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.-Ing. Martin Abendroth

Professor Incharge
Dr.-Ing Arun Prakash and Dr.-Ing Habil Geralf Hutter

March 25, 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 tools, basis splines (B splines) are the basis used to create elements like curves, lines and surfaces. Standard Finite element method uses Lagrangian function as their basics, what-if we use B splines or NURBS(Non-Uniform Rational B-Splines) as basics function for FEM which lead to the formulation of 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.[1]
- 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.[1]
- Due to strong coupling with geometry, optimization problem can be solved effectively. This lead us to use IGA for structural topology optimization.

1.2 Topology optimization using IGA

The main idea behind structural optimization is to get structures with minimum weight 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 as this method is quite effective for minimum weight topology. [2]

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. Based on the compliance and volume constrains, we used optimality criterion and Method of moving asymptotes to solve constrained based non-linear optimization problem. The optimization of structure is performed until the termination criteria is satisfied.

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 are used to get desired properties for interactive geometrical design.[1]. 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.[1].

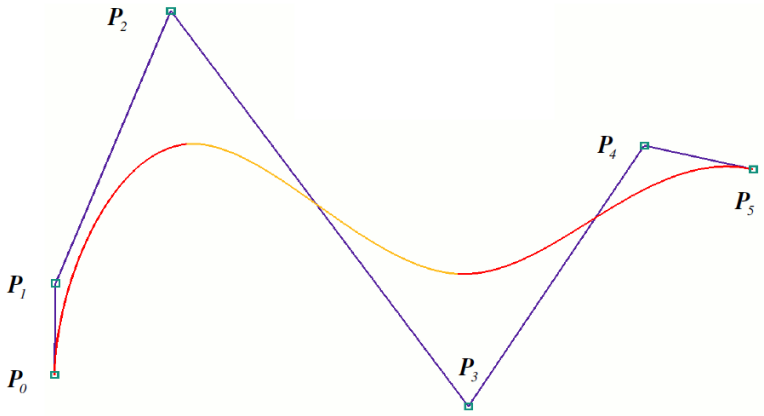


Figure 1: Cubic B-spline with 6 control points.[3]

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.

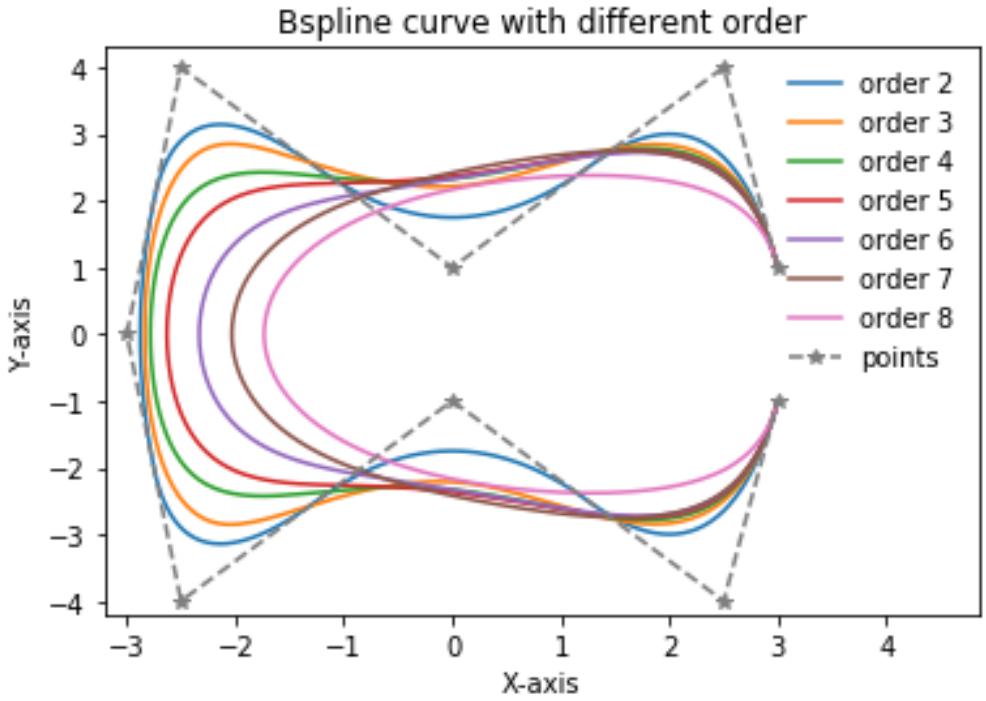


Figure 2: B-spline with 9 control points plotted for different degrees.

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 else clamped .As shown in Fig[3].

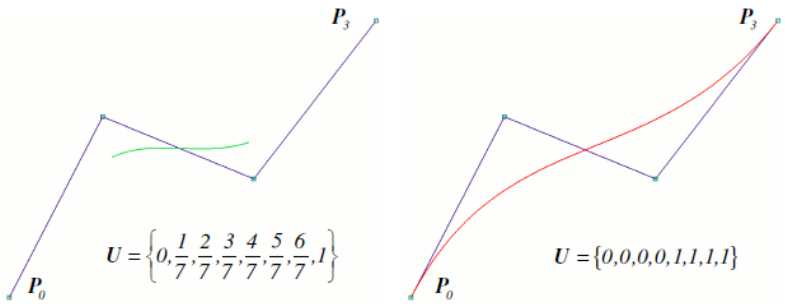


Figure 3: B-spline curve of open and clamped knot vector.[3]

```
def knot_vector(self, n, degree):
    '''
    This function return knotvector based on the number of control
        points and degree of Bspline
    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

    Test case
    ------
    test command -
                  pytest test_Inputs.py::test__knotvector_true
    '''
```

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 [4].

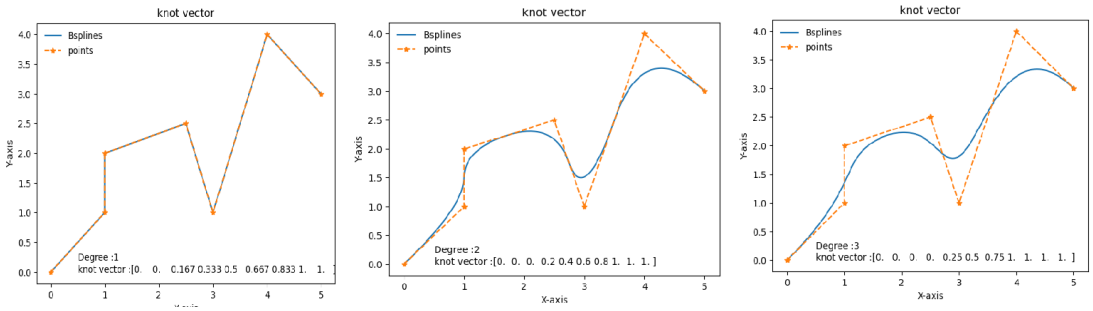


Figure 4: Knot vector and B-spline curve change due to change in degree.

2.4 B-spline Basis function

B-spline basis function is defined using recursive formula i.e Cox-de-Boor formulation, starting with the zeroth order basis function (p = 0).[1].

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

and for a polynomial order p greater then one

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

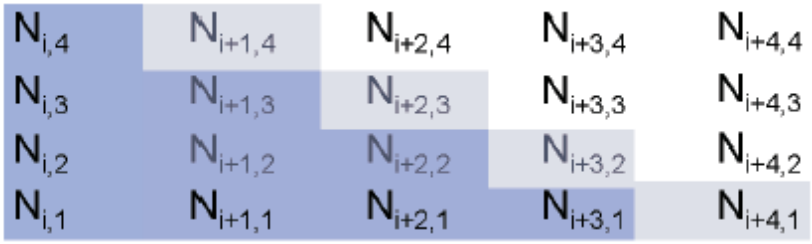


Figure 5: B-spline basis dependancy on other basis.[5]

B-splines function are not interpolatory i.e, value does not pass through them but the curves are approximated based on the values . 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
Implementation of equation 2.2 for the documentation

Generates Basis functions.
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 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.[4].

- 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].
      """
```

Properties of B-spline basis function derivatives Some properties of derivatives of B-spline basis function. The values for the test cases are obtained from the NURBS book.[4].

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

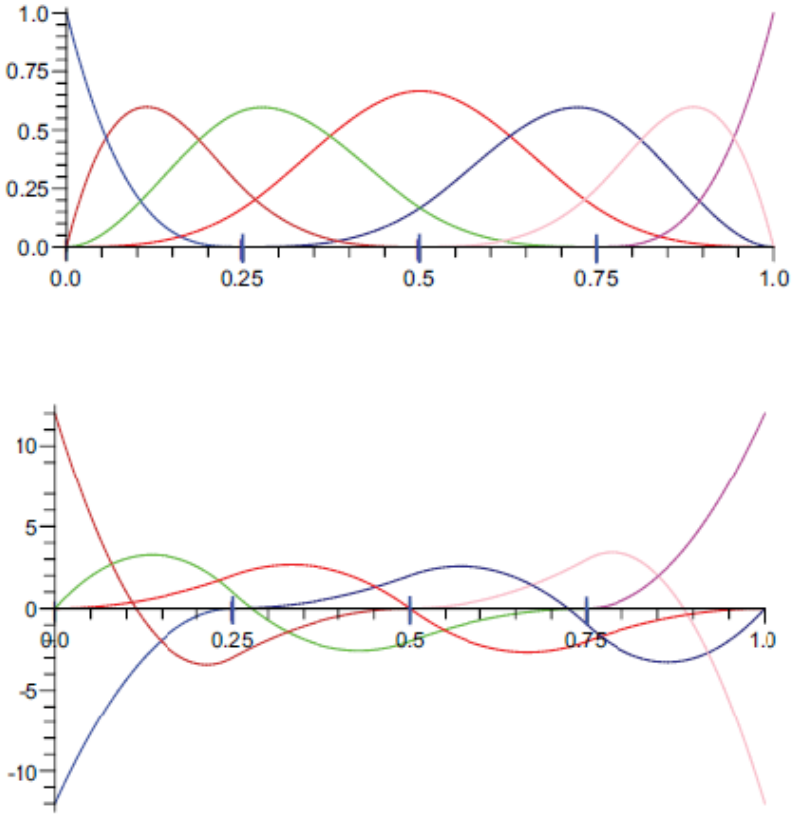


Figure 6: a)Cubic B-spline basis function b) Cubic B-spline derivative function .[4]

- 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.

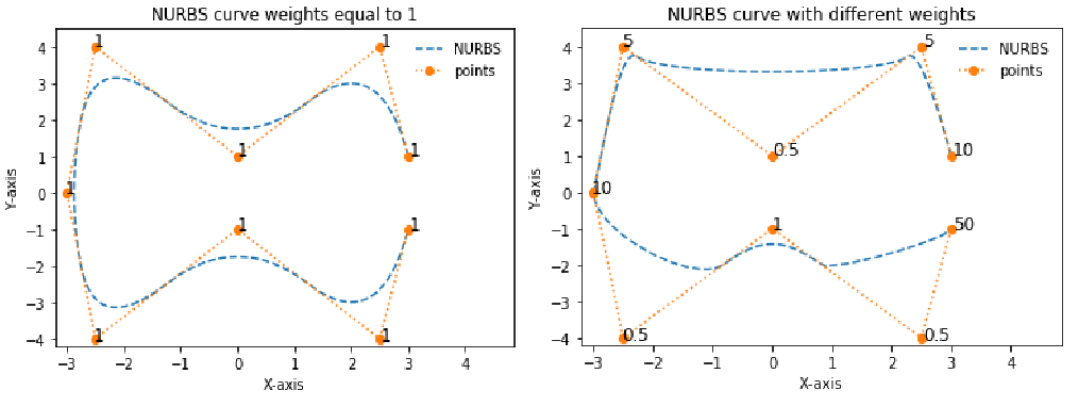


Figure 7: a) B-spline b) NURBS

3.1 NURBS basis function

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

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

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 [5]

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

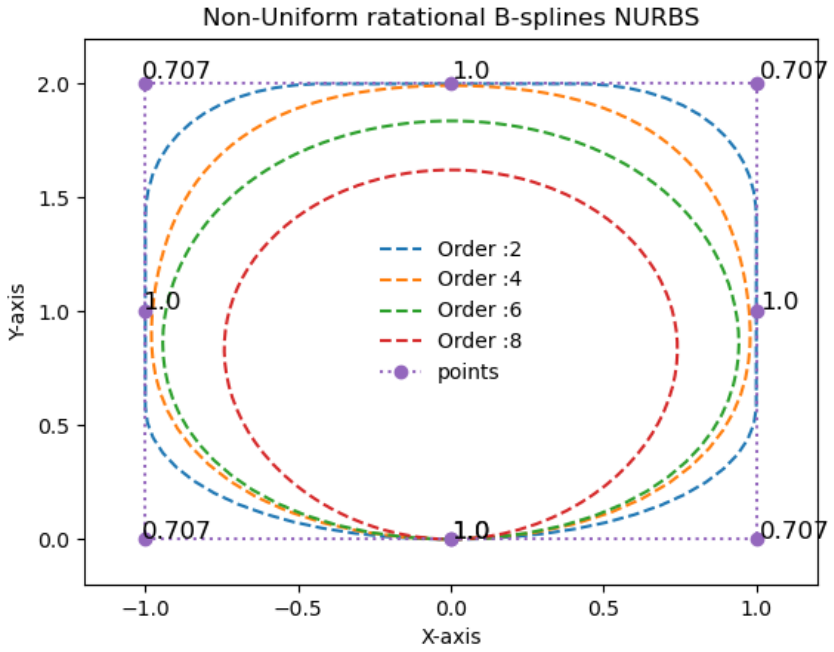


Figure 8: Circle from NURBS

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.

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

where the trivariant function is given as

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

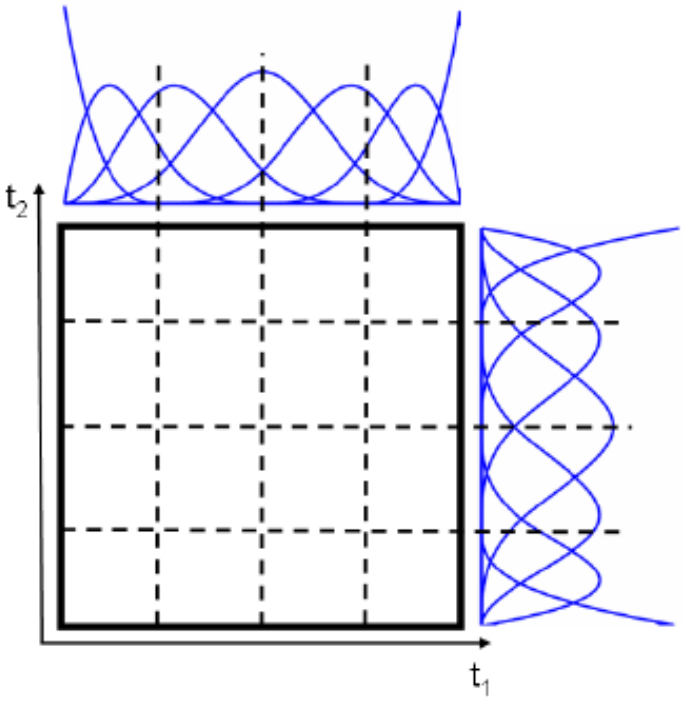


Figure 9: NURBS surface.[5]

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

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

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

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

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):
    '''
    NURBS volume which is a tensor product of NURBS basis along
        xi,eta,neta direction.
    Parameters
    ----------
    Ux,Uy,Uz : float
               knot index along xi,eta,neta direction
    weights : Array
               An array of NURBS weights defined for each knot.
    xdegree,ydegree,zdegree : int
                  degree of the knotvector along xi,eta,neta
                  0-constant, 1-linear, 2-quadratic, 3-cubic
    xknotvector,yknotvector,zknotvector : Array
                    knotvectors defined along xi,eta,neta
    Returns
    -------
    R - Array.
        Trivariant function
    dR_dx,dR_dy,dR_dz -Array
        Derivative of Trivariant function along xi,eta,neta direction.
    '''
```

The values for the test cases are obtained from the NURBS book[4].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

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

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.

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

Degree's of freedom vector qα is given as

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

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.

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

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

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

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

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

The force vector is expressed as

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

$$\sum_{e=1}^{nele} \left[\mathbf{K}_e \mathbf{U}_e = \mathbf{F}_e\right] \tag{4.17}$$

where K_e is Iso-geometric element's stiffness matrix, U_e is displacement vector and F_e force vector.

