Personal Programming Project 2020-21

ISO-GEOMETRIC ANALYSIS BASED TOPOLOGY OPTIMIZATION (IGTO)

TECHNISCHE UNIVERSITÄT BERGAKADEMIE FREIBERG

Die Ressourcenuniversität. Seit 1765.

Viswambhar Reddy Yasa

Computational Material Science

65074

Supervised by
Dr.Martin Abendroth

February 26, 2021

Contents

1 Introduction

ISO-Geometric analysis(IGA) is a recently developed method in which the Computer Aided Design (CAD) is integrated with Computer Aided Engineering (CAE). In Computer Aided Engineering, we utilize finite element method to find approximate solution of the problem. This approximation is done by discretizing the geometry into smaller parts known as elements.

In many standard design tool, basis splines (B splines) are the design tool used to create elements like curves, lines and surfaces. Standard Finite element method uses Lagrangian function as the basics, what-if we use B splines or NURBS(Non-Uniform Rational B-Splines) as basics function for FEM which is ISO-geometric analysis. In ISO-geometric analysis, there is a tight coupling between design(CAD) and analysis (CAE) such that the geometry is captured exactly eliminating any error due to geometric approximation.

1.1 Advantages of IGA over standard FEM

- Smooth contact surface can be obtained without geometrical approximation, leading to more physically accurate contact stresses.[2]
- Due to integrating of CAE and CAD, we save a lot time but eliminating geometric discretization.
- IGA has been applied to cohesive fracture, outlining a framework for modeling debonding along material interfaces using NURBS.[2]
- Due to strong coupling with geometry, optimization problem can be solved effectively. This lead us to using IGA for structural topology optimization.

1.2 Topology optimization using IGA

The main idea behind structural optimization is to get minimum weight structures for different stresses and design constraints. Topology optimization is a mathematical method of optimizing the distribution of material within the domain so that the structure would satisfy the design parameters. The design parameters include geometry of the domain, applied load and position of the load, the amount of mass which has remain after optimization. We implemented SIMP(Solid Isotropic Material with Penalization) method in order to remove porosity and this method is quite effective for minimum weight topology. [3]

The main objective of personal programming project is to develop a python code which could implement topology optimization of a structure using a meshless method(IGA). First, the structure is analysed using IGA and compliance is calculated. In personal programming project, We developed a python code for 3D Nurbs volume. FEM is implemented for a cantilever beam and topology optimization is implemented and plotted in python.

adddd more

2 Basis Splines

In CAD to represent geometrical objects we require different type of discretisation which cannot be achieved using Bezier curves or Lagrange polynominal. So basis splines based method NURBS is used to get desired properties for interactive geometrical design.[2]. Fig[1] shows a Cubic B-spline.

To construct a B-spline , the control points and degree of the curve have to be specified. Based on which the knot vector is developed.We would discuss about in this session[2].

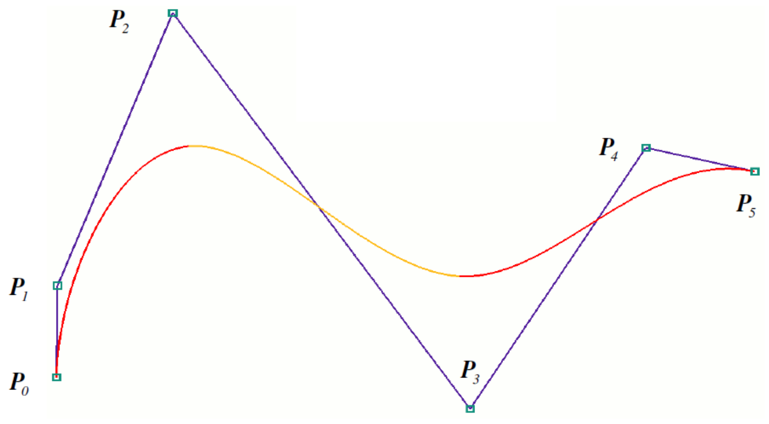


Figure 1: Cubic B-spline with 6 control points.[1]

2.1 Control points

In FEM, we need nodes to generate mesh similarly in IGA we require control points which are co-ordinates of the curve. The shape of the curve depends on the control point position and degree of the B-spline. Control points for complex geometries can be download as text file from any design software.

2.2 Degree of the B-spline

The order or degree of B-splines represents the number of control point which influence the position of the curve. This can be clearly seen in Fig[2]. The maximum degree of the B-spline curve is one less then the number of control points.

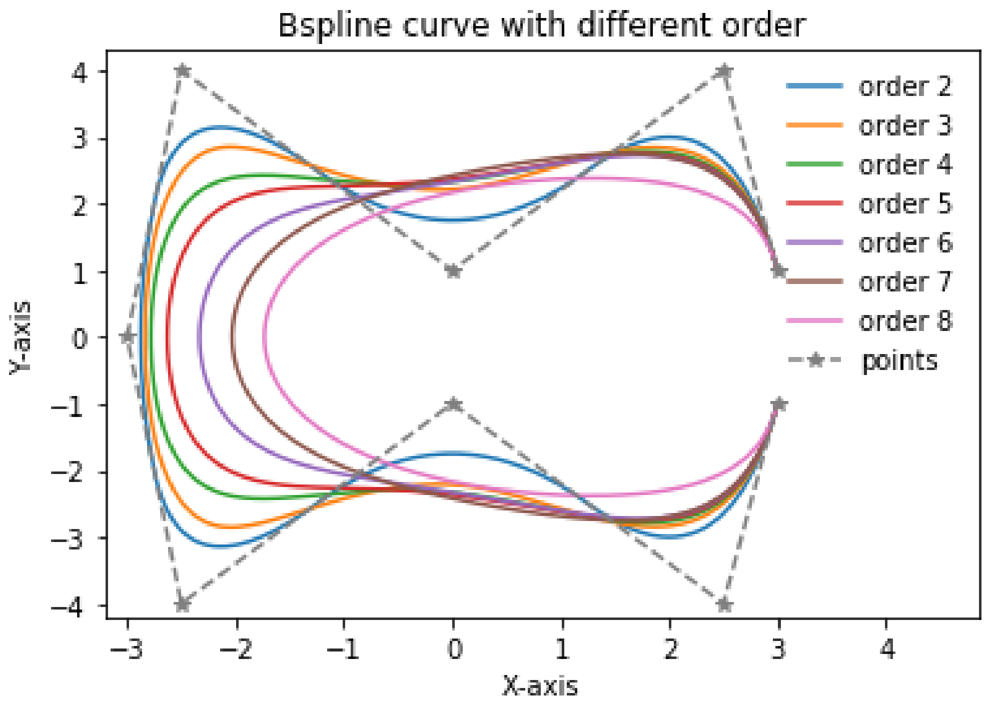


Figure 2: B-spline with 9 control points plotted for different degrees.

2.3 Generation of Knot vector

The spacing of knots in the knot vector has a significant effect on the shape of a B-spline curve. In IGA, knot vector is an array of values arranged in ascending order. A knot vector divides the knot span into elements known as patches.

Knot vector $\xi = \{\xi_1, \xi_2, ..., \xi_{n+p+1}\}$,$n$ is the number of functions, p refers to the basis functions degree.

The simplest method of specifying knot vector is by evenly spacing. If the first and the last knots appear $p + 1$ times, the knot vector is said to be open.As shown in Fig[3].

