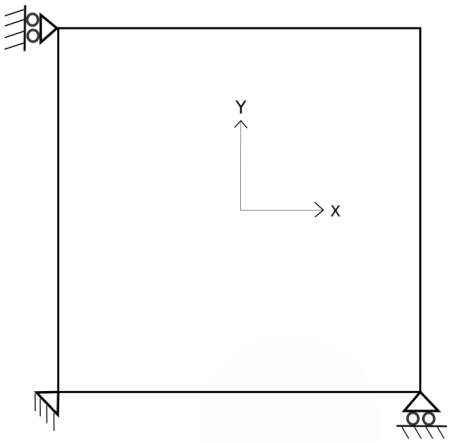

Figure 15: 2D Plate

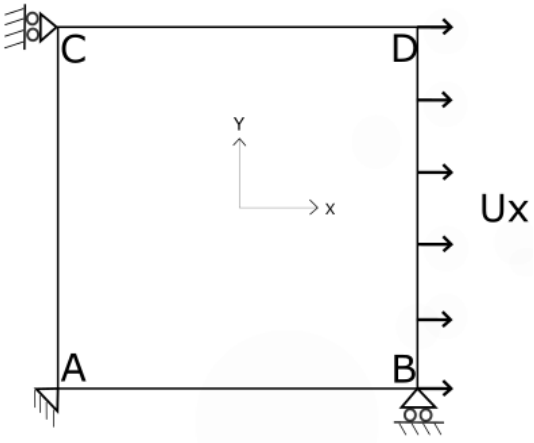

Figure 16: 2D Plate with loading

7.1.2 Parametric details for the plate with single element

The 2nd order NURBS curve is used in both ξ and η directions.

1. Physical details for the geometry:
   L = 10 # Length of the plate in mm
   H = 10 # Height of the plate in mm
   T = 1 # Thickness of the plate in mm

2. Parametric details of the geometry:
   Ξ = [0,0,1,1] # Knot vector in ξ direction
   H = [0,0,1,1] # Knot vector in η direction

   Degree of the curve
   p=1 # Degree of the curve in ξ direction
   q=1 # Degree of the curve in η direction

   Number of control points in each direction
   n_{cp}^{ξ} = len(Ξ) - (p+1) #No.of control points in ξ direction (4-(1+1) = 2)
   n_{cp}^{η} = len(H) - (q+1) #No.of control points in η direction (4-(1+1) = 2)

3. Total number of control points for the geometry
   $n_{cp} = n_{cp}^{\xi} * n_{cp}^{\eta}$ = 2*2 = 4
   The control points are given by

i	$P_{i,0}$	$P_{i,1}$
0	$(0, 0, 0, 1)$	$(0, 10, 0, 1)$
1	$(10, 0, 0, 1)$	$(10, 10, 0, 1)$

with the fourth value in the parentheses being weights of respective control points. As this is the case of single element, there is no need for the control point assembly array and knot vector connectivity matrix.

7.1.3 Results and discussions

Considering the accuracy of the IGA simulation results over FEM results, IGA code generated results can be compared with the Abaqus inbuilt element generated results. An Abaqus plane strain full integration element (**CPE4**) [1] is used for this purpose.
The below figures show the values of displacements (U) and reaction forces (RF) for both Abaqus and IGA element.

A similar contour is used for the program generated results and Abaqus results for easy comparison.
Figure(17) and Figure(18) show the displacement (U1) values of the single CPE4 element and single IGA element at 100 % loading in x-direction respectively.

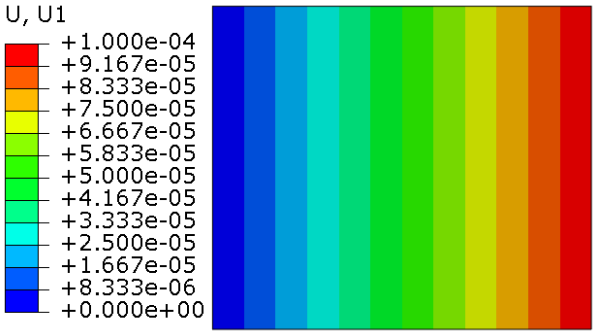

Figure 17: CPE4 Element:U1
Abaqus generated result

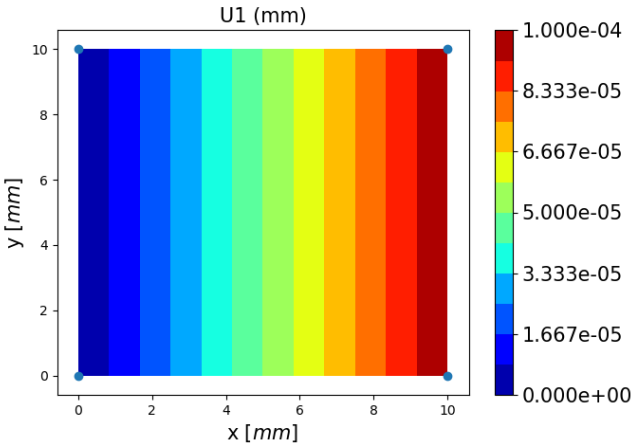

Figure 18: IGA Element:U1
Program generated result

Figure(19) and Figure(20) show the displacement (U2) values of the single CPE4 element and single IGA element at 100 % loading in y-direction respectively.

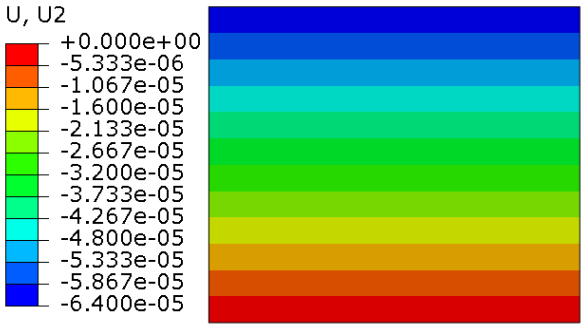

Figure 19: CPE4 Element:U2
Abaqus generated result

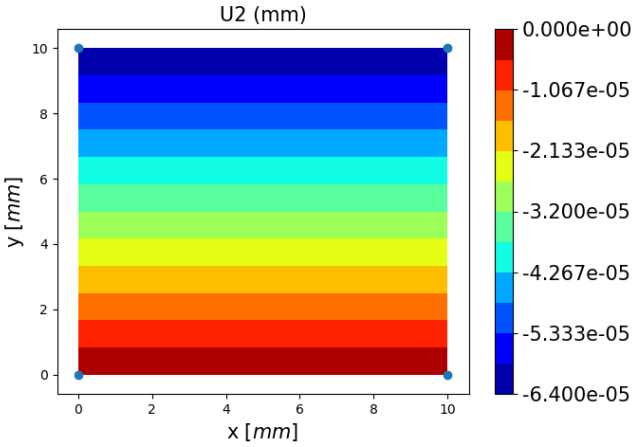

Figure 20: IGA Element:U2
Program generated result

Figure(21) and Figure(22) show the Reaction force values (RF1) of the single CPE4 element and single IGA element at 100 % loading in x-direction respectively.

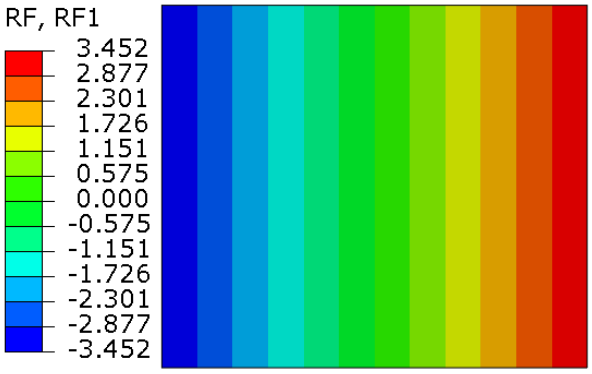

Figure 21: CPE4 Element:RF1
Abaqus generated result

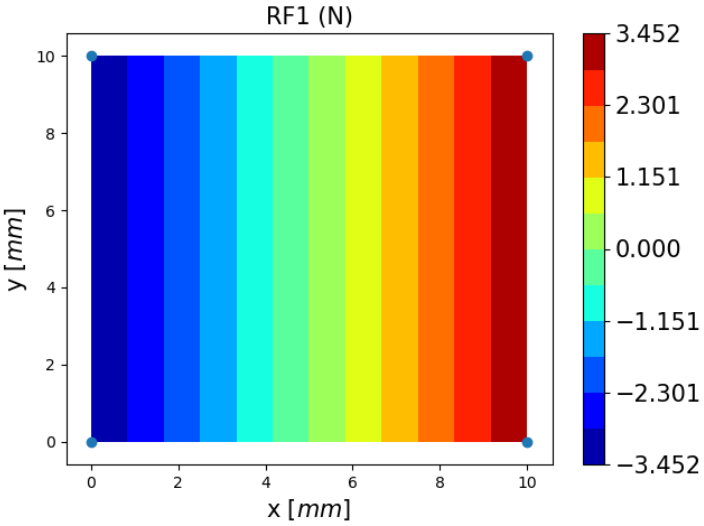

Figure 22: IGA Element:RF1
Program generated result

Reaction force in the y-direction is not reported since the loading is given only in the x-direction, and for the given fixed BCS, the RF2 is negligible.

7.1.4 Conclusion

As shown in figures above the values generated by IGA code are in-line with the results of the Abaqus element. So it can be concluded that IGA code written works well with 2D one element case.

7.2 2D Plate under pure Electrical loading

7.2.1 Problem description

A 2D plate is subjected to Electrical loading as shown in Figure(24).The material used is PZT-PIC151 ceramics, and the dielectric constants can be seen in Appendix (13.1) The movement of bottom edge AB is fixed in y-direction and left edge AC in x-direction.The top edge CD is grounded (potential = 0 volts). An electrical potential V of 100 Volts is applied on bottom edge AB as shown in Fig. (24)

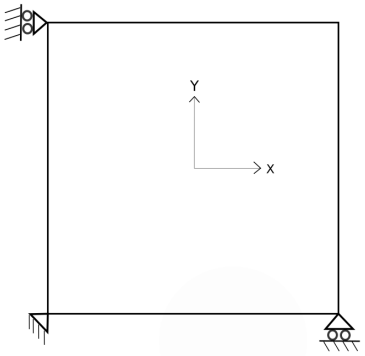

Figure 23: 2D Plate

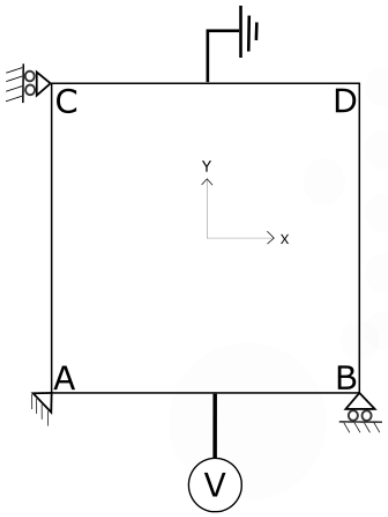

Figure 24: 2D Plate with pure electrical loading

7.2.2 Parametric details for the plate with single element

The parametric details for the geometry are the same as in section 7.1.2

7.2.3 Results and discussions

In this section the comparison is made between IGA code generated result and Abaqus plane strain full integration piezoelectric element (**CPE4E**) [1].
The below figures shows the values of Electrical potentials (EPOT) and reactive electrical nodal charge (RCHG) for both Abaqus and IGA element.

Electro-mechanical coupling is deactivated in this case by giving all the piezoelectric constants a value of "zero"
A similar contour is used for the program generated results and the Abaqus results for easy comparison.

