Project 4

Omkar Mulekar
AERO 4630-002
Aerospace Structural Dynamics

5 April 2019

Part 1a: Plate Clamped at One End

Vibrational analysis was performed on a plate of length $L = 1$m, width $W = 1$m, and height $H = 0.001$m. In this first part, the plate is clamped at one end, and an initial force of $F = 100$N is applied to the other end over the area $0.98L \leq x \leq L$, $0.48W \leq y \leq 0.51W$. To display the vibration of the plate, the displacements at several points on the plate are plotted over time. The respective natural frequencies at each point are also plotted. See Figures 1-4.

Please note that the code finds the natural frequencies by determining the indices of each relative maximum in the list of displacements. Then, it calls times using the first two indices and subtracts them to obtain the period. Then the natural frequency $\omega_n = 2 * pi/T$. Because the plate has so many degrees of freedom, there are many modes of vibration. In early cycles, there are multiple dominating modes of vibration, and the method by which the code used in this project determines natural frequency can find multiple frequencies. Thus, plots of natural frequency presented in this document may show unexpected jumps in frequency between points on the plate.

Part 1b: Plate Clamped at One End (but bigger)

The same analysis was performed for a plate of length $L = 2$m and width $W = 2$m. See Figures 5-8. The amplitudes appear to have doubled, while the natural frequencies are the same.