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

$$E_i = E_i\left(x_i\right) = x_i^p E_0, \quad x_i \in]\,0,1] \tag{4.18}$$

the element density is given as

$$x_i = \begin{cases} 1 & \text{if} \quad \mathbf{x} \in \Omega_s \\ 0 & \text{if} \quad \mathbf{x} \in \Omega \backslash \Omega_s \end{cases} \tag{4.19}$$

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

$$x_{r,s} = \sum_{i=0}^{n_1} \sum_{i=1}^{n_2} R_{i,j}(r,s) x_{ij}^p \tag{4.20}$$

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.

$$\mathrm{E}_i = \mathrm{E}_i\left(x_i\right) = \mathrm{E}_{\min} + x_i^p\left(\mathrm{E}_0 - \mathrm{E}_{\min}\right), \quad x_i \in [0,1] \tag{4.21}$$

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,

$$c(\overline{\mathbf{x}}) = \mathbf{F}^{\mathrm{T}}\mathbf{U}(\overline{\mathbf{x}}) \tag{4.22}$$

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:

$$\text{minimize} \ : c(x) = U^T K U = \sum_{e=1}^{N} E_e\left(x_e\right) u_e^T k_0 u_e \tag{4.23}$$

$$\text{subject to } : \begin{cases} \frac{V(x)}{V_0} = f \\ KU = F \\ 0 \leq x \leq 1 \end{cases} \tag{4.24}$$

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

$$\frac{\partial c(\overline{\mathbf{x}})}{\partial x_e} = \sum_{i \in N_e} \frac{\partial c(\overline{\mathbf{x}})}{\partial \tilde{x}_i} \frac{\partial \tilde{x}_i}{\partial x_e} \tag{4.25}$$

$$\begin{aligned} \frac{\partial c(\overline{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} &= \mathbf{F}^{\mathrm{T}} \frac{\partial \mathbf{U}(\overline{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} \\ &= \mathbf{U}(\overline{\mathbf{x}})^{\mathrm{T}} \mathbf{K}(\overline{\mathbf{x}}) \frac{\partial \mathbf{U}(\overline{\mathbf{x}})}{\partial \overline{\mathbf{x}}_i} \end{aligned} \tag{4.26}$$

$$\mathbf{K}(\tilde{\mathbf{x}})\mathbf{U}(\overline{\mathbf{x}}) = \mathbf{F} \tag{4.27}$$

The above equation(4.27) is derivatived with respect to element density.

$$\frac{\partial \mathbf{K}(\overline{\mathbf{x}})}{\partial \overline{\mathbf{x}}_i} \mathbf{U}(\overline{\mathbf{x}}) + \mathbf{K}(\overline{\mathbf{x}}) \frac{\partial \mathbf{U}(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} = \mathbf{0} \tag{4.28}$$

which yields,

$$\frac{\partial \mathbf{U}(\tilde{\mathbf{x}})}{\partial \overline{\mathbf{x}}_i} = -\mathbf{K}(\tilde{\mathbf{x}})^{-1} \frac{\partial \mathbf{K}(\overline{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} \mathbf{U}(\overline{\mathbf{x}}) \tag{4.29}$$

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

$$\frac{\partial c(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}_i} = -\mathbf{u}_i(\tilde{\mathbf{x}})^{\mathrm{T}} \left[\mid p\tilde{\mathrm{X}}_i^{p-1} \left(\mathrm{E}_0 - \mathrm{E}_{\min} \right) \mathbf{k}_i^0 \right] \mathbf{u}_i(\overline{\mathbf{x}}) \tag{4.30}$$

Volume constrain:

$$\frac{\partial v(\tilde{\mathbf{x}})}{\partial x_e} = \sum_{i \in N_e} \frac{\partial v(\overline{\mathbf{x}})}{\partial \tilde{x}_i} \frac{\partial \bar{x}_i}{\partial x_e} \tag{4.31}$$

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

4.3.3 Sensitivity Analysis

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

$$\bar{x}_e = \frac{\sum_{i \in N_c} H_{ei} x_i}{\sum_{i \in N_c} H_{ei}} \tag{4.32}$$

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

$$H_{ei} = \max\left(0, r_{\min} - \Delta(e,i)\right) \tag{4.33}$$

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:

$$\frac{dc}{dx_j} = \sum_{e \in N_j} \frac{1}{\sum_{i \in N_c} H_{ei}} H_{je} \frac{dc}{\bar{x}_e} \tag{4.34}$$

$$\frac{dV}{dx_j} = \sum_{\epsilon \in N_j} \frac{1}{\sum_{i \in N_e} H_{ei}} H_{je} \frac{dV}{\tilde{x}_e} \tag{4.35}$$

```
def Knearestneighbours(rmin,nelx,nely,nelz):
    '''
    The code has been taken from An efficient 3D topology optimization
        code written in Matlab section sensitivity analysis
    To perform sensitivity analysis, we require weight function.This
        functions provides the weight factor
    Parameters
    ----------
    rmin : int
        The minimum radius in which the elements are influenced.
    nelx,nely,nelz : int
        No of element in x ,y and z direction.

    Returns
    -------
    H : array
        A 2d array containing the weight factor of each element.
```

```
DH : array
    Sum of the weight factor (change in weight factors).
'''
```

4.3.4 Optimality criteria method

Optimality criteria is penalty based optimization method where weights are added to Lagrangian multiplier. To solve structural optimization problem using optimality criteria (OC) method, the constrains $0 \leq \mathrm{x} \leq 1$ is inactive. The convergence can able be achieved if the KKT condition are satisfied.[6]

$$\frac{\partial c(\overline{\mathbf{x}})}{\partial x_e} + \lambda \frac{\partial v(\overline{\mathbf{x}})}{\partial x_e} = 0 \tag{4.36}$$

The implementation of OC update scheme is given below.

$$x_e^{\text{new}} = \begin{cases} \max\left(0, x_e - m\right), & \text{if } x_e B_e^{\eta} \leq \max\left(0, x_e - m\right) \\ \min\left(1, x_e + m\right), & \text{if } x_e B_e^{\eta} \geq \min\left(1, x_e - m\right) \\ x_e B_e^{\eta}, & \text{otherwise} \end{cases} \tag{4.37}$$

where m is move limit and η is numerical damping coefficient and optimality condition B_e is obtained from equation(4.38).

$$B_e = -\frac{\partial c(\overline{\mathbf{x}})}{\partial x_e} \left(\lambda \frac{\partial v(\overline{\mathbf{x}})}{\partial x_e} \right)^{-1} \tag{4.38}$$

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.

```
def
    optimality_criteria(dfun,dCon,initial_X,constrain,residual=None,H=None,DH=None,
oc_disp=True,g=1,tol=0.01,move=0.1,neta=0.5,initial_value=0,final_value=1e09):
    '''
Optimality criteria is a penalty based method to solve constrain based
    problem.
    Parameters
    ----------
    dfun : array
        derivatives of the function which are to be optimized.
    dCon : array
        derivatives of the constrains .
    initial_X : array
        initial set of values which are required to solve the problem.
    constrain : array
              set of constrains.
    H : array
        weight factor required for sensitivity analysis.
    DH : array
        change in weight factor.
    oc_disp : bool, optional
        If TRUE the values at each iteration is printed. The default is
            True.
    g : float, optional
        penality value to remove gray scale values. The default is 1.
    tol : float, optional
        Tolerance . The default is 0.01.
    move : float, optional
        the step which has to be taken while find the optimization. The
            default is 0.1.
    neta : float, optional
        Numerical damping co-efficient. The default is 0.5.
    initial_value : int, optional
        The initial range to find Lagrangian multiplier. The default is
            0.
    final_value : TYPE, optional
        The final range to find Lagrangian multiplier. The default is
            1e09.
    Returns
    -------
    X_new : array
        The optimized values obtained after optimization.
```

```
'''
```

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

```
(12) pytest
   test_optimization.py::test__optimality_criteria_simple_function

(13) 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 sub problem is given as [7]

$$\text{minimize} \quad \tilde{f}_0^{(k)}(\mathrm{x}) + a_0 z + \sum_{i=1}^{m} \left(c_i y_i + \frac{1}{2} d_i y_i^2 \right) \tag{4.39}$$

An approximate function is build to satisfy the above problem

$$\tilde{f}_i^{(k)}(\mathrm{x}) = \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j} + \frac{q_{ij}^{(k)}}{x_j - l_j^{(k)}} \right) + r_i^{(k)}, \quad i = 0, 1, \ldots, m \tag{4.40}$$

where p_{ij} and q_{ij} are given as

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

$$q_{ij}^{(k)} = \left(x_j^{(k)} - l_j^{(k)} \right)^2 \left(0.001 \left(\frac{\partial f_i}{\partial x_j} \left(\mathrm{x}^{(k)} \right) \right)^+ + 1.001 \left(\frac{\partial f_i}{\partial x_j} \left(\mathrm{x}^{(k)} \right) \right)^- + \frac{10^{-5}}{x_j^{\max} - x_j^{\min}} \right) \tag{4.42}$$

$$r_i^{(k)} = f_i \left(\mathrm{x}^{(k)} \right) - \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j^{(k)}} + \frac{q_{ij}^{(k)}}{x_j^{(k)} - l_j^{(k)}} \right) \tag{4.43}$$

The lower and upper bound are formulated as below

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

$$\beta_j^{(k)} = \min\left\{x_j^{\max}, u_j^{(k)} - 0.1\left(u_j^{(k)} - x_j^{(k)}\right), \quad x_j^{(k)} + 0.5\left(x_j^{\max} - x_j^{\min}\right)\right\} \tag{4.45}$$

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

$$\begin{aligned} l_j^{(k)} &= x_j^{(k)} - 0.5\left(x_j^{\max} - x_j^{\min}\right) \\ u_j^{(k)} &= x_j^{(k)} + 0.5\left(x_j^{\max} - x_j^{\min}\right) \end{aligned} \tag{4.46}$$

For iteration greater then 3:

$$\begin{aligned} l_j^{(k)} &= x_j^{(k)} - \gamma_j^{(k)}\left(x_j^{(k-1)} - l_j^{(k-1)}\right) \\ u_j^{(k)} &= x_j^{(k)} + \gamma_j^{(k)}\left(u_j^{(k-1)} - x_j^{(k-1)}\right), \end{aligned} \tag{4.47}$$

where γ is

$$\gamma_j^{(k)} = \begin{cases} 0.7 & \text{if } \left(x_j^{(k)} - x_j^{(k-1)}\right)\left(x_j^{(k-1)} - x_j^{(k-2)}\right) < 0 \\ 1.2 & \text{if } \left(x_j^{(k)} - x_j^{(k-1)}\right)\left(x_j^{(k-1)} - x_j^{(k-2)}\right) > 0 \\ 1 & \text{if } \left(x_j^{(k)} - x_j^{(k-1)}\right)\left(x_j^{(k-1)} - x_j^{(k-2)}\right) = 0 \end{cases} \tag{4.48}$$

MMA ALGORTHIM

1. Choose an initial design x_0 and other objective function for initial design
2. Check for convergence using do while i.e change less then 0.01 (x_k-x_{k-1})
   a) Based on iteration the lower and upper bounds are calculated.
   b) Update lower and upper asymptotes.
   c) calculate the derivatives.
   d) solve the sub problem using primal-dual method. We obtained updated element density x_k.
   e) store previous two iteration values x_{k-2},x_{k-1} which would be the input for next iteration.

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(4.49) To satisfy constrained problem, we build a Lagrangian function.

$$\begin{aligned} L =& \psi(x,\lambda) + (a_0 - \zeta)\, z + \sum_{j=1}^{n} \left(\xi_j \left(\alpha_j - x_j\right) + \eta_j \left(x_j - \beta_j\right)\right) + \\ &+ \sum_{i=1}^{m} \left(c_i y_i + \frac{1}{2} d_i y_i^2 - \lambda_i a_i z - \lambda_i y_i - \lambda_i b_i - \mu_i y_i \right) \end{aligned} \tag{4.49}$$

```
def Moving_asymptoes(dfun0,dcon,f0,c0,x0,x1,x2,L,U,loop,nel,vel,
Xmin,Xmax,ma_disp,m=0.2,m_tol=0.1):
    '''
    Based on the MMA paper by Krister Svanberg whose formulation are
        used to build this function
    MMA and GMMA by Krister Svanberg
    Optimization and Systems Theory,
    KTH, Stockholm, Sweden.
    The equation are provided in the paper and are referred in this
        function accordingly.
    This is the main function which performs active set strategy based
        constrained problem optimization (KKT condition are satisfied
        using Newton raphson)
    Parameters
    ----------
    dfun0 : array
        Contains the derivatives of the function.
    dcon : array
        Contains the derivatives of the constrains.
    f0 : array
        inital value of the function for value x0.
    c0 : array
        initial values of the constraind for values x0.
    x0 : array
        Initial guess values and later the current iteration values of X.
    x1 : array
        X values from previous iteration (k-1).
    x2 : array
        X values from previous iteration (k-2).
```

```
L : array
    Lower limit values.
U : array
    Upper limit values.
loop : int
    Current iteration number.
nel : int
    Number of variables in the function. [x1,x2,x3,...]
vel : int
    Number of constrains.
Xmin : array
    The minimum values of X.
Xmax : array
    The maximum value of X.
ma_disp : bool
    If TRUE , prints the values.
m : float, optional
    The length of the step movement. The default is 0.2.
m_tol : bool, optional
    Tolerance for MMA. The default is 0.1.

Returns
-------
x  : array
       Optimized values of the array which satisfys KKT condition
Lower : array
          The lower limit for the values of X
Upper : array
       Upper limit for the values of X
Test case
---------
Test command -
         pytest test_optimization.py::test__MMA_literature_equation

'''
```

Test command to check the implementation of MMA with quadratic equation and KKT condition provided in literature.[7]

```
(14) 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 formulated to be use iso-parametric method.Similar to FEM, IGA can be divided into sub programs

- Pre-Processing
- Processing
- Post-Processing

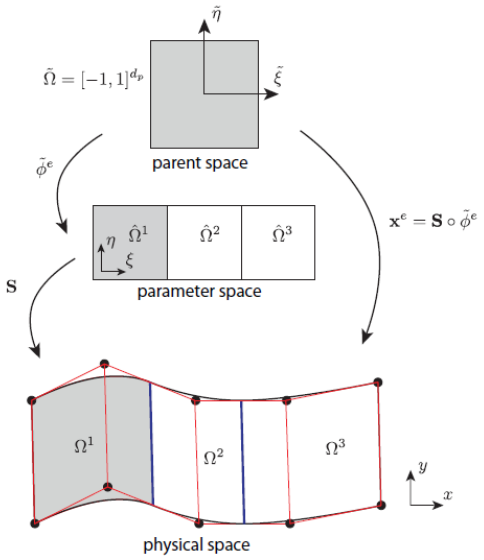


Figure 10: Relevant spaces.[1]

5.1 Relevant spaces

In FEM, there are parent and unit space. The core concepts in IGA is 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]. To implement numerical integration such as gauss quadrature this space is required.

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 stage, the input parameters have to be defined based on which the following parameters for IGA are formulated.

5.2.1 Generation of NURBS parameters

Based on 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). The NURBS parameters like control points and knot vectors and knot spans to represent geometry are generated.

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.

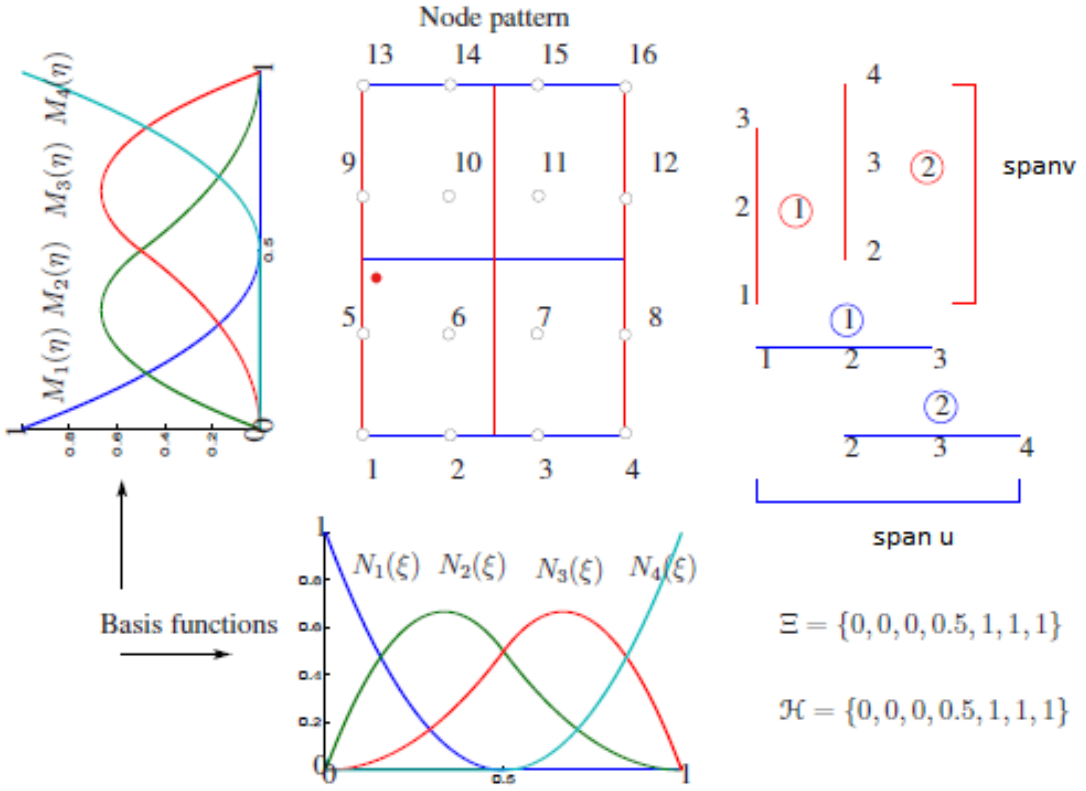


Figure 11: Bi-quadratic NURBS used to map a 2d surface,can be extended to 3d by adding Tri-quadratic NURBS and span W.[1]

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

(16) 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 using 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.

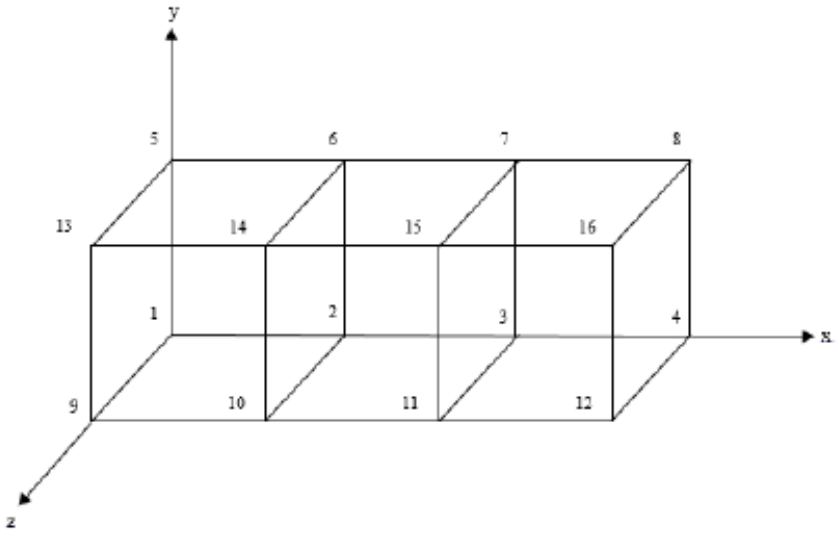


Figure 12: Control point assembly and node numbering

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[12].

```
def controlpointassembly(nx,ny,nz,nU,nV,nW,xdegree,ydegree,zdegree,
knotconnectivityU,knotconnectivityV,knotconnectivityW):
    '''
    Based on Isogeometric analysis: an overview and computer
        implementation aspects paper page - 50,51
    IGFEM 6.1. Data structure- Listing 2: - Which is implemented for a
        2D element extended it for 3D elements.
```

```
This function generates a control point assembly matrix used in
    element routine
EX:(nx,ny,nz):(2,1,1) degree:(2,1,1)
    front face numbering
    3--4--5
    |     |
    0--1--2

    back face numbering
    9--10--11
    |      |
    6--7--8
Parameters
----------
nx,ny,nz : int
        Number of element in x,y and z direction.
nU,nV,nW : int
           length of knot vector along xi,eta,neta direction.
xdegree,ydegree,zdegree : int
                         degree of the knotvector along xi,eta,
                          0-constant, 1-linear, 2-quadratic, 3-cubic
knotconnectivityU,knotconnectivityV,knotconnectivityW : Array
  An array which contains the length of each element and number of
     knot in it.
     Ex-knotvector:[0,1,2,3,4,5,6]
Returns
-------
elements_assembly : array
                  2d matrix which contains the ordering of the
                      element based on the degree and knot span.
                  EX:[[0,1,2,3,4,5,6,7,8,9,10,11]]
'''
#Inititation of dimentions for 2D,3D matrices
elements_assembly=np.zeros(((nU*nV*nW),(xdegree+1)*(ydegree+1)*(zdegree+1)))
elements_order=elementorder(nx,ny,nz)
a=0
#looped over the length of the knot in xi,eta,neta direction : No-of
    elements
for i in range(nW):
    for j in range(nV):
        for k in range(nU):
            c=0
            #looped over knot span in xi,eta,neta direction : No-of
```

```
                    knots within each element
                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]]
#element index are generated based on knot index and element order
                            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

```
(17) pytest test_geometry.py::test__single_element_Assembly

(18) pytest test_geometry.py::test__element_Assembly_C0_continuity

(19) 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 class is used to perform switch cases funtionality to implement
        boundary conditions.
    Option :
```

```
        0- Cantilever beam with load along the bottom edge of the
           free end.
        1- Simple supported with load at bottom center.
        2- Cantilever beam with load at corner point of the free end.
        3- Cantilever beam with point load at the free end (2d case
           loading at y=height and x=length).
        4- Cantilever beam with two forces at either end
        5- Cantilever beam with load along the center of the free end
    '''
    def __init__(self,CONTROL_POINTS,length,height,width,bc_disp):
```

5.3 Processing

We need to compute global stiffness matrix $\mathbf{K}$ and global force vector $\mathbf{F}$. 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

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

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.

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

```
def
    jacobian(Xi,Eta,Nta,Uspan,Vspan,Wspan,X,Y,Z,weights,xdegree,xknot_vector,ydegree,
yknot_vector,zdegree,zknot_vector):
    '''
    This function creates jacobian which maps from global to parameter
        space and parameter to unit space.
    Parameters
    ----------
    Xi,Eta,Nta : float
                  Values obtained from unittoparametric function along
                     xi,eta,neta direction.
    Uspan,Vspan,Wspan : array
                       A 2d array containing the knot values of the
                          element.
    X,Y,Z : array
               A 2d array containing the control point co-ordinates of
                  x,y and z direction.
    weights : array
            An array which contains the weights of each control point
               (NURBS)
    xdegree,ydegree,zdegree : int
                  degree of the knotvector along xi,eta,neta
                  0-constant, 1-linear, 2-quadratic, 3-cubic
    xknot_vector,yknot_vector,zknot_vector : array
                   knot vector along xi,eta,neta direction.

