Project 3

Omkar Mulekar
AERO 4630-002
Aerospace Structural Dynamics

8 March 2019

Part 1a: Non-dimensionalization and Weak Form

The governing equation

$$\text{div}\boldsymbol{\sigma} + \boldsymbol{f} = \rho\ddot{\boldsymbol{u}} \tag{1}$$

can be put in its weak form. By redefining the second derivative and converting to index notation, the governing equation becomes

$$\sigma_{ij,j} + f_i = \frac{\rho(u_i^{n+1} - 2u_i^n - u_i^{n-1})}{(\Delta t)^2} \tag{2}$$

which can be rearranged to

$$\frac{1}{\rho}\sigma_{ij,j}(\Delta t)^2 + \frac{1}{\rho}f_i(\Delta t)^2 = u_i^{n+1} - 2u_i^n - u_i^{n-1} \tag{3}$$

By integrating over the body Ω, the integral becomes

$$\int_\Omega \frac{1}{\rho}\sigma_{ij,j}(\Delta t)^2 dV + \int_\Omega \frac{1}{\rho}f_i(\Delta t)^2 dV = \int_\Omega (u_i^{n+1} - 2u_i^n - u_i^{n-1})dV \tag{4}$$

We can multiply by a perturbation v_i and apply the chain rule to the first term.

$$\int_\Omega \frac{(\Delta t)^2}{\rho}\sigma_{ij,j}v_i dV = \int_\Omega \frac{(\Delta t)^2}{\rho}(\sigma_{ij}v_i)_{,j}dV - \int_\Omega \frac{(\Delta t)^2}{\rho}\sigma_{ij}v_{i,j}dV \tag{5}$$

Applying the divergence theorem yields

$$\int_\Omega \frac{(\Delta t)^2}{\rho}\sigma_{ij,j}v_i dV = \int_{\partial\Omega} \frac{(\Delta t)^2}{\rho}\sigma_{ij}n_j v_i dA - \int_\Omega \frac{(\Delta t)^2}{\rho}\sigma_{ij}v_{i,j}dV \tag{6}$$

The governing equation in its weak form becomes

$$-\int_{\Omega}\frac{(\Delta t)^2}{\rho}\sigma_{ij}(\boldsymbol{u}^{n+1})v_i dV+\int_{\partial\Omega}\frac{(\Delta t)^2}{\rho}t_i v_i dA+\int_{\Omega}\frac{(\Delta t)^2}{\rho}f_i v_i dV=\int_{\Omega}v_i(u_i^{n+1}-2u_i^n-u_i^{n-1})dV \tag{7}$$

Length terms, traction and stress terms, and time terms in the weak form equation can be nondimensionalized by the beam length L, Young's modulus E, and characteristic time $\tilde{t}$.

$$-\int_{\Omega}\frac{(\Delta t)^2}{\rho(\tilde{t})^2}\frac{\sigma_{ij}(\boldsymbol{u}^{n+1})v_i}{EL^2}dV+\int_{\partial\Omega}\frac{(\Delta t)^2}{\rho(\tilde{t})^2}\frac{t_i v_i}{EL}dA+\int_{\Omega}\frac{(\Delta t)^2}{\rho(\tilde{t})^2}\frac{f_i v_i}{L}dV=\int_{\Omega}\frac{v_i(u_i^{n+1}-2u_i^n-u_i^{n-1})}{L^2}dV \tag{8}$$

Part 1b: Just Displacement

Deflection was determined at the center of a beam with length $L = 1$m, width $W = 0.2$m, height $H = 0.2$m, Youngs Modulus $E = 200$GPa, Poisson ratio $\nu = 0.3$, and density 7800kg m^{-3}. The beam is clamped at both ends, and a downward force of $F = 100$N is applied over the area on the top surface near the center, within 0.2 m along the width, and within 0.02 m along the length of the beam.