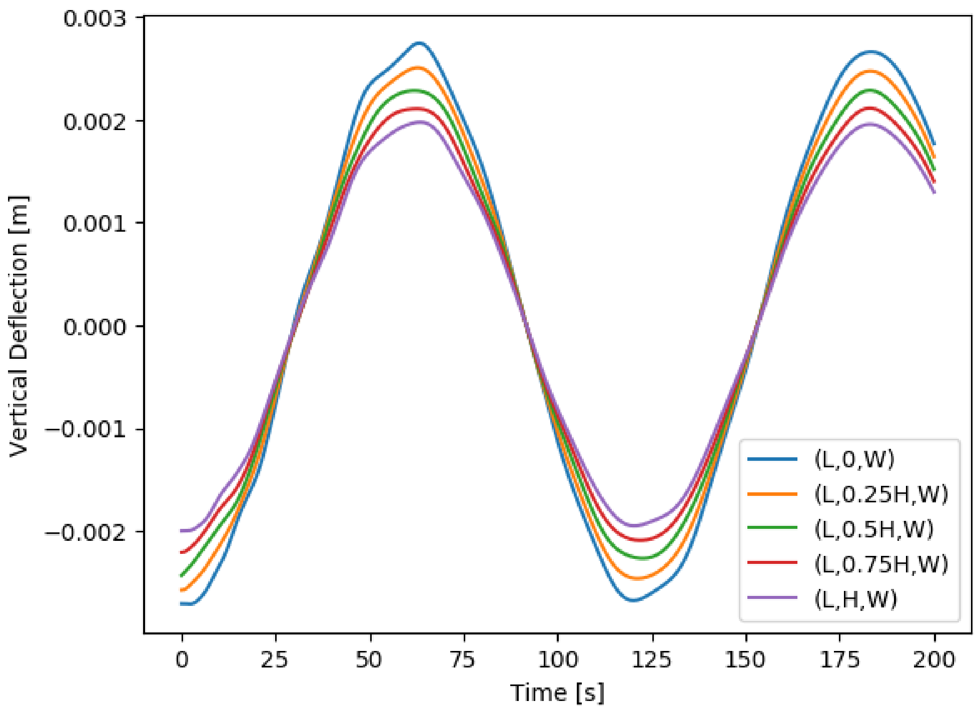

Figure 1: 1A: Displacements along width of 1m plate at $L = L$

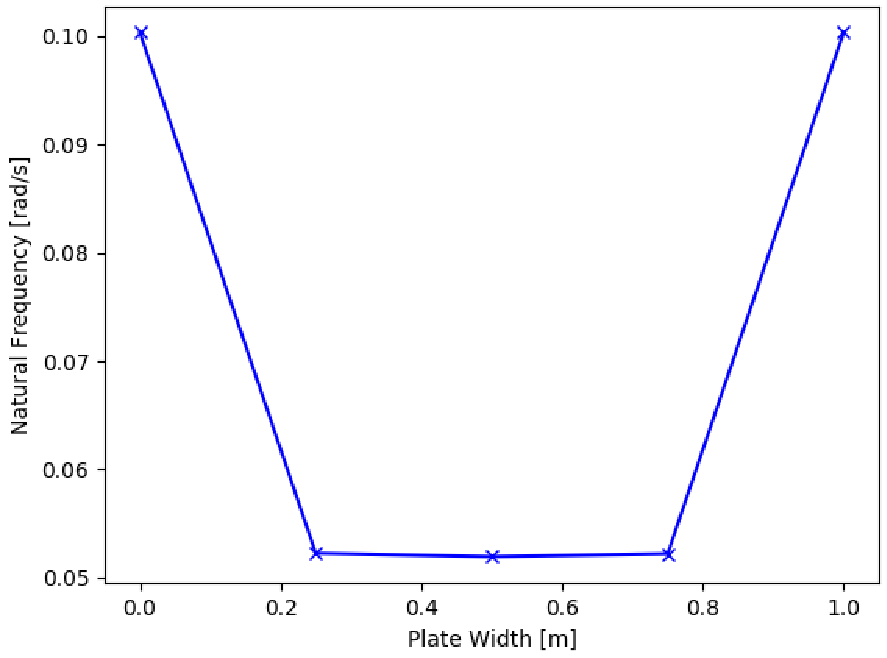

Figure 2: 1A: Natural Frequencies along width of 1m plate at $L = L$

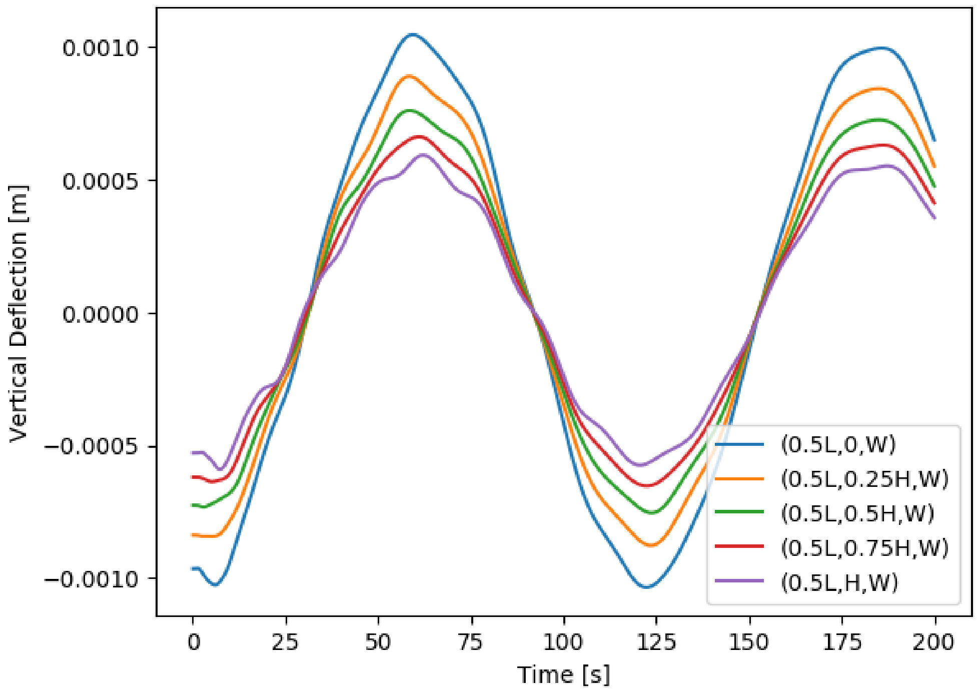

Figure 3: 1A: Displacements along width of 1m plate at $L = 0.5L$

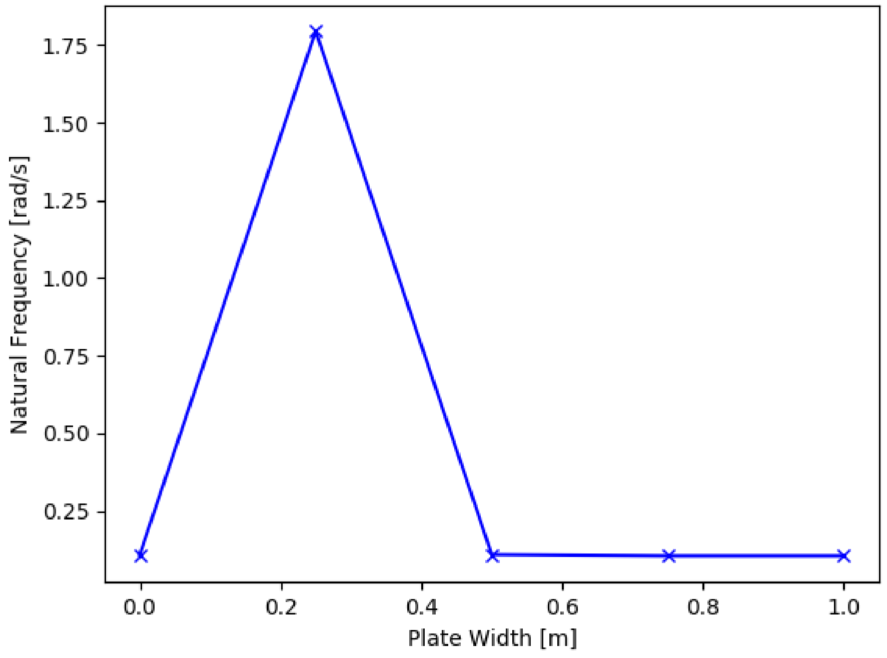

Figure 4: 1A: Natural Frequencies along width of 1m plate at $L = 0.5L$

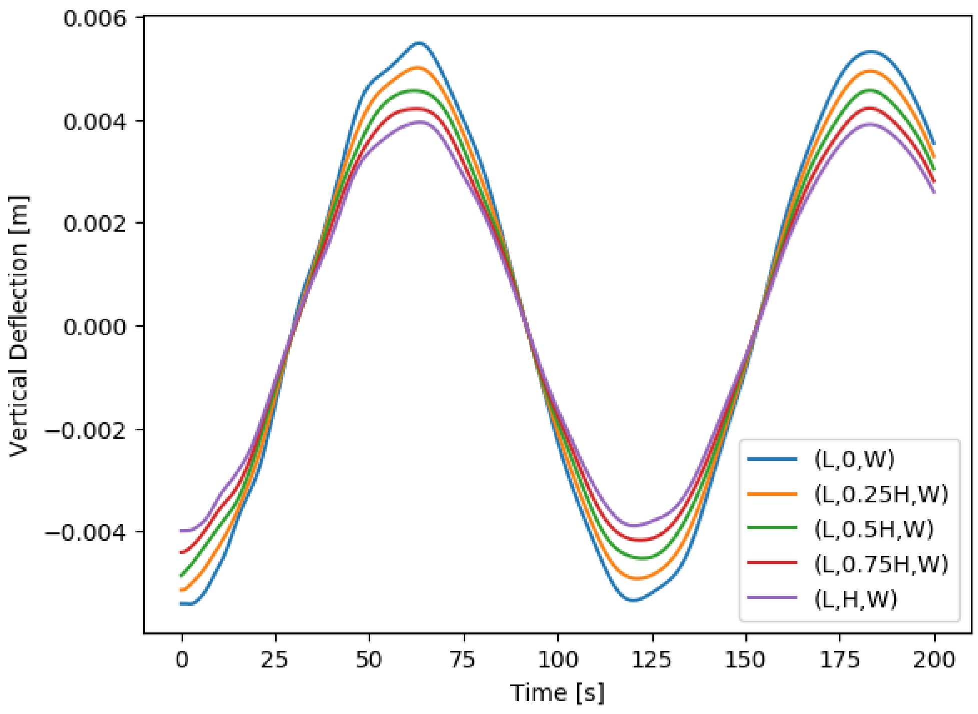

Figure 5: 1B: Displacements along width of 2m plate at $L = L$