Figure(25) and Figure(26) show the Electrical potential (EPOT) values of the single CPE4E element and single IGA element at 100 % loading respectively.

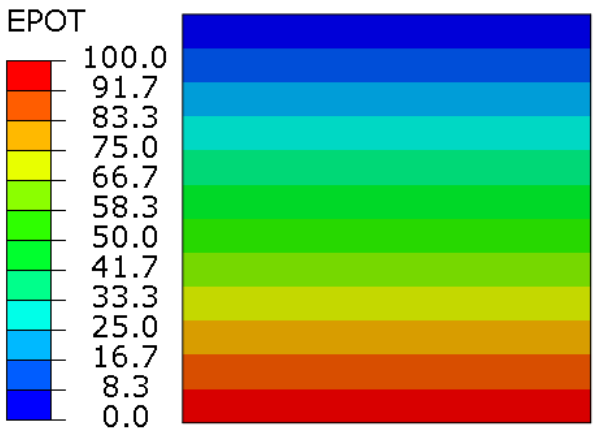

Figure 25: CPE4E Element:EPOT
Abaqus generated result

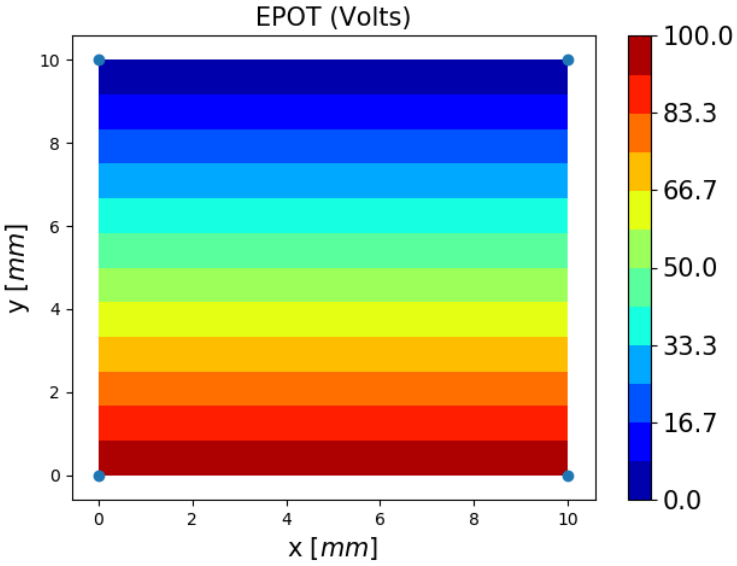

Figure 26: IGA Element:EPOT
Program generated result

Figure(27) and Figure(28) show the reactive nodal charge (RCHG) values of the single CPE4E element and single IGA element at 100 % loading respectively.

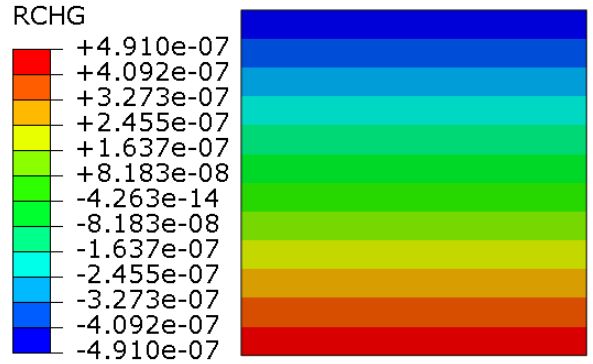

Figure 27: CPE4 Element:RCHG
Abaqus generated result

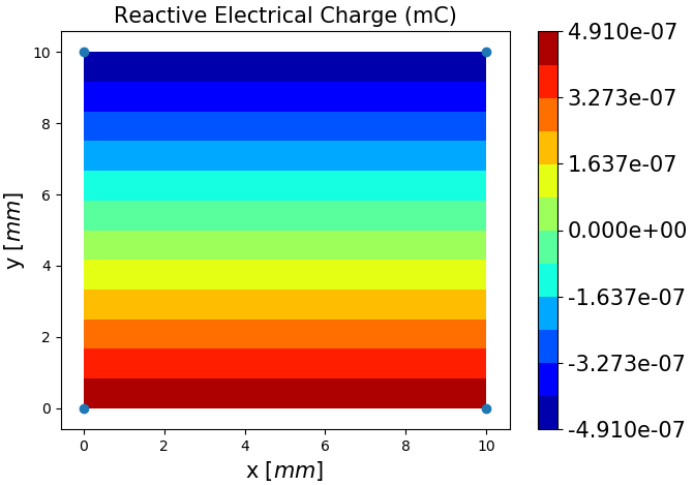

Figure 28: IGA Element:RCHG
Program generated result

7.2.4 Conclusion

As shown in the figures above, the values generated by the IGA code are in line with the results of the Abaqus element. So it can be concluded that for electrical analysis, IGA code written works well with 2D one element case.

7.3 2D Piezoelectric plate under mechanical loading

7.3.1 Problem description

A 2D piezoelectric plate subjected to mechanical displacements is considered, as shown in Fig. (30) . The material used is PZT-PIC151 ceramics, and the properties can be seen in the Appendix (13.1)

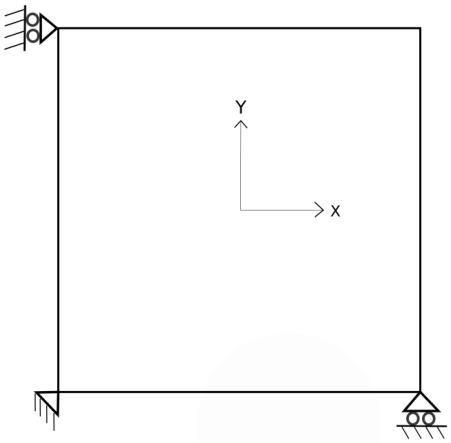

Figure 29: 2D Piezoelectric Plate

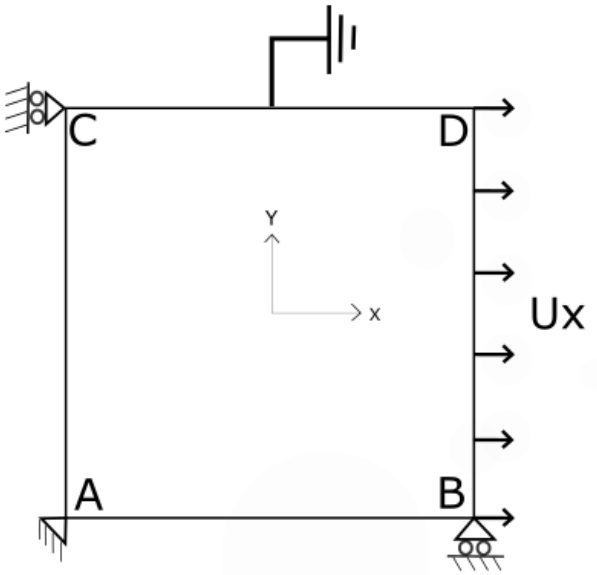

Figure 30: 2D Piezoelectric Plate with loading

The movement of the bottom edge AB and left edge AC of 2D piezoelectric plate is fixed in y-direction and x-direction respectively, as shown in figure(30). The top edge CD is grounded (Electric potential $\Phi = 0$), and a displacement load of 100 nm (1e-4 mm) is applied on the right edge BD. The results for a single element case and multiple elements are discussed in the below sections.
The results generated by IGA code is compared with inbuilt Abaqus piezoelectric element **CPE4E**.

7.3.2 Parametric details for the plate with single element

The parametric details for the geometry are the same as in section 7.1.2

7.3.3 Results and discussions

Abaqus plane strain full integration piezoelectric element (**CPE4E**) is used for analysis. The below figures show the values of displacements (U), electrical potentials (EPOT) and the reaction forces (RF) for both Abaqus and IGA element.

Figure(31) and Figure(32) show the displacement (U1) values of the single CPE4E element and single IGA element at 100 % loading in x-direction respectively.

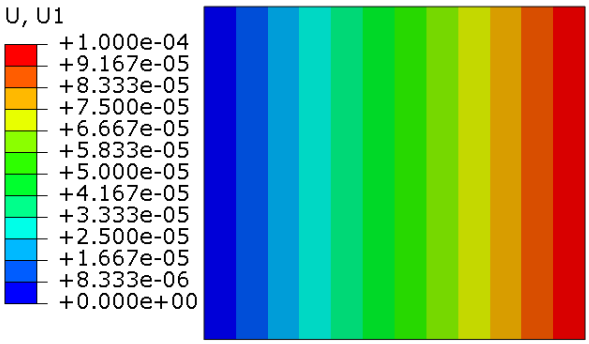

Figure 31: CPE4E Element:U1
Abaqus generated result

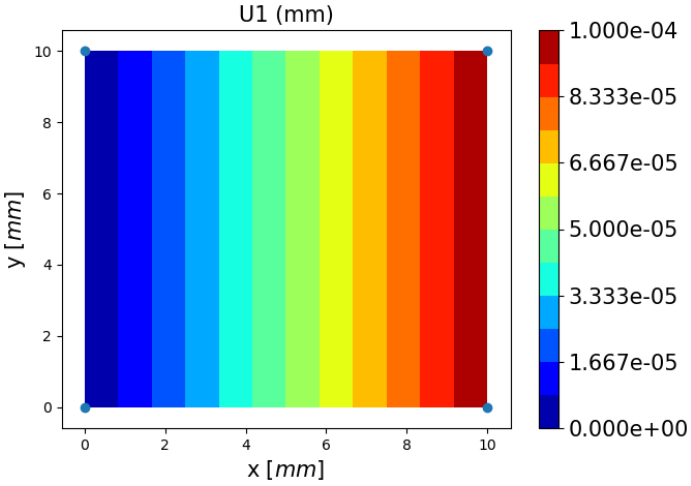

Figure 32: IGA Piezoelectric Element:U1
Program generated result

Figure(33) and Figure(34) show the displacement (U2) values of the single CPE4E element and single IGA element at 100 % loading in y-direction respectively.

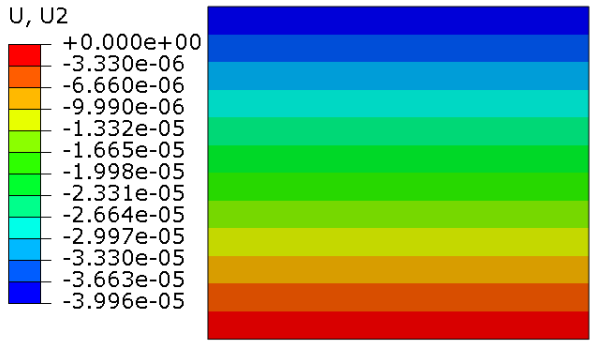

Figure 33: CPE4E Element:U2
Abaqus generated result

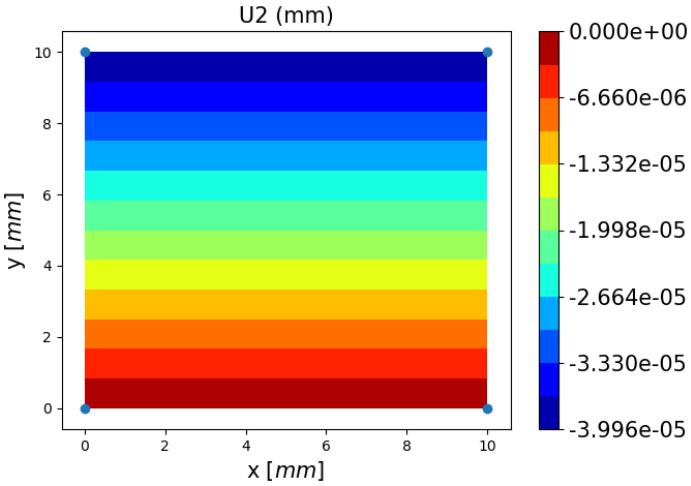

Figure 34: IGA Piezoelectric Element:U2
Program generated result

Figure(35) and Figure(36) show the Reaction force values (RF1) of the single CPE4E element and single IGA element at 100 % loading in x-direction respectively.

