TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Personal Programming Project report on

Implementation of Health Monitoring Prognosis of 1D Structure using Spectral Finite Element Method

by
Venkatasubramanian Sundaramoorthy
64143

Guided by
Dr. Geralf Hütter

April 28, 2020

1 Introduction

1.1 Motivation

Wave propagation and vibration techniques are used to evaluate the health of the structure. These techniques are applied in Structural Health Monitoring. Based on the dynamic characteristics, the damage characteristics are determined, known as the inverse problem. That drives me to analyze wave propagation in 1-dimensional Structure.

1.2 Aim and Problem definition

Wave propagation in a 1-dimensional element in Heterogeneous medium of length l and density ρ with force applied in the middle of the structure is analyzed by implementing Spectral Finite Element with a polynomial of order $n = 2$.
Random Heterogeneous model is described by the variation of wave velocity and density in particular elements.
Damage has been added randomly in the 1-dimensional structure and based on the structural response the location of the damage is recognized.

2 Theory

2.1 Structural Health Monitoring (SHM)

Structural Health Monitoring is a process of accessing the state of health of a structure and predicting its remaining life.(Gopalakrishnan, Ruzzene, & Hanagud, 2011)
The need for Structural Health monitoring is that the man-made structures will start to degrade once they put into service. Corrosion, wear, and overloads are the factors for degradation. Structural Health Monitoring gives the tools for periodic or constant monitoring of critical structures to prevent the failures and need for corrective action. Structural Health Monitoring has applications in aerospace, mechanical, and civil engineering.

2.2 Spectral Finite Element Method(SFEM)

Spectral Finite Element Method is a powerful tool for solving wave propagation problems.
Two different Spectral FEM approaches.(Kudela, Krawczuk, & Ostachowicz, 2007)
1. Fast Fourier Transformation
2. Time-domain method

by using the Time-Domain approach, wave propagation in a 1-Dimensional structure is carried out. The formulation of Stiffness and the Mass matrix is similar to classical Finite Element formulation.

2.2.1 Elastic Wave equation and Element Matrices

Weak form of 1-Dimensional elastic wave equation is given below,

$$\rho\frac{\partial^2 u}{\partial t^2} = \frac{\partial(\mu\frac{\partial u}{\partial x})}{\partial x} + f \tag{1}$$

(Yao & Xiang, 2014)

where ρ is the density, μ is the shear modulus of the element and f is the applied force.
Approximation of the u(x,t) is given by,

$$u(x,t) \approx \bar{u}(x,t) = \sum_{i=1}^{N+1} u_i(t) l_i(x) \tag{2}$$

(Kudela et al., 2007)

where N order of Lagrange polynomial, $u_i(t)$ coefficients of displacements, $l_i(x)$ shape function. Ordinary Differential equation of wave propagation problem is reduced to Matrix form mentioned below (damping is not considered),

$$M\frac{\partial^2 u(t)}{\partial t^2} + Ku(t) = f(t) \tag{3}$$

(Kudela et al., 2007)

where M is the Global Mass Matrix and K is the Global Stiffness Matrix,and f the vector of time dependent excitation signal.
Corresponding element stiffness and mass matrix and Force vectors are mentioned below.

$$M^e = \int_{-1}^{1} \rho(x) l_j(x) l_i(x) dx \approx \sum_{i=1}^{N+1} w_i \rho(\xi) l_j(\xi) l_i(\xi) J d\xi \tag{4}$$

$$K^e = \int_{-1}^{1} \mu(x) \partial x l_j(x) \partial x l_i(x) dx \approx \sum_{i=1}^{N+1} w_i \mu(\xi) \partial(\xi) l_j(\xi) \partial(\xi) l_i(\xi) \frac{d\xi}{dx} d\xi \tag{5}$$

$$f^e = \int_{-1}^{1} l_i(x) f(x) dx \approx \sum_{i=1}^{N+1} w_i l_i(\xi) f(\xi) J d\xi \tag{6}$$

(Kudela et al., 2007)

where l_i represents Lagrange polynomial, μ represents element shear modulus, ρ is the element density , J is the Jacobian, $\partial(\xi) l_i(\xi)$ is the derivative of the Lagrange Polynomial, w_i is the weight of the Gauss Lobatto Legendre points. μ and ρ varies inside the element which makes it difficult to find an analytical solution.(Gopalakrishnan et al., 2011)
Jacobian is given by, (from the NFEM lectures)

$$J = \frac{dx}{d\xi} \tag{7}$$

2.2.2 Gauss Lobatto Legendre Points (GLL)

Element Matrices K^e, M^e, F^e are calculated numerically using GLL integration rule. Roots and the weights of the GLL points is found by,

$$w_i = \frac{2}{n(n-1)[P_{n-1}(\xi_i)]^2} \tag{8}$$

(Kudela et al., 2007)

$$P'_n(x) = \frac{n}{x^2-1}[xP_n - P_{n-1}] \tag{9}$$

(Gopalakrishnan et al., 2011)

where n is the order of the polynomial, and P_n is the Lagrange polynomial. Increase in Polynomial order will cause the densification problem at the boundary points in the element. In realistic condition the order of the polynomial should be less than n$\leq$4.

2.2.3 Lagrange Polynomial and First derivative of Lagrange Polynomial

Lagrange Polynomials are used as the shape function in the element level.

$$l_i^n(\xi) = \Pi_{j \neq i}^{n+1} \frac{\xi - \xi_j}{\xi_i - \xi_j} \qquad wherei, j = 1, 2, ..., n+1 \tag{10}$$

(Gopalakrishnan et al., 2011)

Where ξ_i is taken based on the GLL points. Lagrange polynomial has a fundamental property as shown below. This will decrease the Mass matrix into diagonal which can be used in solution and for a higher dimension this property helps to calculate the inverse matrix easily.

$$l_i^n(\xi_j) = \delta_{ij} \tag{11}$$

(Kudela et al., 2007)

The derivative of the Lagrange polynomial is computed by using the following formula and applied in the calculation of the Element Stiffness matrix.

$$d\xi l_k(\xi_i) = \sum_{j=0}^{N} d_i j l_k(\xi_j) \qquad k = 0, ..., N \tag{12}$$

$$d_{ij} = \begin{Bmatrix} \frac{-1}{4}N(N+1), & \text{if } i = j = 0 \\ \frac{L_N(\xi_i}{L_N(\xi_j} \frac{1}{\xi_i - \xi_j}, & \text{if } 0 \leq i \leq N; 0 \leq j \leq N; \quad i \neq j \\ 0, & \text{if } 1 \leq \ i = j \leq N-1 \\ \frac{1}{4}N(N+1), & \text{if } i = j = N \end{Bmatrix} \tag{13}$$

(Funaro, 1993)

Where n is the order of the polynomial, L_N is the Legendre polynomial,$l_k(\xi_j)$ is the Lagrange Polynomial.

2.2.4 Legendre Polynomials

The Legendre polynomials are applied to determine the first derivative of the Lagrange polynomial. The formula for the Legendre polynomial is given below,

$$P_n = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n \tag{14}$$

(Aboites, 2019)

The value of P at $n = 0$ is 1 and at $n = 1$ is x. where n is the order of the polynomial.

2.2.5 Random Heterogeneous Model

Heterogeneous model is generally represented as the variation in the wave velocity and the density to a certain grid points.(Kasahara, Korneev, & Zhdanov, 2010)

2.2.6 Force

Gaussian wave is used as the force vector of the time dependent signal.(Gopalakrishnan et al., 2011). The formula for calculating Gaussian wave is shown below,

$$f = u_0 \exp -(\frac{t^2}{\sigma^2}) \tag{15}$$

(Gopalakrishnan et al., 2011)

where u_o is the amplitude of displacement,t is the time period and σ is the standard deviation. The first derivative of Gaussian wave is used in this problem and it is given below,

$$f = -2\frac{1}{\sigma^2}u_0 t \exp -(\frac{t^2}{\sigma^2}) \tag{16}$$

2.2.7 Newmark Scheme

The solution of the problem is determined by applying the Newmark scheme (Kudela et al., 2007) by using NFEM Lectures the formula for the Newmark scheme is shown below,

$$\ddot{u} = \frac{u^{new} - 2u + u^{old}}{dt^2} \tag{17}$$

$$u^{new} = u(t+dt), u = u(t), u^{old} = u(t-dt) \tag{18}$$

The solution of the matrix formation is given by,

$$u^{new} = dt^2[M^{-1}[f(t) - Ku(t))]] + 2u - u^{old} \tag{19}$$

2.3 Damage Introduction in the Structure

The flaws in the structure will reduce the stiffness of the structure. Modeling the flaws in the materials is implemented by adjusting the properties of the material. Modification of the material properties P (Young's Modulus, the shear modulus) provided in the below equation,

$$P = \alpha P \tag{20}$$

(Gopalakrishnan et al., 2011)

α is the damage parameter which takes the values as $\alpha \leq 1$ If there is no damage then α is zero and if the element if fully damaged α is 1

2.3.1 Damage Vector

In wave propagation, the structure is changed by the fluctuations in the stiffness matrix. K_d and K_0 are the damaged and the healthy structure stiffness matrices, and similarly, M_d and M_0 are the damaged and the healthy structure mass matrices.

$$\Delta K u_d = D \tag{21}$$

(Carrion, Doyle, & Lozano, 2003)

$$\Delta K = K_0 - K_d \tag{22}$$

(Carrion et al., 2003)

The vector D has the data about damage, then it is a direct process to calculate the damage information. The value of the damage vector is calculated by the changes in the stiffness matrix as mentioned in the above equation. The response of the damaged structure is governed by the following equation,

$$M_0[\Delta\ddot{u}] + K_0[\Delta u] = \Delta K u_d \tag{23}$$

(Carrion et al., 2003)

where $\Delta\ddot{u} = \ddot{u}_d - \ddot{u}_0$ and $\Delta u = u_d - u_0 . \ddot{u}_d$ is the damaged structure acceleration and $\ddot{u}_0$ is the healthy structure acceleration, which can be calculated using the Newmark scheme. similarly, u_d is the damaged structure displacement and the u_0 is the healthy structure displacement

2.3.2 Root Mean Square Error

By calculating Root mean square error between the true and the calculated damage vector, the location of the damage is predicted by the root mean square value.

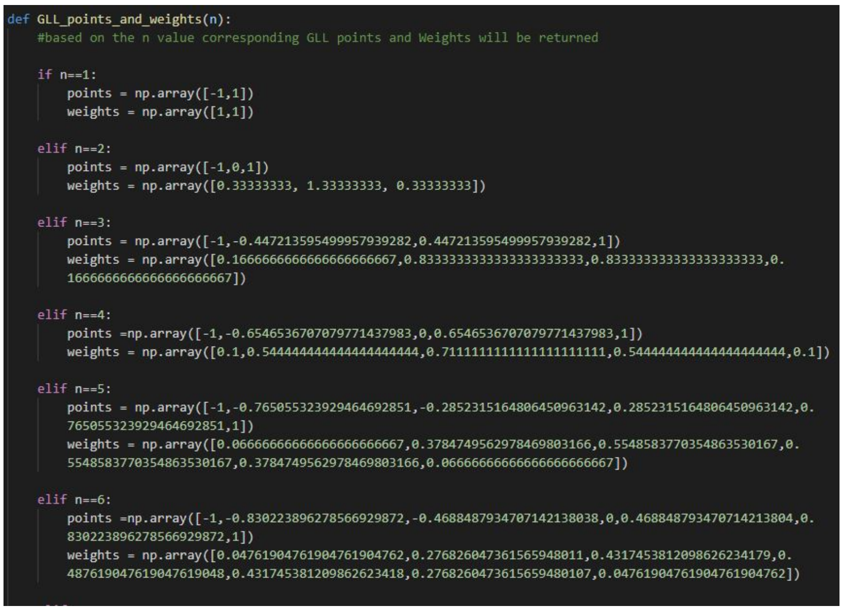
$$rmse = \sqrt{\frac{Predicted\ damage\ vector - True\ damage\ vector}{n}} \tag{24}$$

3 Numerical Methods and Implementation Details

In the following section, the details of the numerical techniques are explained.

3.1 Gauss Lobatto Legendre Points

Gauss Lobatto Legendre points are used for calculating the Lagrange polynomials in the element and Legendre polynomials for the calculation of the first derivative of the Lagrange polynomial. The GLL function takes the polynomial order from the user and returns the corresponding weights and the roots. The values are taken from (for PPP, 2020)
Polynomial order up to $n = 9$ is used and in the following picture polynomial order, up to $n = 6$ is given.