The deflection was determined to be 3.75×10^{-8} m downward, and the Paraview output is shown in Fig 1

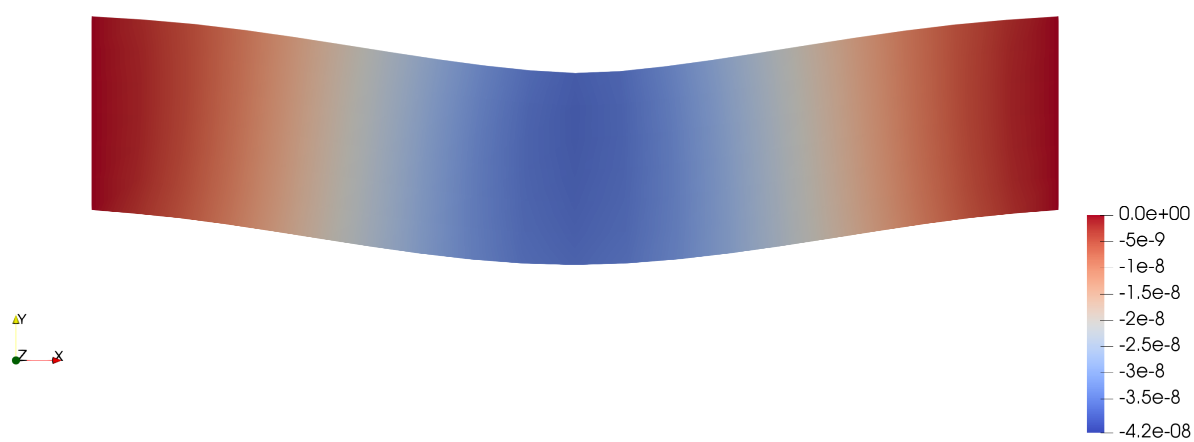

Figure 1: Paraview output for problem 1b

The code used for part 1b can be seen in **Appendix 1b**.

Part 1c: Free Vibration

The force was removed, and the calculated deflection was set as an initial condition of the beam. A free vibration was modeled, and a time-plot can be seen in Fig 2.

The period of oscillation was determined to be 0.0011 seconds, and the natural frequency was determined to be 5681 rad/s or 904.2 Hz.

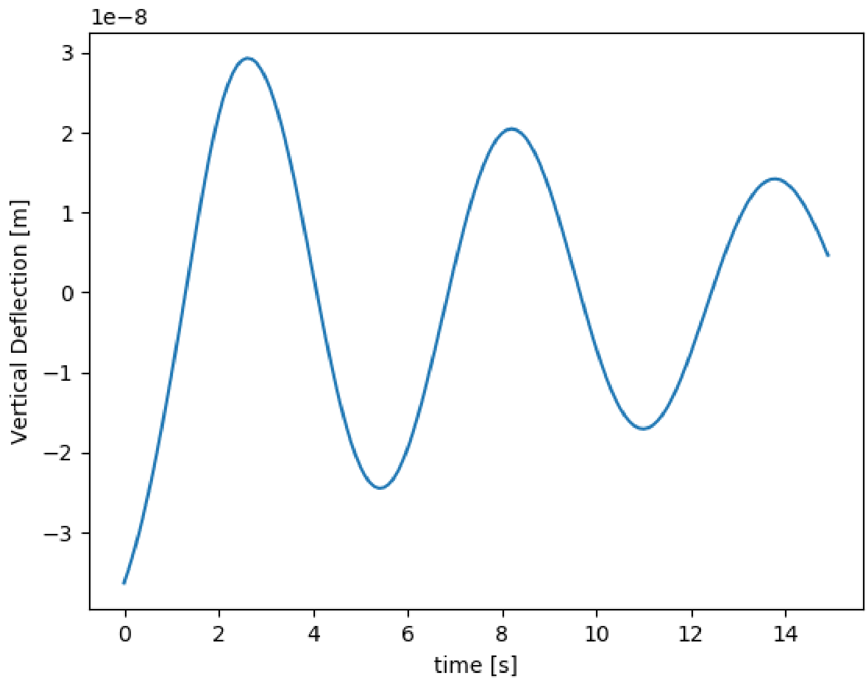

Figure 2: Vertical Deflection vs Time

The code used for Problem 1c is shown in **Appendix 1c**.

Part 1d: Natural Frequency, Changing Dimensions

Changing Width

Next, the beam width was varied to assess how natural frequency changes with beam width. Widths of 0.2, 0.4, 0.6, 0.8, and 1.0 m were assessed. A plot of natural frequencies over beam width is shown in Fig 3.

The Python code used for part 1d-i is shown in **Appendix 1d-i**.