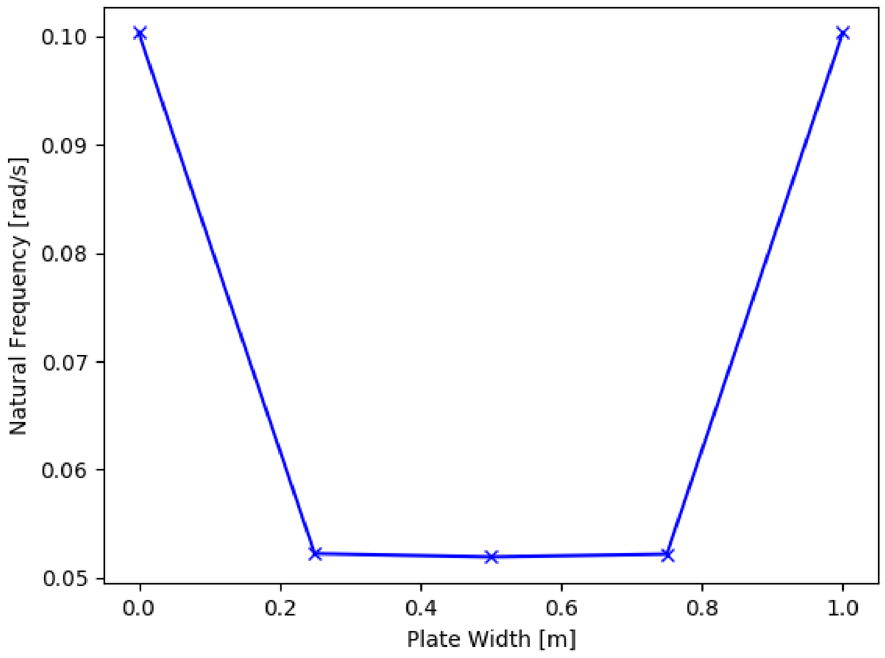

Figure 6: 1B: Natural Frequencies along width of 2m plate at $L = L$

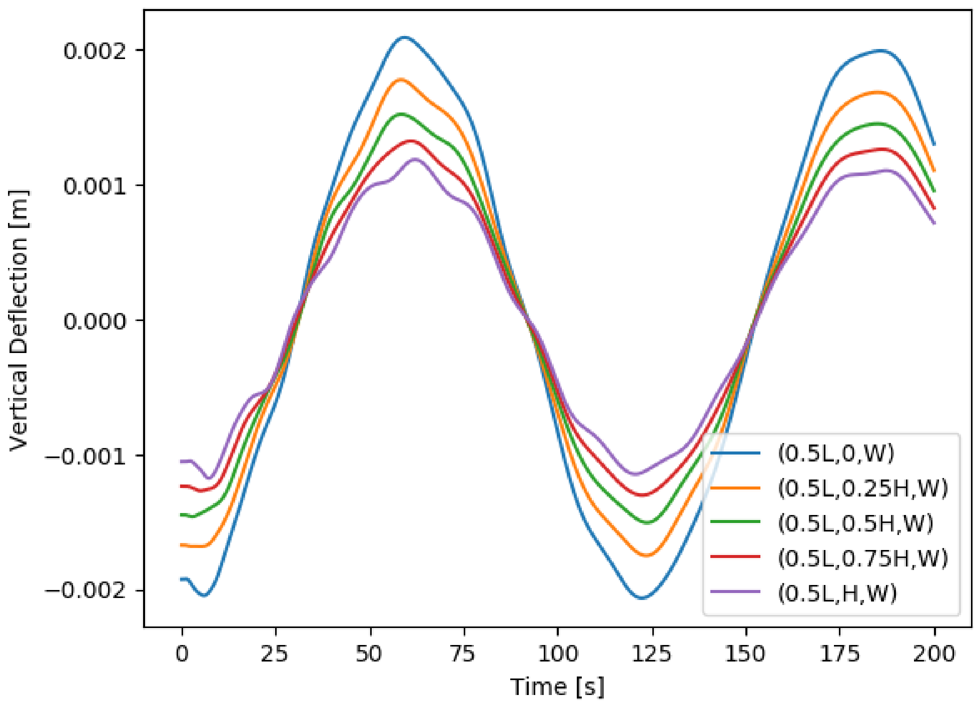

Figure 7: 1B: Displacements along width of 2m plate at $L = 0.5L$

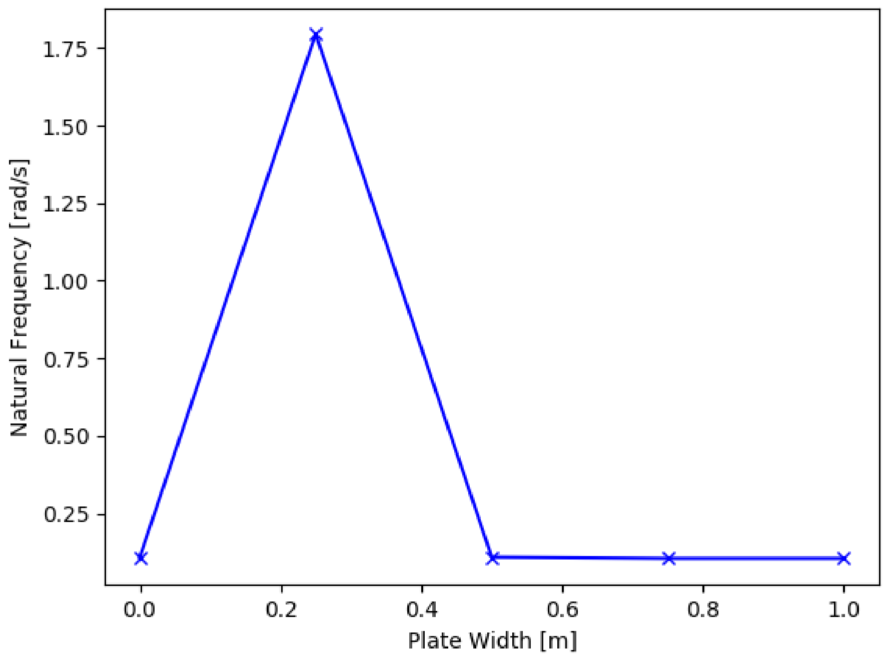

Figure 8: 1B: Natural Frequencies along width of 2m plate at $L = 0.5L$

Part 1c: Plate Clamped at Two Ends

The 1m plate was then clamped at 2 ends ($x = 0$ and $x = L$). See Figures 9-12. The amplitudes can be seen on the displacement plots.