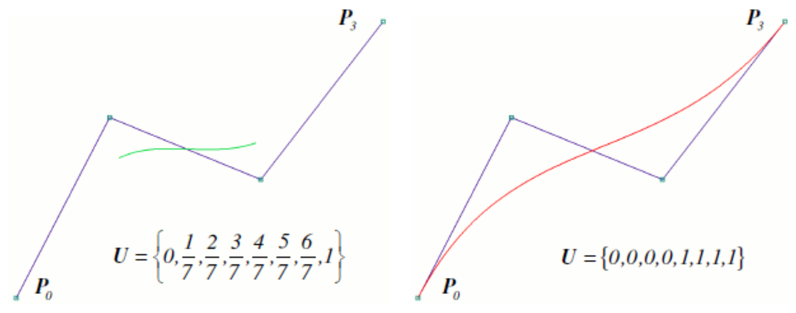

P_3
P_0
$U = \left\{0, \frac{1}{7}, \frac{2}{7}, \frac{3}{7}, \frac{4}{7}, \frac{5}{7}, \frac{6}{7}, 1\right\}$
P_3
P_0
$U = \{0,0,0,0,1,1,1,1\}$

Figure 3: B-spline curve of open and clamped knot vector.[1]

```
---------------------------------------------------------------------
def knot_vector(self, n, degree):
        '''
        This function return knot vector based on the number of
        control points and degree of B-spline
        Parameters
        ----------
        control_points : int
            number of points along with weights.
        degree : int
            order of B-splines  0-constant, 1-linear, 2-quadratic
            ,3-cubic.

        Returns
        -------
        knotvector - an array containing knots based on
          control points
        '''
---------------------------------------------------------------------
```

The function knot vector is implemented as method in Input class. To run the test case.

```
-------------------------------------------------------------------
```

```
(1) pytest test_Inputs.py::test__knotvector_true
--------------------------------------------------------------
```

The function output is compared with the value obtained from NURBS book [5].

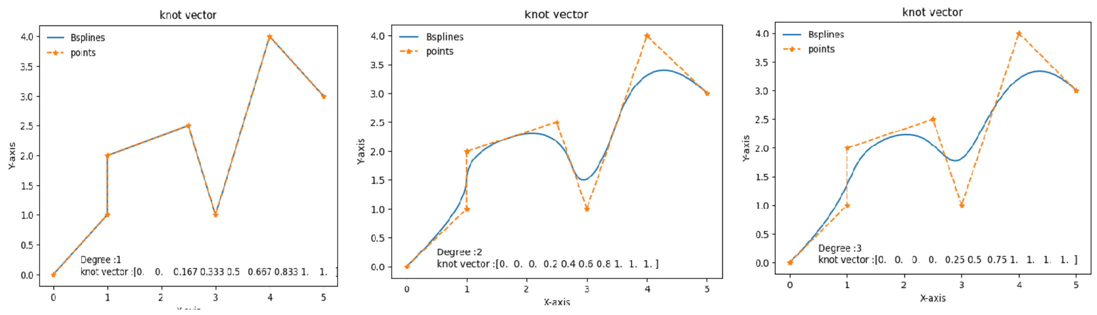


Figure 4: Knot vector and B-spline curve change due to change in degree.

2.4 B-spline Basis function

B-spline basis function is defined as recurrence formula using Cox-de-Boor formula, starting with the zeroth order basis function (p = 0).[2].

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \le \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

and for a polynomial order p greater then one

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi). \tag{2.2}$$

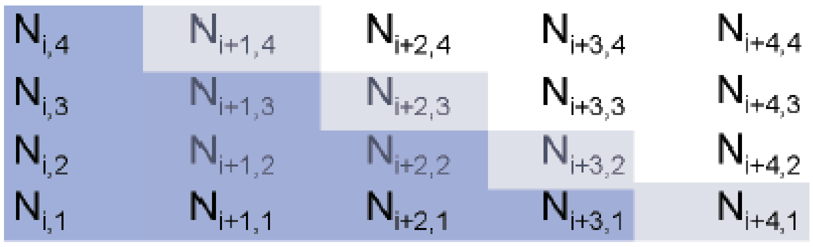


Figure 5: B-spline basis dependancy on other basis.[4]

B-splines function are not interpolatory i.e, value donot pass through them but the curves are approximated. Fig[5] show how the higher order basis functions depend on lower order basis functions.

```
---------------------------------------------------------------------
def bspline_basis(knot_index,degree,U,knotvector):
    '''
    modified version based on Algorthim 2.2 THE NURBS book pg70

    Parameters
    ----------
    knotindex : int
        DESCRIPTION. The default is ''.

    degree : int
        order of B-splines  0-constant, 1-linear, 2-quadratic,
        3-cubic
    U : int
            The value whose basis are to be found
    knotvector : array or list
             list of knot values.
    Returns
    -------
    Basis array of dimension degree+1
        It contains non-zero cox de boor based basis function
            Ex= if degree=2 and knotindex=4
            basis=[N 2_0, N 3_1, N 4_2]

    '''
---------------------------------------------------------------------
```

2.4.1 Properties of B-spline basis function

Some properties of B-spline basis function. The values for the test cases are obtained from the NURBS book.[5].

- They constitute a partition of unity.
- Each basis function is non-negative over the entire parametric domain
- They are linearly independent

Given below are respective commands to run the test cases

```
-----------------------------------------------------------------
(2) pytest test_geometry.py::test__Bspline_basis_sum_equal_to_one_true

(3) pytest test_geometry.py::test__Bspline_basis_all_values_positive_true

(4) pytest test_geometry.py::test__Bspline_basis_equal_true
-----------------------------------------------------------------
```

2.4.2 Derivatives of B-spline functions

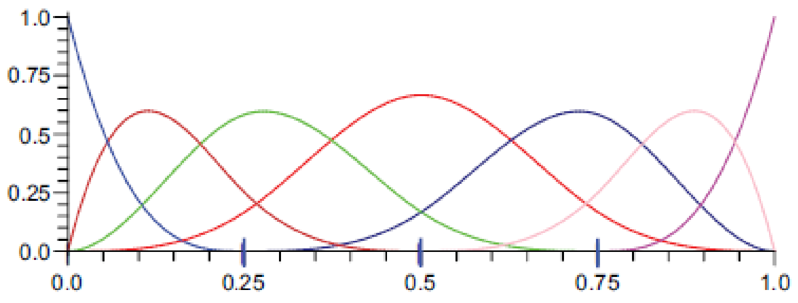
To implement FEM, we require the derivatives of the basis function. For IGA the first order derivatives are sufficient.

$$\frac{d}{d\xi}N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i}N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}}N_{i+1,p-1}(\xi) \tag{2.3}$$

```
--------------------------------------------------------------
def derbspline_basis(knot_index,degree,U,knotvector,order=1):
    """
    Modified version based on the alogorithm A2.3
    in NURBS Book page no.72
    Parameters
    ----------
    knot_index : integer
                The position of the value U in knotvector
    degree : int
        order of B-splines  0-constant, 1-linear, 2-quadratic,
         3-cubic
    U : int
            The value whose basis are to be found
    knotvector : array or list
            list of knot values.

order :int
default :1

    Returns
    -------
    Array
        Derivatives of Basis array of dimension degree+1
        It contains non-zero cox de boor based basis function
            Ex= if degree=2 and knotindex=4
            der_basis=[N' 2_0, N' 3_1, N' 4_2].
    """
-----------------------------------------------------------------
```



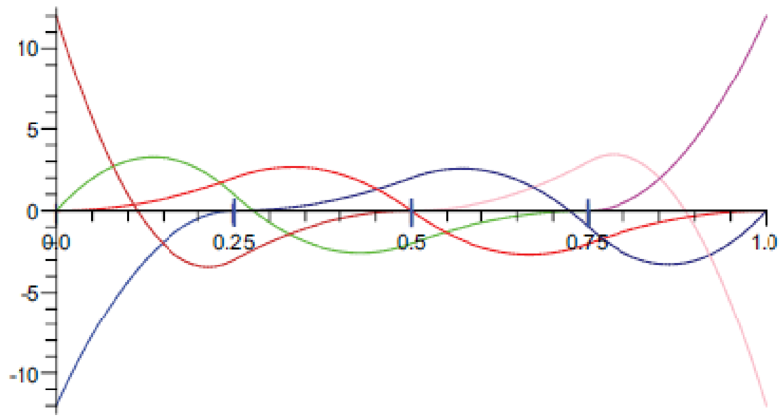


Figure 6: a)Cubic B-spline basis function b) Cubic B-spline derivative function .[5]

Properties of B-spline basis function Some properties of derivatives of B-spline basis function. The values for the test cases are obtained from the NURBS book.[5].

- Derivative of basis functions sum upto to zero $\sum_{i=0}^{n} dN_{i,p}(\xi) = 0$.

- They are linearly independent

Given below are respective commands to run the test cases.

```
-----------------------------------------------------------------
(5) pytest test_geometry.py::test__derbspline_basis_sum_equal_zero_true

(6) pytest test_geometry.py::test__derbspline_basis_equal_true
-----------------------------------------------------------------
```

3 NURBS (Non-Uniform Rational B-splines)

B-spline are useful for free-form drawing but lack the flexibility to represent simple geometry like circle and ellipsoid. Therefore NURBS are used which make use of weight functions and have the ability to form exact representation of circles, paraboloids and ellipsoids.