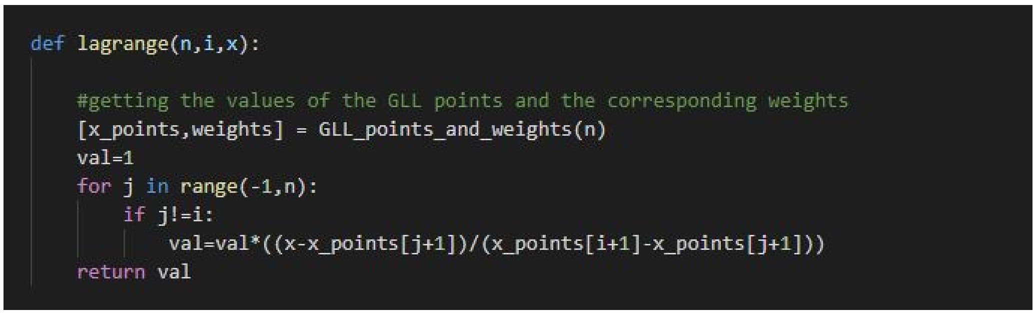


```
def GLL_points_and_weights(n):
    #based on the n value corresponding GLL points and Weights will be returned

    if n==1:
        points = np.array([-1,1])
        weights = np.array([1,1])

    elif n==2:
        points = np.array([-1,0,1])
        weights = np.array([0.33333333, 1.33333333, 0.33333333])

    elif n==3:
        points = np.array([-1,-0.4472135954999579392 82,0.4472135954999579392 82,1])
        weights = np.array([0.16666666666666666667,0.83333333333333333333,0.83333333333333333333,0.
        16666666666666666667])

    elif n==4:
        points =np.array([-1,-0.6546536707079771437983,0,0.6546536707079771437983,1])
        weights = np.array([0.1,0.5444444444444444444 4,0.71111111111111111111,0.5444444444444444444 4,0.1])

    elif n==5:
        points = np.array([-1,-0.7650553239294646928 51,-0.2852315164806450963142,0.2852315164806450963142,0.
        7650553239294646928 51,1])
        weights = np.array([0.06666666666666666666 7,0.3784749562978469803166,0.5548583770354863530167,0.
        5548583770354863530167,0.3784749562978469803166,0.06666666666666666666 7])

    elif n==6:
        points =np.array([-1,-0.8302238962785669298 72,-0.4688487934707142138038,0,0.4688487934707142138 04,0.
        8302238962785669298 72,1])
        weights = np.array([0.04761904761904761904762,0.2768260473615659480 11,0.4317453812098626234179,0.
        4876190476190476190 48,0.4317453812098626234 18,0.2768260473615659480107,0.04761904761904761904762])
```

Figure 1: Function to return the polynomial roots and the corresponding weights based on the polynomial order n

3.2 Lagrange Polynomial

Lagrange polynomials are used as the shape function in the element level. The following picture shows the function to compute the Lagrange polynomial of order n.



```
def lagrange(n,i,x):

    #getting the values of the GLL points and the corresponding weights
    [x_points,weights] = GLL_points_and_weights(n)
    val=1
    for j in range(-1,n):
        if j!=i:
            val=val*((x-x_points[j+1])/(x_points[i+1]-x_points[j+1]))
    return val
```

Figure 2: Function which returns the Lagrange polynomial of order n

Lagrange polynomial of order $n = 5$ is presented below and the densification in the boundaries are identified,.

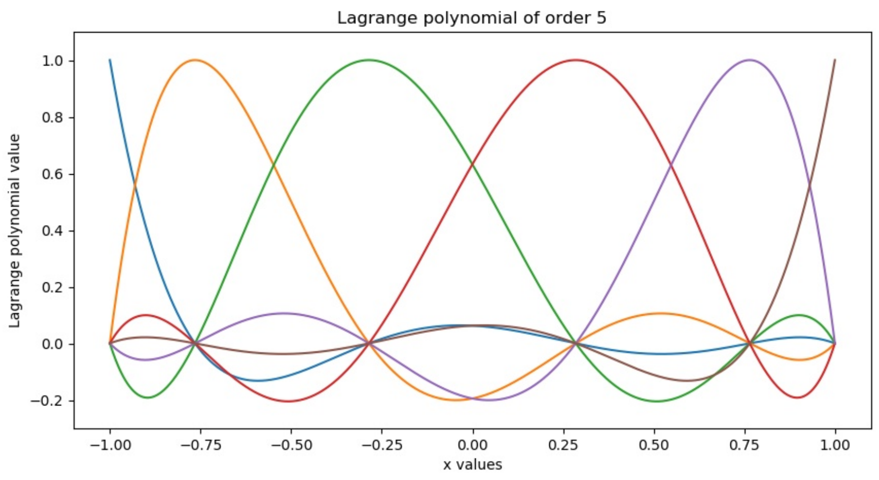
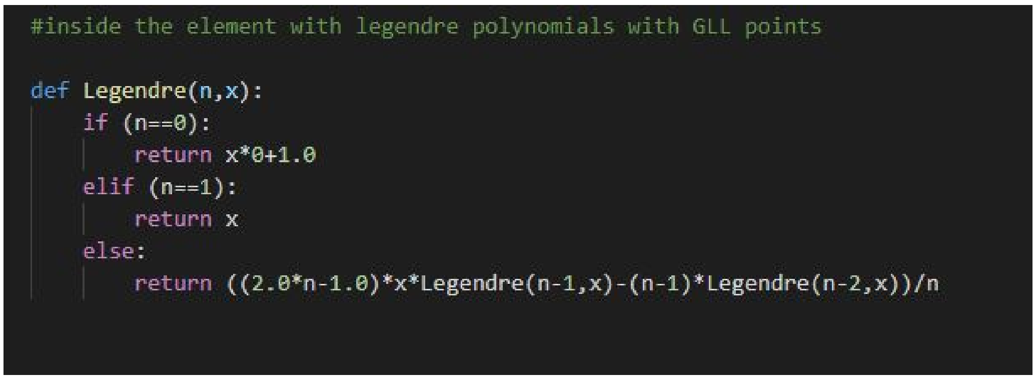


Figure 3: Lagrange polynomial of order $n = 5$ and densification in the boundaries

The densification around the boundaries will be added, if the polynomial order n is higher than 4, to resolve the problem the time step needs to be increased which will be expensive. In realistic simulation polynomial order $n \leq 4$ is applied.

3.3 Legendre Polynomials and First derivative of Lagrange polynomial

Legendre polynomials are utilized to determine the first derivatives of the Lagrange polynomials used in the calculation of the K^e at the element level.
Legendre Polynomial function accepts the polynomial order n and the corresponding GLL points x from the GLL function



```
#inside the element with legendre polynomials with GLL points

def Legendre(n,x):
    if (n==0):
        return x*0+1.0
    elif (n==1):
        return x
    else:
        return ((2.0*n-1.0)*x*Legendre(n-1,x)-(n-1)*Legendre(n-2,x))/n
```

Figure 4: Legendre polynomial Function

Legendre polynomials of order $n = 5$ is given below,

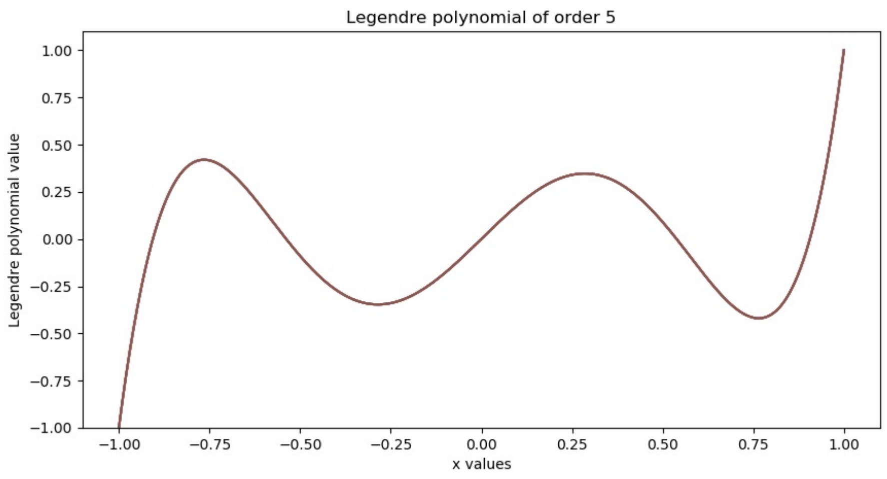
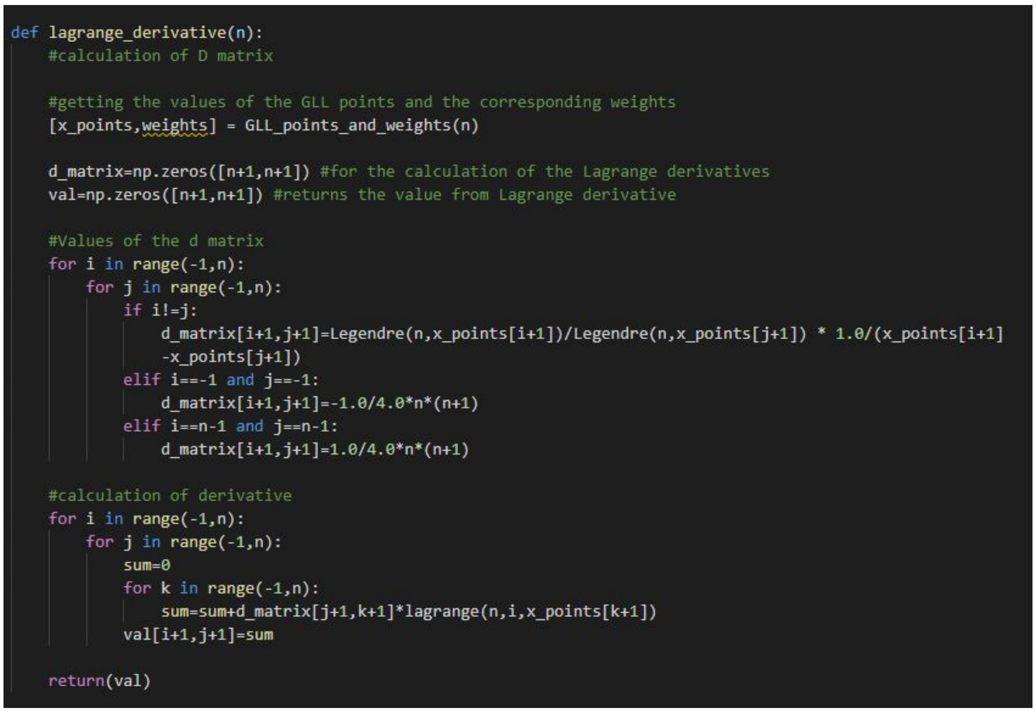


Figure 5: Legendre polynomial of order $n = 5$

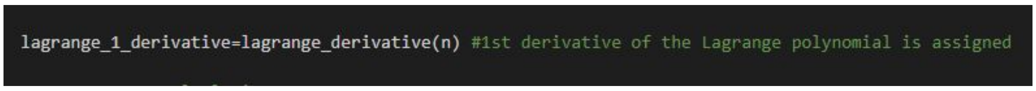
The derivatives of the Lagrange polynomials are computed by utilizing the following functions. The function takes the order of the polynomial n and returns the first derivative.



```python
def lagrange_derivative(n):
    #calculation of D matrix

    #getting the values of the GLL points and the corresponding weights
    [x_points,weights] = GLL_points_and_weights(n)

    d_matrix=np.zeros([n+1,n+1]) #for the calculation of the Lagrange derivatives
    val=np.zeros([n+1,n+1]) #returns the value from Lagrange derivative

    #Values of the d matrix
    for i in range(-1,n):
        for j in range(-1,n):
            if i!=j:
                d_matrix[i+1,j+1]=Legendre(n,x_points[i+1])/Legendre(n,x_points[j+1]) * 1.0/(x_points[i+1]
                -x_points[j+1])
            elif i==-1 and j==-1:
                d_matrix[i+1,j+1]=-1.0/4.0*n*(n+1)
            elif i==n-1 and j==n-1:
                d_matrix[i+1,j+1]=1.0/4.0*n*(n+1)

    #calculation of derivative
    for i in range(-1,n):
        for j in range(-1,n):
            sum=0
            for k in range(-1,n):
                sum=sum+d_matrix[j+1,k+1]*lagrange(n,i,x_points[k+1])
            val[i+1,j+1]=sum

    return(val)
```

Figure 6: Lagrange polynomial derivative Function

The values of the first derivative of Legendre polynomials are assigned is shown below,