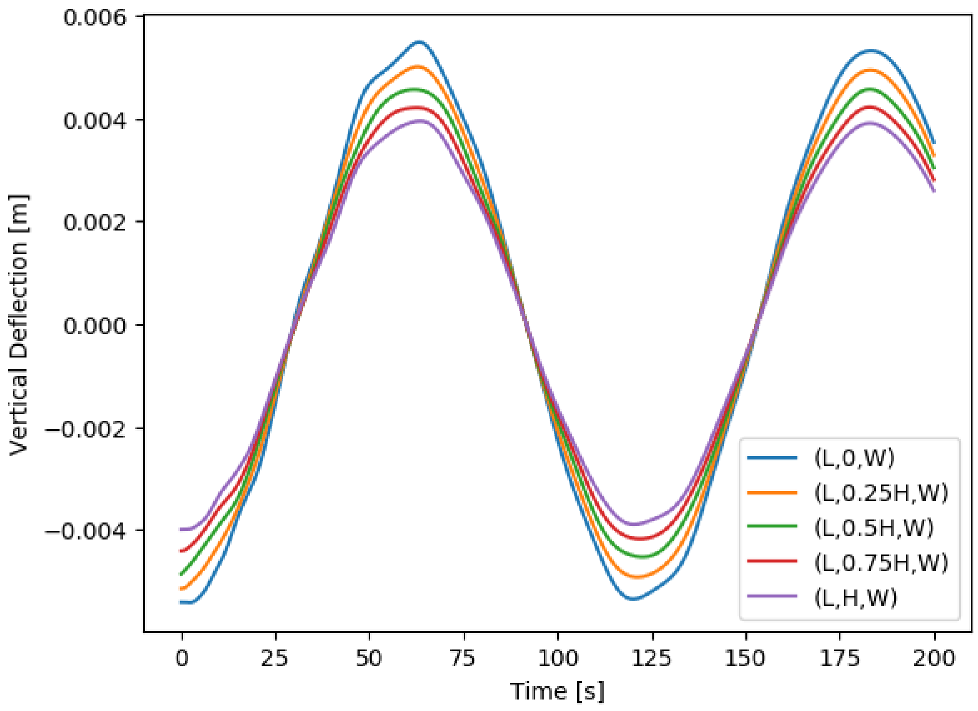

Figure 9: 1C: Displacements along width of 1m plate at $L = L$

Part 1d: Plate Clamped on All Ends

Vibrational analysis was then performed for plates of length/width 1m and 2m. See Figures 13-20. Again, the amplitudes appear to double while the frequencies do not change from the 1m to the 2m simulations.