    Returns
```

```
-------
jacobian1 : array
              A jacobian matrix which maps physical space to
                 parameteric space
det_jacobian1 : float
              determinant of jacobian1 (physical space to
                 parameteric space)
det_jacobian2 : float
                 determinant of jacobian 2 (parameteric space to
                    unit space)
nurbs : array
           Trivariant function obtained from trilinear derivative
              function
'''
```

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

---

```
(20) pytest test_element_routine.py::test__unit_to_parametric_space_true

(21) 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.

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

Iso-geometric analysis 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 $\mathbf{K}_e : \mathbf{K}(\mathrm{sctr}\,B, \mathrm{sctr}\,B) = \mathbf{K}(\mathrm{sctr}\,B, \mathrm{sctr}\,B) + \mathbf{K}_e$.
2. End loop over elements.
3. Calculate nodal displacements U from 5.14.

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.

```
(22) pytest test_element_routine.py::test__gauss_quadrature_1point_true

(23) 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,

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

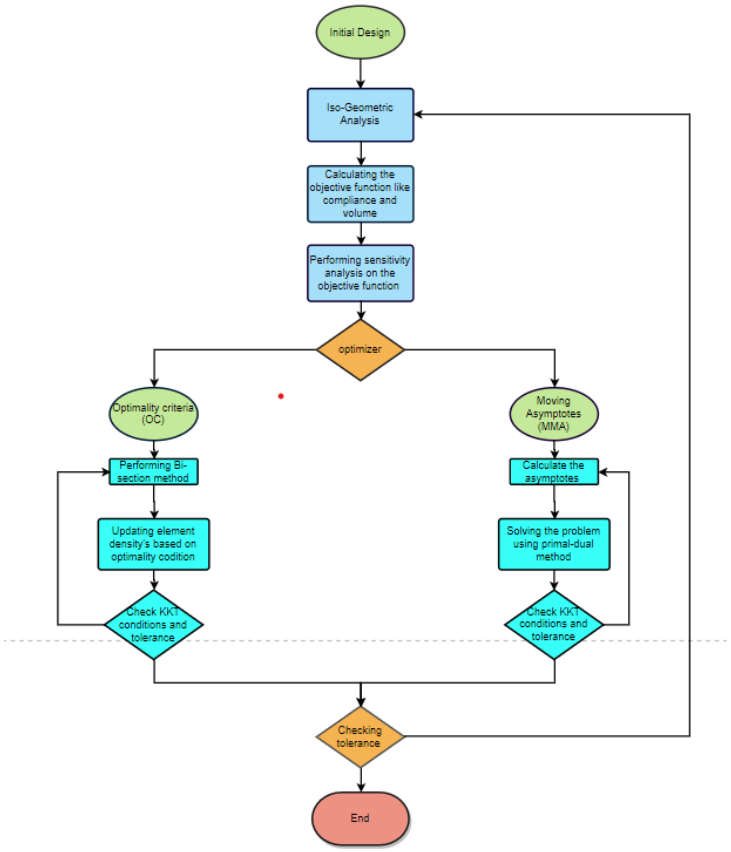


Figure 13: Topology work flowchart

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.

5.4 Post-processing

5.4.1 Mapping displacements

The nodal displacement obtained from equ(5.14) can be mapped directly to the respective nodes to get the deformation of structure.

$$[\mathbf{Newcontrolpoints}] = [\mathbf{Controlpoints}] + [\mathbf{U}] \tag{5.15}$$

5.4.2 Mapping Strains and Stresses

In Iso-geometric analysis, the strains and stresses are interpolated at the gauss points because of this the mapping requires a complex procedure which depends on the size of each knot vector and span of knot vector (element size).

$$\epsilon_i = B_{ij}u_j \tag{5.16}$$

$$\sigma_i = C_{ij}\epsilon_j = C_{ij}B_{jk}u_k \tag{5.17}$$

The above equation are looped over gauss points. Below python function performs that

---

```
def stress_strain_element_routine(X,Y,Z,weights,displacements,E,v,
Uspan,Vspan,Wspan,xdegree,xknot_vector,ydegree,
yknot_vector,zdegree,zknot_vector):
    '''
    This function builds stress and strains at gauss quadrature based on
        the displacement
    Parameters
    ----------
    X,Y,Z : array
              A 2d array containing the control point co-ordinates of
                 x,y and z direction.
    weights : array
```

```
            An array which contains the weights of each control point
                (NURBS)
Uspan,Vspan,Wspan : array
                    A 2d array containing the knot values of the
                        element.
E :int
        Young's modulus
v : float
        poission's ratio
xdegree,ydegree,zdegree : int
                            degree of the knotvector along xi,eta,neta
                                0-constant, 1-linear, 2-quadratic,
                                    3-cubic
xknot_vector,yknot_vector,zknot_vector : array
                                        knot vector along xi,eta,neta
                                            direction.
Returns
-------
gp_strain : array
           element strains at gauss quadrature
gp_stress : array
           element stress at gauss quadrature
R : array
          Trivariant basis functions mapped according to equ. 4.8
             obtained from jacobian
B : array
        strain displacement matrix obtained from jacobian
'''
```

5.4.3 Mapping element densities (Topology optimization)

The element density x_i is an array and based on the element number obtained from element order. The element density array is mapped into 3D array.

5.5 Visualization

Scientific visualization is a branch which deal with visualization of data along with user interface.

5.5.1 VTK (Visualization Tool-kit)

VTK is a open source library used for scientific visualization. This tool can be used to plot topology optimization which is based on volume analysis. For using VTK, we need to generate vtk readable file which contains the data in VTK format. This is achieved by using python library PYEVTK which is used to generate structured or unstructured VTK files from control point and physical parameters. Below is the python function to generate structured VTK files

---

```
def VTK(CONTROL_POINTS,element_density,nx,ny,nz,file_name):
    '''
    For visualization of the mesh, we generate a VTK file containing all
        the information
    Parameters
    ----------
    CONTROL_POINTS : array
                      An array of co-ordinates of control points.
    element_density : array
                      A 1D array conatining the element density obtained
                          after topology optimization.
    nx,ny,nz : int
                Number of element along x,y,z direction
    file_name : str
                   path and name of the output VTK file
    Returns
    -------
           A VTK file is generated with given file name at the given
               path
    '''
    width1=120
    print('%' * width1)
    print('\n Creating VTK file ')
    from pyevtk.hl import gridToVTK
    from numpy import array,unique,reshape

    x = array(unique(CONTROL_POINTS[:, 0]), dtype='float64')
    y = array(unique(CONTROL_POINTS[:, 1]), dtype='float64')
    z = array(unique(CONTROL_POINTS[:, 2]), dtype='float64')
    volume = (element_density.reshape((nx, ny, nz), order='F'))
    #print(volume)
    path='./'+file_name
    #print('"'+path+'"')
```

```
gridToVTK("./cantilever_beam", x, y, z, cellData={"Volume1": volume})
print('\n VTK file generated \n')
print('%' * width1)
pass
```

5.5.2 Visualization of deformed structure and topology

Visualization of VTK file can be done using paraview or open source VTK library pyvista.

1. **PARAVIEW**

Paraview is an open-source software, built on top of VTK with GUI. We can build visualization to analyze data using qualitative and quantitative techniques.
Procedure :

1. Open Paraview.
2. select the vtk file, the name of the file would appear in pipeline browser.
3. Change the visibility but clicking on the eye.
4. Click on Display panel and change the representation to SURFACE WITH EDGES and colouring option to VOLUME Fig(15).

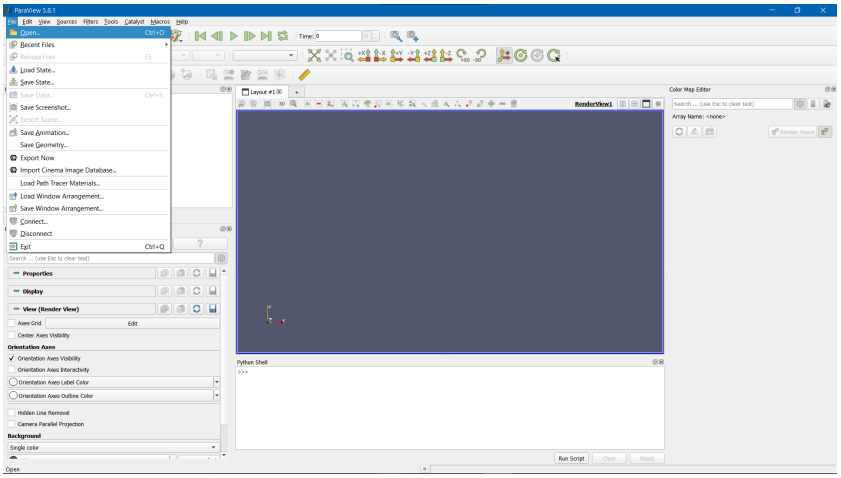


Figure 14: Paraview- Selection of file

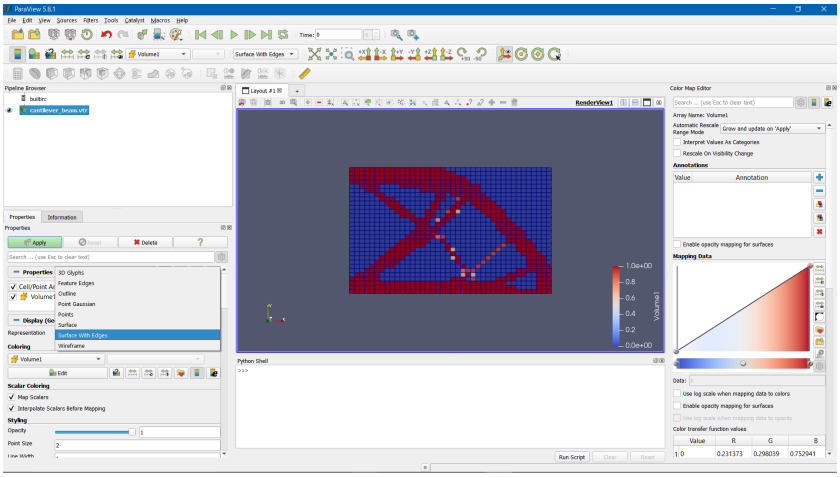


Figure 15: Paraview- Changing the options

2. **PYVISTA**

A open-source python library used for scientific visualization. It can be used to plot VTK file. The main advantage of PYVISTA over PARAVIEW is not need of generating a VTK file for visualization and can be plotted similar to Mathplotlib library.

Python function based on pyvista to plot 3D structures.

To plot deformed structure.

```
def mesh_vis(CP,U,nx,ny,nz,optimizer):
    '''
    Function is built on pyvista to visulaize VTK files
    Parameters
    ----------
    CP : array
        Control point co-ordinates.
    U : array
            Displacements obtained due to loading
    nx,ny,nz : int
              No of elements along x,y,z direction.
    optimizer : str
                  Name of the optimizer used for Topology optimization.
    Returns
    -------
    deformed and undeforemd structure is plotted and save in results
        folder based on loop number and optimizer.
    '''
```

To plot the modified Topology of the structure.

```
def element_density_slided1(i,CP,nx,ny,nz,element_density,optimizer):
    '''
    Function is built on pyvista to visulaize VTK files
    Parameters
    ----------
    i : int
        Loop number used in saving the plot name.
    CP : array
        Control point co-ordinates.
    nx,ny,nz : int
               No of elements along x,y,z direction.
    element_density : array
                      An 1D array containing the volume of the optimized
                          structure.
    optimizer : str
                  Name of the optimizer used for Topology optimization.
    Returns
    -------
    A 3D plot with XY ,ZX, YZ projection are plotted and save in results
        folder based on loop number and optimizer.
    '''
```

6 Testing and Verification

To run all the test cases, copy all test files in same folder and enter PYTEST command on the terminal.

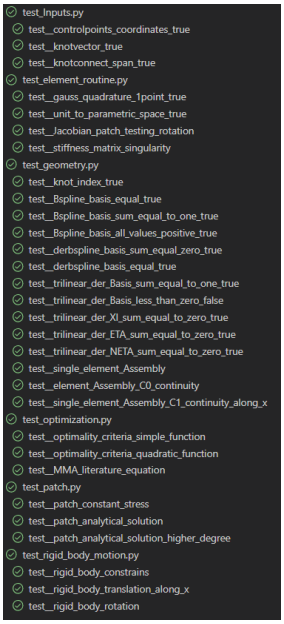


Figure 16: List of test cases

6.1 Unit Testing

Unit testing is performed to detect any error in the code. We have perform functional unit testing i.e each function is test with expected values.
To perform unit testing in python, we used a python testing module **PYTEST**.

1. test_Inputs.py : Test cases for input parameters
2. test_geometry.py : Test cases on shape function and assembly
3. test_element_routine.py : Test cases on integration scheme and sanity checks on other element routine functions.

A detailed information is provided for each test cases in the code.Test command to run the test cases.

```
pytest test_Inputs.py
pytest test_geometry.py
pytest test_element_routine.py
```

6.2 Functional testing

To find the optimal values for given parameter, we have used optimizers like OC and MMA to obtain minima for given constrains.OC is penalty based method and MMA is based on active set strategy to solve KKT condition.

6.2.1 Optimality Criterion (OC)

In this method we used weighted penalty parameters and the volume constrain is given as sum of each element should be equal to one.

Verification-1 OC used to solve a linear function with following constrains given below

$$f(x1, x2) = x1 + x2 - 1 = 0 \tag{6.1}$$

$$\sum_{i=0}^{N} x_i = 0 \text{ and } \quad 0 <= x_i <= 1 \tag{6.2}$$

Expected Output=[0.5,0.5]
Command to run the test case.

```
pytest test_optimizers:: test__optimality_criteria_simple_function
```

Result Test case passed.

Verification-2 OC used to solve a quadratic function with following constrains given below

$$f(x1, x2) = (\text{x0} + \text{x1})\text{^}2 - 2 * (\text{x0} - \text{x1}) \tag{6.3}$$

$$\sum_{i=0}^{N} x_i = 0 \text{ and } \quad 0 <= x <= 1 \tag{6.4}$$

Expected Output=[0,1]
Command to run the test case.

```
pytest test_optimizers:: test__optimality_criteria_quadratic_function
```

Result Test case passed.

6.2.2 Method of Moving Asymptotes (MMA)

MMA is a based on active set strategy method and to solve the problem gradient based method are used which requires some initial guess.

Verification MMA used to solve a quadratic function with following constrains given below. The objective functions and constrain obtained from literature [7]

$$f(x) = \mathrm{x(1)\hat{}2 + x(2)\hat{}2 + x(3)\hat{}2} \tag{6.5}$$

$$\begin{aligned} \text{subject to} & (\mathrm{x(1)} - 5)\mathrm{\hat{}2} + (\mathrm{x(2)} - 2)\mathrm{\hat{}2} + (\mathrm{x(3)} - 1) \wedge 2 =< 9 \\ & (\mathrm{x(1)} - 3)\mathrm{\hat{}2} + (\mathrm{x(2)} - 4)\mathrm{\hat{}2} + (\mathrm{x(3)} - 3) \wedge 2 =< 9 \\ & 0 =< \mathrm{x}(j) =< 5, \text{ for j} = 1, 2, 3. \end{aligned} \tag{6.6}$$

Expected Output=[2.0175, 1.7800, 1.237]
Command to run the test case.