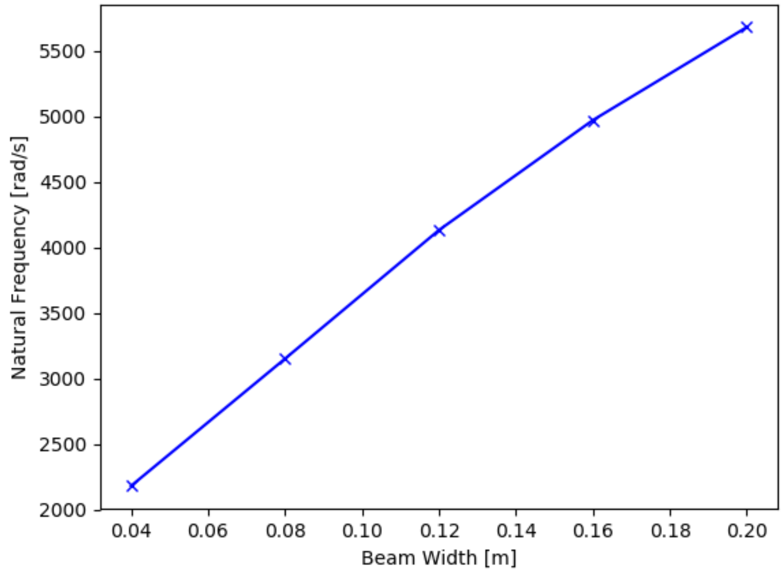

Figure 3: Natural Frequency over Beam Width for Problem 1d

Changing Height

Next, the beam height was varied to assess how natural frequency changes with beam height. Heights of 0.2, 0.4, 0.6, 0.8, and 1.0 m were assessed. A plot of natural frequencies over beam Height is shown in Fig 4.

The Python code used for part 1d-ii is shown in **Appendix 1d-ii**.