In Fig[7] NURBS with weight equal to one represent B-splines.

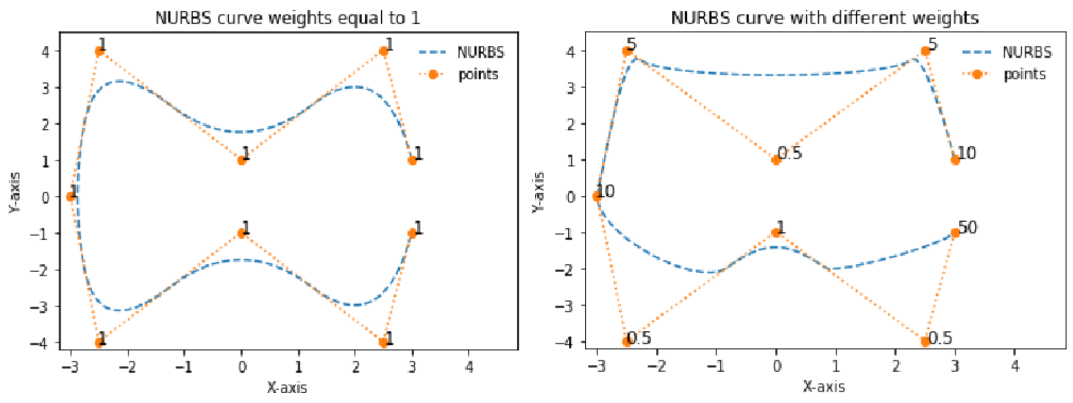


Figure 7: a) B-spline b) NURBS

3.1 NURBS basis function

Similar to B-spline basis function, NURBS depend on additional weight parameter. It is defined as:

$$R_{i,p}(\xi) = \frac{N_{i,p}(\xi)w_i}{W(\xi)} = \frac{N_{i,p}(\xi)w_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i},p}(\xi)w_i} \tag{3.1}$$

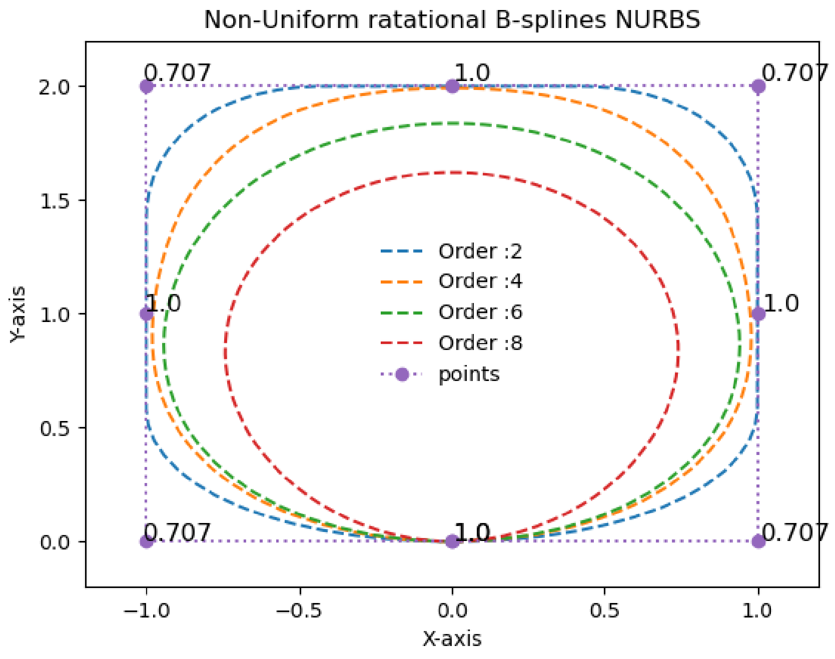


Figure 8: Circle from NURBS

By selecting appropriate weights, we can describe many type of curves.

NURBS has to satisfy all properties of B-splines, NURBS are function of B-splines and form superset of them.

3.2 NURBS derivative function

The first derivative of a NURBS basis function is computed using the quotient rule as [4]

$$\frac{d}{d\xi}R_{i,p}(\xi) = w_i \frac{N'_{i,p}(\xi)W(\xi) - N_{i,p}(\xi)W'(\xi)}{W(\xi)^2} \tag{3.2}$$

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

$$W'(\xi) = \sum_{\hat{i}=1}^{n} N'_{i,p}(\xi) w_i^n \tag{3.3}$$

3.3 NURBS volume

The most important relation required for IGA is the NURBS volume which is the tensor product of NURBS basis in 3 directions.

$$V(\xi, \eta, \zeta) = \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{l} R_{i,j,k}^{p,q,r}(\xi, \eta, \zeta) P_{i,j,k} \tag{3.4}$$

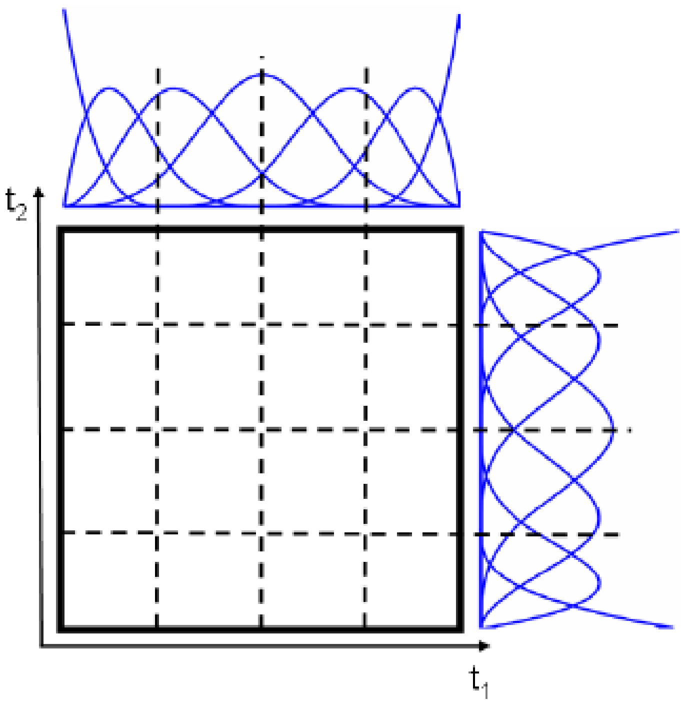


Figure 9: NURBS surface.[4]

where the trivariant function is given as

$$R_{i,j,k}^{p,q,r}(\xi, \eta, \zeta) = \frac{N_{i,p}(\xi) N_{j,q}(\eta) N_{k,r}(\zeta) w_{i,j,k}}{\sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{l} N_{i,p}(\xi) N_{j,q}(\eta) N_{k,r}(\zeta) w_{i,j,k}} \tag{3.5}$$

For easy representation,As shown in fig[9] a surface is considered. In 2D, Control points form a net and knot vector along t1,t2 form patches. In 3D, knot vector along U,V,W form a patches.