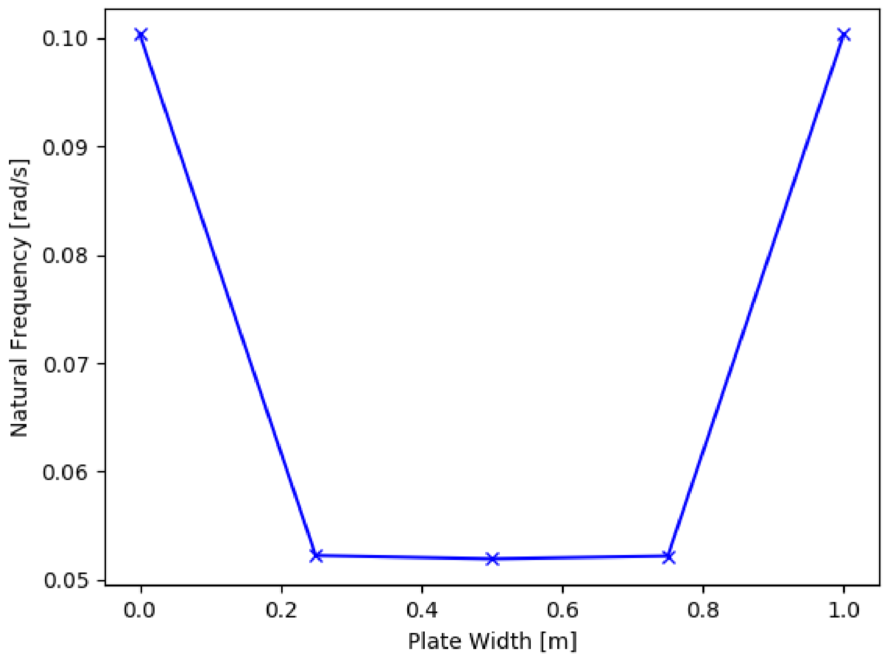

Figure 10: 1C: Natural Frequencies along width of 1m plate at $L = L$

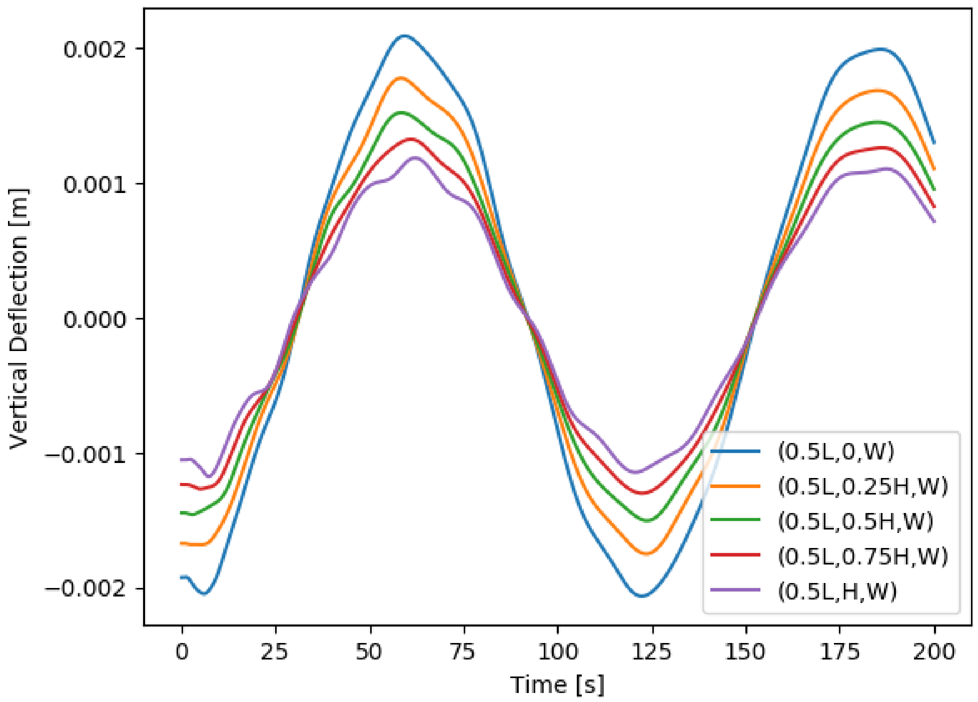

Figure 11: 1C: Displacements along width of 1m plate at $L = 0.5L$

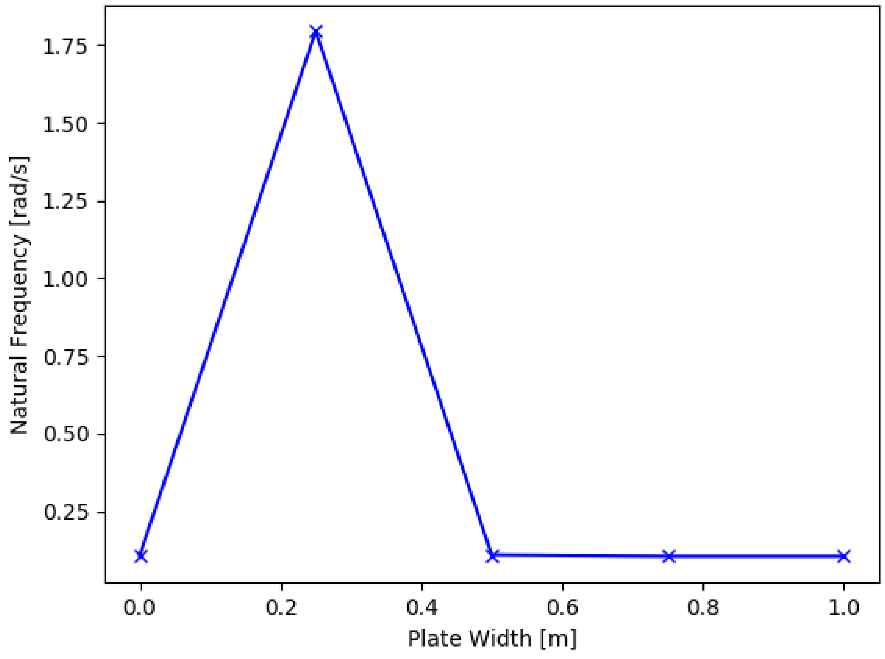

Figure 12: 1C: Natural Frequencies along width of 1m plate at $L = 0.5L$

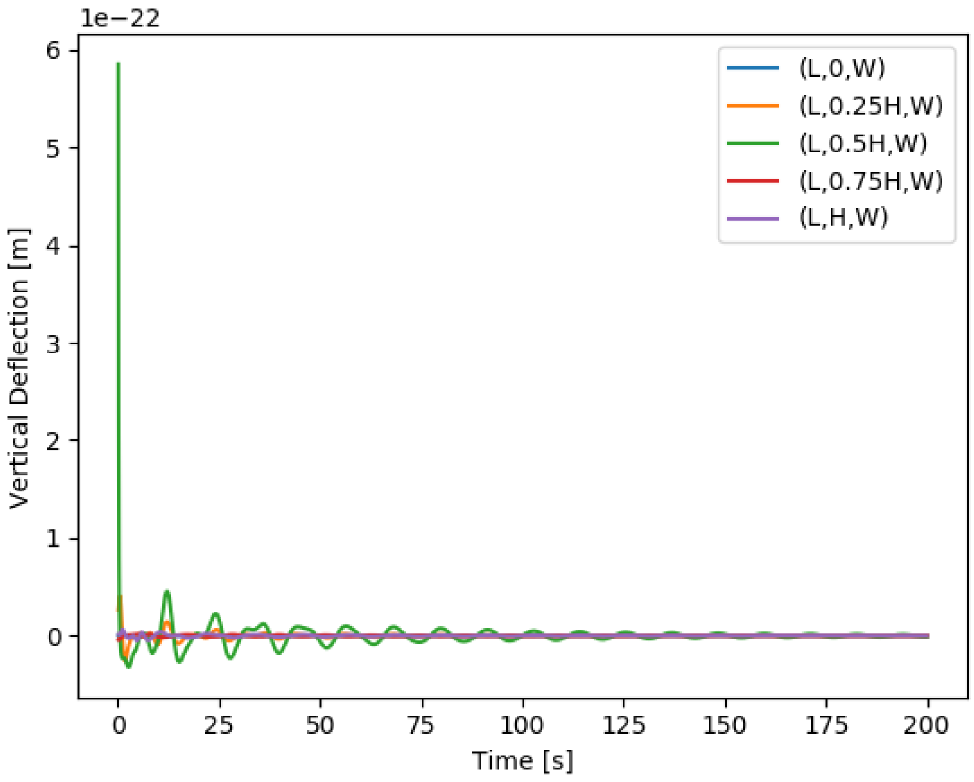

Figure 13: 1D: Displacements along width of 1m plate at $L = L$

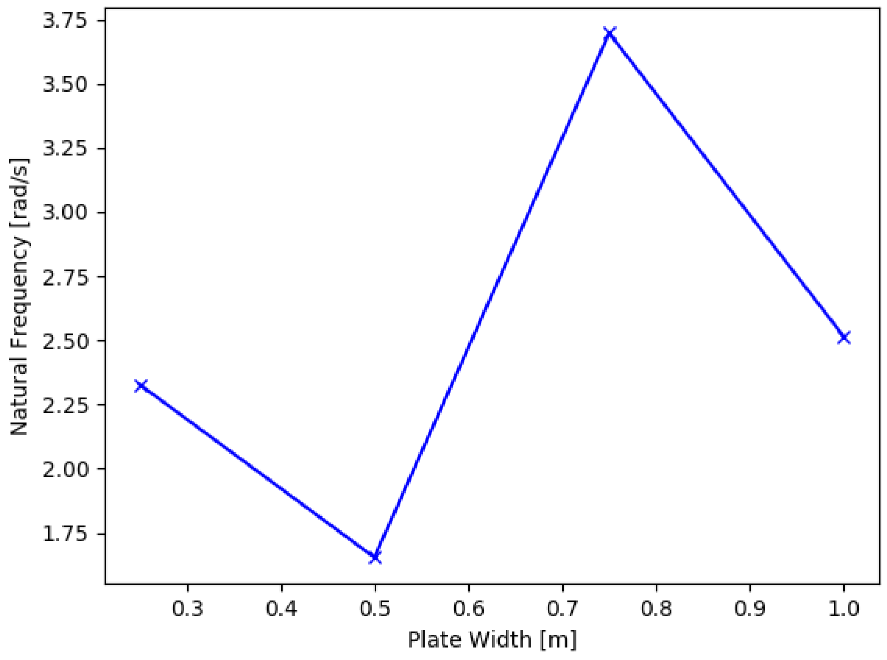

Figure 14: 1D: Natural Frequencies along width of 1m plate at $L = L$

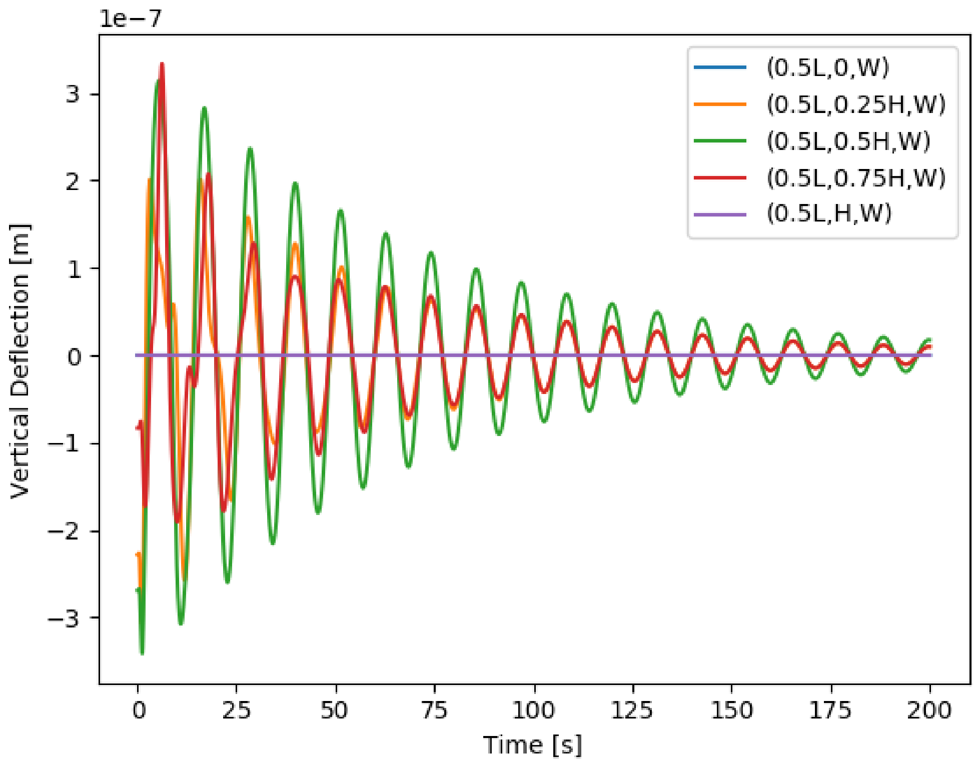

Figure 15: 1D: Displacements along width of 1m plate at $L = 0.5L$

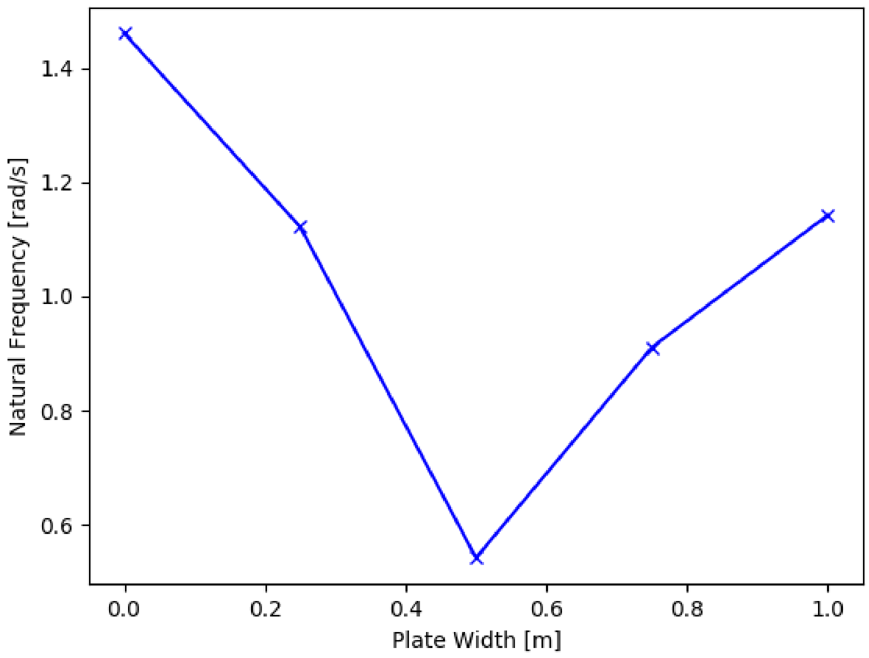

Figure 16: 1D: Natural Frequencies along width of 1m plate at $L = 0.5L$

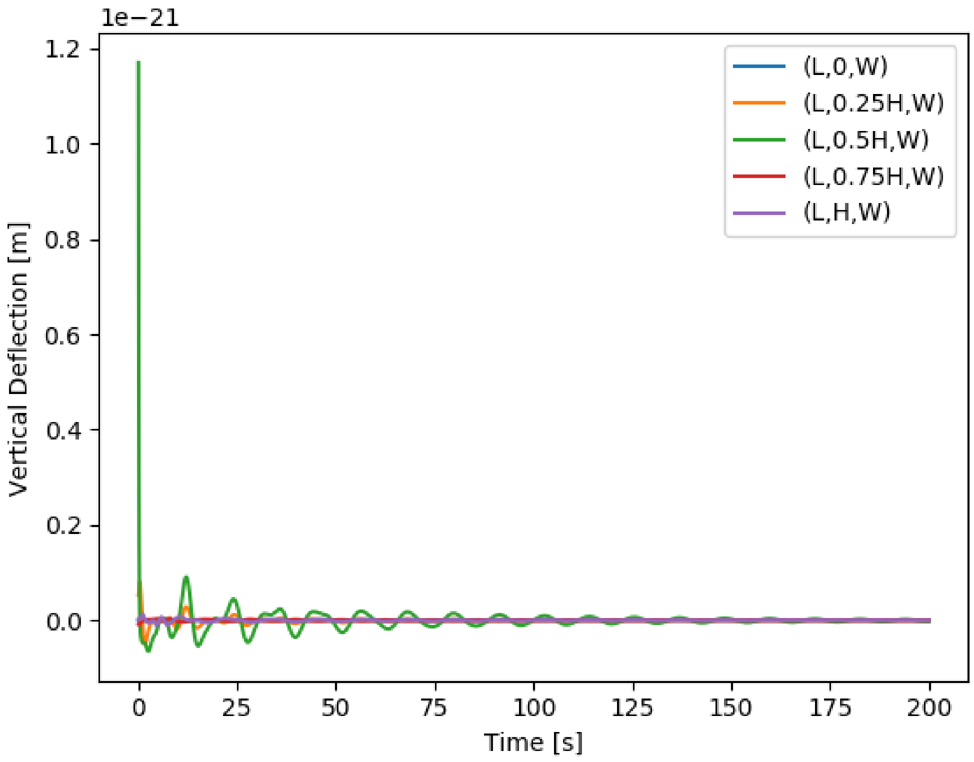

Figure 17: 1D: Displacements along width of 2m plate at $L = L$

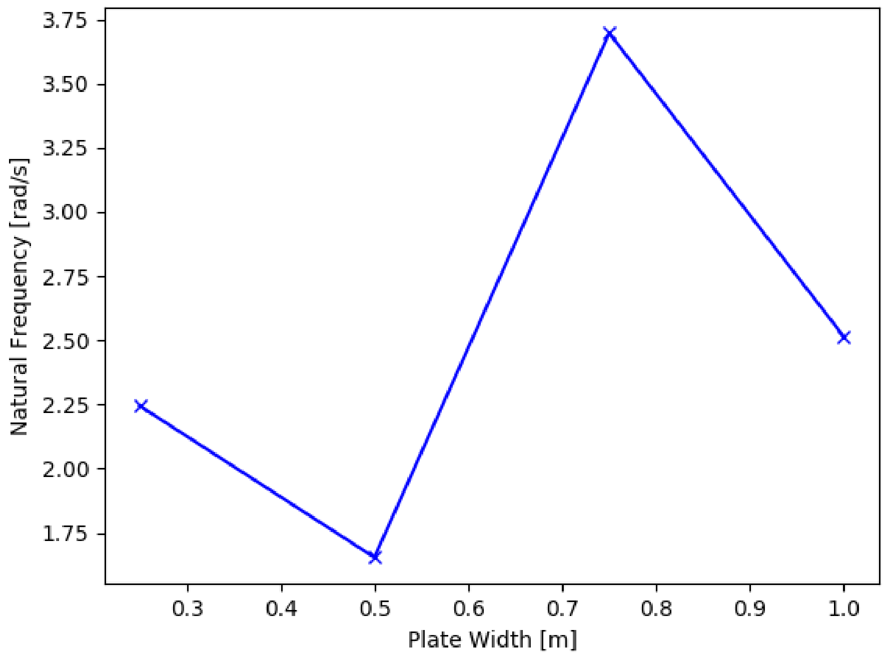

Figure 18: 1D: Natural Frequencies along width of 2m plate at $L = L$

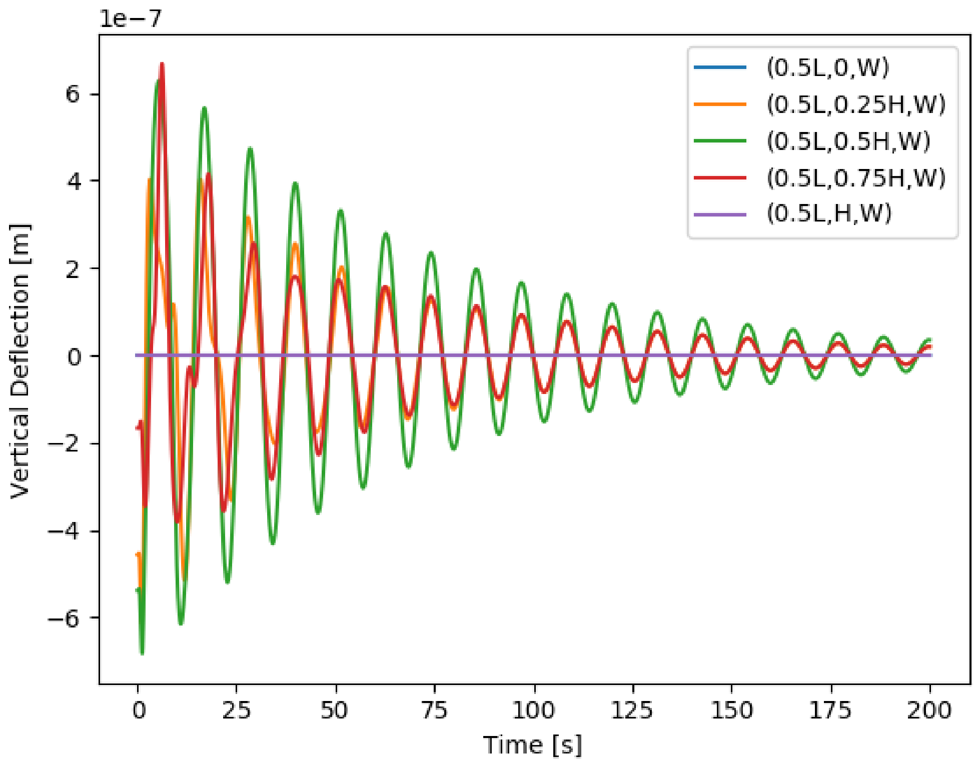

Figure 19: 1D: Displacements along width of 2m plate at $L = 0.5L$

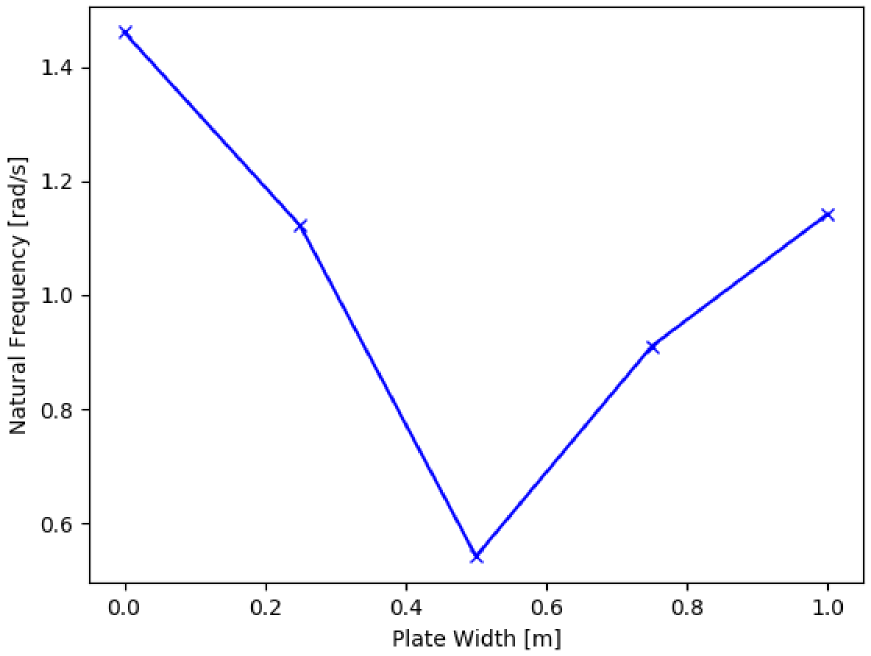

Figure 20: 1D: Natural Frequencies along width of 2m plate at $L = 0.5L$

Appendix 1a-i: Code for Problem 1a

```
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
from scipy.signal import savgol_filter

#==========================================================