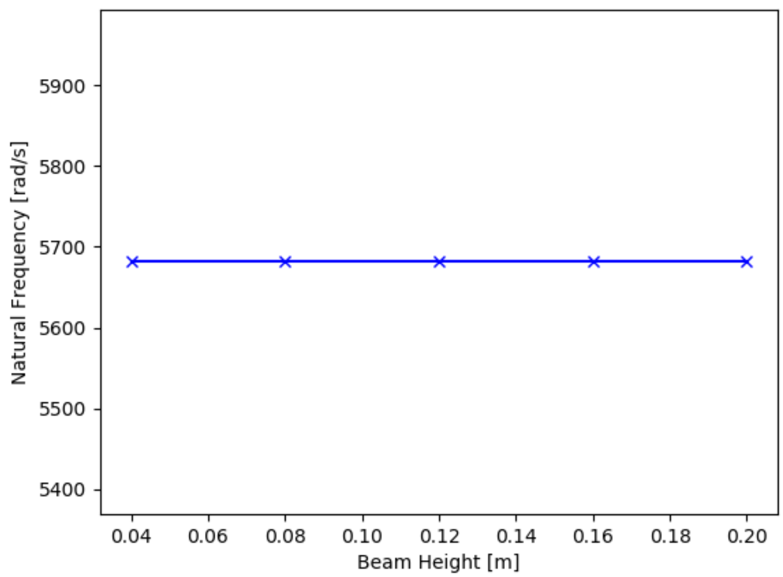

Figure 4: Natural Frequency over Beam Height for Problem 1d

Appendix 1b: Code for Problem 1b

```
"""
Python script for Part 1b of Project 2a

Original Author:  Vinamra Agrawal
Date:             January 25, 2019

Edited By:          Omkar Mulekar
Date:             February 10, 2019

"""

from __future__ import print_function
from fenics import *
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from ufl import nabla_div
import math

#==========================================================================
# Define System Properties
#==========================================================================
length = 1;
W = 0.2;
H = 0.2;

a = 0.04/length;
b = 0.4*H/length;
area = a*b;

F = -100

youngs = 200e9 # Youngs
nu = 0.3 # Poisson
rho = 7800 # Density


# Lame parameters
mu = (youngs)/(2*(1+nu))
lambda_ = (nu*youngs)/((1+nu)*(1-2*nu))

g = 10
```

```
traction_applied = F/area

#==========================================================================
#  Dimensionless parameters
#==========================================================================
l_nd = length/length
w_nd = W/length
h_nd = H/length

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#========================================================================
# Boundaries and Geometry
#========================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,6,6)
V = VectorFunctionSpace(mesh,'P',1)

tol = 1E-14

def boundary_left(x,on_boundary):
    return (on_boundary and near(x[0],0,tol))

def boundary_right(x,on_boundary):
    return on_boundary and near(x[0],l_nd,tol)

bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)
bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)


#========================================================================
def epsilon(u):
    return 0.5*(nabla_grad(u) + nabla_grad(u).T)

def sigma(u):
    return lambda_nd*nabla_div(u)*Identity(d) + mu_nd*(epsilon(u) +
        epsilon(u).T)

#========================================================================
# First we solve the problem of a cantelever beam under fixed
# load.
#========================================================================
```

```
u_init = TrialFunction(V)
d = u_init.geometric_dimension()
v = TestFunction(V)
f = Constant((0.0,0.0,0.0))
T_init = Expression(('0.0', 'x[0] >= 0.48*l && x[0] <= .52*l &&
   near(x[1],w) && x[2] >= 0.3*h && x[2] <= 0.7*h? A : 0.0' ,'0.0'
   ), degree=1, l=l_nd, w=w_nd,h=h_nd, A=traction_nd)
F_init = inner(sigma(u_init),epsilon(v))*dx - dot(f,v)*dx - dot(
   T_init,v)*ds
a_init, L_init = lhs(F_init), rhs(F_init)

print("Solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right])
w_nd = u_init(l_nd/2.0,w_nd/2.0,h_nd/2.0)
w = w_nd * length
print(w[1])

vtkfile_u = File('deflection.pvd')
vtkfile_u << u_init
```

Appendix 1c: Code for Problem 1c

```
"""
Python script for Part 1c of Project 2a

Original Author:  Vinamra Agrawal
Date:             January 25, 2019

Edited By:           Omkar Mulekar
Date:             February 28, 2019

"""

from __future__ import print_function
from fenics import *
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from ufl import nabla_div
import math
import numpy as np
from scipy.signal import argrelextrema

#==========================================================================
# Define System Properties
#==========================================================================
length = 1;
W = 0.2;
H = 0.2;

a = 0.04*length;
b = 0.4*H;
area = a*b;

F = -100

youngs = 200e9 # Youngs
nu = 0.3 # Poisson
rho = 7800 # Density


# Lame parameters
mu = (youngs)/(2*(1+nu))
lambda_ = (nu*youngs)/((1+nu)*(1-2*nu))
```

```
g = 10

traction_applied = F/area