```
pytest test_optimizers:: test__MMA_literature_equation
```

Result Test case passed.

6.3 Sanity checks and Patch Testing

All the required parameter are provided within the test case, only the test command is required to run the test case.

6.3.1 Sanity check for Jacobian

Aim : Jacobian is volume conserving and rotating Jacobian matrix should change it's value.[8]
Expected Output : The determinant of Jacobian should not change even after rotation.
Test command : pytest test_element_routine.py : : test__Jacobian_patch_testing_rotation.
Result : Test case passed.

6.3.2 Sanity check for Stiffness matrix

Aim : The stiffness matrix has to be positive definite.[8]
Expected Output : The eigenvalues of the global stiffness matrix should not be

negative.
Test command : pytest test_element_routine.py : : test__stiffness_matrix_singularity
Result : Test case passed.

6.3.3 Eigenvalue Stiffness Test

Aim : We check rigid body motion for constrained and unconstrained global stiffness matrix [8]
Expected Output : The number of zero eigenvalues from global stiffness matrix represent the rigid body motion of the structure.
unconstrained - 6 zeros eigenvalues (3-rotation, 3-translation)
constrained - 0 zeros eigenvalues
Test command : pytest test_rigid_body_motion.py : : test__rigid_body_constrains
Result : Test case passed.

6.3.4 Rigid body Translation test

We take a 3 x 3 x 3 structure with interior nodes, we apply displacement boundary condition on all exterior nodes and calculate the residual force. From the residual forces , the interior node displacements are calculated .[8]
Aim : Rigid body translation along x direction-global stiffness matrix should be able to represent rigid body translation.
Expected Output : The displacement of the external and internal nodes should be same.
Test command : pytest test_rigid_body_motion.py : : test__rigid_body_translation_along_x
Result : Test case passed.

6.3.5 Rigid body Rotation test

Similar to rigid body translation but we use Euler Bunge transformation matrix.[8]

$$\mathbf{R} = \begin{bmatrix} \cos(\beta)\cos(\gamma) & -\cos(\beta)\sin(\gamma) & \sin(\beta) \\ \cos(a)\sin(\gamma)+\sin(a)\sin(\beta)\cos(\gamma) & \cos(a)\cos(\gamma)-\sin(a)\sin(\beta)\sin(\gamma) & -\sin(a)\cos(\beta) \\ \sin(a)\sin(\gamma)-\cos(a)\sin(\beta)\cos(\gamma) & \sin(a)\cos(\gamma)+\cos(a)\sin(\beta)\sin(\gamma) & \cos(a)\cos(\beta) \end{bmatrix} \tag{6.7}$$

where α, β, γ are the respective angles with co-ordinate axis.
The external nodes are rotation and internal nodes displacement are calculated.
Aim : Rigid body rotation - global stiffness matrix should be able to represent rigid body rotation .

Expected Output : The displacement of the internal node should match the coordinates obtained after performing rotation using equ.(6.7) to reference system.
Test command : pytest test_rigid_body_motion.py : : test__rigid_body_rotation
Result : Test case passed.

6.3.6 Single element patch test

Aim : Check the strains and stresses generated for single element under tensile loading.[8]
Expected Output : The stress generated within the element should be same at each gauss point 2x2x2 (8 gauss points).
Test command : pytest test_rigid_body_motion.py : : test__patch_constant_stress
Result : Test case passed.

6.3.7 Analytical solution Patch test

Analytical solution for 2d cantilever beam:

$$u_x = -\frac{P}{6EI}\left[(6L-3x)x + (2+v)\left[y^2 - \frac{D^2}{4}\right]\right] \tag{6.8}$$

$$U_y = \frac{P}{6EI}\left[3vy^2\langle L-x) + (4+5v)\frac{D^2x}{4} + (3L-x)x^2\right] \tag{6.9}$$

Aim : The deflection of cantilever beam with point load at the free end.
Expected Output : The maximum deflection obtained from analytical solution(6.9) should match the numerical solution
Test command : pytest test_rigid_body_motion.py : : test__patch_analytical_solution
Result : Test case passed.

6.3.8 Analytical solution Higher Order patch test

Aim : The deflection of cantilever beam with point load at the free end using higher order shape function.
Expected Output : The maximum deflection obtained from analytical solution (6.8) should match the numerical solution.
Test command :
pytest test_rigid_body_motion.py : : test__patch_analytical_solution_higher_degree
Result : Test case passed. [4]

7 User manual

The following steps should be performed to run the program and test cases. All the files are written in python.

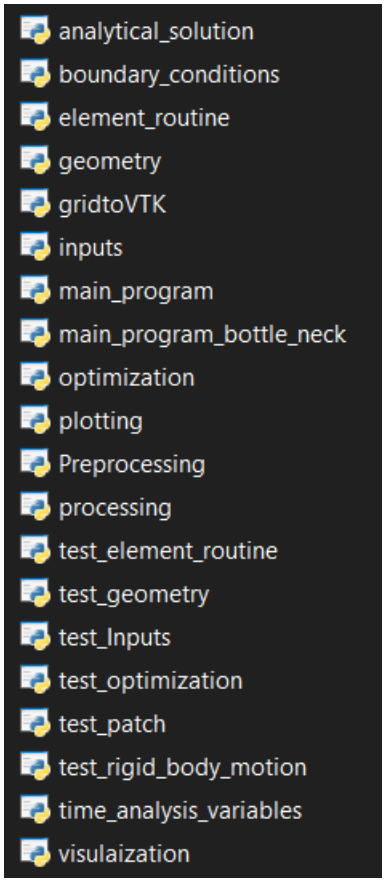


Figure 17: List of all python files

7.1 Procedure to run the program

All python files have to be placed in same folder and working directory has to be same to run the python program. Any python environment can be used to run this python code.

1. A file name main_program.py is the starting point of the IGTO program. command: **python main_program.py**

(a) Required python external libraries like 'numpy', 'matplotlib', 'pyvista', 'pyEVTK','pytest' are checked and installed.

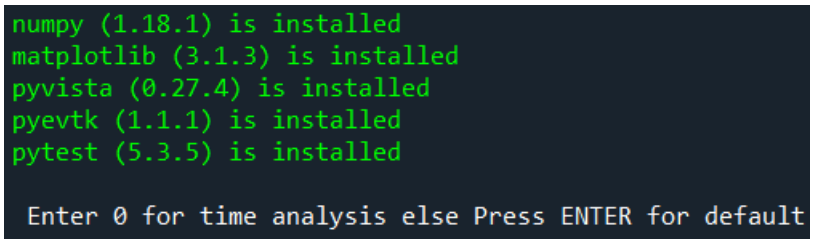


Figure 18: Installed python libraries

2. A prompt appears which asks the user to choose if time analysis has to be performed or not.
   0- Time analysis (log files are generated)
   Enter - For normal execution without any log files.

3. Then the inputs have to be given or press enter to run default values.

```
l h w nx ny nz load volume_fra penal rmin E v density BC_op Ft_op
    verbose
8 5 1 35 25 3 -100 0.4 3 1.5 150000 0.3 7850 0 1 1
INPUTS WITH DEFAULT VALUES:
Length(l)                  : 8
Height(h)                  : 5
Width(w)                   : 1
nx(no of elements along x) : 35
ny(no of elements along y) : 25
nz(no of elements along z) : 3
load                       : -100
volume_fraction            : 0.4
penal                      : 3
rmin                       : 1.5
E(Youngs modulus)          : 150000
v(poisson ratio)           : 0.3
density                    : 7850
BC_op                      : 0
Ft_op                      : 1
verbose                    : 1
```

Boundary condition option (BC_op):

```
Option :
0- Cantilever beam with load along the bottom edge of the free end.
1- Simple supported with point load at bottom center.
2- Cantilever beam with point load at bottom of the free end.
3- Cantilever beam with point load at the free end (2d case
    loading at y=height and x=length).
4- Cantilever beam with two forces at top and bottom of the free
    end .
5- Cantilever beam with point load at the center of the free end.
6- Simple supported with load at top center.
```

Optimizer option (Ft_op):

```
Option :
0-OC (optimality criterion)

1-MMA (method of moving asymptotes)
```

Verbose:

```
Option :
0-Will not print plots using pyvista only VTK file is generated.

1- Plots are generated and stored in results folder.
```

4. Plots are stored in respective sub folder with optimizer names in results folder.

5. A copy of input values and time log file are stored in log_files folder.

7.2 Procedure to run the test case

We used pytest to perform unit, functional and patch testing. All test_ files are the python files which contain respective test cases.
All files should be placed in the same folder and the working directory has to be same, so that test case can run.
Command : **pytest**

This command can be used to run all the test cases or test commands to run the respective files.

```
pytest test_Inputs.py
pytest test_geometry.py
pytest test_element_routine.py
pytest test_optimization.py
pytest test_patch.py
pytest test_rigid_body_motion.py
```

8 Result validation and discussion

8.1 Validation test case-1

8.1.1 Problem description

A cantilever beam is subjected to load at the free end. As shown in figure -59

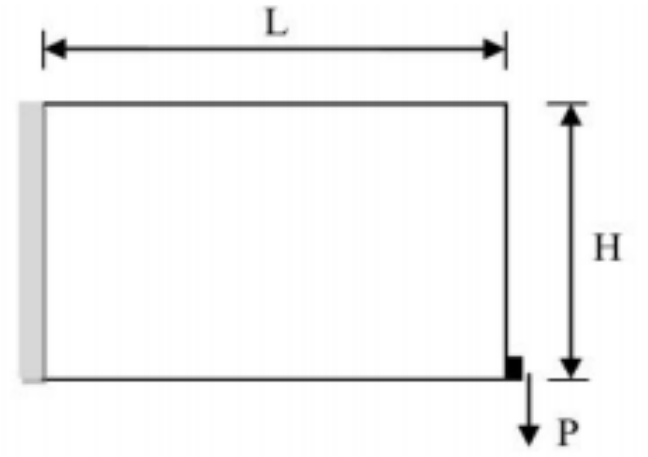


Figure 19:
Cantilever beam with load at free end

8.1.2 Input parameters

The input parameters are Length 8cm, Height 5cm, Width 1cm, Young's modulus 150000 N/m, Poisson ratio 0.3, load 100 N, volume fraction 0.4(optimized volume V/original volume V0), penalization factor 3, minimum radius 1.5
Degree of the curve
p=1,q=1,r=1.

8.1.3 Problem solved using MMA optimizer

Cantilever beam with load at free end with density 7850 kg/m^3. The problem is solved using Moving Asymptotes method to get optimized structure. The following list can be used to obtain the below results.

```
INPUTS:
8 5 1 35 25 3 -100 0.4 3 1.5 150000 0.35 7850 0 1 1
```

The comparison of the results from literature[9] and program generated optimized structure is done in figure (20) and (21).
The change in the structure and it's compliance are plotted at regular intervals in figure (23) and change in volume fraction at each iteration is plotted in figure (24).
Measure of discreteness- The number of elements which are in the between 0 and 1 as shown in figure (25).

Figure 20: IGTO validation-1 for MMA
Results from literature[9]

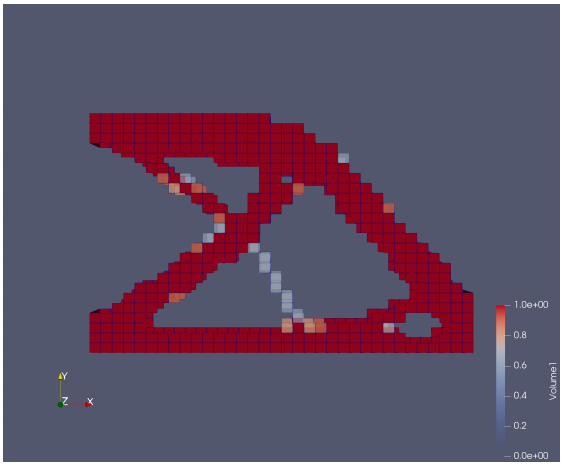


Figure 21: Optimized structure using MMA method
Program generated result, plotted in paraview

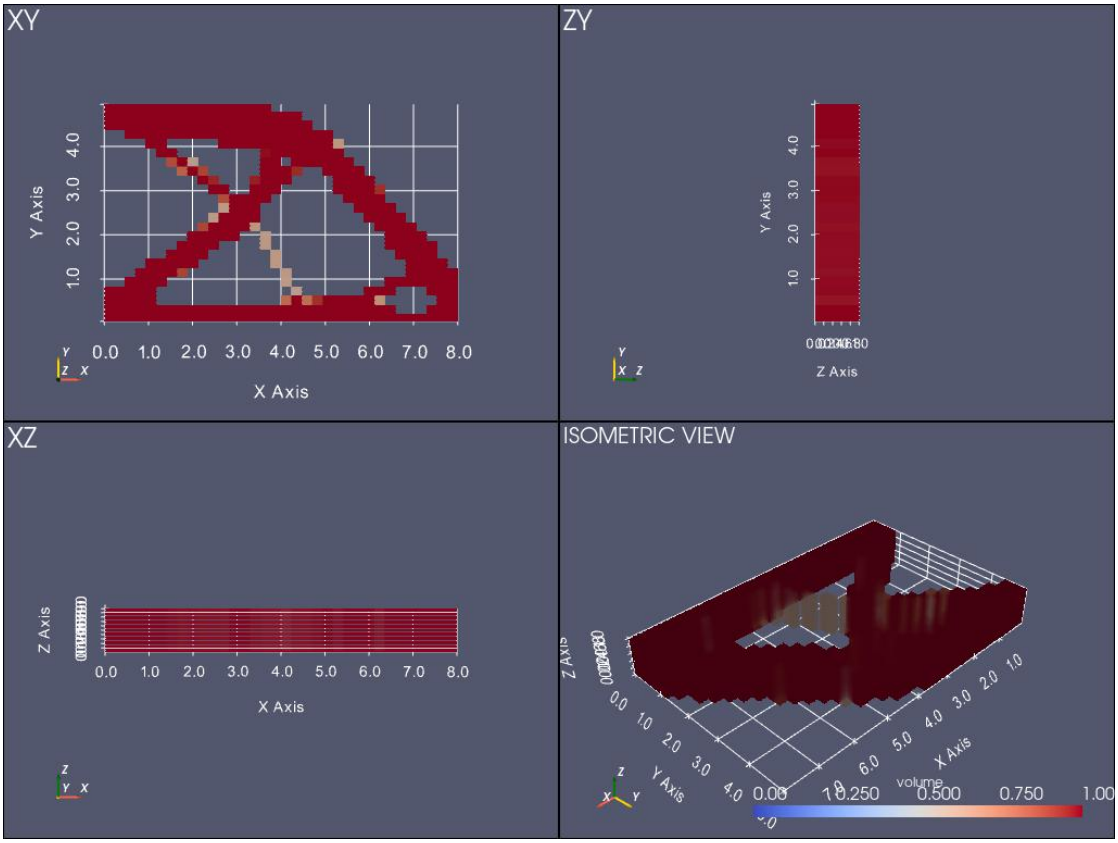


Figure 22: Topology optimized structure of cantilever beam with volume fraction 0.4, penalty 3 and rmin 1.5, plotted using pyvista

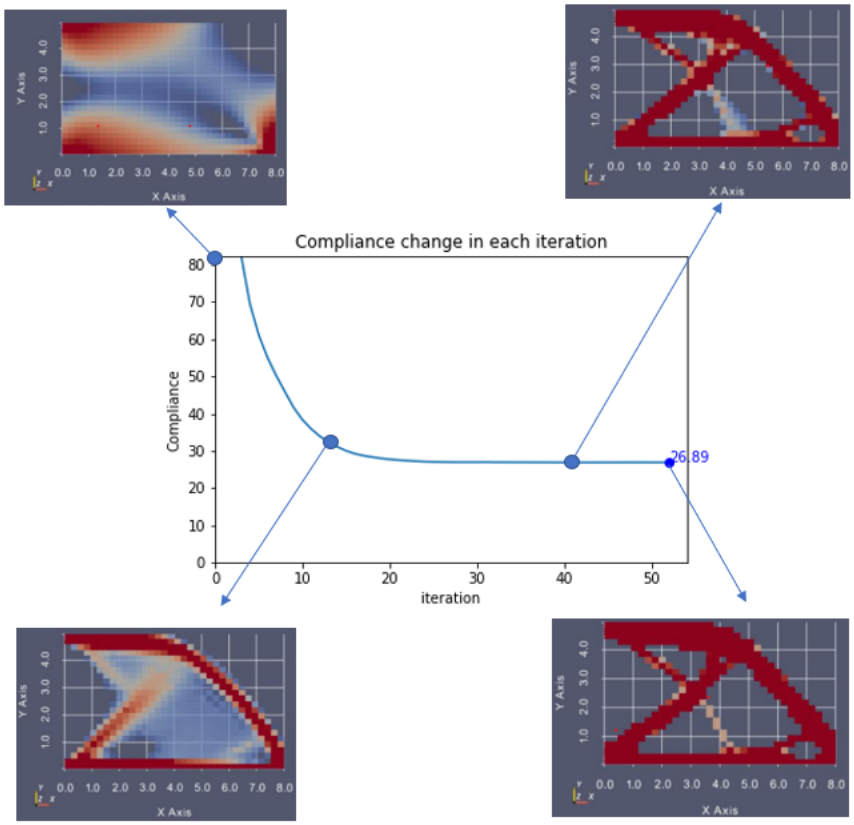


Figure 23: Change of compliance at every iteration, plotted structure at 1,10,40 and last iterations

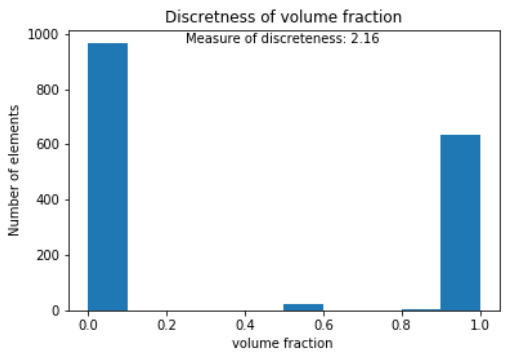


Figure 24: Mesaure of discretness

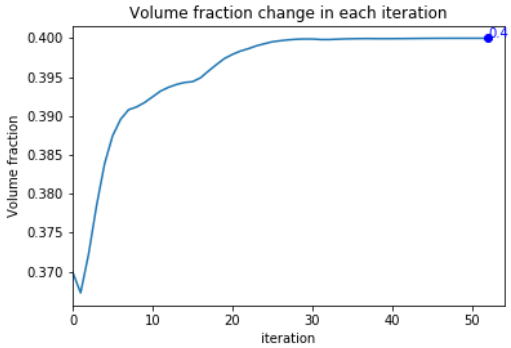


Figure 25: Change in volume fraction at each iteration

8.1.4 Problem solved using OC optimizer

Cantilever beam with load at free end with density 7850 kg/m^3. The problem is solved using optimality criteria method to get optimized structure. The following list can be used to obtain the below results.