3.3.1 Derivative of Trivariant basis function

Derivative of trivariant basis function follow chain rule

$$\frac{\partial R_{i,j,k}^{p,q,r}}{\partial \xi} = \frac{N'_{i,p}N_{j,q}N_{k,r}W - N_{i,p}N_{j,q}N_{k,r}W'_{\xi}}{W^2}w_{i,j} \tag{3.6}$$

$$\frac{\partial R_{i,j,k}^{p,q,r}}{\partial \eta} = \frac{N_{i,p}N'_{j,q}N_{k,r}W - N_{i,p}N_{j,q}N_{k,r}W'_{\eta}}{W^2}w_{i,j} \tag{3.7}$$

$$\frac{\partial R_{i,j,k}^{p,q,r}}{\partial \zeta} = \frac{N_{i,p}N_{j,q}N'_{k,r}W - N_{i,p}N_{j,q}N_{k,r}W'_{\zeta}}{W^2}w_{i,j} \tag{3.8}$$

where

$$W = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l} N_{i,p}N_{j,q}N_{k,r}w_{i,j} \tag{3.9}$$

$$W'_{\xi} = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{k} N'_{i,p}N_{j,q}N_{k,r}w_{i,j} \tag{3.10}$$

$$W'_{\eta} = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{k} N_{i,p}N'_{j,q}N_{k,r}w_{i,j} \tag{3.11}$$

$$W'_{\zeta} = \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{k} N_{i,p}N_{j,q}N'_{k,r}w_{i,j} \tag{3.12}$$

```
------------------------------------------------------------
def trilinear_der(Ux,Uy,Uz,weights,xdegree,xknotvector,ydegree,
yknotvector,zdegree,zknotvector):
    '''


    Parameters
    ----------
    Ux : TYPE
        DESCRIPTION.
    Uy : TYPE
        DESCRIPTION.
    Uz : TYPE
```

```
        DESCRIPTION.
    weights : TYPE
        DESCRIPTION.
    xdegree : TYPE, optional
        DESCRIPTION. The default is XI_DEGREE.
    xknotvector : TYPE, optional
        DESCRIPTION. The default is XI_KNOTVECTOR.
    ydegree : TYPE, optional
        DESCRIPTION. The default is ETA_DEGREE.
    yknotvector : TYPE, optional
        DESCRIPTION. The default is ETA_KNOTVECTOR.
    zdegree : TYPE, optional
        DESCRIPTION. The default is ETA_DEGREE.
    zknotvector : TYPE, optional
        DESCRIPTION. The default is ETA_KNOTVECTOR.

    Returns
    -------
    None.
An array
    '''
------------------------------------------------------------
```

The values for the test cases are obtained from the NURBS book[5].Given below are respective commands to run the test cases

```
-------------------------------------------------------------
(7) pytest test_geometry.py::test__trilinear_der_Basis_sum_equal_to_one_true

(8) pytest test_geometry.py::test__trilinear_der_Basis_less_than_zero_false

(9) pytest test_geometry.py::test__trilinear_der_XI_sum_equal_to_zero_true

(10) pytest test_geometry.py::test__trilinear_der_ETA_sum_equal_to_zero_true

(11) pytest test_geometry.py::test__trilinear_der_NETA_sum_equal_to_zero_true
-------------------------------------------------------------
```

4 Background Theory

4.1 Constitutive equation and their weak form

Based on conservation of linear momentum. The equilibrium equation is given below.

$$\sigma_{ij,j} + f_i = 0 \tag{4.1}$$

The $\sigma_{ij,j}$ is known as stress tensor and f$_i$ is body force vector.

The weak form of the equ(4.1) can be obtained by using virtual displacment principle and integrated over the domain Ω.

$$\int_{\Omega} \delta u_i \left(\sigma_{ij,j} + f_i\right) d\Omega = 0 \tag{4.2}$$

Integration by parts yields

$$\int_{\Gamma} \left(\delta u_i \sigma_{ij} n_j\right) d\Gamma + \int_{\Omega} \left(f_i \delta u_i - \delta u_{i,j} \sigma_{ij}\right) d\Omega = 0 \tag{4.3}$$

where Γ is surface integral. based on cauchy's stress formula and kinematic strain relation.

$$T_i = \sigma_{ij} n_j \tag{4.4}$$

$$\epsilon_{ij} = \begin{cases} u_{i,j} & i = j \\ u_{i,j} + u_{j,i} & i \neq j \end{cases} \tag{4.5}$$

Then the weak form is reduced to

$$\sum_i \left[\int_{\Omega} \left(f_i \delta u_i\right) d\Omega + \int_{\Gamma} \left(T_i \delta u_i\right) d\Gamma - \int_{\Omega} \left(\delta u_{i,j} \sigma_{ij}\right) d\Omega \right] = 0 \tag{4.6}$$

4.2 IGA Formulation

Similar to FEM, we use Galerkin approximation such that the virtual displacements are given as.

$$\delta u_i = \sum_{m=1}^{n} \delta u_i^m R_m \tag{4.7}$$

δu_i^m are the nodal displacements and R$_m$ are the basis functions.

$$\mathbf{R} = \begin{bmatrix} R_1(\xi,\eta,\zeta) & 0 & 0 & R_2(\xi,\eta,\zeta) & 0 & \ldots & R_{n_c^c}(\xi,\eta,\zeta) & 0 \\ 0 & R_1(\xi,\eta,\zeta) & 0 & 0 & R_2(\xi,\eta,\zeta) & \ldots & 0 & R_{n_{cp}^c}(\xi, \\ 0 & 0 & R_1(\xi,\eta,\zeta) & 0 & 0 & R_2(\xi,\eta,\zeta) & \ldots & 0 \end{bmatrix} \tag{4.8}$$

Degree's of freedom vector qα is given as

$$q_\alpha = \left[u_1^1, u_2^1, u_3^1, \ldots, u_1^n, u_2^n, u_3^n\right] \tag{4.9}$$

The strains are expressed as

$$\delta\epsilon_{ij} = \frac{\partial \epsilon_{ij}}{\partial q_\alpha}\delta q_\alpha \tag{4.10}$$

Then the weak form is expressed as

$$\sum_\alpha \left[\sum_i \left[\int_\Omega \left(f_i \frac{\partial u_i}{\partial q_\alpha}\right) d\Omega + \int_\Gamma \left(T_i \frac{\partial u_i}{\partial q_\alpha}\right) d\Gamma - \int_\Omega \left(\sigma_{ij}\frac{\partial \epsilon_{ij}}{\partial q_\alpha}\right) d\Omega\right]\right] = 0 \tag{4.11}$$

A strain-displacement matrix is expressed in terms of displacements and strain via the kinematic relations. The stress tensor is expressed in terms of displacements by utilizing kinematic and constitutive relations. The constitutive matrix relates stress and strain.

$$\mathbf{B}_{k,\alpha} = \begin{bmatrix} R_{1,x} & 0 & 0 & R_{2,x} & 0 & 0 & .... & R_{n_{cp}^e,x} & 0 & 0 \\ 0 & R_{1,y} & 0 & 0 & R_{2,y} & 0 & .... & 0 & R_{n_{cp}^e,y} & 0 \\ 0 & 0 & R_{1,z} & 0 & 0 & R_{2,z} & .... & 0 & 0 & R_{n_{cp}^e,z} \\ R_{1,y} & R_{1,x} & 0 & R_{2,y} & R_{2,x} & 0 & .... & R_{n_{cp}^e,y} & R_{n_{cp}^e,x} & 0 \\ 0 & R_{1,z} & R_{1,y} & 0 & R_{2,z} & R_{2,y} & .... & 0 & R_{n_{cp}^e,z} & R_{n_{cp}^e,y} \\ R_{1,z} & 0 & R_{1,x} & R_{2,z} & 0 & R_{2,x} & .... & R_{n_{cp}^e,z} & 0 & R_{n_{cp}^e,x} \end{bmatrix} \tag{4.12}$$

$$\epsilon_k = B_{k\alpha}q_\alpha \tag{4.13}$$

$$\sigma_k = C_{kl}\epsilon_l = C_{kl}B_{l\alpha}q_\alpha \tag{4.14}$$

The constitutive matrix is calculated based on material properties. The integral of $\int_\Omega \left(B^TCBq\right)$ as stiffness matrix (K).

$$\int_\Omega \left(B^TCBq\right) d\Omega = Kq \tag{4.15}$$

The force vector is expressed as

$$F_\alpha = \int_\Omega f_i \frac{\partial u_i}{\partial q_\alpha} d\Omega + \int_\Gamma T_i \frac{\partial u_i}{\partial q_\alpha} d\Gamma \tag{4.16}$$

4.3 Topology optimization using IGA

Topology optimization can be described as binary compliance problem which is use to find a "black and white" layout that minimizes the compliance subjected to volume constrain. There are many frameworks like homogenization method and density-based approach.In our case, the material properties are assumed constant within each element. We implement density-based approach as it restricts the formation of pores and solved the minimum compliance effectively.
In density-based approach, the problem is parametrized by the material density distribution. The stiffness tensor are determined using a power-law interpolation function. The power-law implicilty penalizes the intermediate density values to remove pores in structure layout. This penalization method is referred as *Solid Isotropic Material with Penalization* (SIMP).