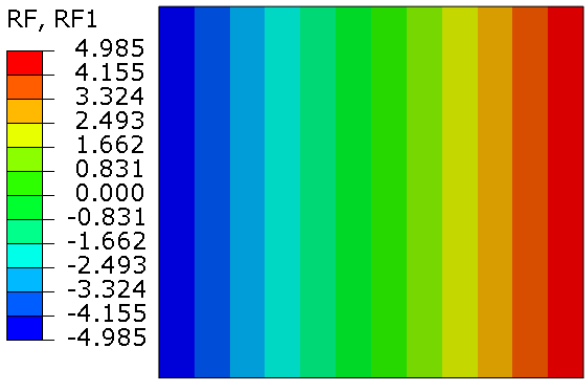

Figure 35: CPE4E Element:RF1
Abaqus generated result

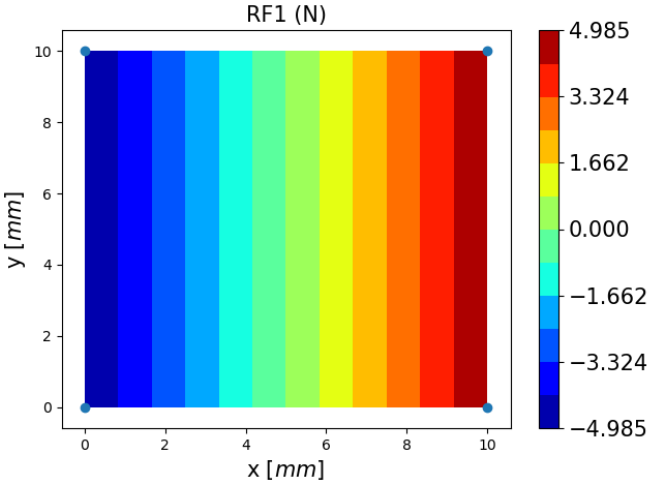

Figure 36: IGA Piezoelectric Element:RF1
Program generated result

Figure(37) and Figure(38) show the Electrical potential values (EPOT) of the single CPE4E element and single IGA element at 100 % loading respectively.

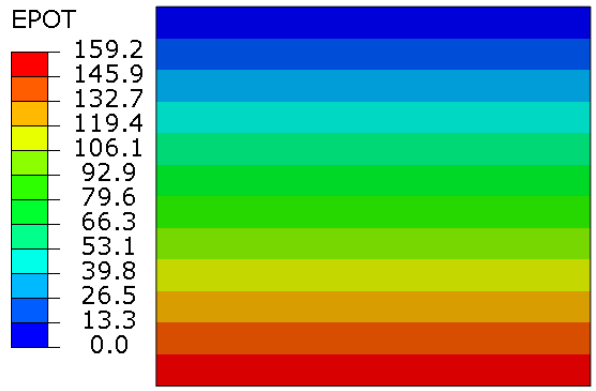

Figure 37: CPE4E Element:EPOT
Abaqus generated result

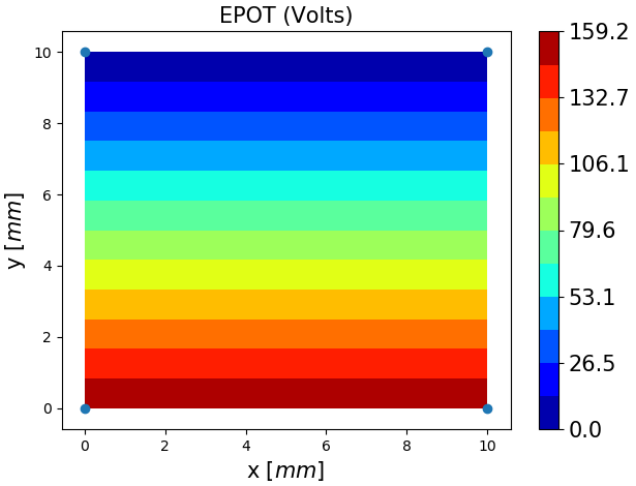

Figure 38: IGA Piezoelectric Element:EPOT
Program generated result

7.3.4 Conclusion

As shown in the figures above, the values generated by the IGA code for an electro-mechanical coupling with mechanical input are in line with the results of the Abaqus element. It can be concluded that IGA code written works well with a 2D one element piezoelectric case with displacement loading.

7.3.5 Comparison of electro-mechanical coupling with pure mechanical case

If we consider the problems in section (8.2) and section (7.3.1), both have same boundary conditions and are loaded in the same way with 100 nm displacement in the x-direction. But in the latter case electro-mechanical coupling is taken into consideration.

As seen in the Fig. (22) and Fig. (36) a difference in reaction force values can be observed due to the considered electro-mechanical coupling in the latter case. Similarly we can find the difference in U2 values as well due to the coupling (refer Fig.(20) and refer Fig. (34))

7.3.6 Parametric details for the plate with 2 elements in x-direction and 3 elements in y-direction

The 2nd order NURBS curve is used in both ξ and η directions.

1. Physical details for the geometry:
   L = 10 # Length of the plate in mm
   H = 10 # Height of the plate in mm
   T = 1 # Thickness of the plate in mm

2. Parametric details of the geometry:
   Ξ = [0,0,1,2,2] # Knot vector in xi direction
   H = [0,0,1,2,3,3] # Knot vector in eta direction

   Degree of the curve
   p=1 # Degree of the curve in ξ direction
   q=1 # Degree of the curve in η direction

   Number of control points in each direction
   n_{cp}^{ξ} = len(Ξ) - (p+1) #No.of control points in ξ direction (5-(1+1) = 3)
   n_{cp}^{η} = len(H) - (q+1) #No.of control points in η direction (6-(1+1) = 4)

3. Total number of control points for the geometry
   $n_{cp} = n_{cp}^{\xi} * n_{cp}^{\eta}$ = 3*4 = 12
   The control points are given by

i	$P_{i,0}$	$P_{i,1}$	$P_{i,2}$
0	(0, 0, 0, 1)	(0, 5, 0, 1)	(0, 10, 0, 1)
1	(3.33, 0, 0, 1)	(3.33, 5, 0, 1)	(3.33, 10, 0, 1)
2	(6.67, 0, 0, 1)	(6.67, 5, 0, 1)	(6.67, 10, 0, 1)
3	(10, 0, 0, 1)	(10, 5, 0, 1)	(10, 10, 0, 1)

   with fourth value in the parentheses being weights of respective control points. A control point assembly array and knot vector connectivity matrix are used to connect the elements.

7.3.7 Results and discussions

Abaqus plane strain full integration piezoelectric element (**CPE4E**) is used for analysis. For the comparison between Abaqus and IGA elements, 2 elements, along the x-direction and 3 elements along the y-direction, are used. A different number of elements are used along each direction in order to verify if the code generates proper results in unsymmetric conditions as well w.r.t number of elements in each direction.
The **blue points** on the program generated results are the final position of the control points at 100 % loading.
The below figures shows the values of displacements (U), electrical potentials (EPOT) and reaction forces (RF) for both Abaqus and IGA elements.

Figure(39) and Figure(40) show the displacement (U1) values of the CPE4E elements and IGA elements at 100 % loading in x-direction respectively.

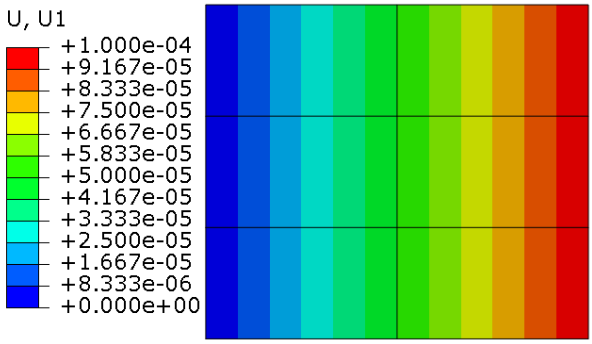

Figure 39: CPE4E Element:U1
Abaqus generated result

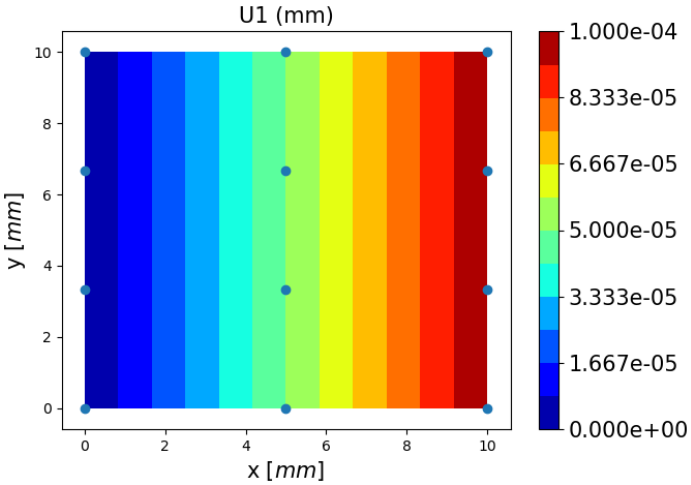

Figure 40: IGA Piezoelectric Element:U1
Program generated result

Figure(41) and Figure(42) show the displacement (U2) values of the CPE4E elements and the IGA elements at 100 % loading in the y-direction respectively.

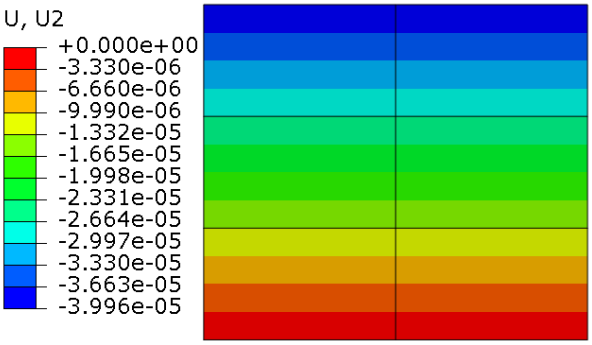

Figure 41: CPE4E Element:U2
Abaqus generated result

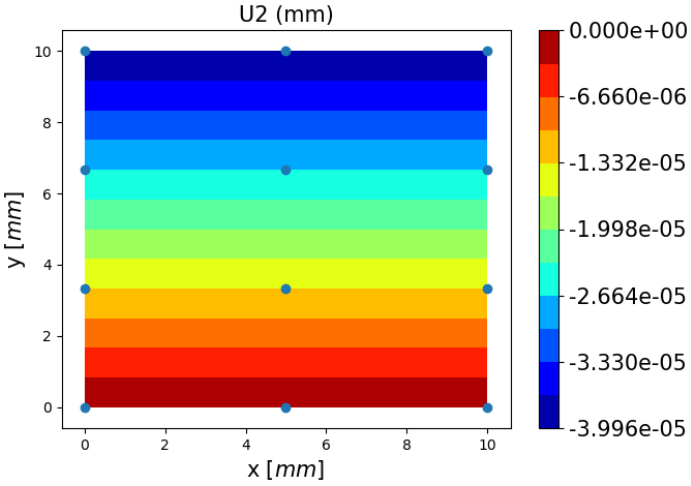

Figure 42: IGA Piezoelectric Element:U2
Program generated result

Figure(43) and Figure(44) show the Reaction force values (RF1) of the CPE4E elements and the IGA elements at 100 % loading in x-direction respectively.