```
INPUTS:
8 5 1 35 25 3 -100 0.4 3 1.5 150000 0.35 7850 0 0 1
```

The comparison of the results from literature[9] and program generated optimized structure is done in figure (26) and (27).
The change in the structure and it's compliance are plotted at regular intervals in figure (31) and change in volume fraction at each iteration is plotted in figure (28).
Measure of discretness- The number of elements which are in the between 0 and 1 as shown in figure (29).

Figure 26: IGTO validation-1 for OC
Results from literature[9]

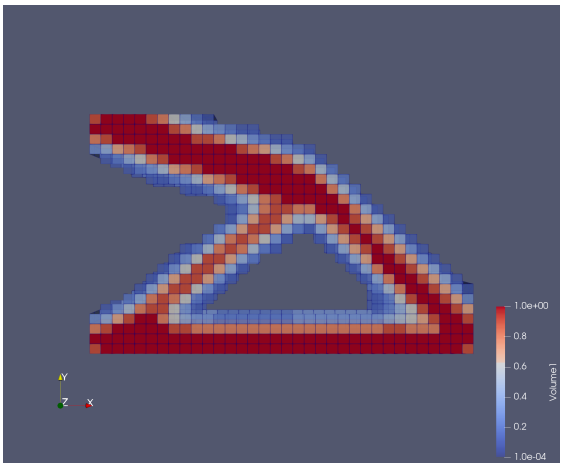


Figure 27: Optimized structure using OC method
Program generated result, plotted in paraview

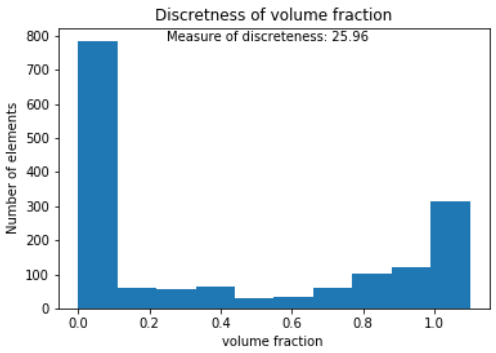


Figure 28: Mesaure of discretness

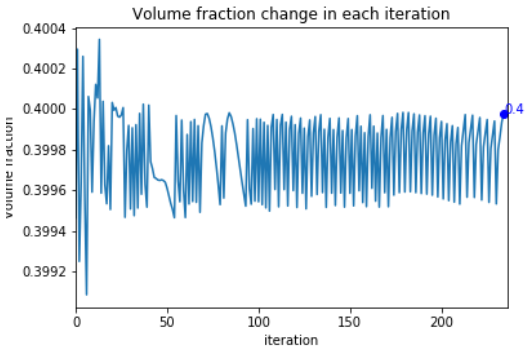


Figure 29: Change in volume fraction at each iteration

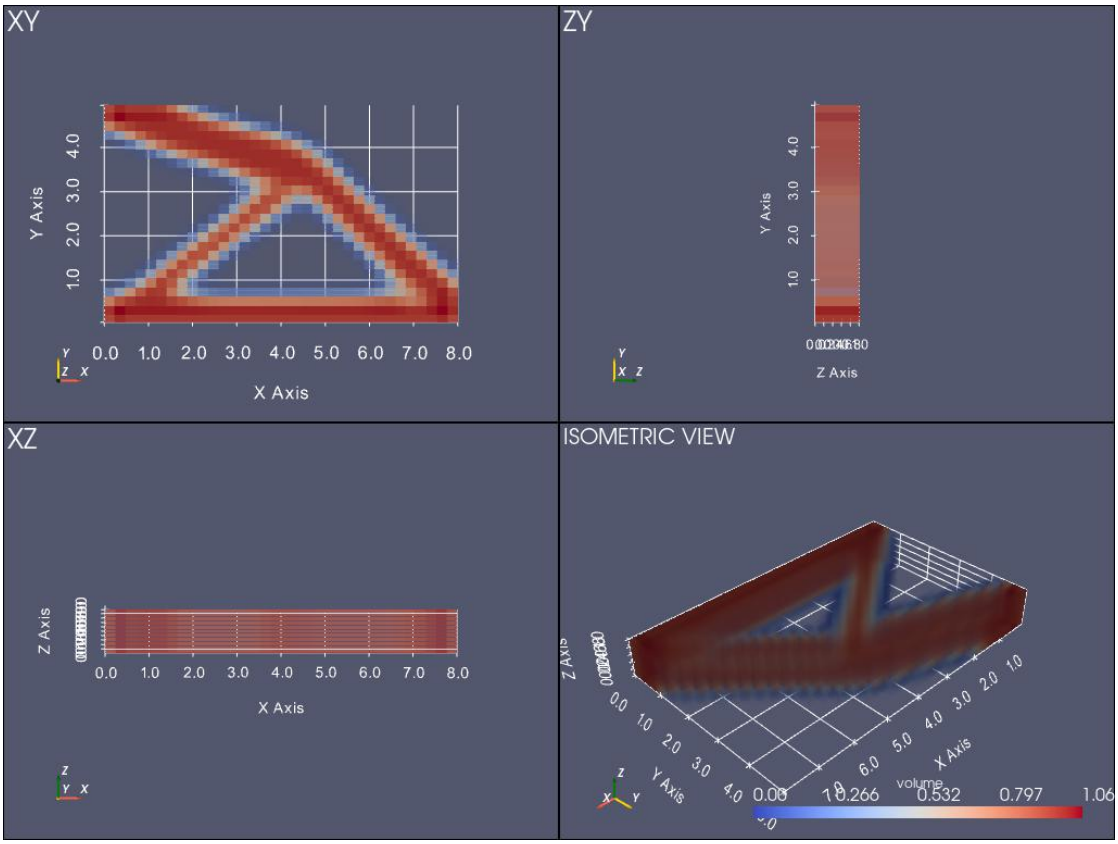


Figure 30: Topology optimized structure of cantilever beam with volume fraction 0.4, penalty 3 and rmin 1.5 plotted using pyvista

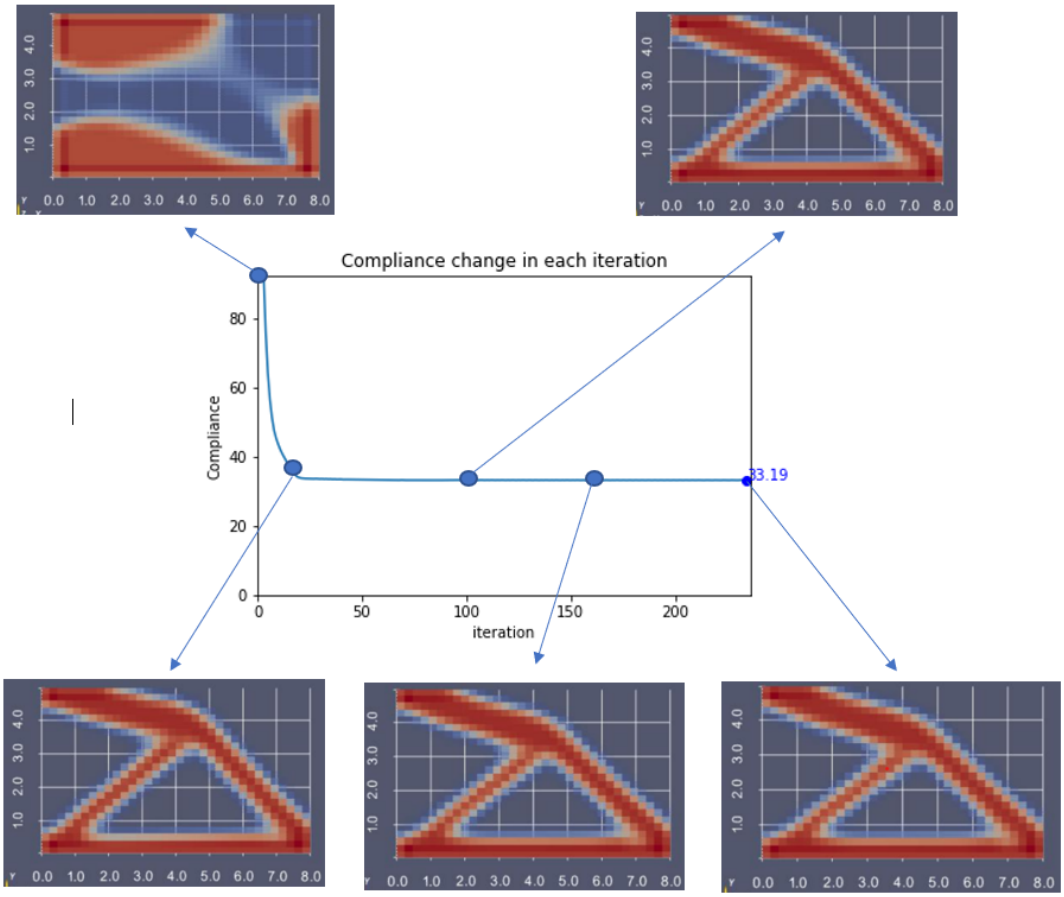


Figure 31: Change of compliance at every iteration, plotted structure at 1,40,100,160 and last iterations

8.1.5 Results discussion

1. Lower compliance is achieved using MMA and the number of iterations required to achieve optimized structure are less using MMA than OC method.

2. From figure (20) and (21), a slight deviation from the original solution can be accounted to the dependency of optimized structure on number of elements and comparison of 2d structure results from paper with program results obtained from 3D analysis.

3. The measure of discreteness for MMA and OC methods shows that MMA penalizes the element volume effectively than OC method.

4. From volume fraction change at each iteration for MMA and OC methods. As MMA is gradient based method it approaches the final volume gradually where as OC is penalty based method therefore fluctuates about the volume constraint.

8.2 Validation test case-2

8.2.1 Problem description

A simple support beam is subjected to load at center of the free end. As shown in figure(32)

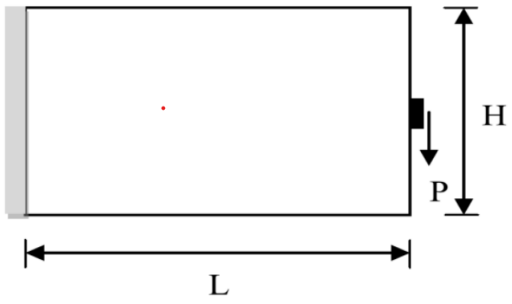


Figure 32:
Cantilever beam with load at center of the free end

8.2.2 Input parameters

The input parameters are Length 8cm, Height 5cm, Width 1cm, Young's modulus 150000 N/m, Poisson ratio 0.3, load 100 N, volume fraction 0.4(optimized volume V/original volume V0), penalization factor 3, minimum radius 1.5
Degree of the curve
p=1,q=1,r=1.

8.2.3 Problem solved using MMA optimizer

Cantilever beam with load at center of the free end with density 7850 kg/m^3. The problem is solved using Moving Asymptotes method to get optimized structure. The following list can be used to obtain the below results.

```
INPUTS:
8 5 1 35 25 3 -100 0.4 5 1.5 150000 0.35 7850 5 1 1
```

The comparison of the results from literature[9] and program generated optimized structure is done in figure (33) and (34).
The change in the structure and it's compliance are plotted at regular intervals in figure (38) and change in volume fraction at each iteration is plotted in figure (35).
Measure of discretness- The number of elements which are in the between 0 and 1 as shown in figure (36).

Figure 33: IGTO validation-2 for MMA **Results from literature**[9]

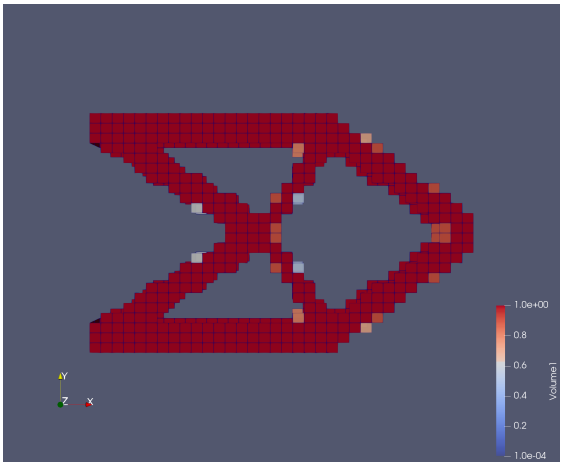


Figure 34: Optimized structure using MMA method
Program generated result, plotted in paraview

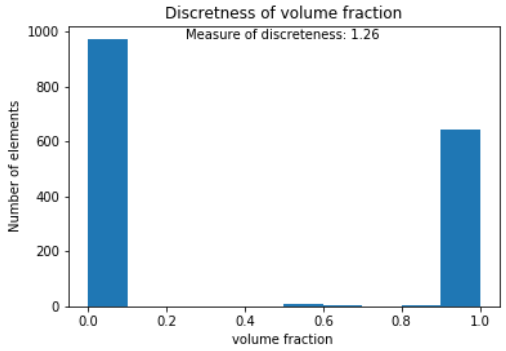


Figure 35: Mesaure of discretness

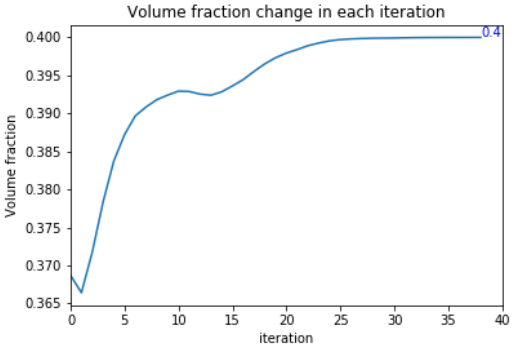


Figure 36: Change in volume fraction at each iteration

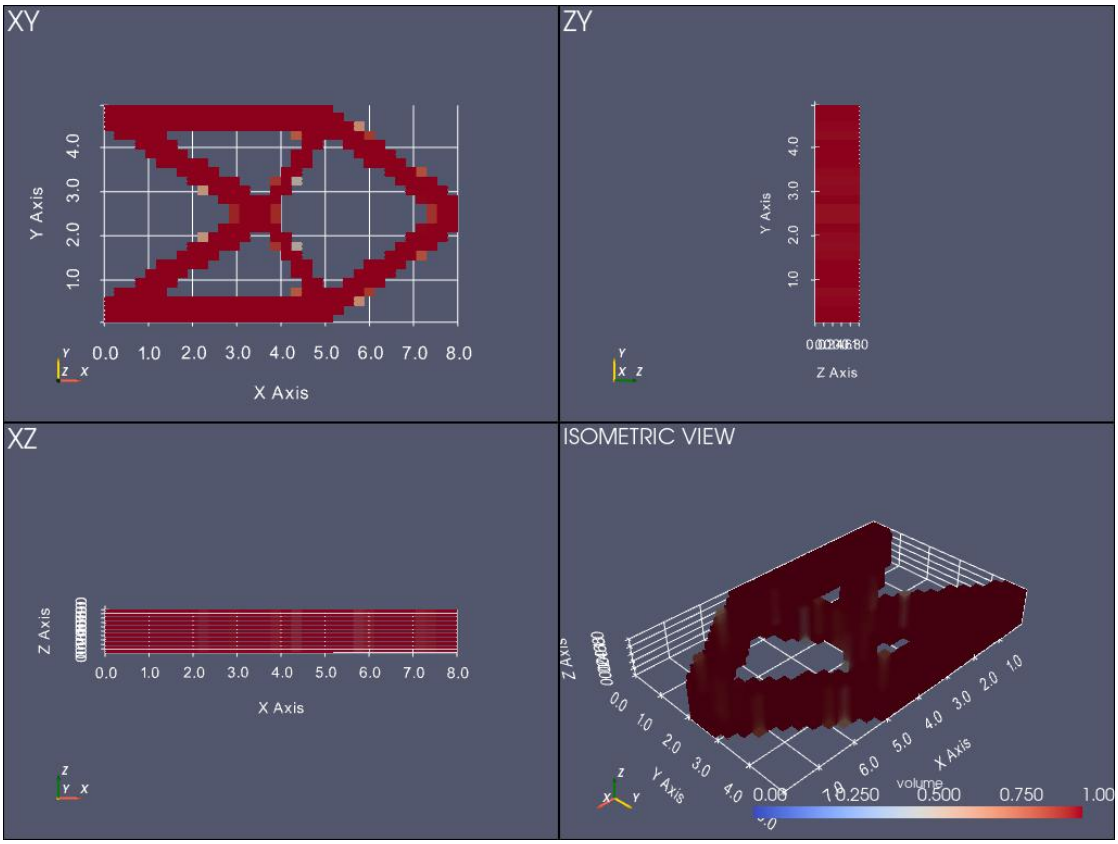


Figure 37: Topology optimized structure of cantilever beam with volume fraction 0.4, penalty 3 and rmin 1.5,plotted using pyvista

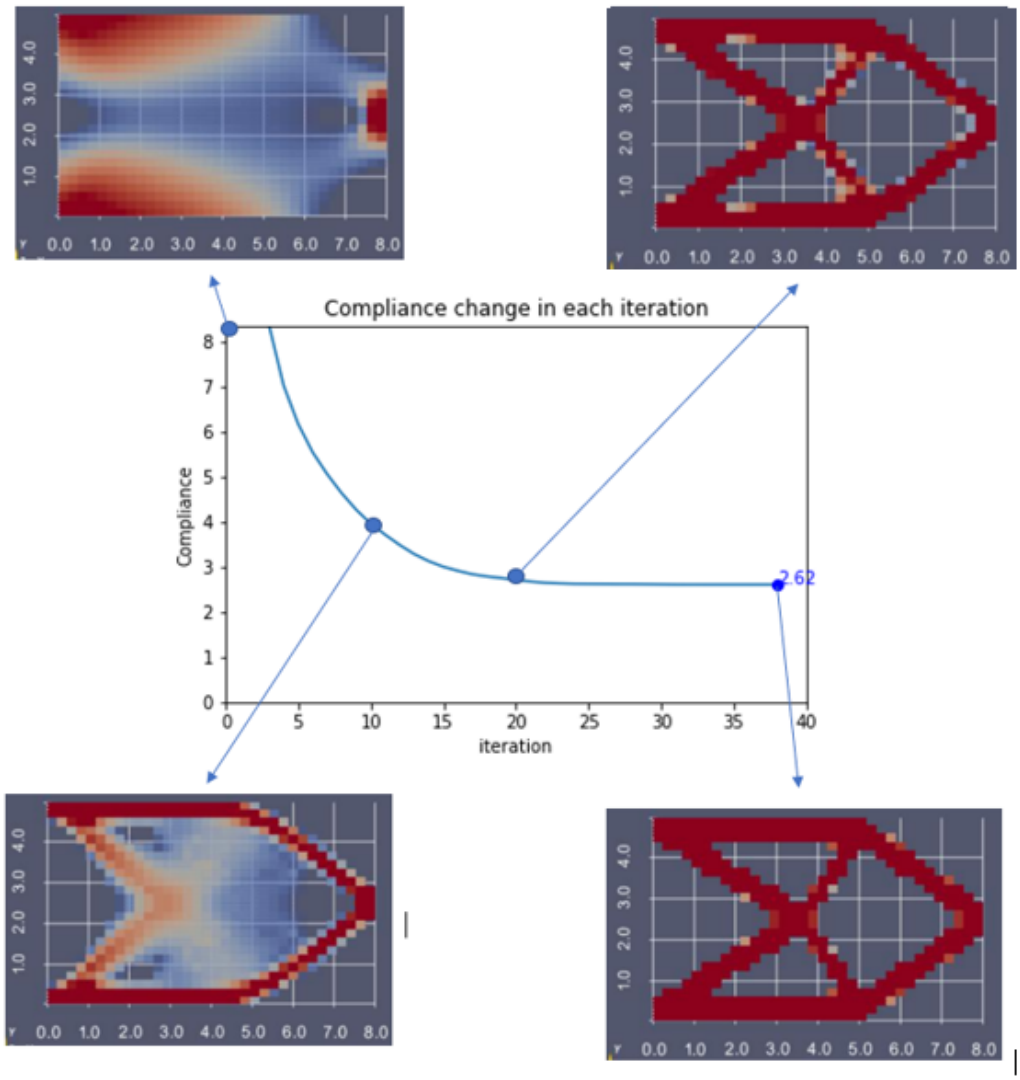


Figure 38: Change of compliance at every iteration, plotted structure at 1,10,20 and last iterations

8.2.4 Problem solved using OC optimizer

Cantilever beam with load at center of the free end with density 7850 kg/m^3. The problem is solved using optimality criteria method to get optimized structure. The following list can be used to obtain the below results.