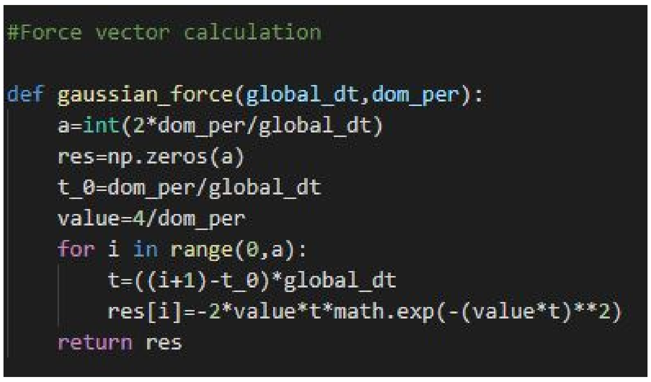


```python
lagrange_1_derivative=lagrange_derivative(n) #1st derivative of the Lagrange polynomial is assigned
```

Figure 7: Lagrange polynomial derivative Assignment

3.4 Force Calculation

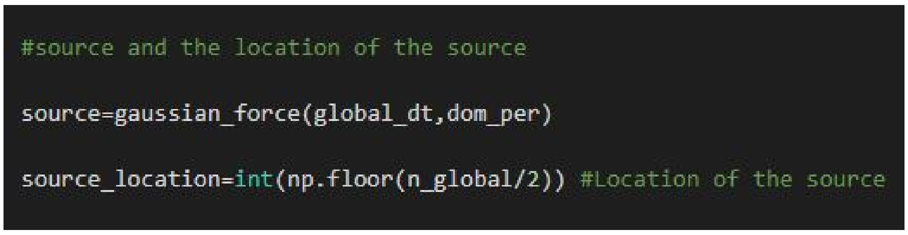
A Gaussian function is utilized for the study of wave propagation and it is calculated by using the values of wave frequency and the time. A Gaussian function is shown below,



```python
#Force vector calculation

def gaussian_force(global_dt,dom_per):
    a=int(2*dom_per/global_dt)
    res=np.zeros(a)
    t_0=dom_per/global_dt
    value=4/dom_per
    for i in range(0,a):
        t=((i+1)-t_0)*global_dt
        res[i]=-2*value*t*math.exp(-(value*t)**2)
    return res
```

Figure 8: Gaussian function

The location of the source and the assignment of the source is shown below,

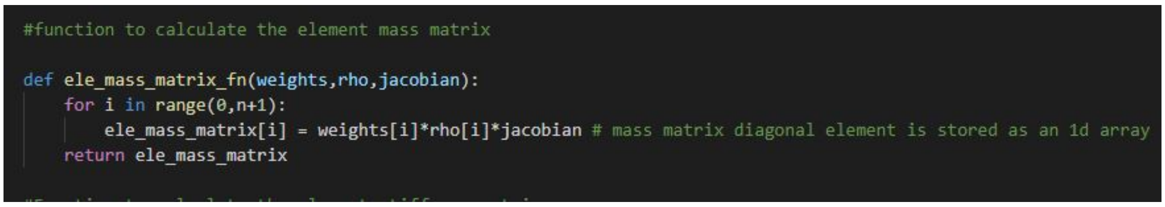


```
#source and the location of the source

source=gaussian_force(global_dt,dom_per)

source_location=int(np.floor(n_global/2)) #Location of the source
```

Figure 9: Source Function call and the location of the source

3.5 Element Mass and Stiffness Matrix

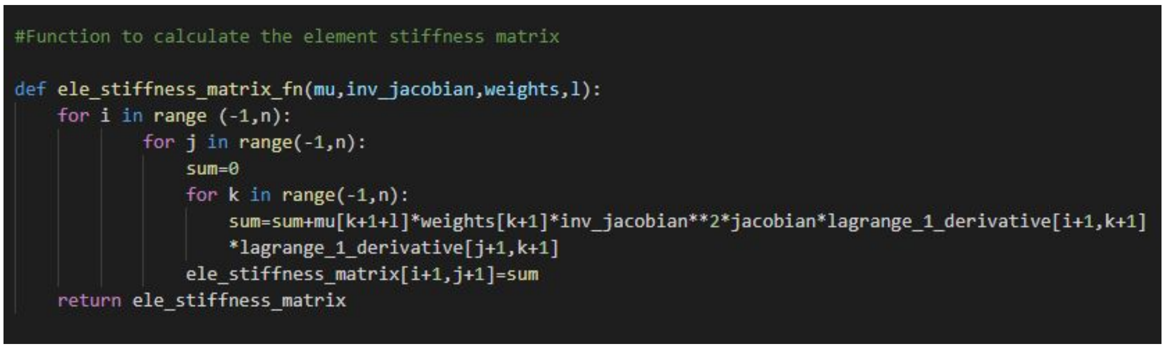
The Element Mass matrix function is shown below, which takes the weights of the values using GLL function, rho, and the jacobian values and returns the element mass matrix.
Mass matrix is diagonal because of the property of the Lagrange polynomial described before. The below function calculates the main diagonal elements of the mass matrix.



```
#function to calculate the element mass matrix

def ele_mass_matrix_fn(weights,rho,jacobian):
    for i in range(0,n+1):
        ele_mass_matrix[i] = weights[i]*rho[i]*jacobian # mass matrix diagonal element is stored as an 1d array
    return ele_mass_matrix
```

Figure 10: Element Mass matrix Function

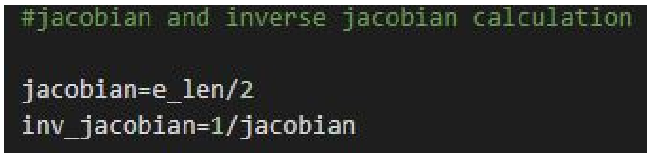
The Element stiffness matrix is computed by the below-mentioned function which takes the value of density μ, the inverse of jacobian, weights of the GLL points, and the first derivative of Lagrange polynomial l.
Element stiffness matrix function is shown below,



```
#Function to calculate the element stiffness matrix

def ele_stiffness_matrix_fn(mu,inv_jacobian,weights,l):
    for i in range (-1,n):
            for j in range(-1,n):
                sum=0
                for k in range(-1,n):
                    sum=sum+mu[k+1+l]*weights[k+1]*inv_jacobian**2*jacobian*lagrange_1_derivative[i+1,k+1]
                    *lagrange_1_derivative[j+1,k+1]
                ele_stiffness_matrix[i+1,j+1]=sum
    return ele_stiffness_matrix
```

Figure 11: Element Stiffness matrix Function

The jacobian and the inverse jacobian is calculated by the below code,

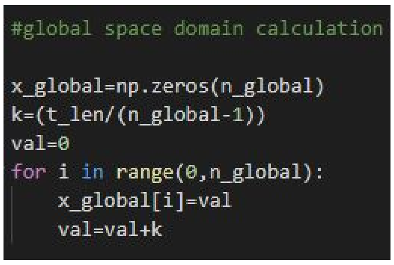


```
#jacobian and inverse jacobian calculation

jacobian=e_len/2
inv_jacobian=1/jacobian
```

Figure 12: Jacobian and Inverse Jacobian

3.6 Global Coordinates

Global Coordinates are calculated by the below-mentioned code. And these values are utilized in plotting the graph.

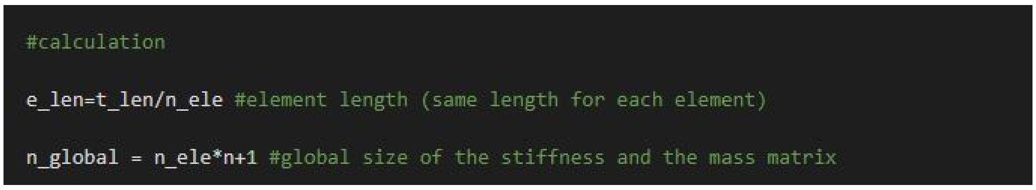


```
#global space domain calculation

x_global=np.zeros(n_global)
k=(t_len/(n_global-1))
val=0
for i in range(0,n_global):
    x_global[i]=val
    val=val+k
```

Figure 13: Global Coordinates calculation

3.7 Basic Calculation

The below-mentioned code will explain the calculation of the size of the Global Mass and Stiffness matrices n_{global}, Length of the element e_{len}.

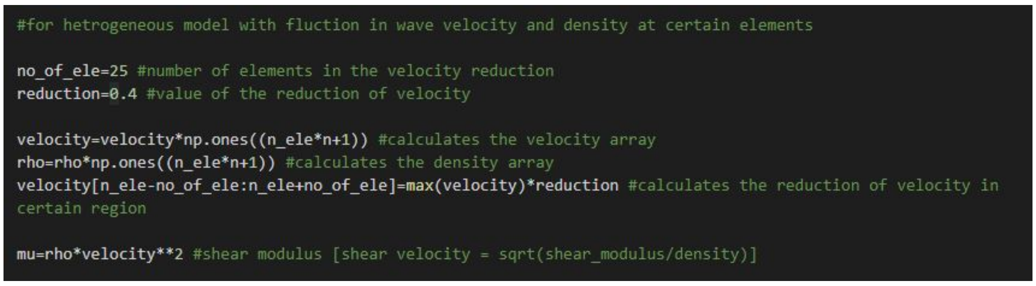


```
#calculation

e_len=t_len/n_ele #element length (same length for each element)

n_global = n_ele*n+1 #global size of the stiffness and the mass matrix
```

Figure 14: Element length and size of Global Matrices calculation

3.8 Heterogeneous medium

As mentioned in the earlier section Heterogeneous medium is introduced by the fluctuation in the wave velocity and the density to particular elements.
The below code shows the Heterogeneous medium calculation.

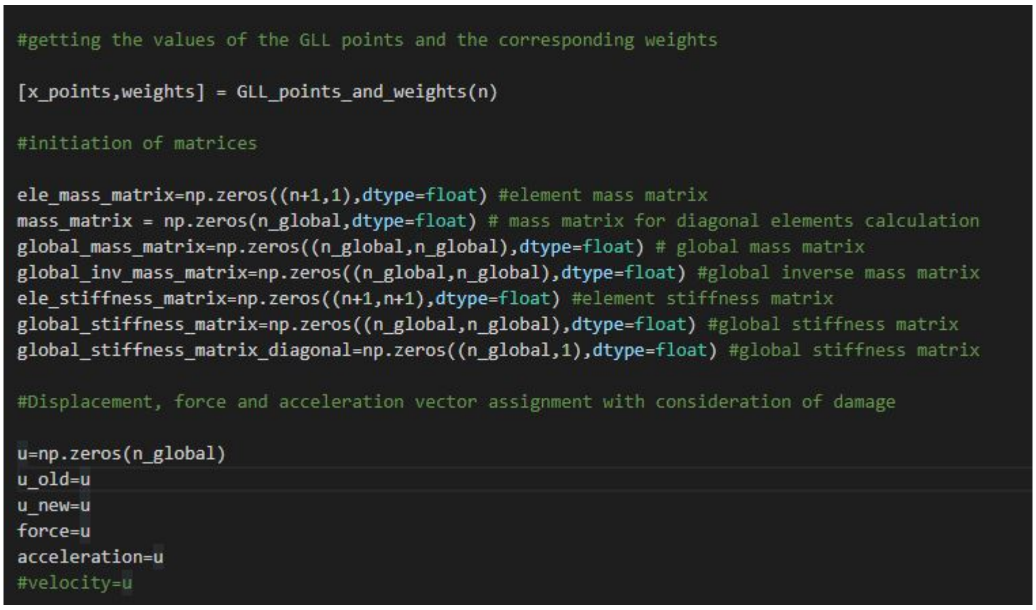


```
#for hetrogeneous model with fluction in wave velocity and density at certain elements

no_of_ele=25 #number of elements in the velocity reduction
reduction=0.4 #value of the reduction of velocity

velocity=velocity*np.ones((n_ele*n+1)) #calculates the velocity array
rho=rho*np.ones((n_ele*n+1)) #calculates the density array
velocity[n_ele-no_of_ele:n_ele+no_of_ele]=max(velocity)*reduction #calculates the reduction of velocity in
certain region

mu=rho*velocity**2 #shear modulus [shear velocity = sqrt(shear_modulus/density)]
```

Figure 15: Velocity and density calculation for Heterogeneous medium

3.9 Matrices Initiation

The Element mass, Stiffness matrices, and the corresponding Global Mass and Stiffness matrices and Displacement vectors are initiated in the below-mentioned code.