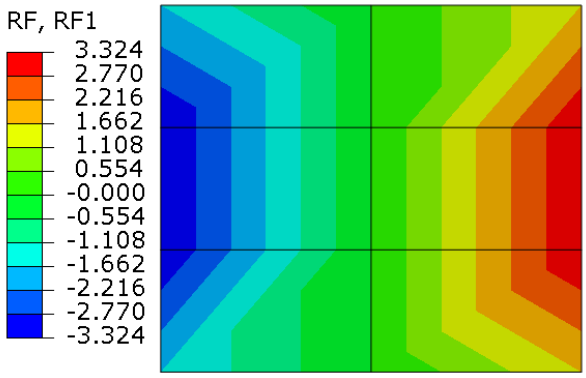

Figure 43: CPE4E Element:RF1
Abaqus generated result

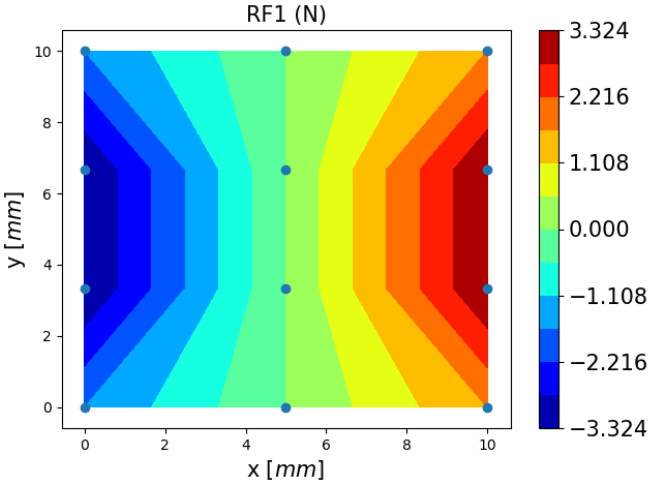

Figure 44: IGA Piezoelectric Element:RF1
Program generated result

Figure(45) and Figure(46) show the Electrical potential values (EPOT) of the CPE4E elements and the IGA elements at 100 % loading respectively.

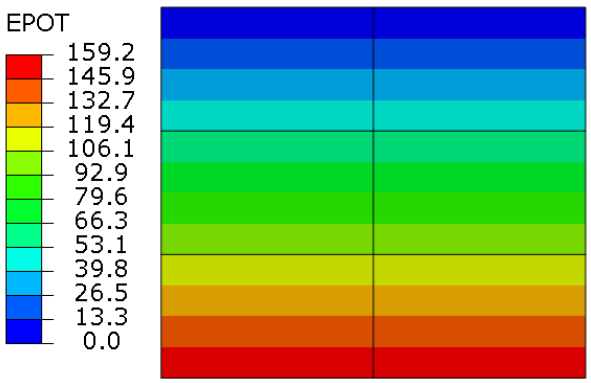

Figure 45: CPE4E Element:EPOT
Abaqus generated result

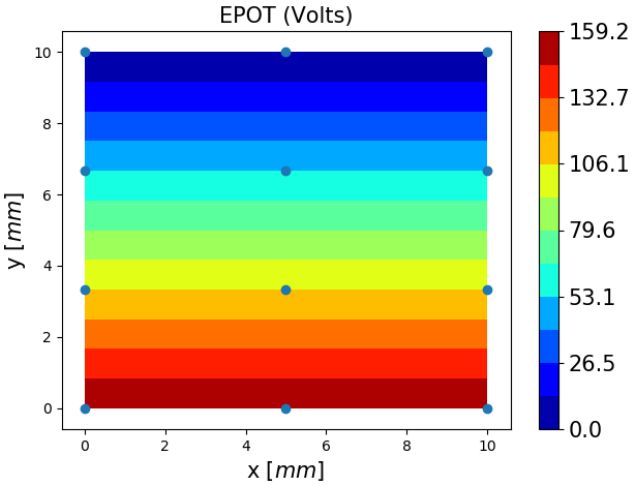

Figure 46: IGA Piezoelectric Element:EPOT
Program generated result

7.3.8 Conclusion

As shown in figures above for a 6 elements case, the results generated by IGA code are in line with the results of the Abaqus elements.

7.4 2D Piezoelectric plate under electrical loading

7.4.1 Problem description

A 2D piezoelectric plate subjected to electrical loading is considered, as shown in Fig. (48) . The material used is PZT-PIC151 ceramics, and the properties can be seen in the Appendix (13.1)

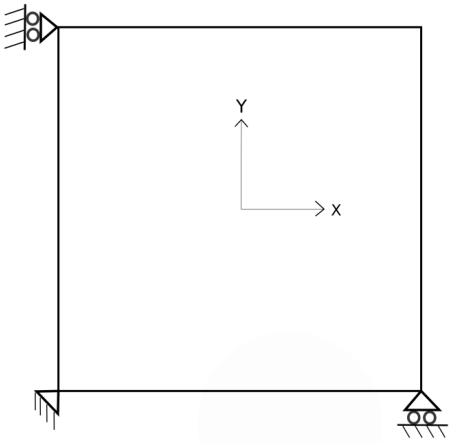

Figure 47: 2D Piezoelectric Plate

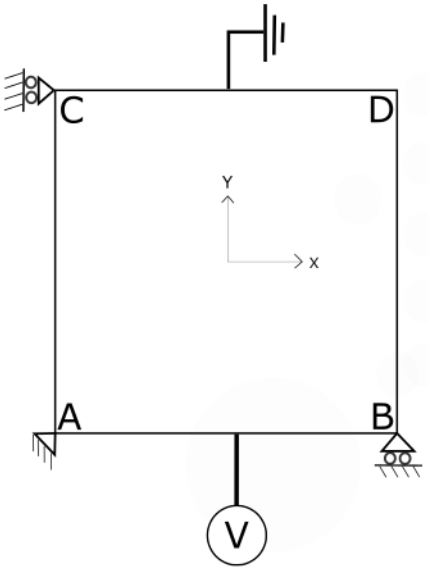

Figure 48: 2D Piezoelectric Plate with loading

The movement of the bottom edge AB and left edge AC of 2D piezoelectric plate is fixed in y-direction and x-direction respectively, as shown in figure(48). The top edge CD is grounded (Electric potential $\Phi = 0$), and an electrical potential of 100 V is applied on the bottom edge AB. The results for a single element case and multiple elements are discussed in the below sections.
The results generated by IGA code is compared with inbuilt Abaqus piezoelectric element **CPE4E**.

7.4.2 Parametric details for the plate with single element

The parametric details for the geometry are the same as in section 7.1.2

7.4.3 Results and discussions

Abaqus plane strain full integration piezoelectric element (**CPE4E**) is used for analysis. The below figures show the values of displacements (U), electrical potentials (EPOT) and the reactive electrical nodal charge (RCHG) for both Abaqus and IGA element.

Figure(49) and Figure(50) show the displacement (U1) values of the single CPE4E element and single IGA element at 100 % loading in x-direction respectively.

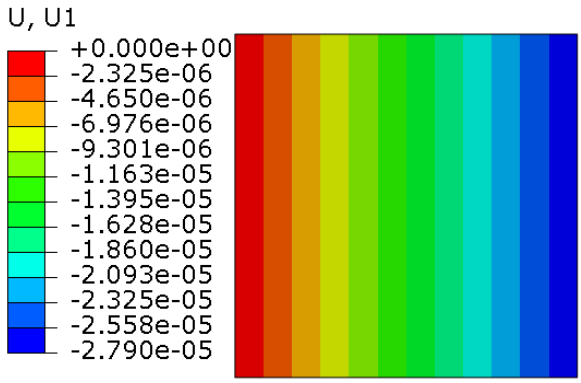

Figure 49: CPE4E Element:U1
Abaqus generated result

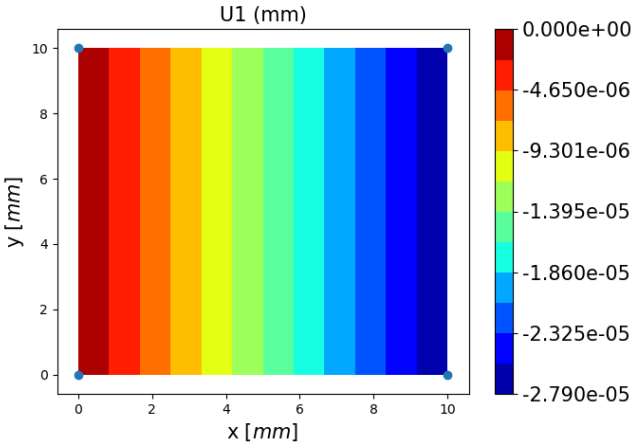

Figure 50: IGA Piezoelectric Element:U1
Program generated result

Figure(51) and Figure(52) show the displacement (U2) values of the single CPE4E element and single IGA element at 100 % loading in y-direction respectively.

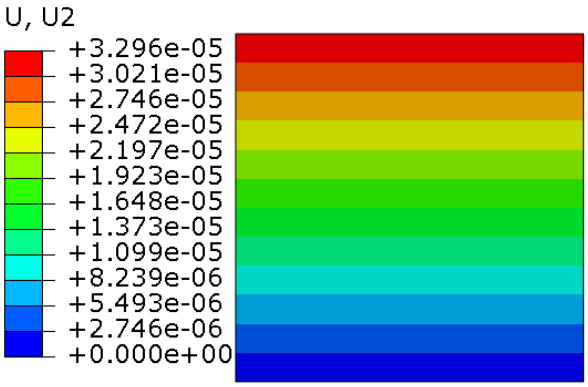

Figure 51: CPE4E Element:U2
Abaqus generated result

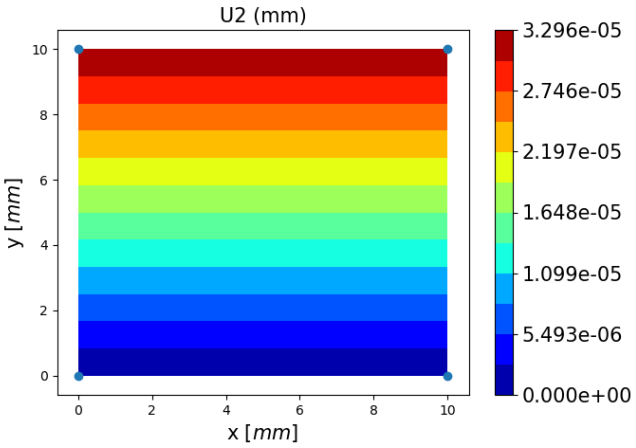

Figure 52: IGA Piezoelectric Element:U2
Program generated result

Figure(53) and Figure(54) show the Electrical potential values (EPOT) of the single CPE4E element and single IGA element at 100 % loading respectively.