#===========================================================================
#  Dimensionless parameters
#===========================================================================
l_nd = length/length
w_nd = W/length
h_nd = H/length

bar_speed = math.sqrt(youngs/rho)
t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 150

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#=========================================================================
# Boundaries and Geometry
#=========================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,6,6)
V = VectorFunctionSpace(mesh, 'P',1)

tol = 1E-14

def boundary_left(x,on_boundary):
    return (on_boundary and near(x[0],0, tol))

def boundary_right(x,on_boundary):
    return on_boundary and near(x[0],l_nd, tol)

bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)
bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)


#=========================================================================
def epsilon(u):
    return 0.5*(nabla_grad(u) + nabla_grad(u).T)
```

```
def sigma(u):
    return lambda_nd*nabla_div(u)*Identity(d) + mu_nd*(epsilon(u) +
        epsilon(u).T)

#=====================================================================
# First we solve the problem of a cantelever beam under fixed
# load.
#=====================================================================
u_init = TrialFunction(V)
d = u_init.geometric_dimension()
v = TestFunction(V)
f = Constant((0.0,0.0,0.0))
T_init = Expression(('0.0', 'x[0] >= 0.48*l && x[0] <= .52*l &&
    near(x[1],w) && x[2] >= 0.3*h && x[2] <= 0.7*h? A : 0.0' ,'0.0'
    ), degree=1, l=l_nd, w=w_nd,h=h_nd, A=traction_nd)
F_init = inner(sigma(u_init),epsilon(v))*dx - dot(f,v)*dx - dot(
    T_init,v)*ds
a_init, L_init = lhs(F_init), rhs(F_init)

print("Solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right])
w_nd = u_init(l_nd/2.0,w_nd/2.0,h_nd/2.0)
w = w_nd * length
print(w[1])

#=====================================================================
# Next we use this as initial condition, let the force go and
# study the vertical vibrations of the beam
#=====================================================================
u_n = interpolate(Constant((0.0,0.0,0.0)),V)
u_n_1 = interpolate(Constant((0.0,0.0,0.0)),V)
u_n.assign(u_init)
u_n_1.assign(u_init)

T_n = Constant((0.0,0.0,0.0))

u = TrialFunction(V)
d = u.geometric_dimension()
v = TestFunction(V)

F = (dt*dt)*inner(sigma(u),epsilon(v))*dx + dot(u,v)*dx - (dt*dt)*
    dot(f,v)*dx - (dt*dt)*dot (T_n,v)*ds - 2.0*dot(u_n,v)*dx + dot(
    u_n_1,v)*dx
```

```
a,L = lhs(F), rhs(F)

xdmffile_u = XDMFFile('results/solution.xdmf')
xdmffile_s = XDMFFile('results/stress.xdmf')

u = Function(V)

u_store = [0] * num_steps
time = [0] * num_steps

index = 0
for n in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    solve(a == L, u, [bc_left, bc_right])
    u_grab = u(0.5,0.1,0.1)
    u_store[n] = u_grab[1]

    if(abs(t-index)<0.01):
        print("Writing output files...")
        xdmffile_u.write(u*length,t)
        W = TensorFunctionSpace(mesh, "Lagrange", 1)
        stress = lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
            + epsilon(u).T)
        xdmffile_s.write(project(stress,W),t)
        index += 1

    time[n] = t
    t+=dt
    u_n_1.assign(u_n)
    u_n.assign(u)


# Get period of oscillation
u_np = np.array(u_store)
min_args = argrelextrema(u_np,np.less)
period = (time[min_args[0][1]] - time[min_args[0][0]])*t_char
nat_freq = 2*math.pi /period
print("Period of Oscillation", period, " seconds")
print("Natural Frequency:    ", nat_freq," rad/s")

plt.figure(1)
plt.plot(time,u_store)
plt.xlabel('time [s]')
plt.ylabel('Vertical Deflection [m]')
```

```
plt.savefig('1cfig.png')
```