4.3.1 Solid Isotropic Material with Penalization (SIMP) Method

SIMP method is a heuristic relation between element density and element Young's modulus. It given by

$$E_i = E_i\left(x_i\right) = x_i^p E_0, \quad x_i \in]\,0,1] \tag{4.17}$$

the element density is given as

$$x_i = \begin{cases} 1 & \text{if} \quad \mathbf{x} \in \Omega_s \\ 0 & \text{if} \quad \mathbf{x} \in \Omega \backslash \Omega_s \end{cases} \tag{4.18}$$

The element densities are approximated by the NURBS basis functions over a patch.

$$x_{r,s} = \sum_{i=0}^{n_1} \sum_{i=1}^{n_2} R_{i,j}(r,s) x_{ij}^p \tag{4.19}$$

where E_0 is the elastic modulus of material, E_{min} is the minimum elastic modulus of the void region which prevents stiffness matrix singularity. **P** is the penalization factor introduced to ensure black and white layout.A modified SIMP method is given below.

$$\mathrm{E}_i = \mathrm{E}_i\left(x_i\right) = \mathrm{E}_{\min} + x_i^p\left(\mathrm{E}_0 - \mathrm{E}_{\min}\right), \quad x_i \in [0,1] \tag{4.20}$$

Advantages of modified SIMP method

1. Independency of the problem on minimum value of material's elastic modulus and the penalization power.

4.3.2 Minimum Compliance formulation

The main objective of minimum compliance formulation is to find material density distribution that minimizes deformation for prescribed support and loading condition. The compliance is given as,

$$c(\overline{\mathbf{x}}) = \mathbf{F}^{\mathrm{T}}\mathbf{U}(\overline{\mathbf{x}}) \tag{4.21}$$

where F is the vector of nodal forces and U(x) is the vector of nodal displacements. The mathematical formulation of the optimization problem reads as follows:

$$\text{minimize } : c(x) = U^T K U = \sum_{e=1}^{N} E_e\left(x_e\right) u_e^T k_0 u_e \tag{4.22}$$

$$\text{subject to } : \begin{cases} \frac{V(x)}{V_0} = f \\ KU = F \\ 0 \leq x \leq 1 \end{cases} \tag{4.23}$$

Where C is compliance, k_0 is the element stiffness matrix, N number of elements, V(x) and V_0 are the material volume and design volume. f is the prescribed volume fraction.
The derivatives of the compliance is

$$\frac{\partial c(\overline{\mathbf{x}})}{\partial x_e} = \sum_{i \in N_e} \frac{\partial c(\overline{\mathbf{x}})}{\partial \tilde{x}_i}\frac{\partial \tilde{x}_i}{\partial x_e} \tag{4.24}$$

$$\begin{aligned} \frac{\partial c(\overline{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} &= \mathbf{F}^{\mathrm{T}}\frac{\partial \mathbf{U}(\overline{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} \\ &= \mathbf{U}(\overline{\mathbf{x}})^{\mathrm{T}}\mathbf{K}(\overline{\mathbf{x}})\frac{\partial \mathbf{U}(\overline{\mathbf{x}})}{\partial \overline{\mathbf{x}}_i} \end{aligned} \tag{4.25}$$

$$\mathbf{K}(\tilde{\mathbf{x}})\mathbf{U}(\overline{\mathbf{x}}) = \mathbf{F} \tag{4.26}$$

The above equation(4.26) is derivatived with respect to element density.

$$\frac{\partial \mathbf{K}(\overline{\mathbf{x}})}{\partial \overline{\mathbf{x}}_i}\mathbf{U}(\overline{\mathbf{x}}) + \mathbf{K}(\overline{\mathbf{x}})\frac{\partial \mathbf{U}(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} = \mathbf{0} \tag{4.27}$$

which yields,

$$\frac{\partial \mathbf{U}(\tilde{\mathbf{x}})}{\partial \overline{\mathbf{x}}_i} = -\mathbf{K}(\tilde{\mathbf{x}})^{-1}\frac{\partial \mathbf{K}(\overline{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i}\mathbf{U}(\overline{\mathbf{x}}) \tag{4.28}$$

On substitution, the objective function is obtained.
Compliance constrain:

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} = -\mathbf{u}_i(\tilde{\mathbf{x}})^{\mathrm{T}} \left[\mid p \tilde{\mathrm{X}}_i^{p-1} \left(\mathrm{E}_0 - \mathrm{E}_{\min}\right) \mathbf{k}_i^0 \right] \mathbf{u}_i(\overline{\mathbf{x}}) \tag{4.29}$$

Volume constrain:

$$\frac{\partial v(\tilde{\mathbf{x}})}{\partial x_e} = \sum_{i \in N_e} \frac{\partial v(\overline{\mathbf{x}})}{\partial \tilde{x}_i} \frac{\partial \bar{x}_i}{\partial x_e} \tag{4.30}$$

The objective function are obtained, we require optimization algorithm to solve them to get minimization.

4.3.3 Sensitivity

To ensure that topology optimization problem would exist and avoid the formation of checker board pattern or pores in the design. This can be achieved by applying filter or sensitivity to objective function like compliance and volume constrains. Basic filter function is defined as

$$\bar{x}_e = \frac{\sum_{i \in N_c} H_{ei} x_i}{\sum_{i \in N_c} H_{ei}} \tag{4.31}$$

where H_{ei} is the weight factor given as.

$$H_{ei} = \max\left(0, r_{\min} - \Delta(e, i)\right) \tag{4.32}$$

Based on filter radius r_{min}, we calculate the center-to-center distance between elements $\Delta(e, i)$.
The sensitivity for the objective function c and the material volume V with respect to design variable x_i. is now given as:

$$\frac{dc}{dx_j} = \sum_{e \in N_j} \frac{1}{\sum_{i \in N_c} H_{ei}} H_{je} \frac{dc}{\bar{x}_e} \tag{4.33}$$

$$\frac{dV}{dx_j} = \sum_{\epsilon \in N_j} \frac{1}{\sum_{i \in N_e} H_{ei}} H_{je} \frac{dV}{\tilde{x}_e} \tag{4.34}$$

```
---------------------------------------------------------------------

def Knearestneighbours(rmin,nelx,nely,nelz):
    '''
    function used to generate weight factor from the given parameters

    Parameters
    ----------
    rmin : float
        minimum radius to describe the radius of the filter.
    nelx : int
        No of elements in x.
    nely : int
        No of elements in y.
    nelz : int
        No of elements in z.

    Returns
    -------
    H : numpy array
        An array of weight function for each element.
    DH : numpy array
        sum of all weight function saved as single array .

    '''
----------------------------------------------------------------------
```

4.3.4 Optimality criteria method

To solve structural optimization problem using optimality criteria (OC) method.In OC method, the constrains 0 ¡ x ¡ 1 is inactive. The convergence can able be achieved if the KKT condition are satisfied.

$$\frac{\partial c(\overline{\mathbf{x}})}{\partial x_e} + \lambda \frac{\partial v(\overline{\mathbf{x}})}{\partial x_e} = 0 \tag{4.35}$$

The implementation of OC update scheme is given below.

$$x_e^{\text{new}} = \begin{cases} \max(0, x_e - m), \text{ if } x_e B_e^\eta \leq \max(0, x_e - m) \\ \min(1, x_e + m), \text{ if } x_e B_e^\eta \geq \min(1, x_e - m) \\ x_e B_e^\eta, \text{ otherwise} \end{cases} \tag{4.36}$$

where m is move limit and η is numerical damping coefficient and optimality condition B$_e$ is obtained from equation(4.37).

$$B_e = -\frac{\partial c(\overline{\mathbf{x}})}{\partial x_e} \left(\lambda \frac{\partial v(\overline{\mathbf{x}})}{\partial x_e} \right)^{-1} \tag{4.37}$$

The unknown value λ is the lagrangian multiplier which is obtained by root finding algorithm such as bi-section method.

Bi-section method

1. Choose lower(l) and upper(u) bound values.
2. Compute midpoint m=(l+u)/2 and calculate f(m)
3. Determination of next subinterval based on sign of the product f(l)x f(m) and f(u)x f(m)
4. Repeat till tolerance is achieved.