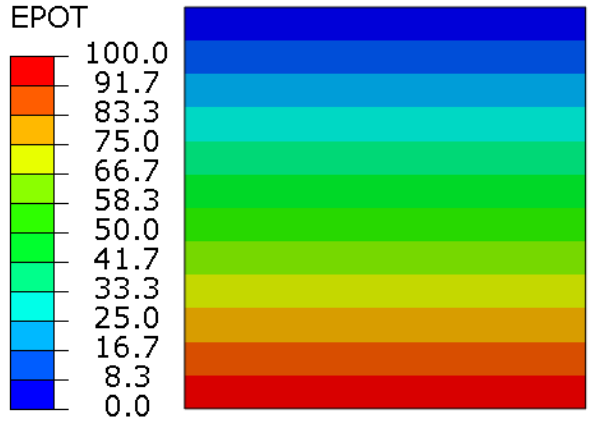

Figure 53: CPE4E Element:EPOT
Abaqus generated result

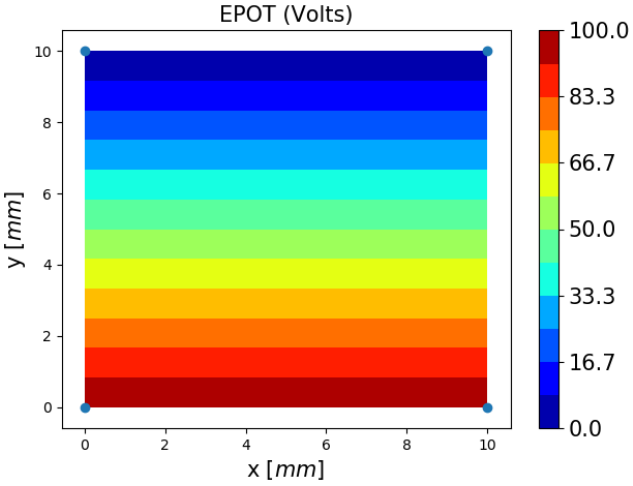

Figure 54: IGA Piezoelectric Element:EPOT
Program generated result

Figure(55) and Figure(56) show the Reactive electrical nodal charge (RCHG) of the single CPE4E element and single IGA element at 100 % loading respectively.

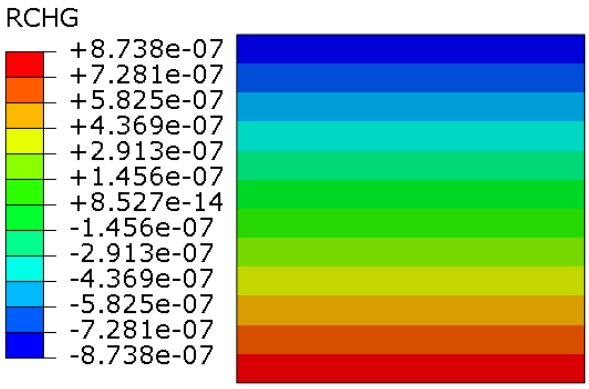

Figure 55: CPE4E Element:RCHG
Abaqus generated result

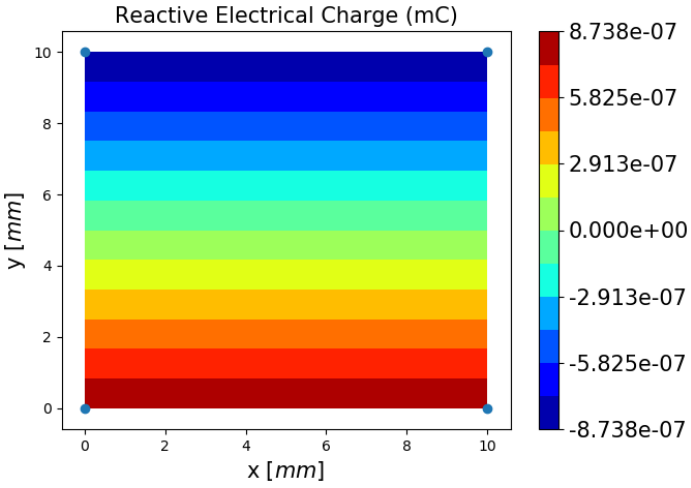

Figure 56: IGA Piezoelectric Element:RCHG
Program generated result

7.4.4 Conclusion

As shown in the figures above, the values generated by the IGA code for an electro-mechanical coupling with electrical input are in line with the results of the Abaqus element. It can be concluded that IGA code written works well with a 2D one element piezoelectric case with a given voltage difference on the surface.

7.4.5 Comparison of electro-mechanical coupling with pure electrical case

Considering the problems in section (7.2.1) and section (7.4.1), the 2D plate is grounded on the top edge and applied a potential of 100 V on the bottom edge. But in the latter case electro-mechanical coupling is taken into consideration. As seen in the Fig. (28) and Fig. (56)

a difference in RCHG values can be observed due to the considered electro-mechanical coupling in the latter case.

7.4.6 Parametric details for the plate with 2 elements in x-direction and 3 elements in y-direction

The parametric details for the geometry are the same as in section 7.3.6

7.4.7 Results and discussions

Abaqus plane strain full integration piezoelectric element (**CPE4E**) is used for analysis. For the comparison between Abaqus and IGA elements, 2 elements, along the x-direction and 3 elements along the y-direction, are used.
The below figures shows the values of displacements (U), electrical potentials (EPOT) and reactive electrical nodal charge (RCHG) for both Abaqus and IGA elements.
Figure(57) and Figure(58) show the displacement (U1) values of the CPE4E elements and IGA elements at 100 % loading in x-direction respectively.

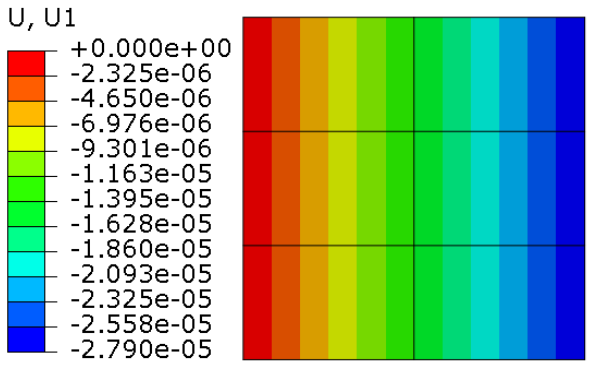

Figure 57: CPE4E Element:U1
Abaqus generated result

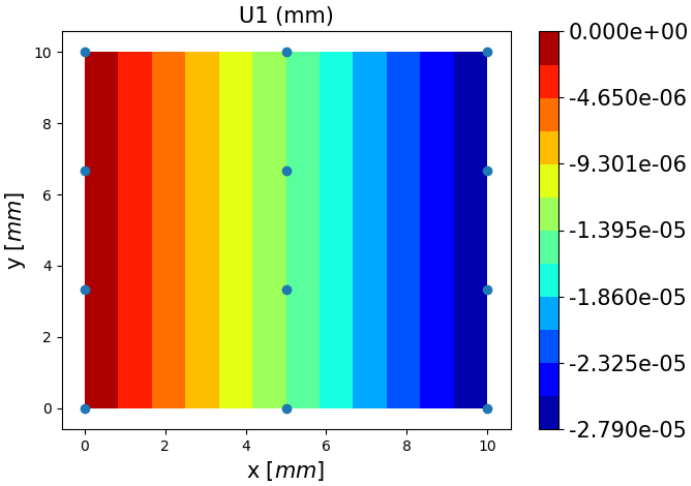

Figure 58: IGA Piezoelectric Element:U1
Program generated result

Figure(59) and Figure(60) show the displacement (U2) values of the CPE4E elements and the IGA elements at 100 % loading in the y-direction respectively.

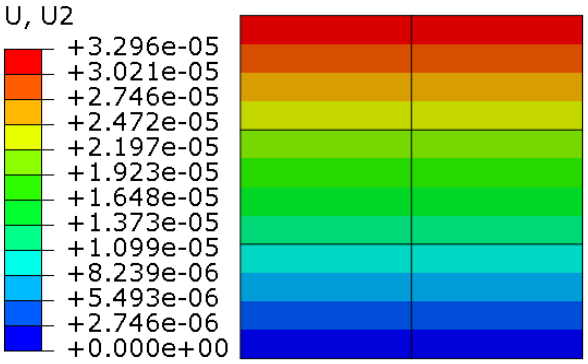

Figure 59: CPE4E Element:U2
Abaqus generated result

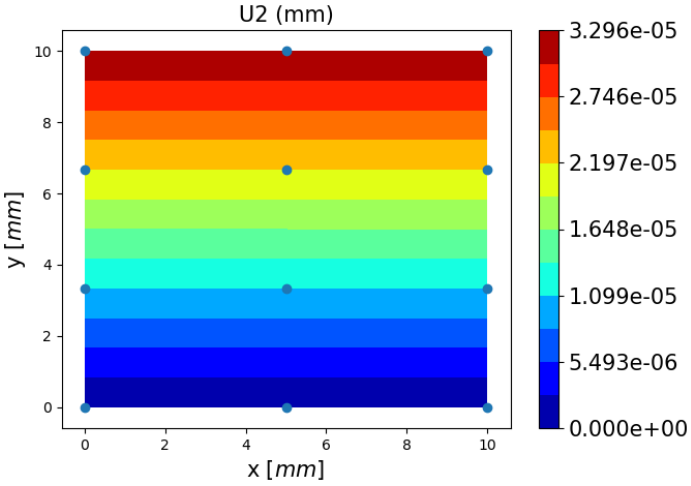

Figure 60: IGA Piezoelectric Element:U2
Program generated result

Figure(61) and Figure(62) show the Electrical potential values (EPOT) of the CPE4E elements and the IGA elements at 100 % loading respectively.

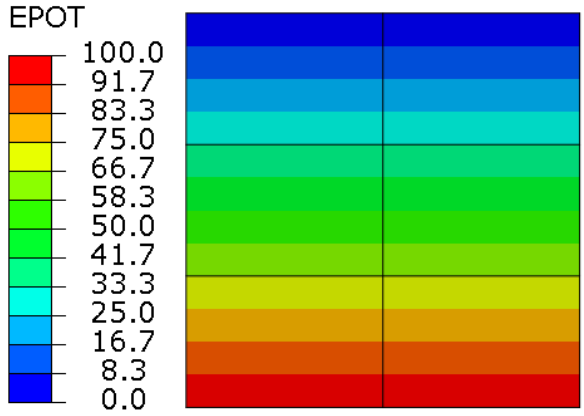

Figure 61: CPE4E Element:EPOT
Abaqus generated result

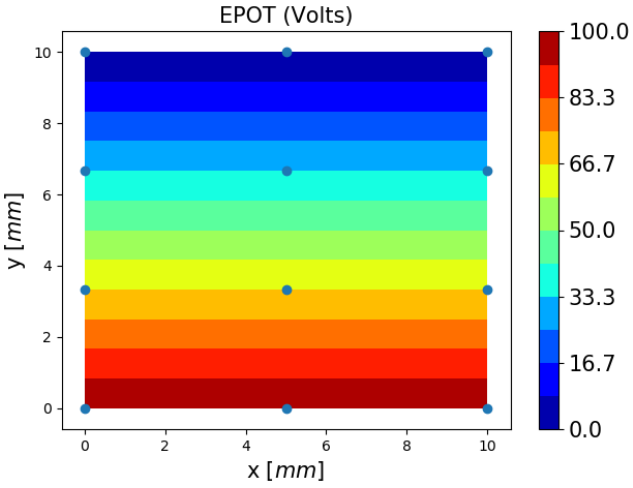

Figure 62: IGA Piezoelectric Element:EPOT
Program generated result

Figure(63) and Figure(64) show the Reactive electrical nodal charge (RCHG) of the CPE4E elements and IGA elements at 100 % loading respectively.