Appendix 1d-i: Code for Problem 1d-i

```
"""
Python script for Part 1d.i of Project 2a

Original Author:  Vinamra Agrawal
Date:             January 25, 2019

Edited By:           Omkar Mulekar
Date:             February 28, 2019

"""

from __future__ import print_function
from fenics import *
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from ufl import nabla_div
import math
import numpy as np
from scipy.signal import argrelextrema

#==========================================================================
# Define System Properties
#==========================================================================
length = 1;
W1 = 0.2;
H = 0.2;
alpha = np.linspace(0.2,1,num=5)
print("alpha = ",alpha)
W = alpha * W1

a = 0.04*length;
b = 0.4*H;
area = a*b;

F = -100

youngs = 200e9 # Youngs
nu = 0.3 # Poisson
rho = 7800 # Density
```

```
# Lame parameters
mu = (youngs)/(2*(1+nu))
lambda_ = (nu*youngs)/((1+nu)*(1-2*nu))

g = 10

traction_applied = F/area


nat_freq = [0] * len(alpha)

print("Beginning Loop...")
for i in range(len(alpha)):
    print("alpha = ", alpha[i])

    #==================================================================
    #  Dimensionless parameters
    #==================================================================
    l_nd = length/length
    w_nd = W[i]/length
    h_nd = H/length

    bar_speed = math.sqrt(youngs/rho)
    t_char = length/bar_speed
    t = 0
    t_i = 0.5
    dt = 0.1
    num_steps = 275

    mu_nd = mu/youngs
    lambda_nd = lambda_/youngs

    traction_nd  = traction_applied/youngs

    #================================================================
    # Boundaries and Geometry
    #================================================================
    mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,6,6)
    V = VectorFunctionSpace(mesh,'P',1)

    tol = 1E-14

    def boundary_left(x,on_boundary):
        return (on_boundary and near(x[0],0,tol))
```

```
def boundary_right(x,on_boundary):
    return on_boundary and near(x[0],l_nd,tol)

bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)
bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)


#==========================================================================
def epsilon(u):
    return 0.5*(nabla_grad(u) + nabla_grad(u).T)

def sigma(u):
    return lambda_nd*nabla_div(u)*Identity(d) + mu_nd*(epsilon(u
        ) + epsilon(u).T)

#==========================================================================
# First we solve the problem of a cantelever beam under fixed
# load.
#==========================================================================
u_init = TrialFunction(V)
d = u_init.geometric_dimension()
v = TestFunction(V)
f = Constant((0.0,0.0,0.0))
T_init = Expression(('0.0', 'x[0] >= 0.48*l && x[0] <= .52*l &&
     near(x[1],w) && x[2] >= 0.3*h && x[2] <= 0.7*h? A : 0.0' ,'
    0.0'), degree=1, l=l_nd, w=w_nd,h=h_nd, A=traction_nd)
F_init = inner(sigma(u_init),epsilon(v))*dx - dot(f,v)*dx - dot
    (T_init,v)*ds
a_init, L_init = lhs(F_init), rhs(F_init)

print("Solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right])
down_nd = u_init(l_nd/2.0,w_nd/2.0,h_nd/2.0)
w = down_nd * length
print("Initial Displacement: ",w[1])

#==========================================================================
# Next we use this as initial condition, let the force go and
# study the vertical vibrations of the beam
#==========================================================================
u_n = interpolate(Constant((0.0,0.0,0.0)),V)
u_n_1 = interpolate(Constant((0.0,0.0,0.0)),V)
u_n.assign(u_init)
u_n_1.assign(u_init)
```

```
T_n = Constant((0.0,0.0,0.0))

u = TrialFunction(V)
d = u.geometric_dimension()
v = TestFunction(V)

F = (dt*dt)*inner(sigma(u),epsilon(v))*dx + dot(u,v)*dx - (dt*
    dt)*dot(f,v)*dx - (dt*dt)*dot (T_n,v)*ds - 2.0*dot(u_n,v)*dx
     + dot(u_n_1,v)*dx
a,L = lhs(F), rhs(F)


u_store = [0] * num_steps
time = [0] * num_steps

index = 0
for n in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    u = Function(V)
    solve(a == L, u, [bc_left,bc_right])
    u_grab = u(l_nd/2.0,w_nd/2.0,h_nd/2.0)
    u_store[n] = u_grab[1]

    if(abs(t-index)<0.01):
        # print("Writing output files...")
        W_ = TensorFunctionSpace(mesh, "Lagrange", 1)
        stress = lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(
            u) + epsilon(u).T)
        index += 1

    time[n] = t
    t+=dt
    u_n_1.assign(u_n)
    u_n.assign(u)

plt.figure(1)
plt.plot(time,u_store)
plt.xlabel('time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.savefig('1dfig_test.png')

# Get period of oscillation
u_np = np.array(u_store)
```

```
    min_args = argrelextrema(u_np,np.greater)
    print("min_args", min_args[0])
    period = (time[min_args[0][1]] - time[min_args[0][0]])*t_char
    nat_freq[i] = 2*math.pi /period
    print("Period of Oscillation", period, " seconds")
    print("Natural Frequency:    ", nat_freq," rad/s")

plt.figure(2)
plt.plot(W,nat_freq,'b-x')
plt.xlabel('Beam Width [m]')
plt.ylabel('Natural Frequency [rad/s]')
plt.savefig('1dfig_alpha.png')
```