```
INPUTS:
8 5 1 35 25 3 -100 0.4 3 1.5 150000 0.35 7850 5 0 1
```

The comparison of the results from literature[9] and program generated optimized structure is done in figure (39) and (40).
The change in the structure and it's compliance are plotted at regular intervals in figure (44) and change in volume fraction at each iteration is plotted in figure (42).
Measure of discretness- The number of elements which are in the between 0 and 1 as shown in figure (41).

Figure 39: IGTO validation-2 for OC
Results from literature[9]

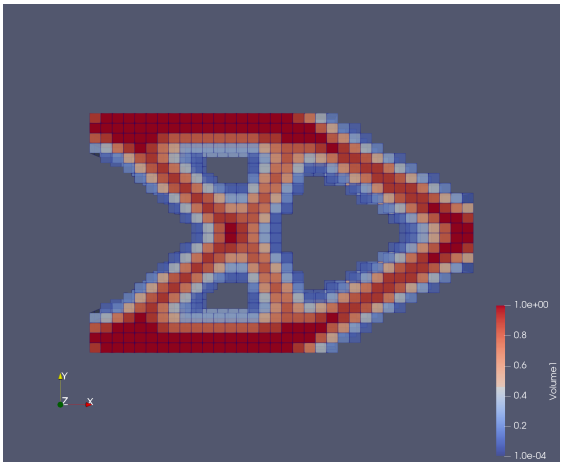


Figure 40: Optimized structure using OC method
Program generated result, plotted in paraview

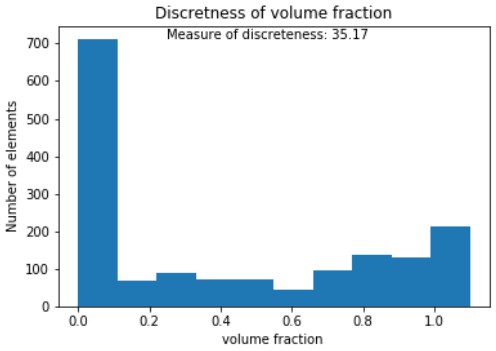


Figure 41: Mesaure of discretness

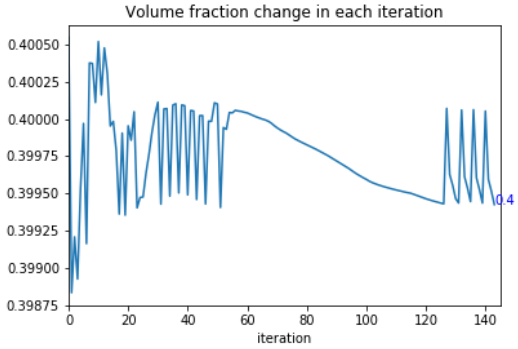


Figure 42: Change in volume fraction at each iteration

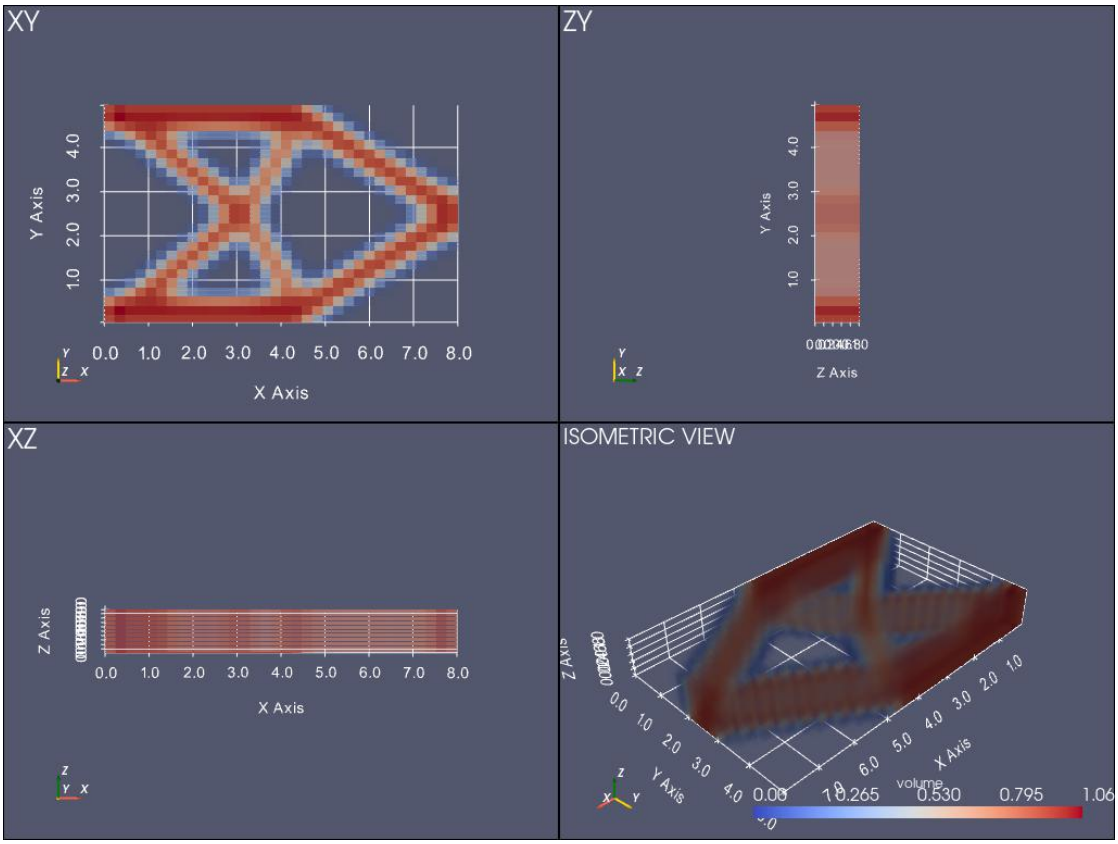


Figure 43: Topology optimized structure of cantilever beam with volume fraction 0.4, penalty 3 and rmin 1.5,plotted using pyvista

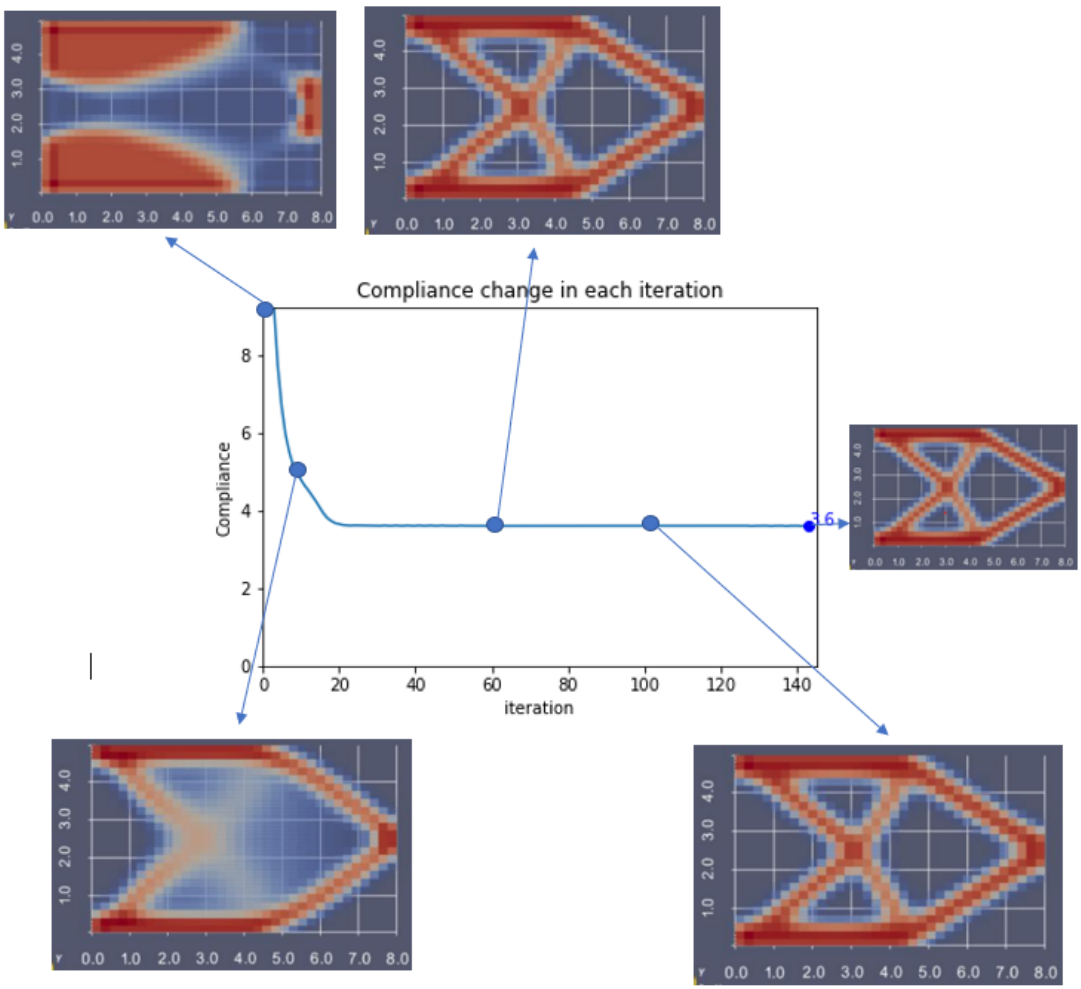


Figure 44: Change of compliance at every iteration, plotted structure at 1,10,60,100 and last iterations

8.2.5 Results discussion

1. Lower compliance is achieved using MMA and the number of iterations required to achieve optimized structure are less using MMA than OC method.
2. The measure of discreteness for MMA and OC methods shows that MMA penalizes the element volume effectively than OC method.
3. From volume fraction change at each iteration for MMA and OC methods. As MMA is gradient based method it approaches the final volume gradually

where as OC is penalty based method therefore fluctuates about the volume constraint.

4. The convergence to optimal solution using OC method is not efficient which can be improved by using filters.

8.3 Validation test case-3

8.3.1 Problem description

A Simple supported beam is subject to load at center. As shown in figure(45)

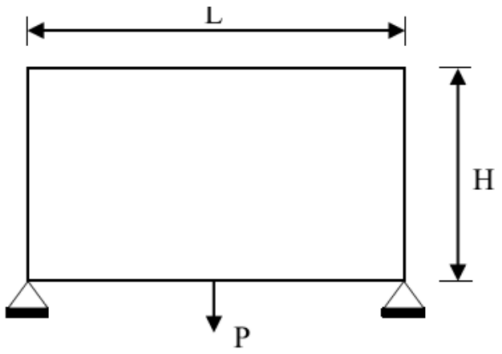


Figure 45:
Simple supported beam with load at center

8.3.2 Input parameters

The input parameters are Length 8cm, Height 5cm, Width 1cm, Young's modulus 150000 N/m, Poisson ratio 0.3, load 100 N, volume fraction 0.4(optimized volume V/original volume V0), penalization factor 3, minimum radius 1.5
Degree of the curve
p=1,q=1,r=1.

8.3.3 Problem solved using MMA optimizer

Simple supported beam with load at center with density 7850 kg/m^3. The problem is solved using Moving Asymptotes method to get optimized structure. The following list can be used to obtain the below results.

```
INPUTS:
8 5 1 35 25 3 -100 0.2 3 1.5 150000 0.35 7850 1 1 1
```

The comparison of the results from literature[9] and program generated optimized structure is done in figure (46) and (47).
The change in the structure and it's compliance are plotted at regular intervals in figure (51) and change in volume fraction at each iteration is plotted in figure (48).
Measure of discretness- The number of elements which are in the between 0 and 1 as shown in figure (49).

Figure 46: IGTO validation-3 for MMA
Results from literature[9]

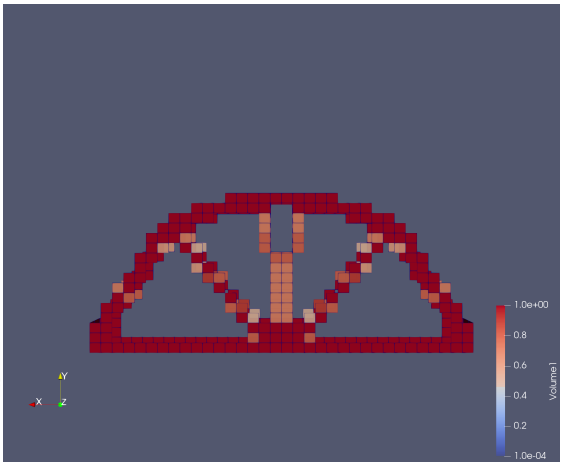


Figure 47: Optimized structure using MMA method
Program generated result, plotted in paraview

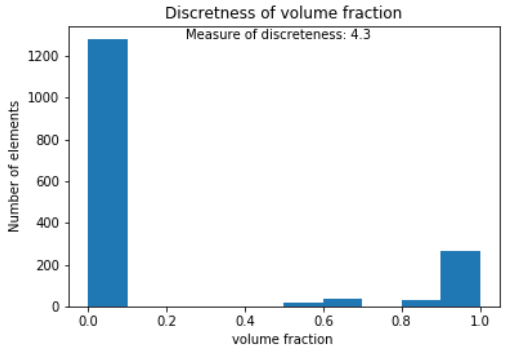


Figure 48: Mesaure of discretness

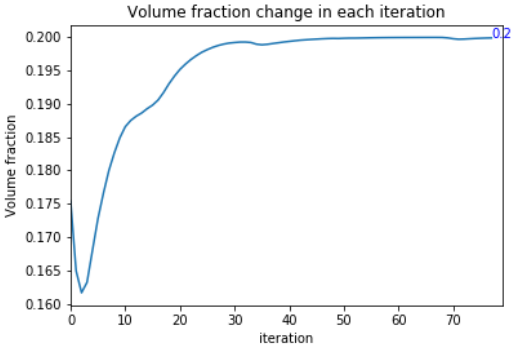


Figure 49: Change in volume fraction at each iteration

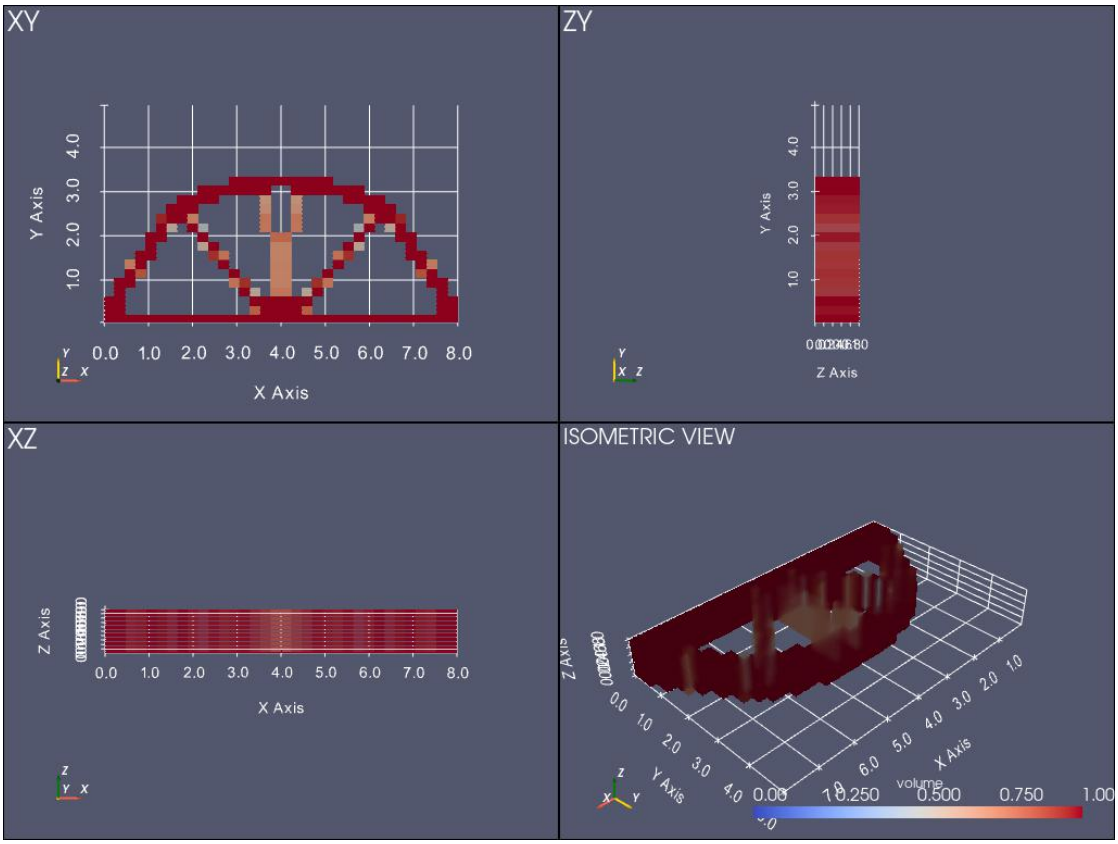


Figure 50: Topology optimized structure of cantilever beam with volume fraction 0.2, penalty 3 and rmin 1.5,plotted using pyvista

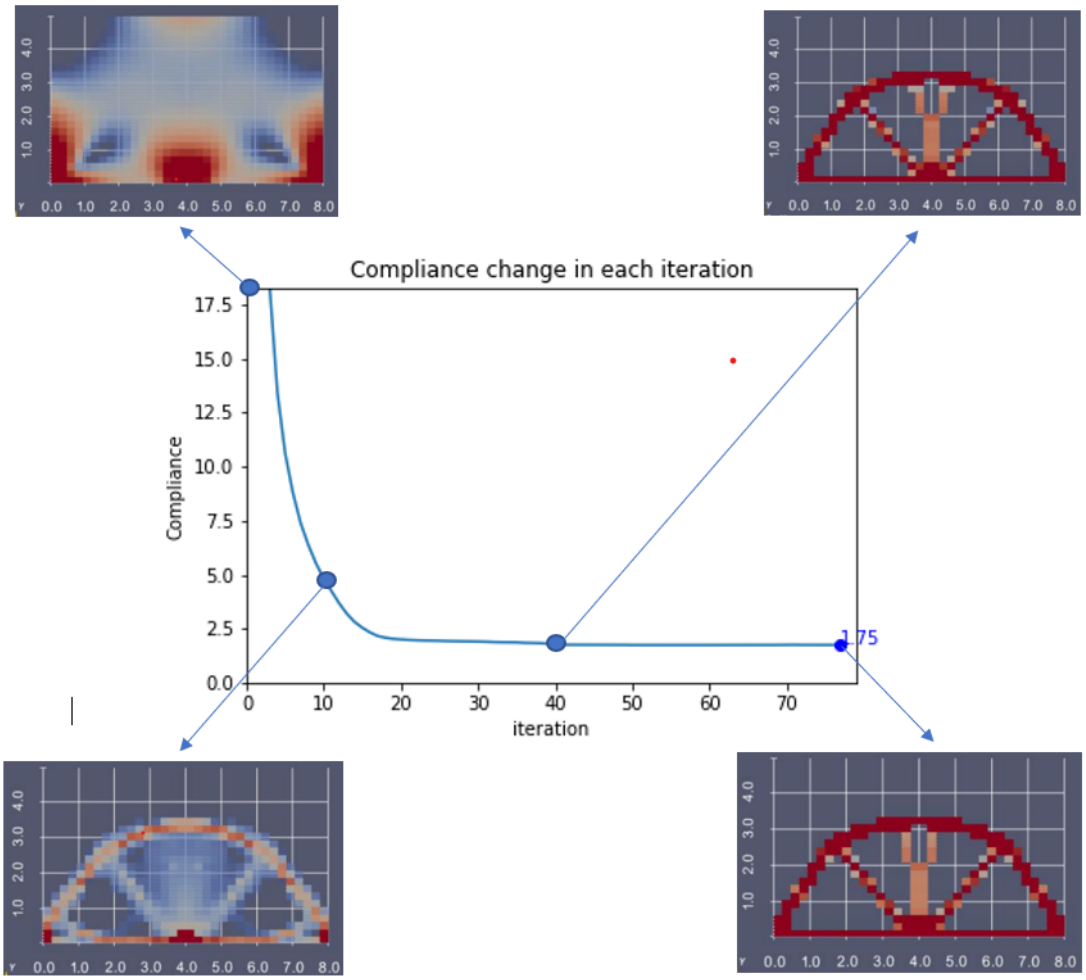


Figure 51: Change of compliance at every iteration, plotted structure at 1,10,40 and last iterations

8.3.4 Problem solved using OC optimizer

Simple supported beam with load at center with density 7850 kg/m^3. The problem is solved using optimality criteria method to get optimized structure. The following list can be used to obtain the below results.