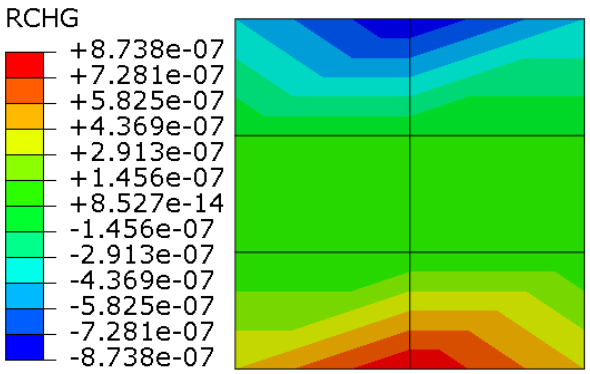

Figure 63: CPE4E Element:RCHG
Abaqus generated result

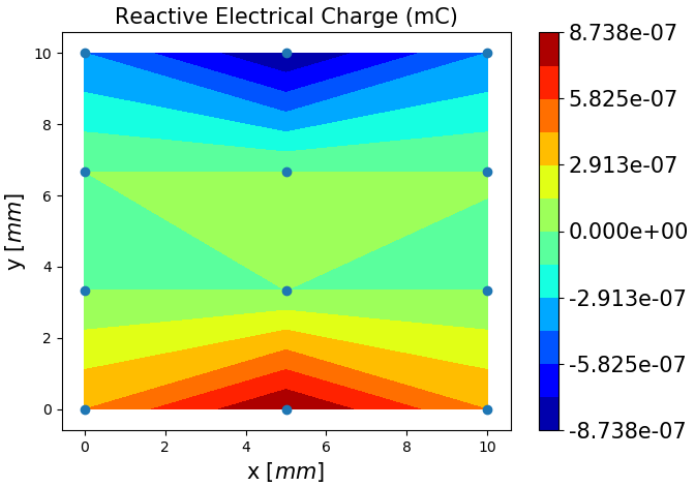

Figure 64: IGA Piezoelectric Element:RCHG
Program generated result

7.4.8 Conclusion

As shown in figures above for a 6 elements case, the results generated by IGA code are in line with the results of the Abaqus elements.

7.5 2D Piezoelectric plate under mechanical loading with higher-order NURBS basis functions

7.5.1 Problem description

The problem statement is same as in section (7.3.1) (2D Piezoelectric plate under mechanical loading).

7.5.2 Order of NURBS basis functions

In the present load case, a 3rd order basis functions in ξ direction and 4th order basis functions in η direction are considered.

7.5.3 Results and discussions

The displacement (U) values and potential (EPOT) values generated with higher-order NURBS basis functions with more number of control points are compared with the results from section (7.3.7) (Results with 6 elements). More control points are used for results only to differentiate between 2nd and higher-order basis functions.
Figure(65) and Figure(66) show the displacement (U1) values of the 2nd order NURBS surface and higher-order NURBS surface at 100 % loading in x-direction respectively.

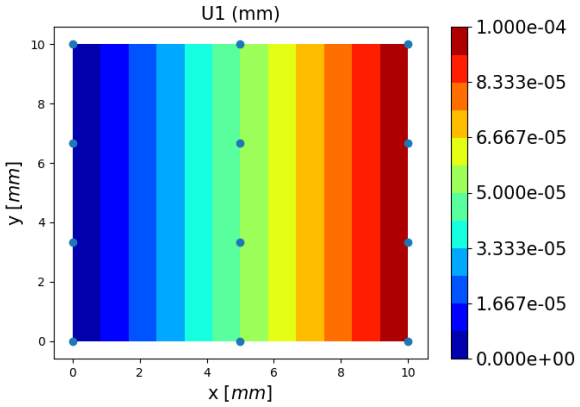

Figure 65: IGA Piezoelectric Element:U1
2nd order NURBS surface

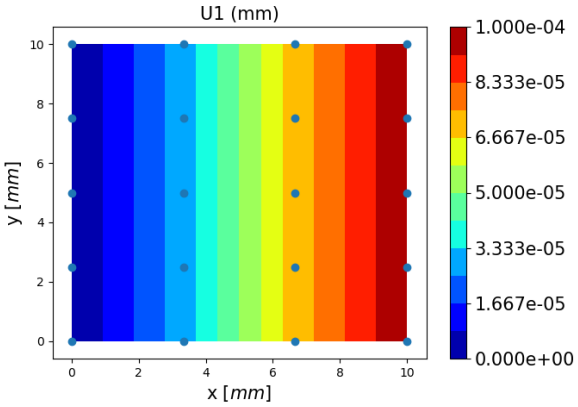

Figure 66: IGA Piezoelectric Element:U1
Higher-order NURBS surface

Figure(67) and Figure(68) show the displacement (U2) values of the 2nd order NURBS surface and higher-order NURBS surface at 100 % loading in the y-direction respectively.

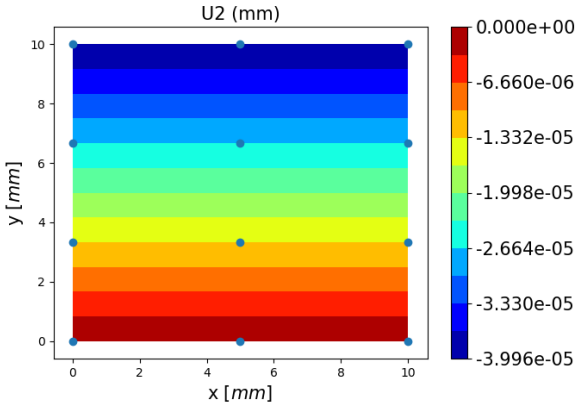

Figure 67: IGA Piezoelectric Element:U2
2nd order NURBS surface

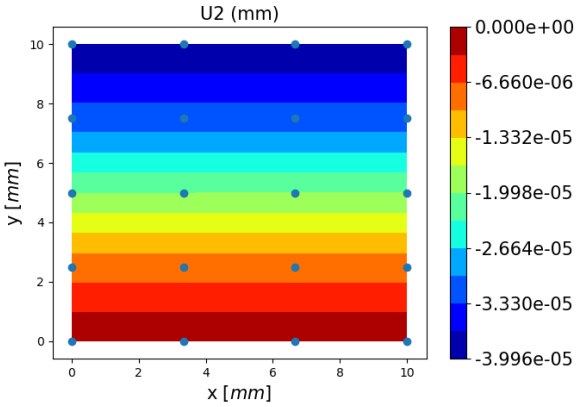

Figure 68: IGA Piezoelectric Element:U2
Higher-order NURBS surface

Figure(69) and Figure(70) show the Electrical potential (EPOT) values of the 2nd order NURBS surface and higher-order NURBS surface at 100 % loading respectively.

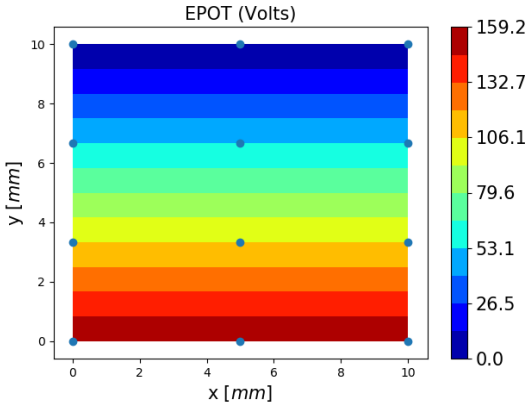

Figure 69: IGA Piezoelectric Element:EPOT **2nd order NURBS surface**

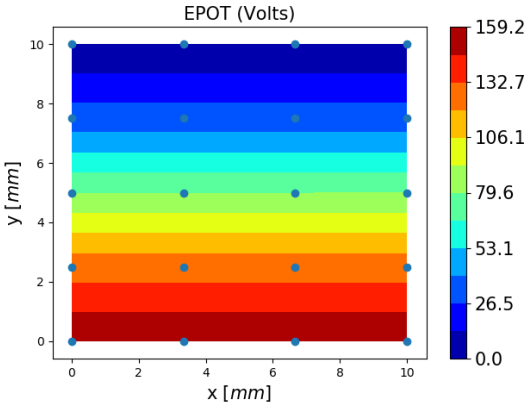

Figure 70: IGA Piezoelectric Element:EPOT **Higher-order NURBS surface**

7.5.4 Conclusion

Since the surface is captured exactly in both the cases (2nd order and higher-order NURBS surface), we can expect the same results for DOF values on control points. The written code gives correct results for higher-order NURBS basis functions provided the control points are on the surface.

8 Bottleneck in the code

8.1 Aim

The test case is for checking the computation time taken by each function in the code to find the possible bottleneck of the written Isogeometric analysis code.

8.2 Problem description

A 2D plate is subjected to mechanical loading as shown in Figure(72). The material used is PZT-PIC151 ceramics. The movement of bottom edge AB is fixed in y-direction and left edge AC in x-direction. A displacement of 100 nm (1e-4 mm) is given on the right edge BD and 200nm (2e-4 mm) on top edge CD. **The analysis is done for 50 control points in each direction and 2nd order NURBS basis functions, to get a clear picture of the analysis time for each function.**

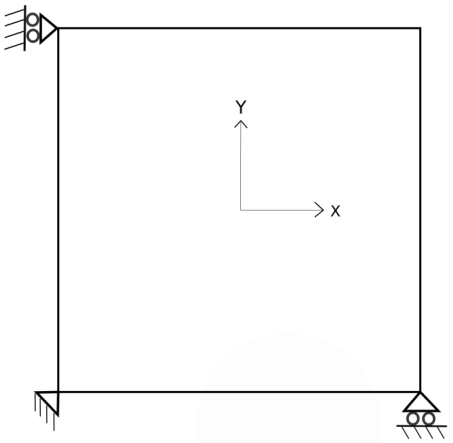

Figure 71: 2D Plate

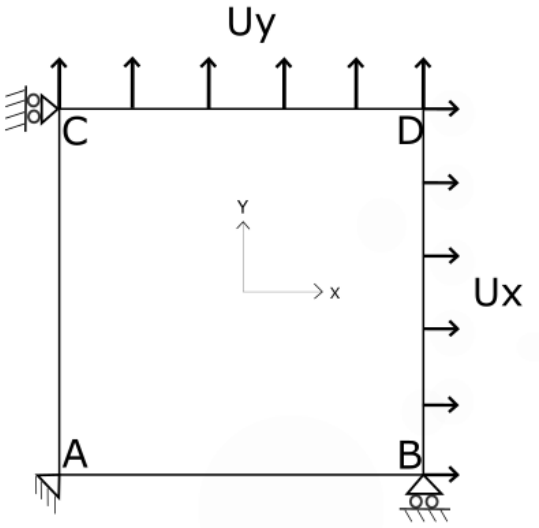

Figure 72: 2D Piezoelectric Plate with loading

timeit package in python is used to compute time taken by a function. A sample code is shown below.

```
#-----------------------------NURBS_Surface_Point-------------------------------#
print('"NURBS_Surface_Point" Function is called for',NURBS_Surface_Point_Count,'times.')
SetUpCode = '''from __main__ import NURBS_Surface_Point
import numpy as np'''
TestCode = '''NURBS_Surface_Point(1, 1, [0., 0., 1., 1.], 1, 1, [0., 0., 1., 1.],
 [[[ 0.    ,  0.    ,  0.    ,  1.    ],[ 0.    , 10.0002,  0.    ,  1.    ]],
 [[10.0001,  0.    ,  0.    ,  1.    ],[10.0001, 10.0002,  0.    ,  1.    ]]], 0.0, 0.0)'''
print('Time taken for single time function execution: ',timeit.timeit(setup = SetUpCode,
                stmt = TestCode,
                number = 1))
print('Total time taken by the function in complete analysis: ',timeit.timeit(setup = SetUpCode,
                stmt = TestCode,
                number = NURBS_Surface_Point_Count))
```