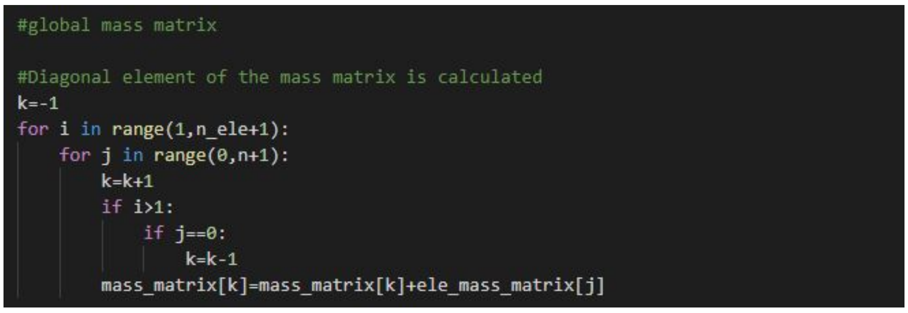


```
#getting the values of the GLL points and the corresponding weights

[x_points,weights] = GLL_points_and_weights(n)

#initiation of matrices

ele_mass_matrix=np.zeros((n+1,1),dtype=float) #element mass matrix
mass_matrix = np.zeros(n_global,dtype=float) # mass matrix for diagonal elements calculation
global_mass_matrix=np.zeros((n_global,n_global),dtype=float) # global mass matrix
global_inv_mass_matrix=np.zeros((n_global,n_global),dtype=float) #global inverse mass matrix
ele_stiffness_matrix=np.zeros((n+1,n+1),dtype=float) #element stiffness matrix
global_stiffness_matrix=np.zeros((n_global,n_global),dtype=float) #global stiffness matrix
global_stiffness_matrix_diagonal=np.zeros((n_global,1),dtype=float) #global stiffness matrix

#Displacement, force and acceleration vector assignment with consideration of damage

u=np.zeros(n_global)
u_old=u
u_new=u
force=u
acceleration=u
#velocity=u
```

Figure 16: Matrices Initiation and Displacement vector Initiation

3.10 Global Mass Matrix

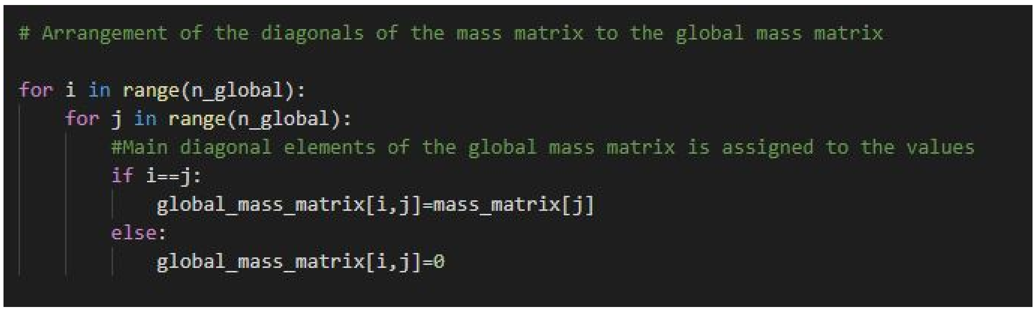
The diagonal elements of the global mass matrix is calculated by the below loop, The Global



```
#global mass matrix

#Diagonal element of the mass matrix is calculated
k=-1
for i in range(1,n_ele+1):
    for j in range(0,n+1):
        k=k+1
        if i>1:
            if j==0:
                k=k-1
        mass_matrix[k]=mass_matrix[k]+ele_mass_matrix[j]
```

Figure 17: Diagonal elements of mass matrix

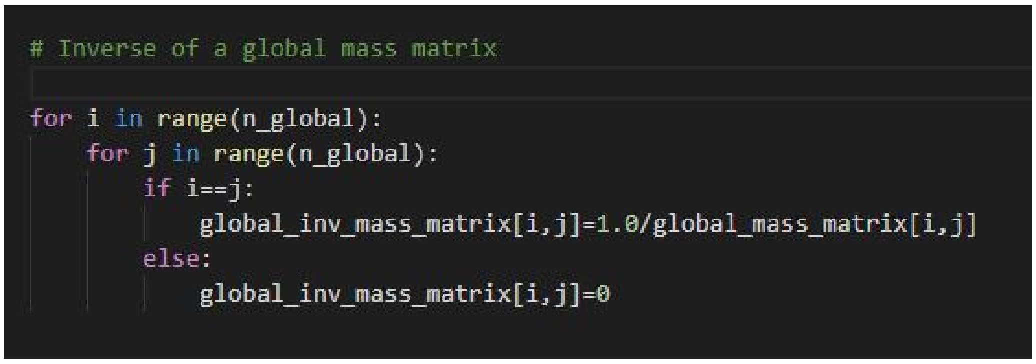
Mass matrix is arranged by using the diagonal elements and the code is mentioned below,



```
# Arrangement of the diagonals of the mass matrix to the global mass matrix

for i in range(n_global):
    for j in range(n_global):
        #Main diagonal elements of the global mass matrix is assigned to the values
        if i==j:
            global_mass_matrix[i,j]=mass_matrix[j]
        else:
            global_mass_matrix[i,j]=0
```

Figure 18: Global Mass matrix is arranged by using the diagonal elements

Inverse of the global mass matrix is calculated by using the global mass matrix and the loops are mentioned below,

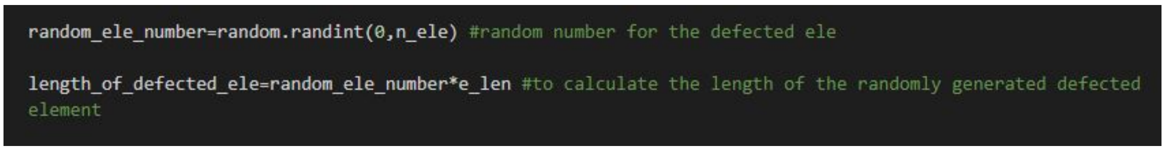


```
# Inverse of a global mass matrix

for i in range(n_global):
    for j in range(n_global):
        if i==j:
            global_inv_mass_matrix[i,j]=1.0/global_mass_matrix[i,j]
        else:
            global_inv_mass_matrix[i,j]=0
```

Figure 19: Inverse of the global mass matrix

3.11 Solution with the consideration of damage

The Damage element number is created randomly and the corresponding location of the damage is calculated by the below-mentioned code.

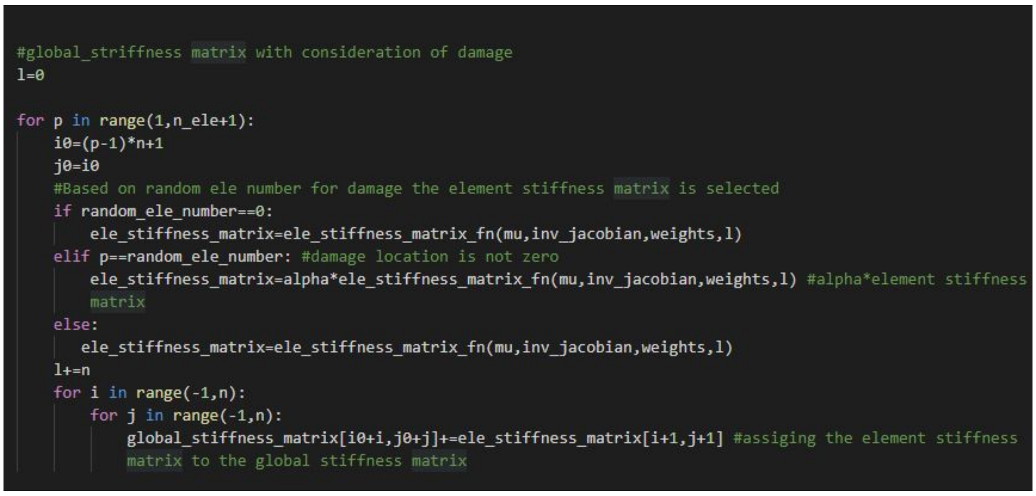


```
random_ele_number=random.randint(0,n_ele) #random number for the defected ele

length_of_defected_ele=random_ele_number*e_len #to calculate the length of the randomly generated defected
element
```

Figure 20: Damage element and the location of the damage

3.11.1 Element Stiffness Matrix function and Calculation of Global Stiffness Matrices

The Global Stiffness matrix is calculated with the consideration of damage. The Damage element number is randomly generated, the element stiffness matrix of the damaged element is decreased by α times. It is arranged to the corresponding location in the global stiffness matrix, which is computed using the below-mentioned code,

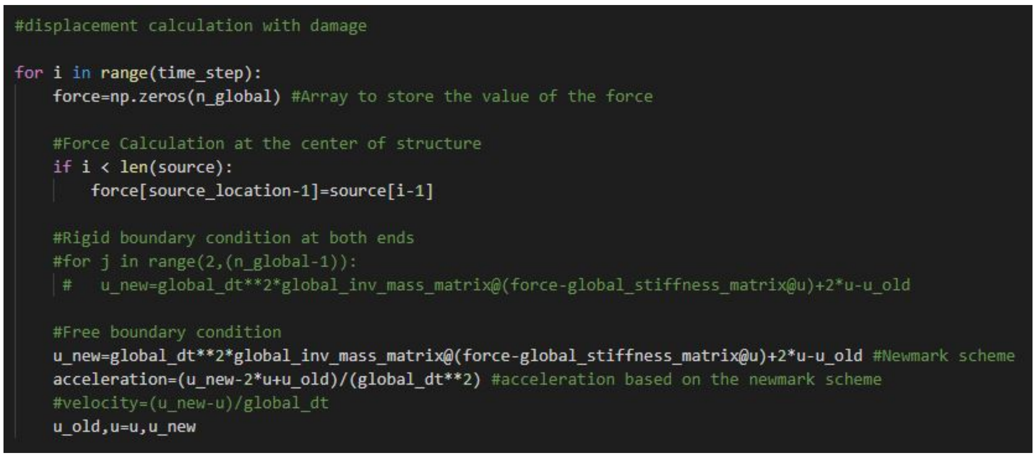


```
#global_striffness matrix with consideration of damage
l=0

for p in range(1,n_ele+1):
    i0=(p-1)*n+1
    j0=i0
    #Based on random ele number for damage the element stiffness matrix is selected
    if random_ele_number==0:
        ele_stiffness_matrix=ele_stiffness_matrix_fn(mu,inv_jacobian,weights,l)
    elif p==random_ele_number: #damage location is not zero
        ele_stiffness_matrix=alpha*ele_stiffness_matrix_fn(mu,inv_jacobian,weights,l) #alpha*element stiffness
        matrix
    else:
       ele_stiffness_matrix=ele_stiffness_matrix_fn(mu,inv_jacobian,weights,l)
    l+=n
    for i in range(-1,n):
        for j in range(-1,n):
            global_stiffness_matrix[i0+i,j0+j]+=ele_stiffness_matrix[i+1,j+1] #assiging the element stiffness
            matrix to the global stiffness matrix
```

Figure 21: Global Stiffness matrix calculation with consideration of damage

3.11.2 Newmark Scheme

With the consideration of damage, Global stiffness matrix is updated and using the Newmark scheme the new displacement values are calculated by the below code (with the source and the location of the source are mentioned before),



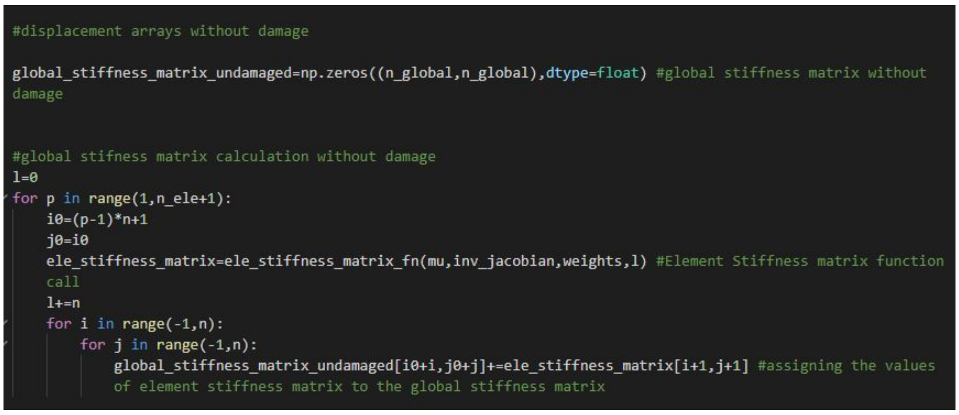
```
#displacement calculation with damage

for i in range(time_step):
    force=np.zeros(n_global) #Array to store the value of the force

    #Force Calculation at the center of structure
    if i < len(source):
        force[source_location-1]=source[i-1]

    #Rigid boundary condition at both ends
    #for j in range(2,(n_global-1)):
    #    u_new=global_dt**2*global_inv_mass_matrix@(force-global_stiffness_matrix@u)+2*u-u_old

    #Free boundary condition
    u_new=global_dt**2*global_inv_mass_matrix@(force-global_stiffness_matrix@u)+2*u-u_old #Newmark scheme
    acceleration=(u_new-2*u+u_old)/(global_dt**2) #acceleration based on the newmark scheme
    #velocity=(u_new-u)/global_dt
    u_old,u=u,u_new
```