Appendix 1d-ii: Code for Problem 1d-ii

```
"""
Python script for Part 1d.ii of Project 2a

Original Author:  Vinamra Agrawal
Date:             January 25, 2019

Edited By:          Omkar Mulekar
Date:             February 28, 2019

"""

from __future__ import print_function
from fenics import *
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from ufl import nabla_div
import math
import numpy as np
from scipy.signal import argrelextrema

#==================================================================
# Define System Properties
#==================================================================
length = 1;
W = 0.2;
H1 = 0.2;
beta = np.linspace(0.2,1,num=5)
print("beta = ",beta)
H = beta * H1
print("H = ", H)




youngs = 200e9 # Youngs
nu = 0.3 # Poisson
rho = 7800 # Density


# Lame parameters
mu = (youngs)/(2*(1+nu))
```

```
lambda_ = (nu*youngs)/((1+nu)*(1-2*nu))

g = 10



nat_freq = [0] * len(beta)

print("Beginning Loop...")
for i in range(len(beta)):
    print("beta = ", beta[i])

    #==================================================================
    #  Dimensionless parameters
    #==================================================================
    l_nd = length/length
    w_nd = W/length
    h_nd = H[i]/length

    bar_speed = math.sqrt(youngs/rho)
    t_char = length/bar_speed
    t = 0
    t_i = 0.5
    dt = 0.1
    num_steps = 275

    mu_nd = mu/youngs
    lambda_nd = lambda_/youngs

    F = -100
    a = 0.04*length;
    b = 0.4*H[i];
    traction_applied = F/(a*b)
    traction_nd  = traction_applied/youngs

    #================================================================
    # Boundaries and Geometry
    #================================================================
    mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,6,6)
    V = VectorFunctionSpace(mesh, 'P',1)

    tol = 1E-14

    def boundary_left(x,on_boundary):
        return (on_boundary and near(x[0],0,tol))
```

```
def boundary_right(x,on_boundary):
    return on_boundary and near(x[0],l_nd,tol)

bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)
bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)


#==================================================================
def epsilon(u):
    return 0.5*(nabla_grad(u) + nabla_grad(u).T)

def sigma(u):
    return lambda_nd*nabla_div(u)*Identity(d) + mu_nd*(epsilon(u
        ) + epsilon(u).T)

#==================================================================
# First we solve the problem of a cantelever beam under fixed
# load.
#==================================================================
u_init = TrialFunction(V)
d = u_init.geometric_dimension()
v = TestFunction(V)
f = Constant((0.0,0.0,0.0))
T_init = Expression(('0.0', 'x[0] >= 0.48*l && x[0] <= .52*l &&
     near(x[1],w) && x[2] >= 0.3*h && x[2] <= 0.7*h? A : 0.0','
    0.0'), degree=1, l=l_nd, w=w_nd,h=h_nd, A=traction_nd)
F_init = inner(sigma(u_init),epsilon(v))*dx - dot(f,v)*dx - dot
    (T_init,v)*ds
a_init, L_init = lhs(F_init), rhs(F_init)

print("Solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right])
down_nd = u_init(l_nd/2.0,w_nd/2.0,h_nd/2.0)
w = down_nd * length
print("Initial Displacement: ",w[1])

#==================================================================
# Next we use this as initial condition, let the force go and
# study the vertical vibrations of the beam
#==================================================================
u_n = interpolate(Constant((0.0,0.0,0.0)),V)
u_n_1 = interpolate(Constant((0.0,0.0,0.0)),V)
u_n.assign(u_init)
```

```
u_n_1.assign(u_init)

T_n = Constant((0.0,0.0,0.0))

u = TrialFunction(V)
d = u.geometric_dimension()
v = TestFunction(V)

F = (dt*dt)*inner(sigma(u),epsilon(v))*dx + dot(u,v)*dx - (dt*
    dt)*dot(f,v)*dx - (dt*dt)*dot (T_n,v)*ds - 2.0*dot(u_n,v)*dx
     + dot(u_n_1,v)*dx
a,L = lhs(F), rhs(F)


u_store = [0] * num_steps
time = [0] * num_steps

index = 0
for n in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    u = Function(V)
    solve(a == L, u, [bc_left,bc_right])
    u_grab = u(l_nd/2.0,w_nd/2.0,h_nd/2.0)
    u_store[n] = u_grab[1]

    if(abs(t-index)<0.01):
        W_ = TensorFunctionSpace(mesh, "Lagrange", 1)
        stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(
            u) + epsilon(u).T)
        index += 1

    time[n] = t
    t+=dt
    u_n_1.assign(u_n)
    u_n.assign(u)

plt.figure(1)
plt.plot(time,u_store)
plt.xlabel('time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.savefig('1dfig_test2.png')

# Get period of oscillation
u_np = np.array(u_store)
```

```
    min_args = argrelextrema(u_np,np.greater)
    print("min_args", min_args[0])
    period = (time[min_args[0][1]] - time[min_args[0][0]])*t_char
    nat_freq[i] = 2*math.pi /period
    print("Period of Oscillation", period, " seconds")
    print("Natural Frequency:   ", nat_freq," rad/s")

plt.figure(2)
plt.plot(H,nat_freq, 'b-x')
plt.xlabel('Beam Height [m]')
plt.ylabel('Natural Frequency [rad/s]')
plt.savefig('1dfig_beta.png')
```