Figure 73:
Sample code calculating the computation time of a function

8.3 Findings and Conclusion

Below is the data generated by using timeit. The code is written to generate the data regarding number of times each function is called in the program and total time taken by each function in the analysis.
From the below values we can see that the time taken by "SurfaceDerivsAlgAuv" Function and "B_matrix" Function is more when compared to other functions apart from element routine.
"SurfaceDerivsAlgAuv" Function is used to compute derivatives of the NURBS bi-variate basis functions.
Derivatives of basis functions are handled differently in IGA compared to FEM because of the recursive nature of the NURBS basis functions. Optimizing "SurfaceDerivsAlgAuv" function is necessary to reduce overall computation time.
Program generated data for 50 control points in each direction

```
"KnotVector" Fucntion is called for 2 times.
Time taken for single time function execution:  1.5964999988682393e-05
Total time taken by the function in complete analysis:  3.021800000624353e-05

"FindSpan" Fucntion is called for 120248 times.
Time taken for single time function execution:  1.0263000007171286e-05
Total time taken by the function in complete analysis:  0.44270469899998943

"BasisFuns" Fucntion is called for 158664 times.
Time taken for single time function execution:  3.306900001120994e-05
Total time taken by the function in complete analysis:  2.1537873399999796

"DersBasisFuns" Fucntion is called for 230496 times.
Time taken for single time function execution:  5.302499999970678e-05
Total time taken by the function in complete analysis:  10.28354588900001

"ControlPointAssembly" Fucntion is called for 4802 times.
Time taken for single time function execution:  0.00010034799998948074
Total time taken by the function in complete analysis:  0.25946407300000374

"materialRoutine" Fucntion is called for 19208 times.
Time taken for single time function execution:  7.469100000889739e-05
Total time taken by the function in complete analysis:  0.7070453440000222

"elementRoutine" Fucntion is called for 4802 times.
Time taken for single time function execution:  0.0065517000000170356
Total time taken by the function in complete analysis:  32.14903563299998

"Jacobian12" Fucntion is called for 38424 times.
Time taken for single time function execution:  0.00047551299999781804
Total time taken by the function in complete analysis:  17.596252571999997

"B_matrix" Fucntion is called for 38424 times.
Time taken for single time function execution:  0.0010496630000034202
Total time taken by the function in complete analysis:  37.19408733499998

"NURBS_Surface_Point" Function is called for 2500 times.
Time taken for single time function execution:  6.0437000001911656e-05
```

```
Total time taken by the function in complete analysis:  0.11756569399994987

"SurfaceDerivsAlgAuv" Function is called for 153700 time.
Time taken for single time function execution:  0.00040823400001954724
Total time taken by the function in complete analysis:  64.39044447399999

"GaussPoints" Function is called for 1 time.
Time taken for single time function execution:  4.390300000522984e-05
Total time taken by the function in complete analysis:  4.903399997147062e-05
```

9 Milestones achieved

The following table describes the proposed coding activities and achieved activities.

Proposed Activities	Achieved
NURBS based 2D geometry generation	yes
Implementation of the Isogeometric Analysis for a single element 2D case	yes
Coupling between mechanical and electrical Dof's	yes
Verification of results with abaqus inbuilt piezoelectric element for mechanical input and electro-mecanical output	yes
Verification of results with abaqus inbuilt piezoelectric element for electrical input and electro-mecanical output	yes
Extra Activities	**Achieved**
Code extension to do analysis with multiple elements	yes
Implementation of Knot and Control point assembly arrays	yes

10 Intricacies of Isogeometric analysis

To develop a code for Isogeometric analysis, it demands the understanding of NURBS theory, apart from the knowledge in the field of FEM. Although the NURBS book gives a clear picture of how to generate a curve and a surface, it is not designed to give a clear insight on how to work with Isogeometric analysis. For example, we can find the derivatives of the NURBS basis functions only but not the derivatives of bivariate basis functions (which are used in NURBS surface). Since basis functions are of recursive nature, it is no more straight forward to derivate the functions and requires a deep insight into the theory that goes with it. One can find much material regarding IGA analysis online, but when it comes to derivatives of bivariate basis functions, there is not much to refer. It took time and effort to get the right combination of material to understand the way to derivate, and the next major task is to write code. As the NURBS basis functions are defined in parametric space, as discussed in section(4.2), it is not straight forward to map physical space on to master space like in FEM. The addition of parametric space makes things a little complicated because the derivatives have to be defined in three different spaces. An introduction paper on IGA by Vishal et al. [2] , gave a clear picture of these mappings. Unlike FEM, which contains a fixed number of nodes, this is not the case with IGA, where the number of control points (equivalent term to nodes in FEM) in each element depends on the parametric details of the NURBS curve/surface. A generalized way has to be adopted to formulate matrices involved in the analysis (like B matrix, Stiffness matrix, DOF array and Global force array) so that it does not require to reformulate the code every time the parametric details change (which is the primary purpose of IGA to reduce the time involved in meshing the geometry as in FEM). Moreover, extending a code from a single element to multiple elements requires two types of connecting arrays namely knot and control points connectivity arrays as discussed in section(4.1.2). So the code has to handle additional connectivity. Lastly defining boundary conditions need special techniques, unlike FEM.

11 How to run the Program

Every test case is provided with a **Readme** (Contains Aim, Expected and Obtained Results and instructions to run the program)in order to facilitate the usage of code.

1. The code is written in python and external libraries numpy, matplotlib.pyplot, sys, path from pathlib,timeit and math are used.

2. Please use any environment which will compile python programs

3. Place all the files in a single folder.

4. A file named Input.py can be edited to change the dimensions of the plate. User can change Length, Height and Thickness of the plate.
   (The results discussed are for Length = 10 mm, Height = 10 mm and Thickness = 1 mm)

```
#---------------------------------Dimensions of 2D Plate-------------------------------------#
Thick   = 1.0  #Thickness of the plate in mm
Length  = 10.0 #Length of the plate    in mm
Height  = 10.0 #Height of the plate    in mm
```

5. Before you run the file, please make sure that the working directory is same as the folder which consists the Program.

6. Use command >>> python Main_Program.py
   to run the program.

7. The contour plots will be saved in the folder **Results.**

8. A "log.txt" file is created in the same folder which contains the values of the results plotted.

12 Conclusion

In this project, an Isogeometric analysis code is developed to create NURBS surface using parametric details and use the same basis functions to analyze Linear elastic mechanical loading and later extended to electro-mechanical coupling. The code is written in such a way that, it can analyze a 2D geometry with up to 4th order NURBS basis functions (the order of the basis functions in each direction can vary, for example, a surface can be generated using 3rd order basis functions in ξ direction and 2nd order basis functions in η direction). The drawback of the written code is, it gives accurate results for a 2nd order NURBS basis functions but fails to produce correct results for higher-order basis functions. This is because unlike FEM, where boundary conditions can be applied on nodes, the same procedure cannot be followed for IGA. The control points may not lie on the surface for higher-order NURBS surface, and points defining the boundary needs to be identified. The results of the IGA code are verified with the Abaqus elements for 2nd order NURBS curve.

12.1 Continuation Strategy

The existing code can be extended by adding boundary condition defining techniques like least square minimization to identify the boundary controlling points and assigning appropriate boundary conditions. The code can be further extended to 3D geometry, but coding assembly arrays (knot connectivity and control points assembly arrays) would be challenging.

13 Appendices

13.1 Material Properties

The material used is PZT-PIC151 ceramics and the properties [4] are as follow. The elastic constants (C) are transversely isotropic, dielectric constants (κ) are anisotropic and material is polarized in Y-direction.

$$C = \begin{bmatrix} 110,000 & 64,000 & 63,000 & 0 & 0 & 0 \\ 64,000 & 100,000 & 64,000 & 0 & 0 & 0 \\ 63,000 & 64,000 & 110,000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20,000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20,000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 23,500 \end{bmatrix} MPa$$

$$e = \begin{bmatrix} 0 & -9.6e-3 & 0 \\ 0 & 15.1e-3 & 0 \\ 0 & -9.6e-3 & 0 \\ 12.0e-6 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 12.0e-6 \end{bmatrix} N/Vmm$$

$$\kappa = \begin{bmatrix} 7.54e-9 & 0 & 0 \\ 0 & 9.82e-9 & 0 \\ 0 & 0 & 7.54e-9 \end{bmatrix} N/V^2$$

where
e are Piezoelectric constants.