Figure 22: Newmark scheme with the consideration of damage

3.12 Solution without consideration of the damage

3.12.1 Global Stiffness Matrix

The true value of the displacement is calculated without considering the damage, as the damage will change the global stiffness matrix.
Global stiffness matrix is calculated without considering the damage is shown below,

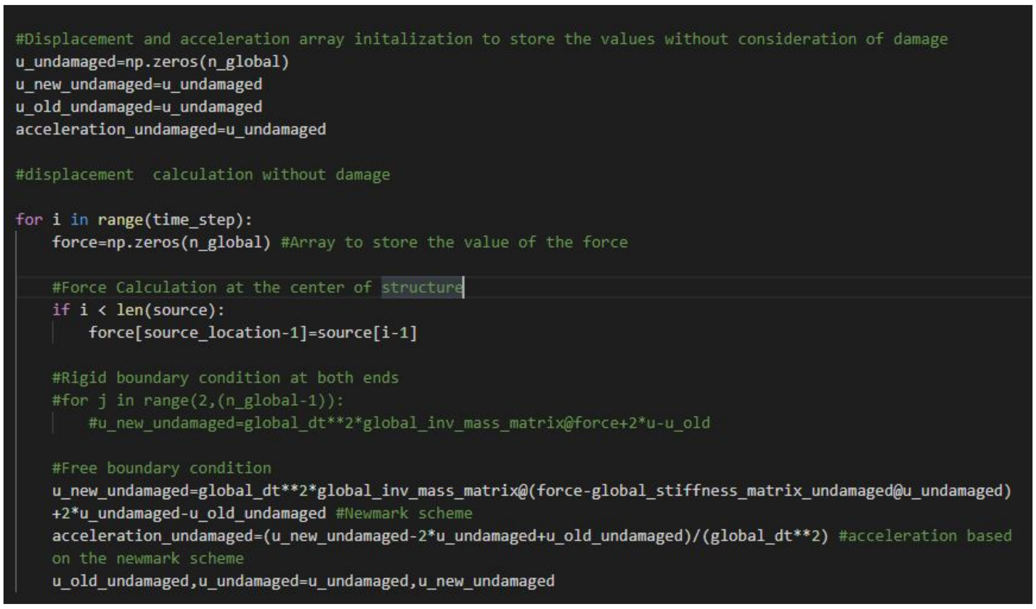


```
#displacement arrays without damage

global_stiffness_matrix_undamaged=np.zeros((n_global,n_global),dtype=float) #global stiffness matrix without
damage


#global stifness matrix calculation without damage
l=0
for p in range(1,n_ele+1):
    i0=(p-1)*n+1
    j0=i0
    ele_stiffness_matrix=ele_stiffness_matrix_fn(mu,inv_jacobian,weights,l) #Element Stiffness matrix function
    call
    l+=n
    for i in range(-1,n):
        for j in range(-1,n):
            global_stiffness_matrix_undamaged[i0+i,j0+j]+=ele_stiffness_matrix[i+1,j+1] #assigning the values
            of element stiffness matrix to the global stiffness matrix
```

Figure 23: Global stiffness matrix without consideration of the damage

3.12.2 Newmark Scheme

Global Stiffness matrix without consideration of the damage, the same Newmark scheme is applied and the displacement is calculated by the below-mentioned code,

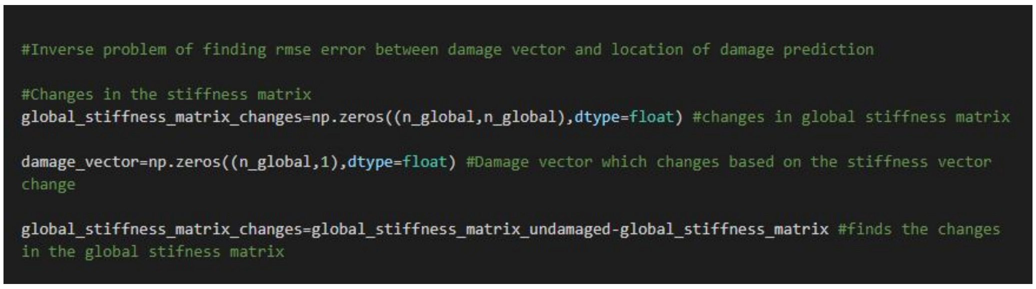


```
#Displacement and acceleration array initalization to store the values without consideration of damage
u_undamaged=np.zeros(n_global)
u_new_undamaged=u_undamaged
u_old_undamaged=u_undamaged
acceleration_undamaged=u_undamaged

#displacement  calculation without damage

for i in range(time_step):
    force=np.zeros(n_global) #Array to store the value of the force

    #Force Calculation at the center of structure
    if i < len(source):
        force[source_location-1]=source[i-1]

    #Rigid boundary condition at both ends
    #for j in range(2,(n_global-1)):
        #u_new_undamaged=global_dt**2*global_inv_mass_matrix@force+2*u-u_old

    #Free boundary condition
    u_new_undamaged=global_dt**2*global_inv_mass_matrix@(force-global_stiffness_matrix_undamaged@u_undamaged)
    +2*u_undamaged-u_old_undamaged #Newmark scheme
    acceleration_undamaged=(u_new_undamaged-2*u_undamaged+u_old_undamaged)/(global_dt**2) #acceleration based
    on the newmark scheme
    u_old_undamaged,u_undamaged=u_undamaged,u_new_undamaged
```

Figure 24: Newmark scheme without consideration of damage

3.13 Damage Vector Calculation

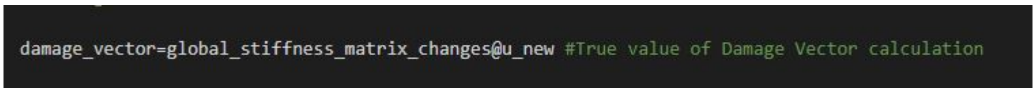
The damage vector has information about the location of the damage and the characteristic of the damage. The damage vector is calculated by calculating the change in the stiffness matrix with the multiplication of the damaged displacement vector. The change in stiffness matrix is the difference between the Global Stiffness matrix with and without consideration of the damage.
The below code will show the change in stiffness matrix,



```
#Inverse problem of finding rmse error between damage vector and location of damage prediction

#Changes in the stiffness matrix
global_stiffness_matrix_changes=np.zeros((n_global,n_global),dtype=float) #changes in global stiffness matrix

damage_vector=np.zeros((n_global,1),dtype=float) #Damage vector which changes based on the stiffness vector
change

global_stiffness_matrix_changes=global_stiffness_matrix_undamaged-global_stiffness_matrix #finds the changes
in the global stifness matrix
```

Figure 25: Change in Stiffness Matrix calculation

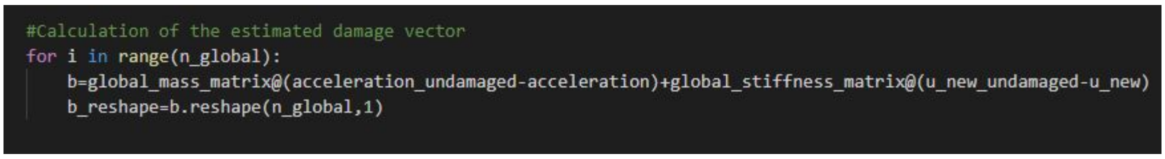
After calculating the change in stiffness matrix the value of Damage vector is calculated by the following code,



```
damage_vector=global_stiffness_matrix_changes@u_new #True value of Damage Vector calculation
```

Figure 26: Damage Vector Calculation

The response of the structure is governed by the equation $M\Delta(\ddot{u}) + K\Delta u$ (Carrion et al., 2003) and it is calculated in the following code, Where, K is the Global stiffness matrix with consideration of damage where the acceleration is calculated by using the Newmark scheme.

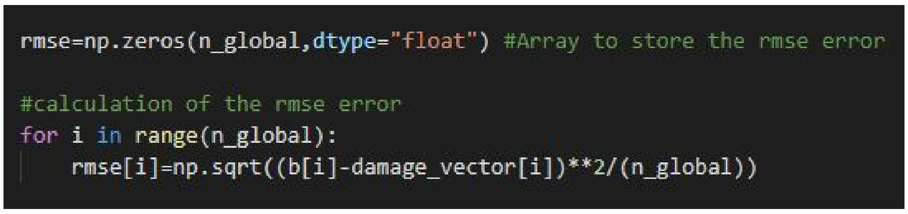


```
#Calculation of the estimated damage vector
for i in range(n_global):
    b=global_mass_matrix@(acceleration_undamaged-acceleration)+global_stiffness_matrix@(u_new_undamaged-u_new)
    b_reshape=b.reshape(n_global,1)
```

Figure 27: Damage Vector Calculation with damaged Stiffness matrix

3.14 Root Mean Square Error (RMSE) between True and Estimated damage Vector

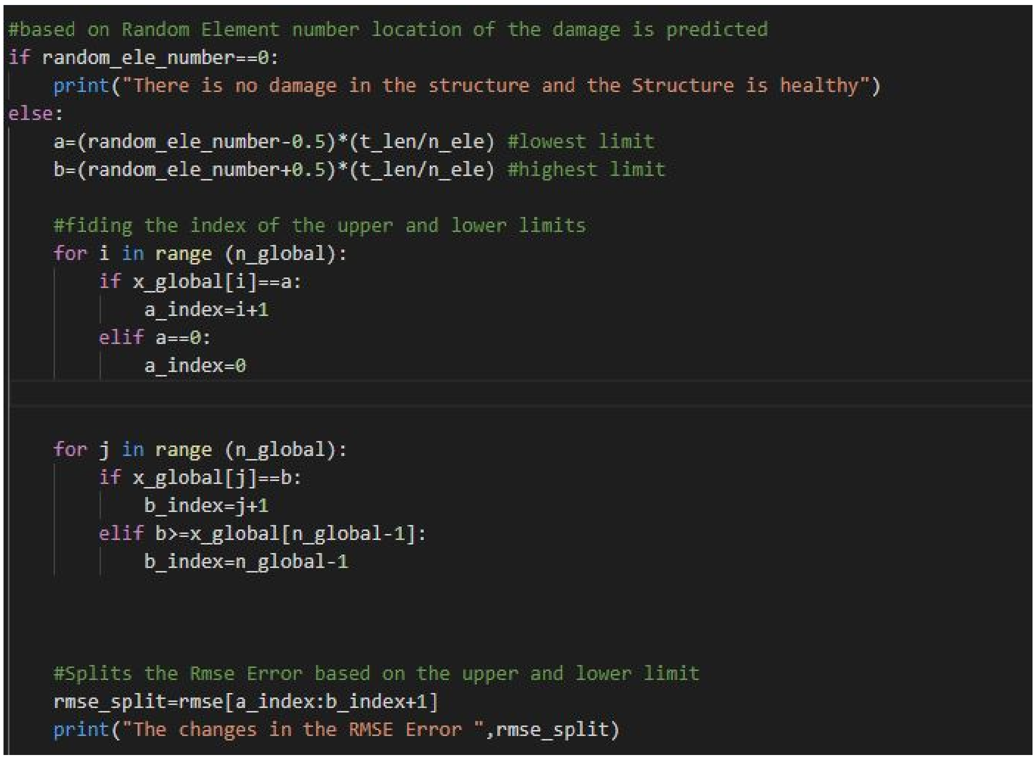
The Root means square error between the true and the estimated damage vector is calculated by the following code.



```
rmse=np.zeros(n_global,dtype="float") #Array to store the rmse error

#calculation of the rmse error
for i in range(n_global):
    rmse[i]=np.sqrt((b[i]-damage_vector[i])**2/(n_global))
```

Figure 28: Root mean square error between True and the Estimated Damage Vector

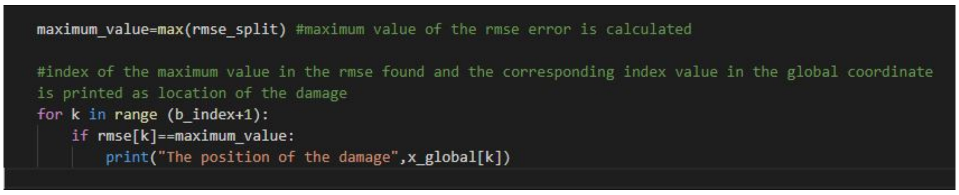
Based on the damage element number two random values are created and the maximum error is calculated it is shown in the following code,



```
#based on Random Element number location of the damage is predicted
if random_ele_number==0:
    print("There is no damage in the structure and the Structure is healthy")
else:
    a=(random_ele_number-0.5)*(t_len/n_ele) #lowest limit
    b=(random_ele_number+0.5)*(t_len/n_ele) #highest limit

    #fiding the index of the upper and lower limits
    for i in range (n_global):
        if x_global[i]==a:
            a_index=i+1
        elif a==0:
            a_index=0


    for j in range (n_global):
        if x_global[j]==b:
            b_index=j+1
        elif b>=x_global[n_global-1]:
            b_index=n_global-1



    #Splits the Rmse Error based on the upper and lower limit
    rmse_split=rmse[a_index:b_index+1]
    print("The changes in the RMSE Error ",rmse_split)
```

Figure 29: Maximum of RMSE value

The maximum value of the Root Mean square is calculated and the corresponding index (Damage Index) in the Root Mean square is calculated. The location of the Damage is the value of the global coordinate at the damage index.
The damage location index and the Location of damage which is calculated by the following code.