```
INPUTS:
8 5 1 35 25 3 -100 0.2 3 1.5 150000 0.35 7850 1 0 1
```

The comparison of the results from literature[9] and program generated optimized structure is done in figure (52) and (53).
The change in the structure and it's compliance are plotted at regular intervals in figure (57) and change in volume fraction at each iteration is plotted in figure (55).
Measure of discretness- The number of elements which are in the between 0 and 1 as shown in figure (54).

Figure 52: IGTO validation-3 for OC
Results from paper[9]

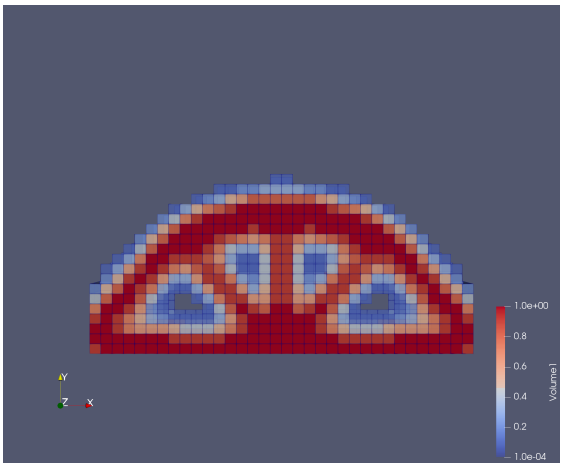


Figure 53: Optimized structure using OC method
Program generated result, plotted in paraview

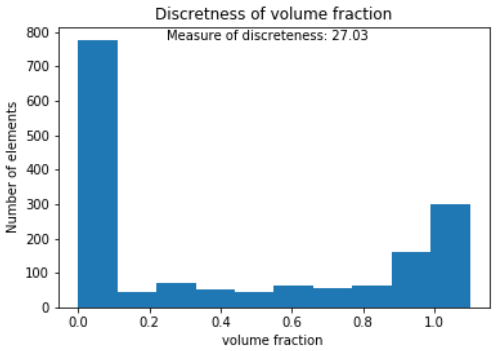


Figure 54: Mesaure of discretness

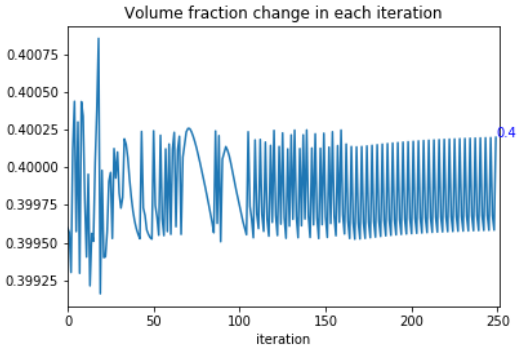


Figure 55: Change in volume fraction at each iteration

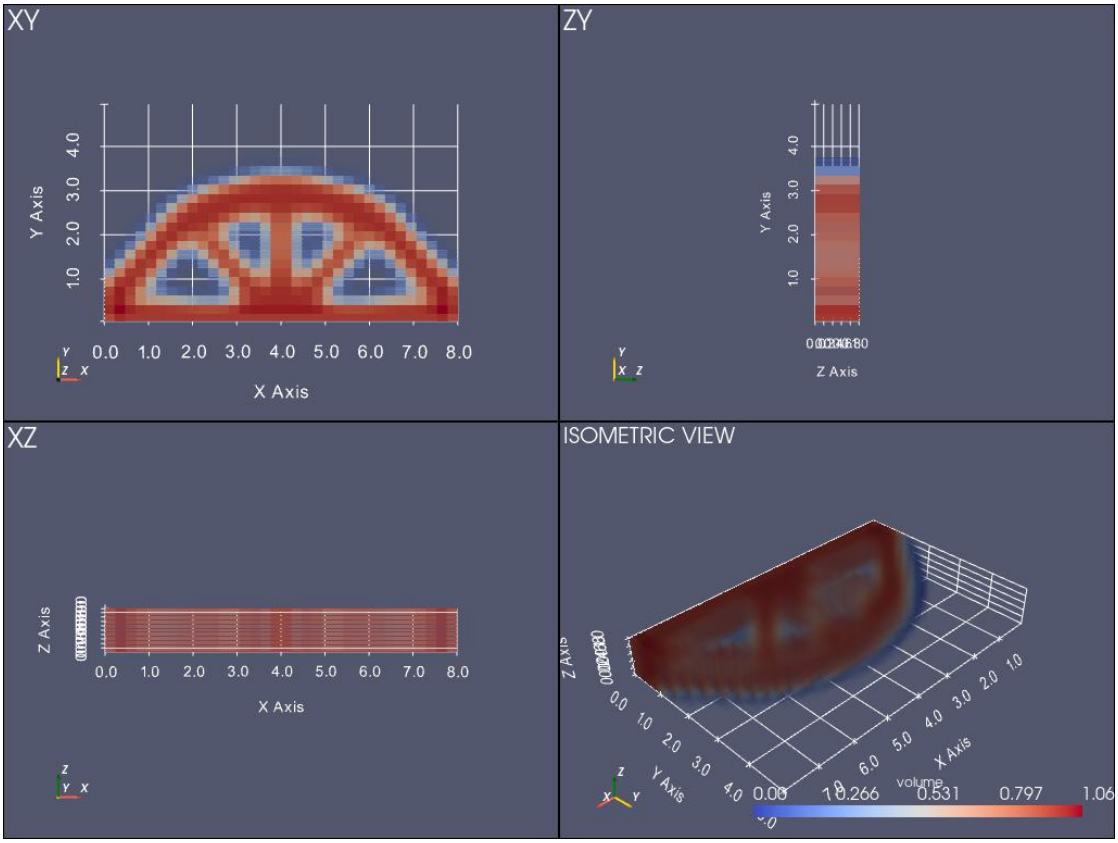


Figure 56: Topology optimized structure of cantilever beam with volume fraction 0.2, penalty 3 and rmin 1.5, plotted using pyvista

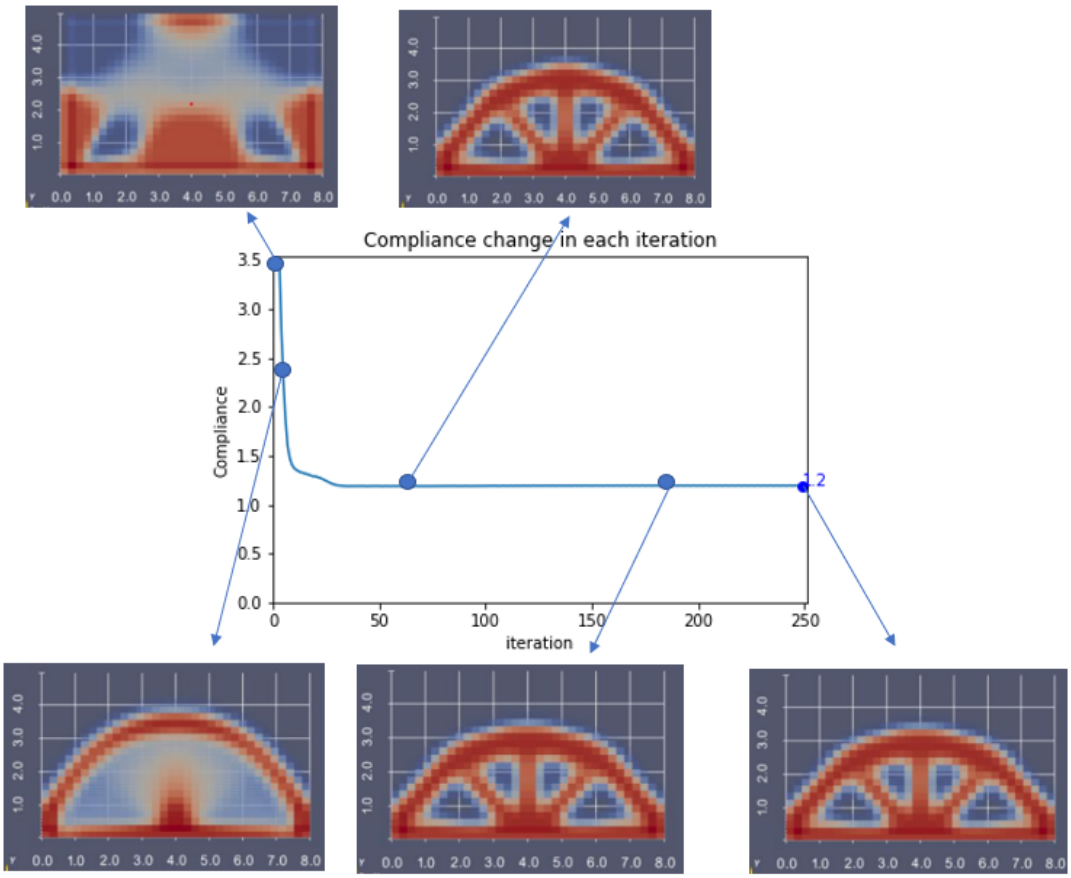


Figure 57: Change of compliance at every iteration, plotted structure at 1,10,60,180 and last iterations

8.3.5 Results discussion

1. Lower compliance is achieved using MMA and the number of iterations required to achieve optimized structure are less using MMA than OC method.
2. The discrepancy in results for both MMA and OC method is because of the 2D analysis is performed in the literature which could account for the straining along z direction.
3. The measure of discreteness for MMA and OC methods shows that MMA penalizes the element volume effectively than OC method.
4. From volume fraction change at each iteration for MMA and OC methods. As MMA is gradient based method it approaches the final volume gradually

where as OC is penalty based method therefore fluctuates about the volume constraint.

5. The convergence to optimal solution using OC method is not efficient which can be improved by using filters.

8.4 Influence of parameters on topology optimization

8.4.1 Problem statement

A cantilever beam is subject to load at the free end. As shown in figure -59

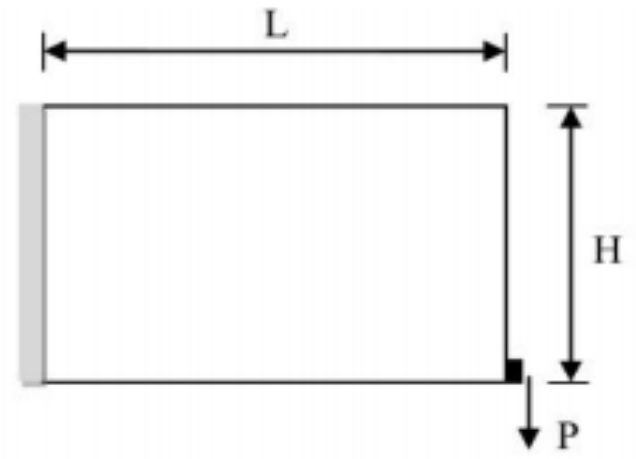


Figure 58:
Cantilever beam with load at free end

8.4.2 Penalization factor

Inputs are same as in validation case-1. Penalization factor is used to remove porosity with in material so checker board pattern can be avoided. The change in topology optimization based on the penalization factor **P** is shown in Table-1. rmin is constant 1.5 and P is changed.

```
INPUTS:
8 5 1 35 25 3 -100 0.4 {P} 1.5 150000 0.35 7850 0 1 1
```

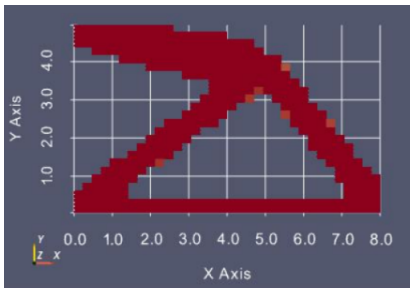
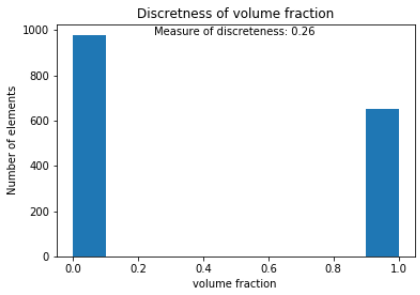
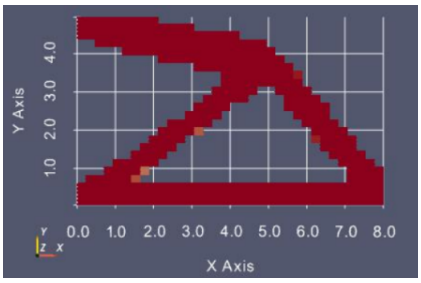
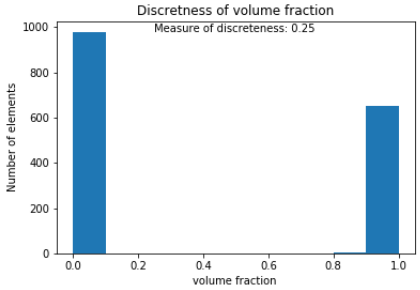
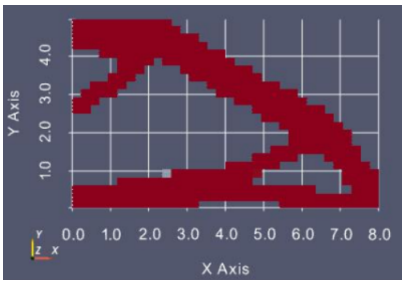
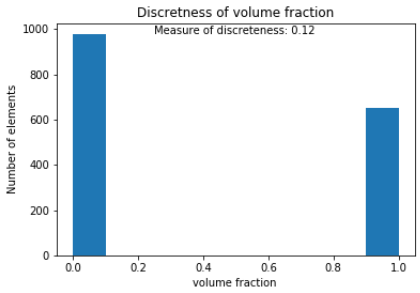
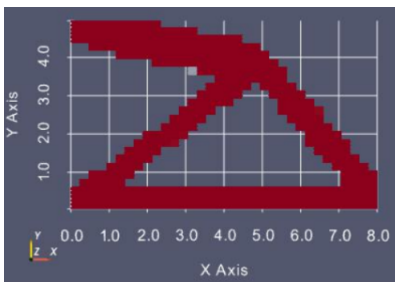
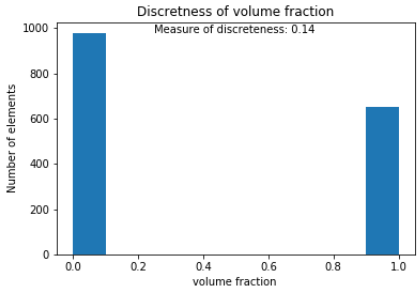
P	optimized structure	Discreteness of volume
5		
10		
20		
30		

Table 1: Influence of penalization factor

8.4.3 Minimum radius (rmin) factor

Inputs are same as in validation case-1. rmin is the minimum distance between the neighbouring elements which is used as a parameter in sensitivity analysis to build weight factor. The change in topology optimization based on the **rmin** factor is shown in Table-2.
P is constant and rmin is changed.

```
INPUTS:
8 5 1 35 25 3 -100 0.4 3.5 {rmin} 150000 0.35 7850 0 1 1
```

8.4.4 Results discussion

1. The topology optimization depends mainly on penalization factor **P** and minimum radius **rmin**.

2. The influence of penalization factor on topology optimization is more elements with intermediate value are penalized. As the value of **P** increases the discreteness decreases. The value of **P** has less influence on the compliance and structure would remain relatively same.As shown in table 1.

3. The **rmin** value has great influence on the structure and the shape of the structure changes with **rmin**. Due to the change in structure, the compliance also changes with **rmin**. As shown in table 2

4. Topology optimization also depends on the type of optimizer being used like OC and MMA

5. MMA would converge faster and has less discrete volume.

6. OC needs more number of iteration to reach optimal solution and has more discrete volume.

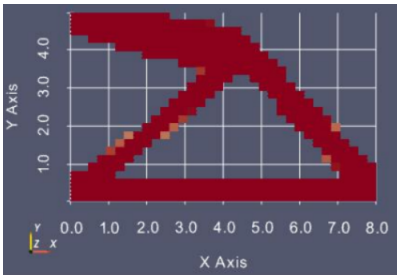
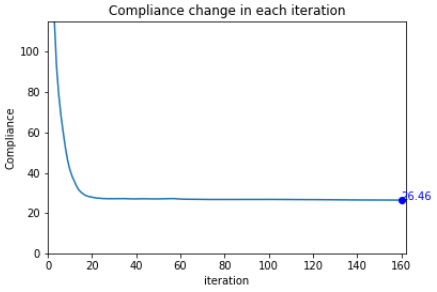
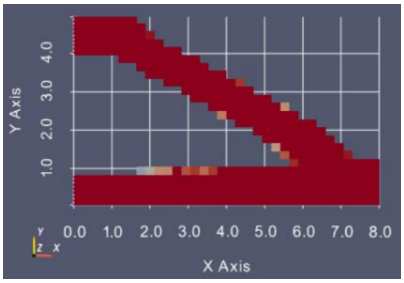
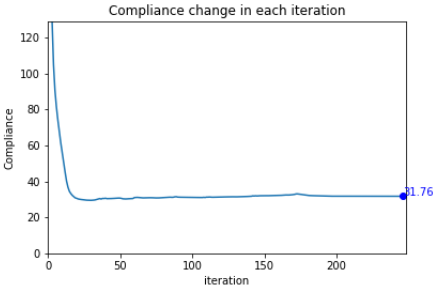
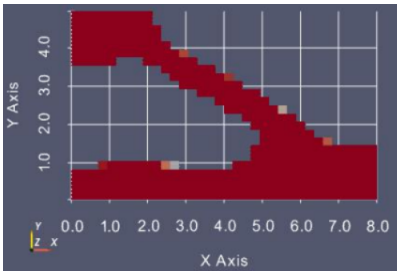
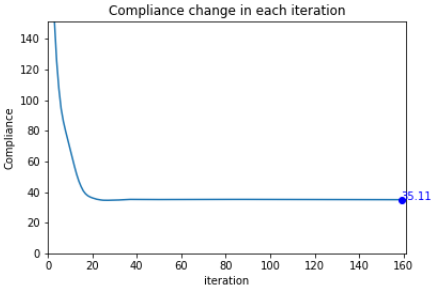
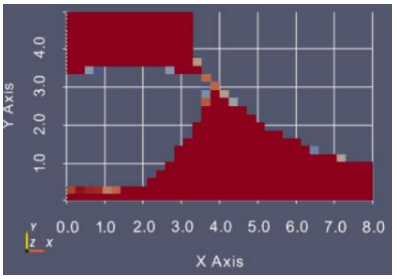
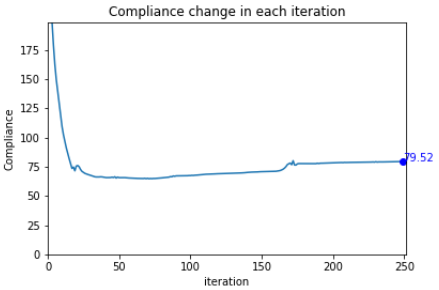
rmin	Optimized structure	Compliance of the structure
2		
5		
8		
15		

Table 2: Influence of rmin factor

9 Time Analysis: Bottleneck in code

The main aim is to find the computation time of each function and find possible bottlenecks in IGTO. The time analysis is performed in **main_program_bottle_neck.py** file, only time module is used in this analysis.

9.1 Problem description

A cantilever beam is subjected to load at the free end. As shown in figure -59

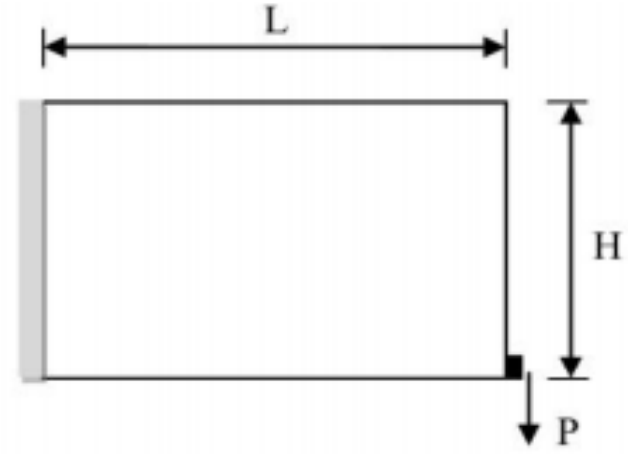


Figure 59:
Cantilever beam with load at free end

9.2 Input parameters

The input parameters are Length 8cm, Height 5cm, Width 1cm, Young's modulus 150000 N/m, Poisson ratio 0.3, load 100 N, volume fraction 0.4(optimized volume V/original volume V0), penalization factor 3, minimum radius 1.5
Degree of the curve
p=1,q=1,r=1.