14 Git log

```
commit 7121cda6949a97187e992facf2540a035ad4dccf
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Apr 5 02:11:40 2020 +0200
Complete working code. Changed the plotting font size and plotted on
↪  scientific scale for some values

commit d0d2647103625a60685698e9696438fa85603604
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Apr 3 02:39:47 2020 +0200
Backup for an working code without error

commit 6f8362ee92d2313c1ebe816b37f669d79b6ded50
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Mar 27 16:44:40 2020 +0100
Complete report with all the results and implementation details

commit 802790e42b9c2532b9d9e65ab59f88d61dc7f3e1
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Mar 27 16:43:14 2020 +0100
Complete Working code for all mechanical, electrical and electro-mechanical
↪  coupling cases

commit a5af9873d7432577ffcc5c9bcbae0ec1f809f641
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Mar 21 21:32:28 2020 +0100
Git Backup before furthur modifications 21st march

commit 00227969b8ce5c4bb44c125a56af9bb277e8de09
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Feb 26 20:11:24 2020 +0100
Code with multiple elements and degree of NURBS Curve

commit 53589f56f1b2eff03db18dcaea7da3d3d50ba749
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Feb 17 13:03:16 2020 +0100
Added results section. In progress

commit ce3cb3b705bbc8b3902768fbbefa93d0f9b950bf
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Feb 17 02:34:50 2020 +0100
Changed geometry file. Code will work for any degree in xi and eta directons.
↪  Have to generalize gauss points

commit 6f2201614f6af8d670418065d05ee61848849d56
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Tue Feb 11 22:08:18 2020 +0100
Code has been extended to NURBS Degree of any order

commit 0f9f04d8e833f96637365080462045d1cfb7440f
```

```
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Feb 8 14:25:12 2020 +0100
Connectivity for more than one element is coded successfully

commit 43acf544de73572e872035b1e9357bb0b7530639
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Feb 3 19:44:31 2020 +0100
Results checked for multiple elements and are correct, Have to check the
↪  results in detail

commit b658ddb716e2b860f710d11c19f8dada97f05fd1
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Feb 2 02:22:58 2020 +0100
Started documenting Problem description

commit 808b5058b947a789645bb7828780492f51642718
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Jan 29 01:13:58 2020 +0100
Derivatives of NURBS bivariate is added to documentation

commit 4fb01b8f7f5cdc96747739ebbdfe81add29c2d12
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Jan 29 00:33:48 2020 +0100
Connectivity for coupling case in progress

commit 7b2f315d2958297cc350ed4228d41af59887ee48
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Jan 27 23:07:24 2020 +0100
Code has been extended for connectivity. Have to debug errors and check the
↪  results

commit f35130a0d00322a757295bbb48a0bab21a6e162c
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Jan 27 01:44:24 2020 +0100
Assignment matrix under progress

commit caa45cb94742e197dec1914a90952481c97cd748
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Jan 25 20:05:31 2020 +0100
Coupling code with output of stress and strain values at gauss points

commit ad32a5346ae3d7f96af88bdbc7c1086c2b847111
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Jan 24 01:29:20 2020 +0100
Updated till post processing of deformed geometry

commit 8d7767407fb6db54b049d863104c5653553a199d
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Thu Jan 23 02:45:42 2020 +0100
Documetation under progress. Written till preprocessign stage of IGA
```

```
commit 6c47769d0a0159cec9c161bc932fe86ce24d95f7
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Thu Jan 23 01:06:06 2020 +0100
Documentation under progress. Documentation as on 22-01-2020

commit aeab0484acff295754ef3b8bd1138b3903de26a3
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Jan 18 17:05:31 2020 +0100
Files have been changed to Displacement driven Analysis, Have to cross check
↪  results with abaqus results

commit c52821a2f05e54fb3a5e7694a5a8fd87513a98d7
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Jan 15 10:03:57 2020 +0100
Files before changing it to displacement driven Analysis

commit cca83c20b0645fe7388235ac2fe61460aa1038e8
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Jan 13 21:09:16 2020 +0100
Changed boundary conditions according to the test case available in UEL

commit 715ad610289b0eb6cc84e123f42b2f797c48e186
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Jan 10 22:50:27 2020 +0100
Polarization values generated are not correct have to check the signs of Be
↪  and K_EE Matrix

commit 281baa2cef8a5a1030ae0e15d94a118ebec5ebe0
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Jan 10 21:37:48 2020 +0100
Extended all the programs to electrical degrees of freedom. Have to debug the
↪  errors for convergence and for the correctness of the results

commit 5c865b1766e067812d7e888b7860319d4f55de36
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Dec 28 14:46:05 2019 +0100
Created seperate files for Main program, Element routine, Material routine,
↪  Matrial properties and Geometry. Sorted the data into files and checked
↪  for convergence. The results generated are satisfactory, Need to perform
↪  more test cases

commit e3b1246e432753dcf1a8a375d5e7510150bc06b5
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Dec 27 21:06:58 2019 +0100
preprocessing_functions is added with NURBS surface point algorithm, Check for
↪  point on NURBS surface,Test case is successful,Also surface is plotted
↪  looping over the knot span

commit 28f28c55d1bd491fd92120e498f054a74dbcf904
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Thu Dec 26 20:05:56 2019 +0100
```

```
Added Findspan, Basis Functions, Derivatives of basis functions, Surface
↪  derivatives and Rational Spline surface derivatives and tested as done
↪  before

commit a05ba916bdc0a5791dc28ce0aeb40133e3d9c65d
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Thu Dec 26 18:38:17 2019 +0100
Created a seperate preprocessing_function.py file to combine and use all the
↪  preprocessing functions created till now

commit 10b424e5ae3036ad0924a64b9b9e1038e74125f4
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Dec 25 20:53:24 2019 +0100
Written Material routine and element routine flow. Have to arrange data into
↪  proper import files and complete main program

commit 20cb0887139aee7f5702731995c9ad1552f61479
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Dec 25 10:51:36 2019 +0100
Stiffnes matrix generated is inline with fem code

commit e9692287d7dd6db0fc01986a05dc4bd746b65246
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Dec 25 02:27:21 2019 +0100
Code till B matrix is done, Have to check for correctness of results

commit 3d7659ab657ef6a4cca0c7c82d76da3b5d9a56d1
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Tue Dec 24 00:38:22 2019 +0100
Generated Local stiffness matrix, Have to debug the errors

commit 00e700db9be0e00c1b6387bb160baff03a972931
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Tue Dec 24 00:01:26 2019 +0100
Copied Derivatives of NURBS Basis functions verified algorithm

commit ca6f4d8171c7ba052b036115663ff13d4550ac21
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Dec 23 22:48:16 2019 +0100
Problem with indexing, Have to sort it out

commit 5a21302ee24c08072db57113599945ae0c1b4a59
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Dec 23 20:38:12 2019 +0100
Have written the flow for derivativ of NURBS Basis function

commit e169a571b158c4f8468bf90c37b8f74773f30d25
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Dec 23 20:36:14 2019 +0100
Have written the flow for derivativ of NURBS Basis function
```

```
commit 83cab6d1814948380d9aca5150fffbcdc72e8da6
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Dec 23 14:37:58 2019 +0100
Should verify the NURBS basis functions derivatives wrt x and y global
↪  co-ordinates

commit 44cb3dd63fcb776cd3efbfed05f253bc6ab504e8
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Dec 22 15:17:51 2019 +0100
No major modifications only changed matrix dimensions

commit 41f0970034f0fb92311509e03605cc39facac7bb
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Dec 22 15:17:12 2019 +0100
Modified SKL matrix dimensions and p is changed to q in loop min()

commit a8fefc2e800ea33e296c1d6016e86d12fce1ab62
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Dec 22 15:15:40 2019 +0100
Jacobian matrix is generated as expected

commit 63de87ddabb1c08731df404fc43d146757249f46
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Dec 21 11:50:07 2019 +0100
Started with stiffness matrix and assembly functions, have to extend the code

commit 5aff7aa584ca82d5573de3ff5e66caea044a3081
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Dec 21 11:48:30 2019 +0100
Changed array dimensions for SKL, the value of k+l should lie between 0 and
↪  d,made changes with that

commit 8cca1e935d58e9632e635db13cd5381624064c4e
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Dec 18 22:33:59 2019 +0100
The function to generate derivatives of a 2D NURBS is coded and test case with
↪  weights as 1 is tested, The results are satisfactory,The result with
↪  different weight functions have to be evaluated with any source

commit 7f17ec887f46179886d67ffb86cbe92a78aa163f
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Dec 18 19:49:38 2019 +0100
Algorithm to find derivative of W(u) is working and tested with example

commit b1e982015325ff6aa0324d03198370ef1f47ec02
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Dec 18 19:49:21 2019 +0100
Algorithm to find derivative of A(u) is working and tested with example

commit ed72cded5952e7f530a5f172da2aa0d7c98bca20
Author: sakiv93 <vikas.sakiv93@gmail.com>
```

```
Date:   Wed Dec 18 19:48:42 2019 +0100
Algorithm is working and tested wit example

commit 93d03a3dd030408efaf81ad603c9bdc613a27a2a
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Dec 18 19:47:58 2019 +0100
Changed function name to NURBS_surface point

commit 8cbee8c2bb0253649fdfe5315440657ad4b52779
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Dec 18 02:03:01 2019 +0100
Derivative at any point is calculated, Have to test it with more examples and
↪  have to solve the issue for last point derivative which cannot be derived
↪  due to no function value at last or end point

commit e2a7de7c3c80feb12bbdbda289c079d0f031b882
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Tue Dec 17 16:10:30 2019 +0100
Function is evaluated with test case

commit e6930551e9d5cde4fdc43a2cd52d356b9391fb1f
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Tue Dec 17 14:30:39 2019 +0100
Algorithm is completed have to check for errors

commit cae24604d3c8e2924fc9e7ba5b2f8869c55b4c60
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Dec 16 17:19:37 2019 +0100
Documentation in progress 16/12

commit 1b9aef3602710da24fcd5f05d73400639a82e815
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Dec 16 16:09:42 2019 +0100
Documentation in progress

commit a8a87e123023ef35d55dd98a2ed1922a20a0b42d
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Dec 16 16:09:15 2019 +0100
Documentation in progress

commit 2ae0cbcb82ee866f22d908d28c547ae56992a879
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Dec 15 14:30:35 2019 +0100
Documentation tex file as on 15-12-19

commit d44772a0cc13e2393173d511c87dd2259b215d5e
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Dec 15 14:30:12 2019 +0100
Documentation as on 15-12-19

commit bb58a6607d0069e5fd50b40b41ab047108c16dbf
```

```
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Nov 29 19:37:28 2019 +0100
Surface has been generated and surface plot is done. Have to solve the problem
↪  function not working for u and v values equal to last knot vector

commit 161e642255cb635633ad5e4491ffdf974094f075
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Thu Nov 28 20:47:21 2019 +0100
Function working and generating desired result. Need to loop over for
↪  extracting 4 points in surface point

commit 395e6023d1294e62ac0ca0a4e6f9685ec293c510
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Mon Nov 25 02:13:44 2019 +0100
Basis function for given index and degree is not giving out desired result.
↪  Have to check for potential errors

commit 5d3a4236c8c2b108b641bfac29244ed5509670f6
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Fri Nov 22 23:01:17 2019 +0100
Test case successful, Function working without errors

commit 1386e28b5f55318f827b3b264d58c0e815118688
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Wed Nov 20 11:26:27 2019 +0100
Developed find position in knot vector using linear search

commit f563a93cc21482ec4acfbd4f8312cc64bc62e63d
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Nov 17 23:57:34 2019 +0100
Plotting is proper with 2d array of control points

commit c08194cf35de72cac1c30f787c241bb140ad2fc7
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Nov 17 23:54:40 2019 +0100
Surface is not generated yet

commit 1ebdd17fd4309fd18a55f869c107e0bd67a3bc73
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Nov 17 15:36:49 2019 +0100
Bspline function is plotted and crossed checked have to write comments

commit 1cdeddca7a1e80bf30d23451d0543fa2ee7a264e
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Nov 17 15:36:14 2019 +0100
Bspline basis function is working fine

commit 0a905e2a9689501605c50bda935438e3a9830439
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sun Nov 17 15:35:22 2019 +0100
Updated knot vector changing order to degree
```

```
commit 6730e792f84b58d513df40455949d6d38cb3ab33
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Nov 16 21:36:30 2019 +0100
Bspline function is generated without errors

commit e0ecf84b5e51c61388b9d14ed4905e7d373d9506
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Nov 16 19:25:48 2019 +0100
Knot vectors generated properly without any error

commit b73ee11f3be716d36c07140c29f9cb597c859f93
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Sat Nov 16 19:25:07 2019 +0100
Code with solved problem for divided by zero error but B spline function is
↪  generating only zero valued array

commit 63d0aeadd6aa8312fad907b650c8c4adf3721017
Author: sakiv93 <vikas.sakiv93@gmail.com>
Date:   Tue Oct 22 20:41:16 2019 +0200
Generated knot vectors
```

References

[1] Abaqus Documentation ABAQUS. Version 6.13. 2014. *Dassault Systemes: 3DS Paris Campus*, 10.

[2] Vishal Agrawal and Sachin S Gautam. Iga: A simplified introduction and implementation details for finite element users. *Journal of The Institution of Engineers (India): Series C*, 100(3):561–585, 2019.

[3] Dr.Hütter. *Non linear finite element methods.* Lecture notes, 2019.

[4] S Kozinov and M Kuna. Simulation of fatigue damage in ferroelectric polycrystals under mechanical/electrical loading. *Journal of the Mechanics and Physics of Solids*, 116:150–170, 2018.

[5] Vinh Phu Nguyen, Robert N Simpson, SPA Bordas, and Timon Rabczuk. An introduction to isogeometric analysis with matlab implementation: Fem and xfem formulations. *arXiv preprint arXiv:1205.2129*, page 26, 2012.

[6] Vinh-Tan Nguyen, Pankaj Kumar, and Jason Yu Chuan Leong. Finite element modellingand simulations of piezoelectric actuators responses with uncertainty quantification. *Computation*, 6(4):60, 2018.

[7] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 2012.

[8] Mit Shah and Ravi Katukam. Stress analysis without meshing iso-geometric analysis finite element method (igafem). *Boeing Summer Internship Project*, 2015.

[9] Martin Siggel, Merlin Pelz, Konstantin Rusch, Jan Kleinert, and DLR Cologne. The tigl geometry library and its current mathematical challenges. In *Workshop DLR/Cologne University*, 2017.