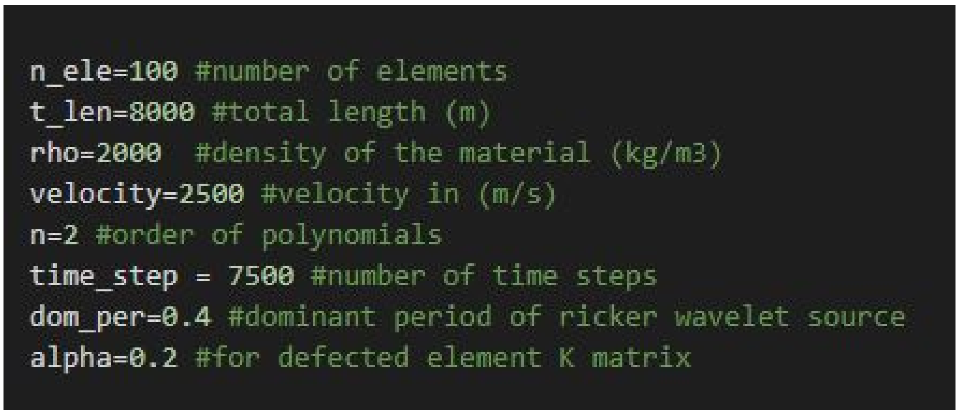
```
maximum_value=max(rmse_split) #maximum value of the rmse error is calculated

#index of the maximum value in the rmse found and the corresponding index value in the global coordinate
is printed as location of the damage
for k in range (b_index+1):
    if rmse[k]==maximum_value:
        print("The position of the damage",x_global[k])
```

Figure 30: Location of damage predicition

4 Results of tests

4.1 Problem setting

1-Dimensional wave propagation in the Heterogeneous medium using the Spectral Finite element method.
In addition to that, the damage is introduced and the location of the damage needs to be predicted.
The material parameters and the specification of the structure are shown in the following code.



```
n_ele=100 #number of elements
t_len=8000 #total length (m)
rho=2000  #density of the material (kg/m3)
velocity=2500 #velocity in (m/s)
n=2 #order of polynomials
time_step = 7500 #number of time steps
dom_per=0.4 #dominant period of ricker wavelet source
alpha=0.2 #for defected element K matrix
```

Figure 31: Material Parameter considered

The location of the source is applied at the middle of the structure.

4.1.1 Without Damage

The damage is not considered and the Free boundary condition is considered, the below figure shows the final displacement,

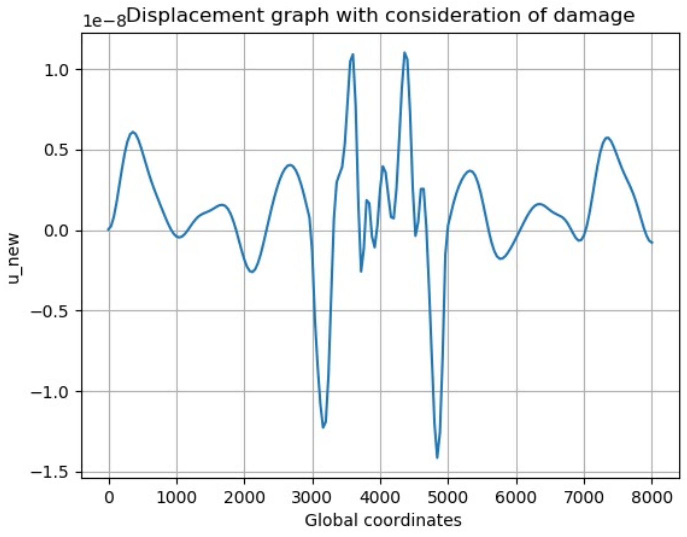


Figure 32: Displacement graph without damage

The force wave is reflected at the boundaries because of the free boundary condition. Because of heterogeneous medium amplitude of the displacement decreases after 25 elements (velocity

reduction elements in both sides).
The true and Estimated Damage vector and the error between them is shown in the below graph,
As there is no damage in the structure, there is no error between the True and the Estimated damage vector

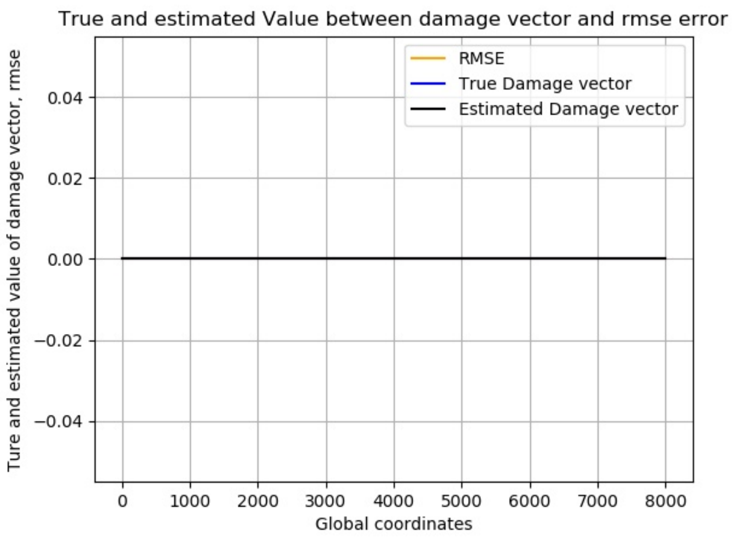
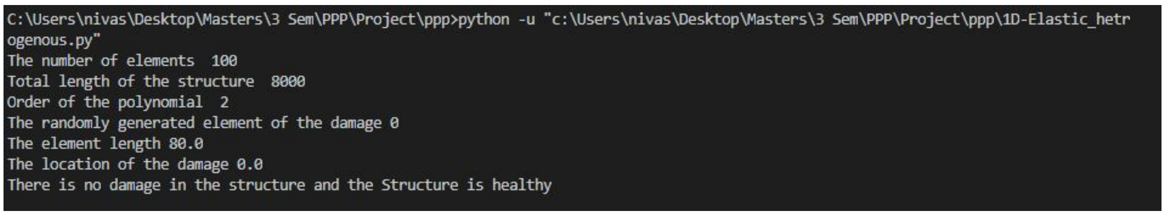


Figure 33: Error between Damage vector graph without damage

The structural health and the location of the damage is shown below,



```
C:\Users\nivas\Desktop\Masters\3 Sem\PPP\Project\ppp>python -u "c:\Users\nivas\Desktop\Masters\3 Sem\PPP\Project\ppp\1D-Elastic_hetr
ogenous.py"
The number of elements  100
Total length of the structure  8000
Order of the polynomial  2
The randomly generated element of the damage 0
The element length 80.0
The location of the damage 0.0
There is no damage in the structure and the Structure is healthy
```

Figure 34: Damage Location without damage

4.1.2 With the consideration of damage

Damage element is generated randomly and the code for the calculation is explained before, The displacement of the damaged structure with force applied to the center of the structure is shown below,

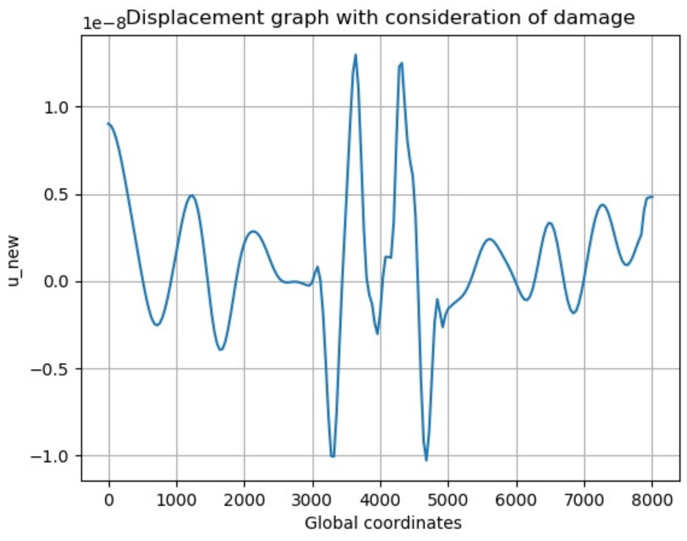


Figure 35: Displacement graph with damage

by comparing the displacement values to the displacement values without consideration of damage, the displacement at elements are deviated because of the damage. The below graph shows the error between the damage vector,

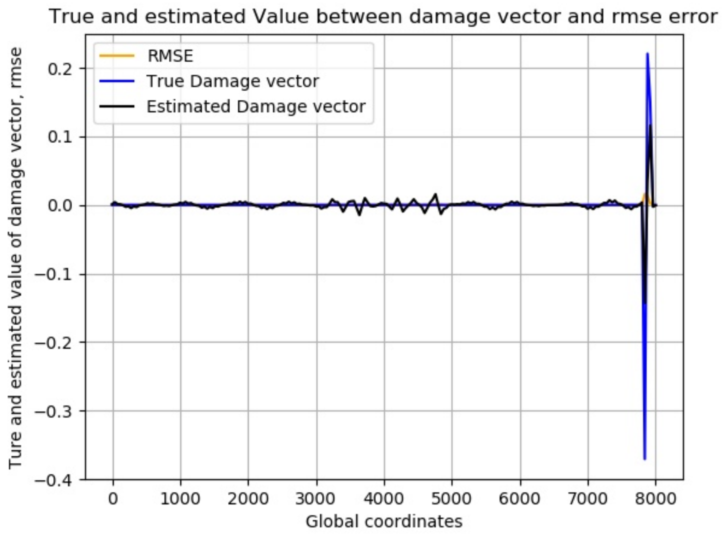
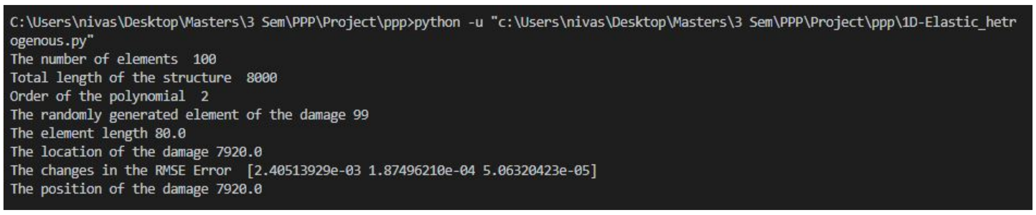


Figure 36: Error between Damage vector graph with damage

With the above graph the location of the damage is near $l = 8000$, and the predicted location of the damage is shown below, RMSE error near the damage location is maximum and it is shown in the graph,



```
C:\Users\nivas\Desktop\Masters\3 Sem\PPP\Project\ppp>python -u "c:\Users\nivas\Desktop\Masters\3 Sem\PPP\Project\ppp\1D-Elastic_hetr
ogenous.py"
The number of elements  100
Total length of the structure  8000
Order of the polynomial  2
The randomly generated element of the damage 99
The element length 80.0
The location of the damage 7920.0
The changes in the RMSE Error  [2.40513929e-03 1.87496210e-04 5.06320423e-05]
The position of the damage 7920.0
```

Figure 37: Damage Location with damage

4.2 Test Cases

As discussed earlier density ρ and shear modulus μ changes inside the element and makes difficult to obtain an analytical solution.

4.2.1 Displacement Analysis

The solution of the displacement will change based on the polynomial order n and the weights, points are changed with corresponding polynomial order.
If the polynomial order is increased there is more number of points in the element level, which gives an accurate result.
By increasing the polynomial order creates the densification near the boundaries and to solve the problem the time step needs to be increased.
The wave Propagation with the highest polynomial order $n = 9$ is considered as the exact solution.
The following test cases are carried out
1) Displacement analysis of polynomial order 2 to 9 with force at center.
2) Displacement analysis of polynomial order 2 to 9 with force at start of the structure.
3) Displacement analysis of polynomial order 2 to 9 with force at end of the structure.