Test command to check the implementation of OC for simple function and quadratic function.

```
---------------------------------------------------------------------
pytest test_optimization.py::test__optimality_criteria_simple_function

pytest test_optimization.py::test__optimality_criteria_quadratic_function

---------------------------------------------------------------------
```

4.3.5 Method of moving Asymptotes

Moving Asymptotes is a gradient based optimization algorithm. MMA subproblem is given as

$$\text{minimize} \quad \tilde{f}_0^{(k)}(\mathrm{x}) + a_0 z + \sum_{i=1}^{m}\left(c_i y_i + \frac{1}{2} d_i y_i^2\right) \tag{4.38}$$

An approximate function is build to satisfy the above problem

$$\tilde{f}_i^{(k)}(\mathrm{x}) = \sum_{j=1}^{n}\left(\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j} + \frac{q_{ij}^{(k)}}{x_j - l_j^{(k)}}\right) + r_i^{(k)}, \quad i = 0, 1, \ldots, m \tag{4.39}$$

where p$_{ij}$ and q$_{ij}$ are given as

$$p_{ij}^{(k)} = \left(u_j^{(k)} - x_j^{(k)}\right)^2 \left(1.001\left(\frac{\partial f_i}{\partial x_j}\left(\mathrm{x}^{(k)}\right)\right)^+ + 0.001\left(\frac{\partial f_i}{\partial x_j}\left(\mathrm{x}^{(k)}\right)\right)^- + \frac{10^{-5}}{x_j^{\max} - x_j^{\min}}\right) \tag{4.40}$$

$$q_{ij}^{(k)} = \left(x_j^{(k)} - l_j^{(k)}\right)^2 \left(0.001 \left(\frac{\partial f_i}{\partial x_j}\left(\mathrm{x}^{(k)}\right)\right)^+ + 1.001 \left(\frac{\partial f_i}{\partial x_j}\left(\mathrm{x}^{(k)}\right)\right)^- + \frac{10^{-5}}{x_j^{\max} - x_j^{\min}}\right) \tag{4.41}$$

$$r_i^{(k)} = f_i\left(\mathrm{x}^{(k)}\right) - \sum_{j=1}^{n}\left(\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j^{(k)}} + \frac{q_{ij}^{(k)}}{x_j^{(k)} - l_j^{(k)}}\right) \tag{4.42}$$

The lower and upper bound are formulated as below

$$\alpha_j^{(k)} = \max\left\{x_j^{\min}, \quad l_j^{(k)} + 0.1\left(x_j^{(k)} - l_j^{(k)}\right), \quad x_j^{(k)} - 0.5\left(x_j^{\max} - x_j^{\min}\right)\right\} \tag{4.43}$$

$$\beta_j^{(k)} = \min\left\{x_j^{\max}, u_j^{(k)} - 0.1\left(u_j^{(k)} - x_j^{(k)}\right), \quad x_j^{(k)} + 0.5\left(x_j^{\max} - x_j^{\min}\right)\right\} \tag{4.44}$$

The lower and Upper Asymptotes are calculated using the below relation
For iteration less then 3:

$$\begin{aligned} l_j^{(k)} &= x_j^{(k)} - 0.5\left(x_j^{\max} - x_j^{\min}\right) \\ u_j^{(k)} &= x_j^{(k)} + 0.5\left(x_j^{\max} - x_j^{\min}\right) \end{aligned} \tag{4.45}$$

For iteration greater then 3:

$$\begin{aligned} l_j^{(k)} &= x_j^{(k)} - \gamma_j^{(k)}\left(x_j^{(k-1)} - l_j^{(k-1)}\right) \\ u_j^{(k)} &= x_j^{(k)} + \gamma_j^{(k)}\left(u_j^{(k-1)} - x_j^{(k-1)}\right), \end{aligned} \tag{4.46}$$

where γ is

$$\gamma_j^{(k)} = \begin{cases} 0.7 & \text{if } \left(x_j^{(k)} - x_j^{(k-1)}\right)\left(x_j^{(k-1)} - x_j^{(k-2)}\right) < 0 \\ 1.2 & \text{if } \left(x_j^{(k)} - x_j^{(k-1)}\right)\left(x_j^{(k-1)} - x_j^{(k-2)}\right) > 0 \\ 1 & \text{if } \left(x_j^{(k)} - x_j^{(k-1)}\right)\left(x_j^{(k-1)} - x_j^{(k-2)}\right) = 0 \end{cases} \tag{4.47}$$

Primal-dual method for solving the Moving Asymptotes method

Prime-dual method is part of active set strategy and constrained based non-linear optimization algorithm.It is gradient based method and used Newton Raphson to solve equ(4.48) To satisfy constrained problem, we build a Lagrangian function.

$$
\begin{aligned}
L =&\psi(x,\lambda) + (a_0 - \zeta)\, z + \sum_{j=1}^{n} \left(\xi_j \left(\alpha_j - x_j\right) + \eta_j \left(x_j - \beta_j\right)\right) + \\
&+ \sum_{i=1}^{m} \left(c_i y_i + \frac{1}{2} d_i y_i^2 - \lambda_i a_i z - \lambda_i y_i - \lambda_i b_i - \mu_i y_i \right)
\end{aligned} \tag{4.48}
$$

ALGORTHIM

Test command to check the implementation of MMA with quadratic equation and KKT condition provided in literature.[citition]

```
-----------------------------------------------------------------------
pytest test_optimization.py::test__MMA_literature_equation
-----------------------------------------------------------------------
```

5 IGTO implementation

In FEM, we use Lagrangian basis function to discretised both geometry and displacement in iso-parametric method. Even IGA can be iso-parametric formulation.Similar to FEM, IGA can be divided into sub programs

- Pre-Processing
- Processing
- Post-Processing

5.1 Relevant spaces

In FEM, there are parent and unit space. The core concepts in IGA are the outlining the relevant spaces. As shown in fig[10]

1. **Parent space**
   This space is also known as unit space. The parent space is defined over the interval Ω=[-1,1].This required to implement numerical integration such as gauss quadrature.

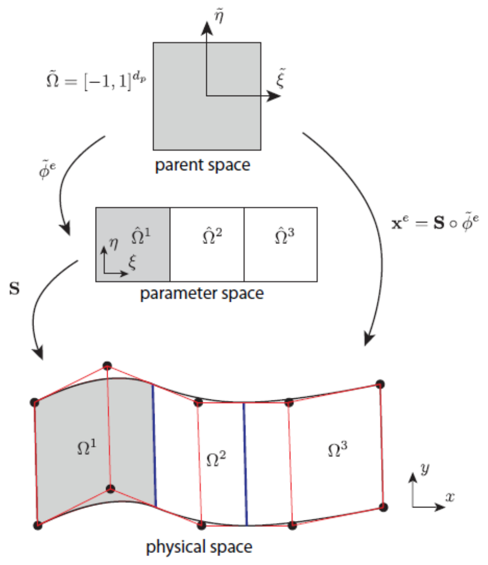


Figure 10: Relevant spaces.[2]

2. **Parameter space**
   This space also referred as pre-image of NURBS mapping is formed by using non-zero intervals of knot vectors. In parametric space, the shapes in physical space are normalized. Due to normalization the domain is defined over Ω=[0,1].

3. **Physical space**
   Physical space is associated with co-ordinate system. NURBS is defined by control points and weights.

5.2 Pre-processing

In pre-processing state, the geometry has to be defined.

5.2.1 Generation of NURBS parameters

Along with physical parameters like length, breath, width, number of elements along x(nx), number of elements along y(ny), number of elements along z(nz), order along x (xidegree),order along y (etadegree) and order along z (netadegree). We require NURBS parameters like control points and knot vectors to represent geometry.

A python class is defined to generate NURBS parameters which is given below.

```
------------------------------------------------------------

class Inputs():
    '''
    A geometry class with NURBS parameter as the method
    Methods:
        crtpts_coordinates : generate control points along with weights

        knot_vector :generates knot vector  based on control points
        generated from crtpts_coordinates and degree

        knotconnect : other parameter required for
        assembly like span, unique knots are .

    '''

    def __init__(self, length=1, height=1, width=1, nx=1, ny=1, nz=1,
    xidegree=1, etadegree=1, netadegree=1)
------------------------------------------------------------
```

"Rhino" a commerical software can be to generate exact parametric values for complex NURBS geometry. Test cases to check NURBS parameters generated through Inputs class.

```
------------------------------------------------------------
(12) pytest test_Inputs.py::test__controlpoints_coordinates_true

(13) pytest test_Inputs.py::test__knotconnect_span_true

------------------------------------------------------------
```

5.2.2 Assembly of Geometry

In IGA, discrization of global geometry into smaller patches is done through assembly array. The assembly array would require a)control points, b)knot vector, c) weights .