```
INPUTS:
8 5 1 35 25 3 -100 0.4 5 1.5 150000 0.35 7850 0 1 1
```

9.3 Analysis and results discussion

A log file is generated with contains the number of times the function was called , total time taken by the function and average time taken by the function.
A separate log file is created for each optimizer. The log contains the top 5 functions with high execution time.A pie chart is plotted based on number of times the function was called as shown in fig.(60)

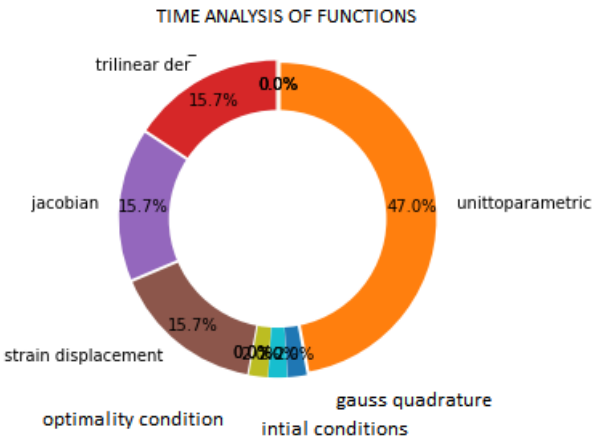


Figure 60: Time analysis pie chart

```
LOG FILE DATA:

AUTHOR : YASA VISWAMBHAR REDDY

!# 1616605044.207525 Topology Optimization using Iso-Geometric Analysis

Measure of discreteness: 2.1574617647058822

Final Volume of the optimised structure : 0.40012254901960786

Mass of the beam at 100% volume      : 314000.0
```

```
Optimized mass of the beam           : 125638.48039215687

Initial Compliance without optimization : 228.7315796089796

Final Compliance with optimization   : 26.888775237117564

Total execution time of the whole program: 1363.7347421646118 sec

The time taken to run Topology optimization: 1346.0096242427826 sec

Execution time of IGA analysis at 100% volume : 11.451846837997437

Top 5 function which have high average execution time

 initial condition
 strain displacement
 unittoparametric
 Compliance matrix
 derbspline basis

 ++++++++++++The function are ordered in decreasing average time of
     execution++++++++++++

--------- initial condition ---------
Total Number of time " initial condition " was called 53 times.
Average time taken by the " initial condition " function during one run
    7.528179096725751e-05 seconds.
Total time taken by the " initial condition " function
    0.0039899349212646484 seconds.

--------- strain displacement ---------
Total Number of time " strain displacement " was called 1396992 times.
Average time taken by the " strain displacement " function during one
    run 4.5424312235716175e-05 seconds.
Total time taken by the " strain displacement " function
    63.45740079879761 seconds.

--------- unittoparametric ---------
Total Number of time " unittoparametric " was called 4190976 times.
Average time taken by the " unittoparametric " function during one run
    3.492859296924389e-06 seconds.
Total time taken by the " unittoparametric " function
```

```
    14.638489484786987 seconds.

--------- Compliance matrix ---------
Total Number of time " Compliance matrix " was called 174624 times.
Average time taken by the " Compliance matrix " function during one run
    3.2285849253336587e-06 seconds.
Total time taken by the " Compliance matrix " function
    0.5637884140014648 seconds.

--------- derbspline basis ---------
Total Number of time " derbspline basis " was called 4190976 times.
Average time taken by the " derbspline basis " function during one run
    1.175686975079883e-05 seconds.
Total time taken by the " derbspline basis " function 49.27275896072388
    seconds.

--------- knot index ---------
Total Number of time " knot index " was called 4190976 times.
Average time taken by the " knot index " function during one run
    1.0294273584875391e-05 seconds.
Total time taken by the " knot index " function 43.14305353164673
    seconds.

--------- Moving asymptoes ---------
Total Number of time " Moving asymptoes " was called 53 times.
Average time taken by the " Moving asymptoes " function during one run
    0.5117317595571842 seconds.
Total time taken by the " Moving asymptoes " function 27.12178325653076
    seconds.

--------- prime dual ---------
Total Number of time " prime dual " was called 424 times.
Average time taken by the " prime dual " function during one run
    0.3181922936214591 seconds.
Total time taken by the " prime dual " function 134.91353249549866
    seconds.

--------- Knearestneighbours ---------
Total Number of time " Knearestneighbours " was called 1 times.
Average time taken by the " Knearestneighbours " function during one
    run 0.1845073699951172 seconds.
Total time taken by the " Knearestneighbours " function
    0.1845073699951172 seconds.
```

```
--------- Folder ---------
Total Number of time " Folder " was called 15 times.
Average time taken by the " Folder " function during one run
    0.10030140876770019 seconds.
Total time taken by the " Folder " function 1.504521131515503 seconds.

--------- Newton method ---------
Total Number of time " Newton method " was called 424 times.
Average time taken by the " Newton method " function during one run
    0.061642896454289275 seconds.
Total time taken by the " Newton method " function 26.136588096618652
    seconds.

--------- Minimizer constrains ---------
Total Number of time " Minimizer constrains " was called 53 times.
Average time taken by the " Minimizer constrains " function during one
    run 0.008919598921289984 seconds.
Total time taken by the " Minimizer constrains " function
    0.47273874282836914 seconds.

--------- linear system_assembly ---------
Total Number of time " linear system_assembly " was called 1204 times.
Average time taken by the " linear system_assembly " function during
    one run 0.006873119115037379 seconds.
Total time taken by the " linear system_assembly " function
    8.275235414505005 seconds.

--------- element routine ---------
Total Number of time " element routine " was called 174624 times.
Average time taken by the " element routine " function during one run
    0.002790476543160177 seconds.
Total time taken by the " element routine " function 487.28417587280273
    seconds.

--------- Asymptoes ---------
Total Number of time " Asymptoes " was called 50 times.
Average time taken by the " Asymptoes " function during one run
    0.0012183094024658203 seconds.
Total time taken by the " Asymptoes " function 0.060915470123291016
    seconds.

--------- line search ---------
```

```
Total Number of time " line search " was called 1204 times.
Average time taken by the " line search " function during one run
    0.0006184371998935839 seconds.
Total time taken by the " line search " function 0.744598388671875
    seconds.

--------- gauss quadrature ---------
Total Number of time " gauss quadrature " was called 174624 times.
Average time taken by the " gauss quadrature " function during one run
    0.0004644662289255068 seconds.
Total time taken by the " gauss quadrature " function 81.1069507598877
    seconds.

--------- assemble ---------
Total Number of time " assemble " was called 88128 times.
Average time taken by the " assemble " function during one run
    0.00046010874347056526 seconds.
Total time taken by the " assemble " function 40.548463344573975
    seconds.

--------- application of BC ---------
Total Number of time " apply BC " was called 54 times.
Average time taken by the " apply BC " function during one run
    0.00038907704529938874 seconds.
Total time taken by the " apply BC " function 0.021010160446166992
    seconds.

--------- optimal condtition ---------
Total Number of time " optimal condtition " was called 1628 times.
Average time taken by the " optimal condtition " function during one
    run 0.00033688706320685307 seconds.
Total time taken by the " optimal condtition " function
    0.5484521389007568 seconds.

--------- jacobian ---------
Total Number of time " jacobian " was called 1396992 times.
Average time taken by the " jacobian " function during one run
    0.00021310742560139894 seconds.
Total time taken by the " jacobian " function 297.7093687057495 seconds.

--------- objective constrains ---------
Total Number of time " objective constrains " was called 53 times.
Average time taken by the " objective constrains " function during one
```

```
    run 0.00019045595852833875 seconds.
Total time taken by the " objective constrains " function
    0.010094165802001953 seconds.

--------- trilinear der ---------
Total Number of time " trilinear der " was called 1396992 times.
Average time taken by the " trilinear der " function during one run
    0.00018529972924741825 seconds.
Total time taken by the " trilinear der " function 258.8622393608093
    seconds.
```

10 Milestones

The following table contains the proposed milestones and their status.

Activities	**Status**
Implementation of B-splines and NURBS based geometry	Yes
Generate controlpoint assembly and knot-connectivity for 3D structure	Yes
Implementation of iso-geometric analysis for 3D structures	Yes
Implementation of topology optimization using SIMP method	Yes
Solving topology optimization problem using OC and MMA method	Yes
Verification of optimized structure with literature results [9] for cantilever and simple supported beam	Yes
Influence of parameters on topology optimization	Yes
Time analysis is performed for all functions	Yes

11 Conclusion

In this project, structural topology optimization is performed on 3D structures. An Iso-geometric code based on NURBS volume is generated for parametric details. The advantage of NURBS over Lagrangian basis function is NURBS has closer relation with geometric which allows us to obtain an complex optimized structure with has lesser compliance. Unlike FEM, application of boundary condition in IGA for higher order basis function is quite difficult and special strategy like least square method have to be used. Only first order NURBS basis functions are used due to difficulty in visualization of the structure for higher order basis. The optimization problem is solved using OC (optimality criterion) and MMA (method of moving asymptotes). From the result analysis, we have found that for iso-geometric topology optimization (IGTO) MMA is better suited as it is gradient based method and converges rapidly compared to OC. With OC, the solution may never reach optimal solution as it fluctuates about the constrain. Performance of OC can be optimized by using additional filters (Heavy side filter).
The influence of parameters like Penalization factor and minimum radius on topology optimization is performed and discreteness of volume in the structure is calculated. To get checkerboard free pattern,$\mathbf{p \geq 3.5}$. The penalization factor depends on Young's modulus and poisson's ration. The results of IGTO are validated with the result present in literature[9] for cantilever beam and simple supported beam. The discrepancy in results show that 3D structural analysis is required to correctly

optimized structure rather than 2D analysis.
The efficiency of IGTO depends on number of elements. As the number of element increase more computational effort is required. Thus IGTO can be extended to high performance computing.

12 GIT Log

```
commit 1eeae10a9e715461563a07130e1df1363caf7439
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Mar 22 17:28:25 2021 +0100

    commenting and doc strings

commit 08e8c92b3c281cd766600b97af3e8f1cf70d8fd3
Merge: f22785c 6665435
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Fri Feb 26 13:45:10 2021 +0100

    Merge branch 'main' of https://github.com/viswambhar-yasa/IGTO into main

commit f22785c80a9d11131ceaa9bb72931fcacc678c4c
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Fri Feb 26 13:44:50 2021 +0100

    few changes

commit 07df0484d05f95476cd1c6d229acdb39d6721e31
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Fri Feb 26 13:42:30 2021 +0100

    added documentation folder

commit 518d7cb8447bcebb23b7a00d4bafed792ebcd9da
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Fri Feb 26 13:41:51 2021 +0100

    Added documentation folder

commit 6665435531712478919c15908c691f5873890a72
Author: YasaV9 <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sun Feb 14 16:04:58 2021 +0100

    Rename Inputs.py to inputs.py

commit 93bf488fb95a26f6dce57883d3f59bd50aff99fc
```

```
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sun Feb 14 00:27:13 2021 +0100

    chenged plotting

commit 8423edbd224f9e69619e9197259865a919b4da36
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sun Feb 14 00:25:31 2021 +0100

    added test cases for geometry patch optimizer

commit a453401c5a34ec696828a0544c7be0c8f5172a2d
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sat Feb 6 23:48:46 2021 +0100

    added plots

commit 87367c092da6aecb12c9e43c3832feb044fa2f9f
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sat Feb 6 23:21:33 2021 +0100

    added plots for time analysis

commit 9162e60f8033f75b872838a1b06ff8f8f1f43c9f
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Fri Feb 5 17:50:02 2021 +0100

    removed some files and added patch test

commit aede9004fedf593563c1892acd93ca49db3432ef
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Fri Feb 5 17:38:06 2021 +0100

    did time analysis and log is generated

commit ef8a59ff4bf1965a6b631ba53c8d92810b3fc511
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sat Jan 30 14:29:44 2021 +0100

    patch test
```

```
commit f558fffd5158a5df479324009b943854ebac401f
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sat Jan 30 14:24:27 2021 +0100

    added time analysis and some test cases

commit 2bf2c8bbf28c199f8c6542e68fe081b7e3d097ba
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sat Jan 23 03:39:08 2021 +0100

    added visualization and tested optimizers

commit b64becceeb6bbf008ea0121b99e0b62a9e7b2805
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Jan 4 23:56:23 2021 +0100

    added pyvitsa visualization scripts

commit 6b065397830c28c2ee59c6e40fd63602f847174c
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Jan 4 23:54:45 2021 +0100

    added MMA along with test cases

commit d1f3564e9d2c293dc3d0b009de0993ffd4995568
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Thu Dec 31 01:22:45 2020 +0100

    made changes to optimizer

commit c17c29e5b0a31b661dc2f7eccd16d7245f3ae5cf
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Wed Dec 30 02:10:22 2020 +0100

    added optimiser and testcases

commit 1c6c587f1b8bcf9c9b141e1bff55896babc50b63
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sat Dec 5 21:12:26 2020 +0100
```

```
    Built connectivity, element routine, boundary cond

commit d3c623e0061bf9f24fcd48148725ce46d378bb16
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Fri Nov 27 21:21:54 2020 +0100

    added test case for jacobian

commit 2342b1a3b8a8fbe873ef4c5ae5d41ff3ec5cc9d3
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Thu Nov 26 22:04:49 2020 +0100

    added jacobian

commit 65c4b6fa9b60c61a550afc1e9861743ac486c8c3
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Nov 23 22:55:33 2020 +0100

    added gauss quadrature with testcases

commit c5bc650dc2ea65ee171b596c178441d2dff499f4
Merge: 1e17b8f e73f962
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Nov 23 22:01:19 2020 +0100

    Merge branch 'main' of https://github.com/viswambhar-yasa/IGTO into main

commit 1e17b8fbd390163b72a1b8c2c81a5dced64e26c6
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Nov 23 21:59:53 2020 +0100

    added trilinear basis function with testcases

commit e73f96234dee715d7f43e33fe50fd8fc809480c3
Author: YasaV9 <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Nov 23 17:18:25 2020 +0100

    Create python-app.yml
```

```
commit 9b37cf0fcbf0f7958a4b164359a62531683db707
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Nov 23 16:51:44 2020 +0100

    added Bsplin efunctions and it's derivatives along with testcases

commit 8e4b143564472dc7482e93c5ae667ec181c18d9e
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Mon Nov 23 16:50:56 2020 +0100

    added geometry and input python files

commit b1aa366b844b865b7f0a1be0605851259fd91d54
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sun Nov 22 18:30:56 2020 +0100

    added bspline basis along with test cases

commit 83f39fd010b81bfdd13f7aebbfcd7d6148a896f3
Author: viswambhar-yasa <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sun Nov 22 15:12:45 2020 +0100

    added input class which buids input parmaters

commit 3aa010832cb58e8c8b6e98367e897b0c3fc454c2
Author: YasaV9 <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Sat Nov 7 09:46:31 2020 +0100

    Add files via upload

    Initial commit -implemented a function which generates
    knot vector based on control points and degree.

commit cfe64604302fa76e0298ddc4a10dc805903e8ee5
Author: YasaV9 <58439259+viswambhar-yasa@users.noreply.github.com>
Date:   Fri Oct 30 13:26:58 2020 +0100

    Initial commit
```

References

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

[2] M. Okruta, *Three-dimensional topology optimization of statically loaded porous and multi-phase structures.* PhD thesis, University of Birmingham, 2014.

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

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

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

[6] K. Liu and A. Tovar, "An efficient 3d topology optimization code written in matlab," *Structural and Multidisciplinary Optimization*, vol. 50, 12 2014.

[7] K. Svanberg, "The method of moving asymptotes—a new method for structural optimization," *International Journal for Numerical Methods in Engineering*, vol. 24, no. 2, pp. 359–373, 1987.

[8] R. L. Taylor, J. C. Simo, O. C. Zienkiewicz, and A. C. H. Chan, "The patch test—a condition for assessing fem convergence," *International Journal for Numerical Methods in Engineering*, vol. 22, no. 1, pp. 39–62, 1986.

[9] M. Tavakkoli, B. Hassani, and H. Ghasemnejad, "Isogeometric topology optimization of structures by using mma," *INTERNATIONAL JOURNAL OF OPTIMIZATION IN CIVIL ENGINEERING*, vol. 3, 01 2013.