4.2.2 Test case 01: Displacement analysis of polynomial order 2 to 9 with force at center

Aim: By applying the force at the center of the structure displacements are analyzed by varying the polynomial order n from 2 to 9. Displacement value of polynomial order $n = 9$ is considered as exact value and the error between the values from polynomial order $n = 2$ to $n = 9$ is plotted.

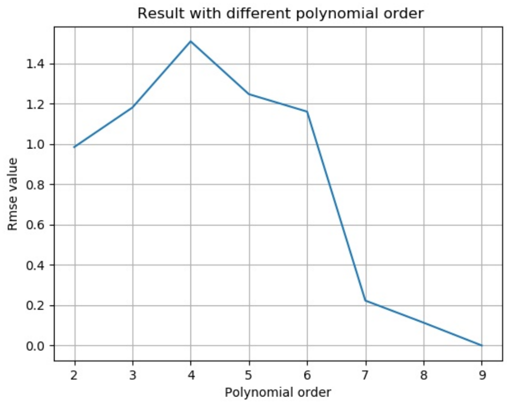


Figure 38: Error between exact $n = 9$ displacement and displacement by varying polynomial order n with force acting on middle of structure

By the above graph it is shown that by increasing the polynomial order decreases the error between the expected value and the exact value.

4.2.3 Test case 02: Displacement analysis of polynomial order 2 to 9 with force at start of the structure

Aim: Structure displacements are analyzed with applied force at the start of the structure by varying the polynomial order n 2 to 9. Exact value of the displacement is considered at polynomial order $n = 9$.

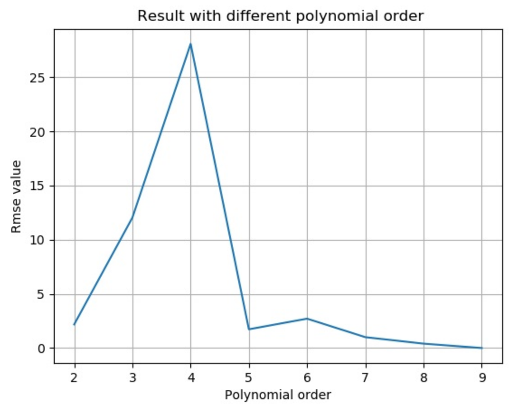


Figure 39: Error between exact $n = 9$ displacement and displacement by varying polynomial order n with force acting on start of structure

In the above graph the error between the exact value and the expected value is decreasing.

4.2.4 Test case 03: Displacement analysis of polynomial order 2 to 9 with force at end of the structure

Aim: Force is applied on the end of the structure and by varying the polynomial order $n = 2$ to $n = 9$,
The error in the displacement values are calculated and plotted in the below graph

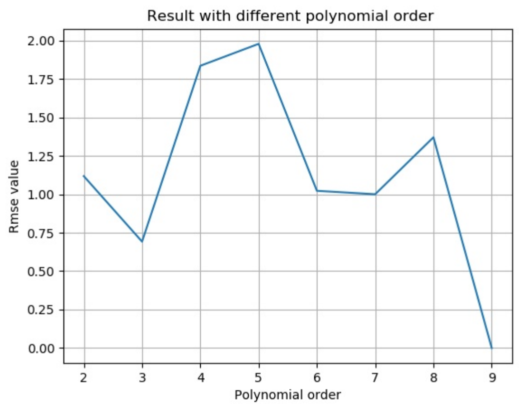


Figure 40: Error between exact $n = 9$ displacement and displacement by varying polynomial order n with force acting on end of structure

The accuracy of the structure is increased by increasing the polynomial order and the time step needs to be increased.
From the above test cases, the accuracy of the displacement is depends on the polynomial order.

4.2.5 Damage Location Analysis

Aim:The damage location is analyzed by the RMSE error graph.Damage element is considered to be 8 ($8^{th} element$).
The below graph shows the error between the damage vector. From the graph, the location of the damage is between 0 to 1000.

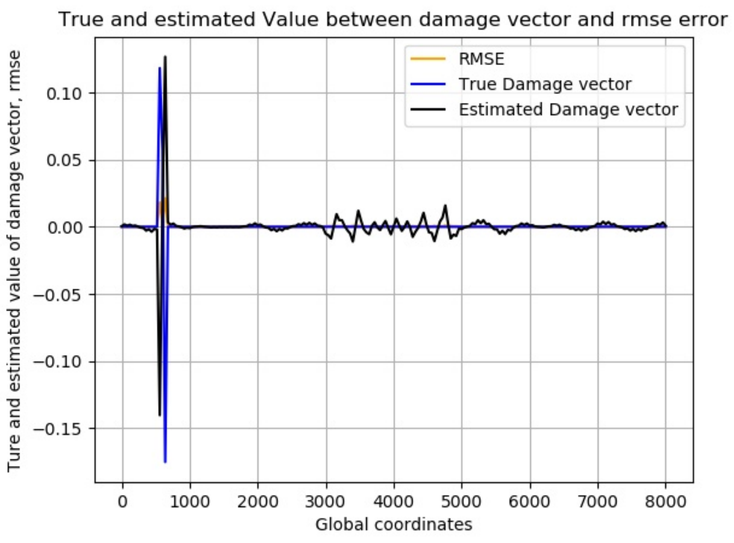


Figure 41: RMSE between Damage Vector with damage

The damage element number is assigned to be eight (8^{th} element) and showed in the below code,

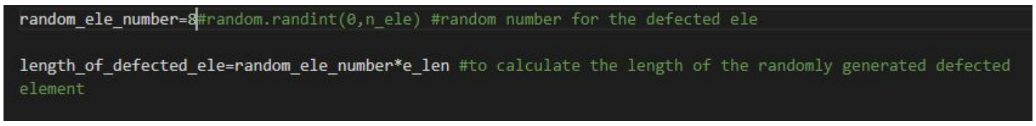

```
random_ele_number=8#random.randint(0,n_ele) #random number for the defected ele

length_of_defected_ele=random_ele_number*e_len #to calculate the length of the randomly generated defected
element
```

Figure 42: True value of Damage Location

The true value of the damage location is calculated by multiplying *random ele number* and the element length *e len* and showed in the above image.
The estimated location of the damage is calculated by the maximum value of the RMSE as explained before.

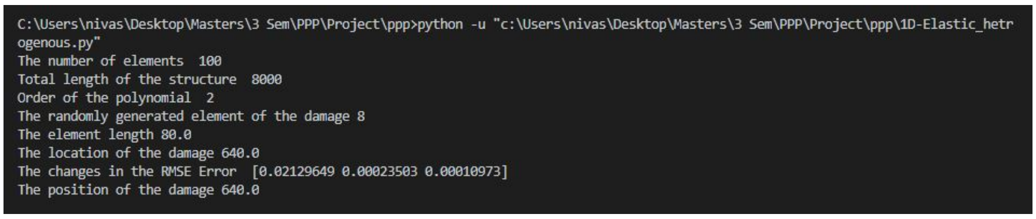
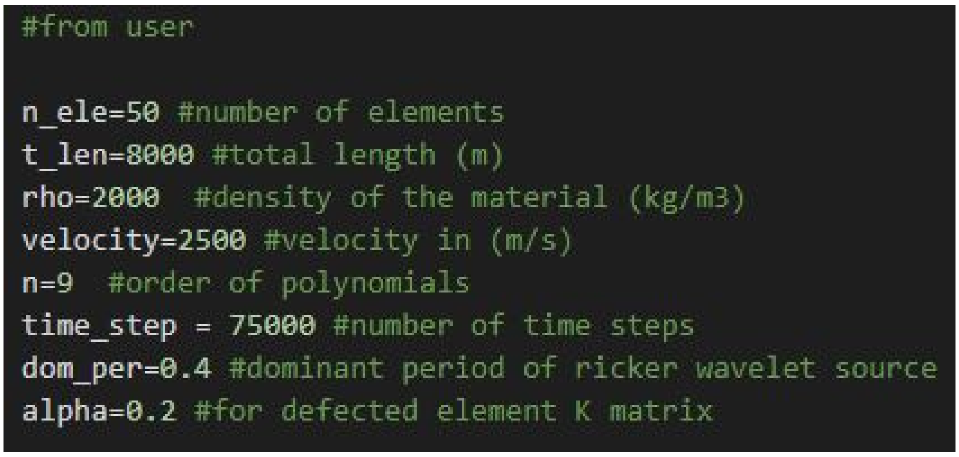

```
C:\Users\nivas\Desktop\Masters\3 Sem\PPP\Project\ppp>python -u "c:\Users\nivas\Desktop\Masters\3 Sem\PPP\Project\ppp\1D-Elastic_hetr
ogenous.py"
The number of elements  100
Total length of the structure  8000
Order of the polynomial  2
The randomly generated element of the damage 8
The element length 80.0
The location of the damage 640.0
The changes in the RMSE Error  [0.02129649 0.00023503 0.00010973]
The position of the damage 640.0
```

Figure 43: Estimated Damage Location

5 Manual

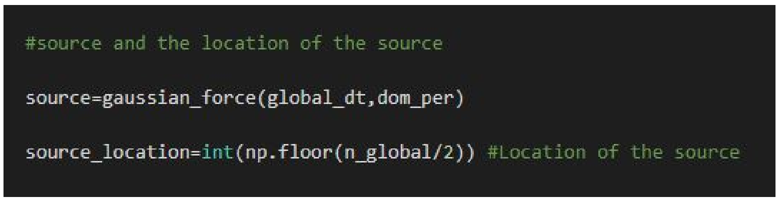
The below-mentioned parameters can be changed by the user.
1.Material parameters and specification of the structure.
2.Location of the source in the structure.
3.Damage Location.
4.The boundary condition of the structure.
The Material parameters and the specification of the structure can be changed by the user and which is shown below.



```
#from user

n_ele=50 #number of elements
t_len=8000 #total length (m)
rho=2000  #density of the material (kg/m3)
velocity=2500 #velocity in (m/s)
n=9  #order of polynomials
time_step = 75000 #number of time steps
dom_per=0.4 #dominant period of ricker wavelet source
alpha=0.2 #for defected element K matrix
```

Figure 44: Material Parameter and Structure Specification

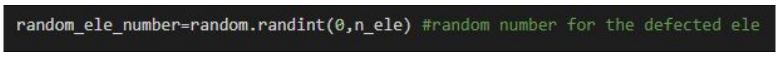
Location of the source in the structure is given below,



```
#source and the location of the source

source=gaussian_force(global_dt,dom_per)

source_location=int(np.floor(n_global/2)) #Location of the source
```

Figure 45: Location of the Source

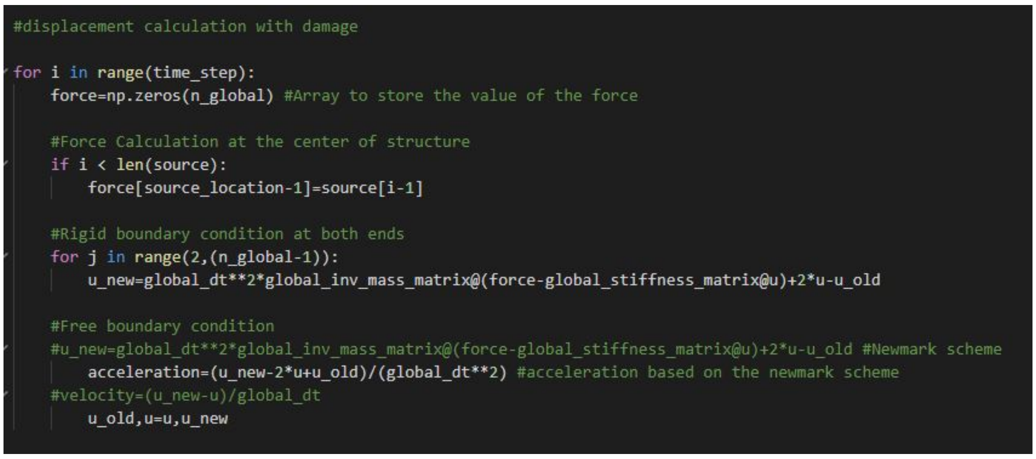
Damage location can be changed by the user and the code is shown below,



```
random_ele_number=random.randint(0,n_ele) #random number for the defected ele
```

Figure 46: Location of the damage element

Boundary condition of the structure can be changed in the below code.



```
#displacement calculation with damage

for i in range(time_step):
    force=np.zeros(n_global) #Array to store the value of the force

    #Force Calculation at the center of structure
    if i < len(source):
        force[source_location-1]=source[i-1]

    #Rigid boundary condition at both ends
    for j in range(2,(n_global-1)):
        u_new=global_dt**2*global_inv_mass_matrix@(force-global_stiffness_matrix@u)+2*u-u_old

    #Free boundary condition
    #u_new=global_dt**2*global_inv_mass_matrix@(force-global_stiffness_matrix@u)+2*u-u_old #Newmark scheme
        acceleration=(u_new-2*u+u_old)/(global_dt**2) #acceleration based on the newmark scheme
    #velocity=(u_new-u)/global_dt
        u_old,u=u,u_new
```

Figure 47: Boundary condition of the structure

After changing the above mentioned parameters run the program to get the result of the structure with the user defined conditions.