The following are the function required to generate a assembly array.

1. **Element order**
A function which generates a 3D array which contains the positions of the elements.

2. **knot connectivity**
An array which contains the length of each element expressed in the form of knot points.

3. **Control point assembly**
An array which contains the control points of the element based on the degree and knot connectivity and element order.

Number of control points in each element is give as n^{cp}=(p+1)*(q+1)*(r+1). For number of elements 2,1,1 respectively in x,y,z direction with order equal to one, n^{cp}=8. The control point array should be as shown in fig[11].

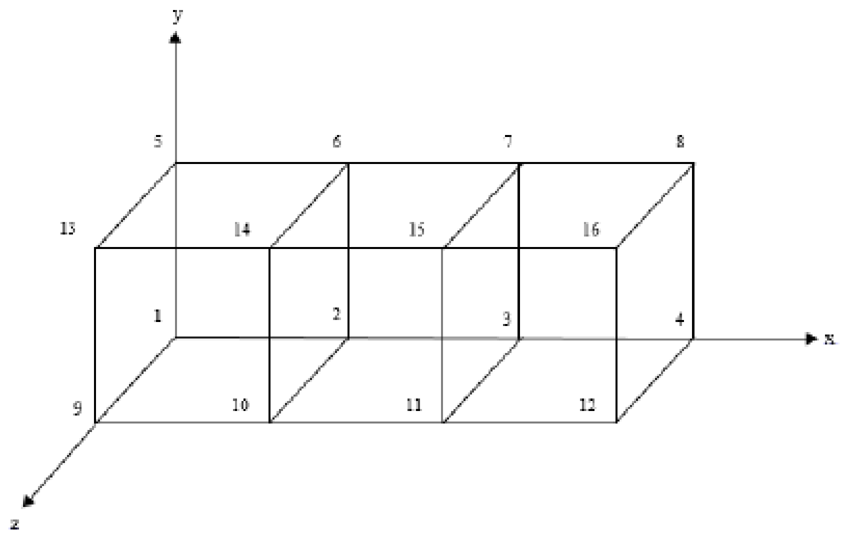


Figure 11: Control point assembly and node numbering

```
-----------------------------------------------------------

def controlpointassembly(n,p,q,nU,nV,nW,xdegree,ydegree,zdegree,
knotconnectivityU,knotconnectivityV,knotconnectivityW):#
    '''
    Inputs:
    n,p,q : int
     knot points in xi,eta,neta direction
    nU,nV,nW :int
     number of element in x,y,z direction
    xdegree,ydegree,zdegree :int
       degree of the NURBS along xi,eta,neta
    knotconnectivityU,knotconnectivityV,knotconnectivityW
    :int

    outputs:
    element assembly : narray
     consist of control points of the elements

    '''
    elements_assembly=np.zeros(((nU*nV*nW),
     (xdegree+1)*(ydegree+1)*(zdegree+1)))
    elements_order=elementorder(n,p,q)
    a=0
    for i in range(nW):
        for j in range(nV):
            for k in range(nU):
                c=0
                for l in range(len(knotconnectivityW[i,:])):
                    for m in range(len(knotconnectivityV[j,:])):
                        for n in range(len(knotconnectivityU[k,:])):
                    elements_assembly[a,c]=elements_order[
                    knotconnectivityU[k,n],
                             knotconnectivityW[i,l],
                             knotconnectivityV[j,m]]
                            c+=1
                a+=1
    elements_assembly=elements_assembly.astype(int)
    return elements_assembly.

    '''
------------------------------------------------------------
```

Following are test commands to check the assembly matrix for different values and order

```
------------------------------------------------------------
(14) pytest test_geometry.py::test__single_element_Assembly
```

```
(15) pytest test_geometry.py::test__element_Assembly_C0_continuity

(16) pytest test_geometry.py::test__single_element_Assembly_C1_continuity_along_x
------------------------------------------------------------
```

5.2.3 Boundary conditions

Implementation of Dirichlet boundary condition for homogeneous case is quite easy as the control points are on the surface and the corresponding control points can be equated to zero. For inhomogeneous case, method like least square have to be implemented which is not in scope of this project.
Boundary condition are implemented in the form a python switch class depending on option respective BC are applied, we get fixed control point nodes and load control point nodes. Give below

```
------------------------------------------------------------
class BC_Switcher(object):
'''
A python switch class'
Methods
 based on option
Ex 0 -cantilever beam with load at the end
INPUTS:
option
 CONTROL_POINTS : numpy array
 Length :float
 height :float
 width  :float

OUTPUTS:
 fixed nodes: numpy array
 load nodes : numpy array

    def __init__(self,CONTROL_POINTS,length,height,width,bc_disp):
        self.CONTROL_POINTS=CONTROL_POINTS
        self.length=length
        self.width=width
        self.height=height
        self.bc_disp=bc_disp
-----------------------------------------------------------------------
```

5.3 Processing

We need to compute global stiffness matrix **K** and global force vector **F**. For this, we requires the derivatives of trilinear basis function which is used to formulate jacobian. A numerical integration scheme is employed to solve volume integrals along with mapping physical space to parent space.

5.3.1 Mapping relevant spaces

The use of NURBS basis introduces a parametric space which needs additional mapping from physical to parameteric space and parameteric space to parent space.As shown in fig[10]

1. **Mapping from physical space to parameter space**

The associated Jacobian is represented as $\mathbf{J}_1$ and is computed as below.

$$\mathbf{J}_1 = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \tag{5.1}$$

The component of the Jacobian 1 are given below

$$\begin{matrix} \frac{\partial x}{\partial \xi} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \xi} x_i & \frac{\partial x}{\partial \eta} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \eta} x_i & \frac{\partial x}{\partial \zeta} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \zeta} x_i \\ \frac{\partial y}{\partial \xi} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \xi} y_i & \frac{\partial y}{\partial \eta} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \eta} y_i & \frac{\partial y}{\partial \zeta} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \zeta} y_i \\ \frac{\partial z}{\partial \xi} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \xi} z_i & \frac{\partial z}{\partial \eta} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \eta} z_i & \frac{\partial z}{\partial \zeta} = \sum_{k=1}^{n_{cp}^e} \frac{\partial \mathbf{R}_k}{\partial \zeta} z_i \end{matrix} \tag{5.2}$$

components of $\mathbf{J}_1$ can be obtained through derivatives of trilinear basis function.

2. **Mapping from parameter space to parent space**

The associated Jacobian is represented as $\mathbf{J}_2$ and is computed as below. The $\bar{\xi}$,$\bar{\eta}$,$\bar{\zeta}$ are the gauss quadrature and ξ_{i+1}, ξ_i are the knot values of that element.

$$\xi = \frac{1}{2}\left[(\xi_{i+1} - \xi_i)\,\bar{\xi} + (\xi_{i+1} + \xi_i)\right] \tag{5.3}$$

$$\eta = \frac{1}{2}\left[(\eta_{i+1} - \eta_i)\,\bar{\eta} + (\eta_{i+1} + \eta_i)\right] \tag{5.4}$$

$$\zeta = \frac{1}{2}\left[(\zeta_{i+1} - \zeta_i)\,\bar{\zeta} + (\zeta_{i+1} + \eta_i)\right] \tag{5.5}$$

The determinent of Jacobian 2 is calculated as shown below.

$$\mathbf{J}_2 = \frac{\partial \xi}{\partial \bar{\xi}} \frac{\partial \eta}{\partial \bar{\eta}} \frac{\partial \zeta}{\partial \bar{\zeta}} \tag{5.6}$$

Below are the test cases to check the functioning of the Jacobian

```
------------------------------------------------------------------
(17) pytest test_element_routine.py::test__unit_to_parametric_space_true

(18) pytest test_element_routine.py::test__Jacobian_patch_testing_rotation
------------------------------------------------------------------
```