6 Citation

a) (Gopalakrishnan et al., 2011): Computational techniques for structural health monitoring

b) (Kudela et al., 2007): Wave propagation modelling in 1D structures using spectral finite elements

c) (Yao & Xiang, 2014): 1D elastic wave equations

d) (Aboites, 2019): Legendre Polynomials

e) (Kasahara et al., 2010): Active geophysical monitoring

f) (for PPP, 2020): GLL points and weights

g) (Carrion et al., 2003): Structural health monitoring and damage detection

h) (Funaro, 1993): First derivative of the Lagrange polynomial with GLL points

References

Aboites, V. (2019). Legendre polynomials: a simple methodology. In *Journal of physics: Conference series* (Vol. 1221, p. 012035).

Carrion, F., Doyle, J., & Lozano, A. (2003). Structural health monitoring and damage detection using a sub-domain inverse method. *Smart materials and structures*, *12*(5), 776.

for PPP, G. P. (2020, February). *Gll points and weights.* `https://keisan.casio.com/exec/system/1280801905`.

Funaro, D. (1993). Fortran routines for spectral methods. *available via anonymous FTP at ftp. ian. pv. cnr. it in pub/splib*.

Gopalakrishnan, S., Ruzzene, M., & Hanagud, S. (2011). *Computational techniques for structural health monitoring.* Springer Science & Business Media.

Kasahara, J., Korneev, V., & Zhdanov, M. S. (2010). *Active geophysical monitoring.* Elsevier.

Kudela, P., Krawczuk, M., & Ostachowicz, W. (2007). Wave propagation modelling in 1d structures using spectral finite elements. *Journal of sound and vibration*, *300*(1-2), 88–100.

Yao, R., & Xiang, Z. (2014). Modified 1d elastic wave equations that retain time synchronization under spatial coordinate transformations. *arXiv preprint arXiv:1404.2838*.

7 Git Log

The history of the git log is mentioned below

7.1 Log History

commit $dd164c979aab3b9fb086c563a33f652c22babbdd$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $SunApr2621:16:372020+0200$
Final commit with changes

commit $b1d4e3c4bbf1d825bbd240954e9a6b950120da96$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $MonApr2012:23:522020+0200$
Plots updated

commit $30240cd49d271ed5f6f1fde05d81e08d959cec7a$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $MonApr2012:05:472020+0200$
comments added in the final commit

commit $f8ae058116b93ac2bc699d927db59d0f3e7ba655$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $MonApr2011:34:312020+0200$
Final commit
commit $d043d97828aaada5b5acccd7125392951cd1c546$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $WedApr1518:55:312020+0200$
rmse based location prediction is checked for 100 elements

commit $00c1ee7ae139755c0dd80195fb673a20b6a56cfc$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $WedApr1518:45:232020+0200$
root mean square of damage vector is calculated and error values are checked

commit $5225f3de1f6dfd6ffbc64a8dfd2b9dd7222915d8$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $WedApr1517:54:332020+0200$
damage vector is calculated and stiffness matrix is predicted

commit $7da16f7ec8394b1f1c24917be5a76def4a8147b4$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $WedApr1512:38:222020+0200$
Acceleration of damaged and $un_{d}amaged$ structure is calculated using newmark s

commit $4147fd1ce8ed484ebc30593c88e9eb85789fd68e$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $TueApr1418:04:402020+0200$ Need to modify the rms algorithm and have to find the relation between damage vector and structural response

commit $aedf90a2c41b5e967644b70aec3018d38f932811$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $TueApr1416:15:502020+0200$
need to check with the damage vector and the rms relation

commit $a469a3e9f393198544565aa25e270ba23b7520f6$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $TueApr1414:04:202020+0200$
The location of the damage is predicted using the minimum of rmse error

commit $f6bb19519136518b00f670f9e350d7fed2ede162$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $TueApr1413:37:462020+0200$
Root mean square error method is computed to calculate the error between the damage and undamaged structural response

commit $ec5d3de69c1a1ecac0df19e1cd8c267bdc4305e1$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $MonApr1315:25:302020+0200$
index of the x_global and corresponding displacement value is calculated

commit $aa7cd9778c6a8636084beb9b9b3349ac371247d6$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $SunApr1220:13:512020+0200$
Damage vector is calculated and checked with damage element

commit $171867567c3a9c05a4e2393432c02c79aaf4967b$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $SunApr1219:49:062020+0200$
random element number and changes in the global stiffness matrix need to change

commit $5a0785c02f1a6a441d282e56363c383ddf78f180$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $SunApr1219:13:172020+0200$
final displacement vector calculation is changed

commit $f060a9d58d38cdf37942461d8ab7b7e9ca37aa60$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $SunApr1218:38:552020+0200$
Mass matrix and global inverse mass matrix is checked

commit $5cdb5a3b839ea5c04024185f53c2330fcbe96ce0$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $SunApr1212:47:112020+0200$
damage vector is calculated which contains the information about the damage

commit $b24801cd6dba069c3f887238b7f12d0e0ea075e3$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $SatApr1119:42:182020+0200$
Global stiffness matrix and displacement without damage is calculated and verified

commit $f9071d88faf2a9655c317134bfd0fec31207652f$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $FriApr1019:10:532020+0200$
to find maximum or minimum of a function using conjugate gradient method

commit $977c91efde874db5337afcad8a8f54da3ca98f79$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $FriApr1017:27:152020+0200$
changes of damage stiffness and random element effect checked and need to find the solution to find the defect location

commit $fb56b69b69b332878f8b38aa9ddb2b57c55ddc99$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $WedApr815:22:582020+0200$
Global stiffness matrix is updated and values updated with defected element

commit $d00fbf22148f4bf383662f2be50d2ae8d16c710a$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $TueApr720:32:442020+0200$
need to check the global stiffness matrix with defect added

commit $e2088bedda9b0fa1af3615d2c3ed9229b417ee52$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $TueApr720:01:042020+0200$
random number is generated and the defected element stiffness matrix is calculated

commit $2dbda1d05dcf1b4fbd8f5919b065e2d6d0570a7f$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $TueApr719:18:022020+0200$
hetrogeneous case calculations done and need to verify the results

commit $51bdb20aa5c764fbdab032ee6ba1f2a68182fc7d$
Author: $Venkatasubramanian < nivass18.vn@gmail.com >$
Date: $SunApr511:07:422020+0200$
Global mass matrix is updated based on hetrogeneous method and values checked

commit 0307ca1fc9d62d2f36dec075b4e1bfe9532c47b2
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: SunApr510 : 47 : 132020 + 0200
Hetrogeneous velocity reduction area defined and values checked

commit 3a7076def76d3e1ceaa1077c0b1fe96c44e68b0a
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriApr323 : 11 : 512020 + 0200
Displacement values are checked

commit 4ce94d2e32786a3f23d7a33ef86ec9925c2ba96c
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriApr321 : 26 : 042020 + 0200
Time extrapolation and the displacement formula added

commit 7188aea456ce3730f8f2e385ad29b6a601131af2
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriApr317 : 08 : 592020 + 0200
Global domain space created with equal spe

commit 8e88261064ee819c6ad37e1709c28305834f59c8
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriApr315 : 02 : 502020 + 0200
GLobal stiffness matrix value checked

commit 486d204453767cabc72644f0d5ccd4e9968ef602
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuApr215 : 40 : 182020 + 0200
Element stiffness function created and values are checked

commit d62112e34559acb65fdc80e0c267ccaa5b9e81f5
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: WedApr115 : 00 : 122020 + 0200
ELement stiffness matrix is created and need to check the values

commit 117f775d4603330db301564ee60b58a0c202f7ee
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar2620 : 43 : 182020 + 0100
legendre and lagrange polynomials added and the first derivative of lagrange polynopmial functions created

commit 614f4500c6b1e1cbbcc4140fadef3c292ba5728a
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar2615 : 54 : 452020 + 0100
ELement level legendre polynomial function creation

```
commit 2b02a9e00f10c446ebfae4ecdb827cd355b81598
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar2615 : 19 : 422020 + 0100
Lagrange polynomial function creation

commit c5db0f31c2841c2edfd0b3974249b30b863f212f
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: TueMar2420 : 23 : 102020 + 0100
Global Inverse mass matrix calculated

commit 9d56b60acb7198e4098637d2e552061302cad1d0
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: TueMar2420 : 06 : 052020 + 0100
arrangement of the diagonals of mass matrix to the global mass matrix

commit 1de0cd2e4d52501398815a8562a63e717d4dc004
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: TueMar2419 : 40 : 542020 + 0100
global mass matrix updated

commit bfc811b9b9261019ca488d092f946f3f54a26c74
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: MonMar1617 : 08 : 472020 + 0100
global mass matrix calculation need to update

commit 3e015e76b76c00d67d90fa005bcf2ab248da0ff9
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: MonMar1616 : 41 : 472020 + 0100
Element mass matrix calculation function changed

commit 6e7e7d83594998d057a89707b85f5c55ae3ce8c7
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar520 : 31 : 212020 + 0100
element mass matrix function created and values are checked

commit fded3816b9f2d0c55062a854bd4839f0ddd32a80
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar519 : 58 : 442020 + 0100
Element mass matrix function created and values are checked

commit e2285dcfad0f88656e5ff8b8d6c0751ce1f80967
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar519 : 07 : 002020 + 0100
element mass matrix function created and values are checked
```

commit 70de17491f762e8c244cc0449b7d8d6295027c81
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar518 : 47 : 592020 + 0100
user inputs and jacobian matrix are calculated

commit f3f76760c3891cc58173999f13335c184a523395
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar518 : 46 : 142020 + 0100
GLL points and weights calculation function is created and the order of polynomial

commit 6034772ce9911917e34cd4a4901e06015474e306
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: ThuMar518 : 44 : 092020 + 0100
Initial commit

commit 7fb9033996215558997249fe86735ad5a85e2791
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: SatFeb1520 : 08 : 482020 + 0100
Checking for the formula of calculating GLL polynomial and its derivative

commit c228ee9380118d3f1d2ce147a27ad5559bc43bc4
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: SatFeb1520 : 07 : 202020 + 0100
Changes in the polynomial weights function

commit 2defd57d54c09c0080180c68f0419b647e02962a
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: SatFeb1520 : 02 : 102020 + 0100
new function is created for calculating the polynomial for GLL and diff of the polynomial function is changed based on the GLL polynomial function

commit 10fb597dcd64d320feee8aa14df3f8e39223ff41
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriFeb1421 : 19 : 292020 + 0100
Changes made in the polynomial weight formula

commit 00b8123a02663e49106a8bd4a81ce5597b93e232
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriFeb1420 : 44 : 002020 + 0100
Lagrange polynomials are checked and densification also checked

commit bd5747c0875f0c7d70fc7a701bc346a75365bec3
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriFeb1418 : 11 : 572020 + 0100
Changes made in the Polynomial and Diff of polynomial

commit 623258e1602e8b9a26c9f3d3d33548316d08bae3
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriFeb1417 : 52 : 052020 + 0100
Changes need to made for GLL polynomial and diff

commit c8f05857fcb8aebab0dd8b49c7a0d9dfffd99225
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: SatFeb120 : 10 : 322020 + 0100
GLL points and weights are calculated using newton raphson method and checked the results

commit 141b7ada33deac0a303fea5a605123b8afe472b3
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: SatFeb118 : 38 : 442020 + 0100
removed the root finding algorithm

commit a6eb4cf9361ea72f2a1235e5bf6f9310afbdd9f4
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: TueJan2820 : 54 : 172020 + 0100
Newton raphson method need to be changed

commit 3775e5aee89c0a4106a74abd80913e340b90366a
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: TueJan2819 : 10 : 212020 + 0100
Derivative part is added for the Newton raphson method of root finding

commit 95426ce64013443515754b589298a0d3a030ea8f
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: MonJan2722 : 05 : 242020 + 0100
Function created with changes in the polynomial and the condition

commit 205e081fb9973d9fe08022ce33fdf44921ea5cf5
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: SatJan2517 : 06 : 342020 + 0100
GLL points and the correspondings weights are added in the function

commit 077dc74f308431e97868902378dd83c55360a089
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriJan2422 : 31 : 062020 + 0100
Polynomial of n order is created

commit d265e661060d9bc6e8c6f11ca26a52d0201176c9
Author: Venkatasubramanian < nivass18.vn@gmail.com >
Date: FriJan2421 : 24 : 062020 + 0100
Initial commit