5.3.2 Building global stiffness matrix

From the relation 4.15, the element stiffness matrix as below.

$$K^{(e)} = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} B^T C B |J| d\bar{\Omega}_e \tag{5.7}$$

Where J $=$J$_1$*J$_2$ obtained from the relation in (5.2) and (5.6).
Since strain-displacement matrix is a function of derivatives in global co-ordinates and gauss quadrature can be applied only in normalized co-ordinate system. The below relation transform normalized derivatives to co-ordinate derivatives using Jacobian.

$$\frac{\partial R_m}{\partial x_i} = \frac{\partial \xi_j}{\partial x_i}\frac{\partial R_m}{\partial \xi_j} = J_{ij}^{-1}\frac{\partial R_m}{\partial \xi_j} \tag{5.8}$$

Element routine procedure:
1. Loop over elements, $e = 1, \ldots,$ nel
(a) Determine the co-ordinates of the element using control point assemby matrix
(b) Determine NURBS coordinates $[\xi_i, \xi_{i+1}] \times [\eta_j, \eta_{j+1}] \times [\zeta_j, \zeta_{j+1}]$ using knotspan U,V,W.
(c) $\mathbf{K}_e = \mathbf{0}$
(d) Loop over Gauss points, $\left\{\bar{\xi}_j, \bar{w}_j\right\}_{\_}$ $j = 1, 2, \ldots, n_{gp}$
i. Compute ξ, η, ζ corresponding to $\bar{\xi}_j$, $\bar{\eta}_j$ and $\bar{\zeta}_j$
ii. Compute $|J_2|$
iii. Compute derivatives of shape functions R at ξ,η and ζ from triliniear basis function.
iv. Compute $\mathbf{J}_1$ using , $\mathbf{R}_{,k}$ and $\mathbf{R}_{,\eta}$.

v. Compute Jacobian inverse $\mathbf{J}_1^{-1}$ and determinant $|J_1|$.
vi. Compute derivatives of shape functions $\mathbf{R_{xx}} = [\mathbf{R}_{,\xi}\mathbf{R}_{,\eta}]\,\mathbf{J}_1^{-1}$
vii. Use $\mathbf{R}_{,\mathbf{x}}$ to build the strain-displacement matrix $\mathbf{B}$
viii. Compute $\mathbf{K}_e = \mathbf{K}_e + \bar{w}_j\,|J_2||J_1|\,\mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}$.
(f) End loop over Gauss points.
(g) Assemble $\mathbf{K}_e : \mathbf{K}(\operatorname{sctr} B, \operatorname{sctr} B) = \mathbf{K}(\operatorname{sctr} B, \operatorname{sctr} B) + \mathbf{K}_e$.
2. End loop over elements.

The above procedure is performed using the following python function.1) gauss quadrature, 2) Jacobian, 3) strain displacement, 4) Compliance matrix, 5) element routine and 6) assemble.

Test commands to check gauss quadrature and global stiffness matrix generated from element routine.

```
-------------------------------------------------------------------
(19) pytest test_element_routine.py::test__gauss_quadrature_1point_true

(20) pytest test_element_routine.py::test__stiffness_matrix_singularity
-------------------------------------------------------------------
```

5.3.3 Topology optimization

Using SIMP method, the element stiffness matrix is interpolated as,

$$k_e\left(\tilde{x}_e\right) = E_e\left(\tilde{x}_e\right) k_e^0 \tag{5.9}$$

where k_e^0 is given as,

$$k_e{}^0 = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} B^T C^0 B|J| d\bar{\Omega}_e \tag{5.10}$$

On substitution, we get,

$$\boldsymbol{K}_e = \sum_j \left(E_{\min} + \tilde{\rho}\left(\boldsymbol{x}_j^q\right)^p \left(E_0 - E_{\min}\right)\right) \boldsymbol{B}_j^T \boldsymbol{C}_0 \boldsymbol{B}_j \left|\boldsymbol{J}_j\right| \omega_j \tag{5.11}$$

The derivative of objective function

$$\frac{\partial \boldsymbol{K}_e}{\partial \rho_i} = \frac{\partial \boldsymbol{K}_e}{\partial \tilde{\rho}\left(\boldsymbol{x}_e^c\right)} \frac{\partial \tilde{\rho}\left(\boldsymbol{x}_e^c\right)}{\partial \rho_i} = p\left(\tilde{\rho}\left(\boldsymbol{x}_e^c\right)\right)^{p-1}\left(E_0 - E_{\min}\right) \boldsymbol{K}_e^0 \frac{\partial \tilde{\rho}\left(\boldsymbol{x}_e^c\right)}{\partial \rho_i} \tag{5.12}$$

The element density is given as,

$$\tilde{\rho}\left(x_e^c\right) = \sum_i R_i\left(x_e^c, y_e^c\right) x_i \tag{5.13}$$

The nodal displacements vector U(x) is the obtained from the equilibrium equation:

$$K(\tilde{x})U(\tilde{x}) = F \tag{5.14}$$

F is the vector of nodal forces and it is independent of the physical densities x.

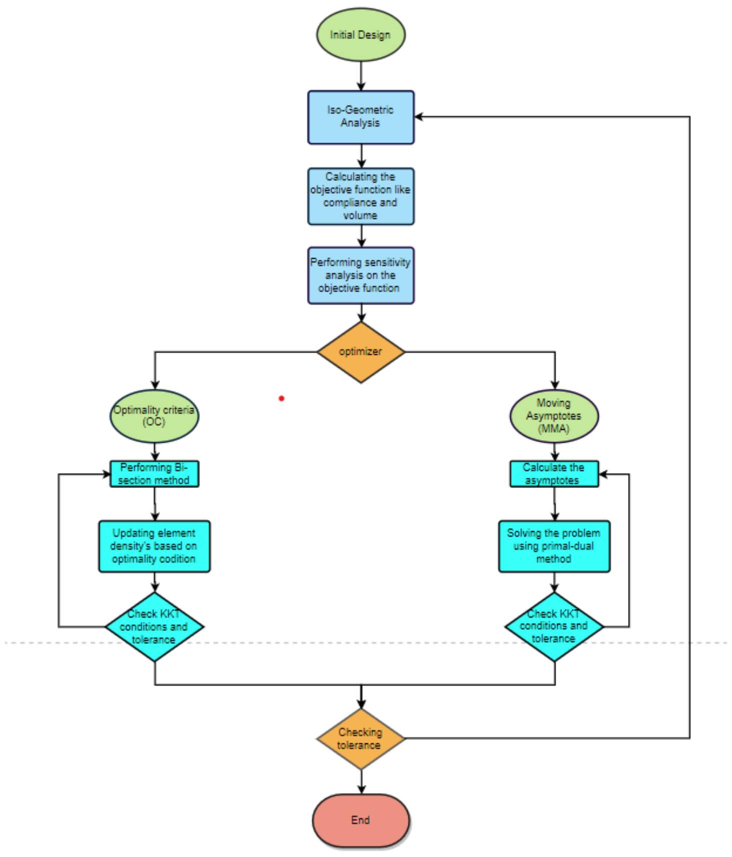

Initial Design
Iso-Geometric Analysis
Calculating the objective function like compliance and volume
Performing sensitivity analysis on the objective function
optimizer
Optimality criteria (OC)
Moving Asymptotes (MMA)
Performing Bi-section method
Calculate the asymptotes
Updating element density's based on optimality codition
Solving the problem using primal-dual method
Check KKT conditions and tolerance
Check KKT conditions and tolerance
Checking tolerance
End

Figure 12: Topology work flowchart

5.4 Post-processing

References

[1] P. S. Lockyer. *Controlling the interpolation of NURBS curves and surfaces.* PhD thesis, University of Birmingham, 2007.

[2] V. P. Nguyen, C. Anitescu, S. P. Bordas, and T. Rabczuk. Isogeometric analysis: An overview and computer implementation aspects. *Mathematics and Computers in Simulation*, 117:89–116, 2015.

[3] M. Okruta. Three-dimensional topology optimization of statically loaded porous and multi-phase structures. 2014.

[4] B. C. OWENS. *IMPLEMENTATION OF B-SPLINES IN A CONVENTIONAL FINITE ELEMENT FRAMEWORK.* PhD thesis, Texas A&M University, 2009.

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