#  Dimensional parameters
#==========================================================
length = 1.0
W = 1.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==========================================================
#  Dimensionless parameters
#==========================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#=============================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

boundary_left = 'near(x[0],0)'
bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

tol = 1E-14

#=============================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
   && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init, u_init, bc_left)

#==================================================================
# 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 = Expression(('near(x[0], l) ? (t <= t_i ? A : 0.0) :
    0.0', '0.0', '0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
    t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps
```

```
for n in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    u = Function(V)
    solve(a == L, u, bc_left)

    u_grab = u(length,widths[0]*W,H)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(length,widths[1]*W,H)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(length,widths[2]*W,H)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(length,widths[3]*W,H)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(length,widths[4]*W,H)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]

    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length,t)
    #   stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
    #   index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(L,0,W)')
plt.plot(time,displacements1,label='(L,0.25H,W)')
plt.plot(time,displacements2,label='(L,0.5H,W)')
plt.plot(time,displacements3,label='(L,0.75H,W)')
plt.plot(time,displacements4,label='(L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_a/1a_i_disps.png',bbox_inches='tight')


nat_freq = [0]*len(widths)

displacements0 = savgol_filter(displacements0, 51, 4)
u_np = np.array(displacements0)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[2] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period

displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[4] = 2*math.pi /period

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

Appendix 1a-ii: Code for Problem 1a

```
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
from scipy.signal import savgol_filter

#==================================================================
#  Dimensional parameters
#==================================================================
length = 1.0
W = 1.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==================================================================
#  Dimensionless parameters
#==================================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#==========================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

boundary_left = 'near(x[0],0)'
bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

tol = 1E-14

#==========================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
   && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init, u_init, bc_left)

#=============================================================
# 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 = Expression(('near(x[0], l) ? (t <= t_i ? A : 0.0) :
    0.0', '0.0', '0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
    t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps
```

```
for n in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    u = Function(V)
    solve(a == L, u, bc_left)

    u_grab = u(0.5*length,widths[0]*W,H)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(0.5*length,widths[1]*W,H)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(0.5*length,widths[2]*W,H)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(0.5*length,widths[3]*W,H)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(0.5*length,widths[4]*W,H)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]

    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length,t)
    #   stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
    #   index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(0.5L,0,W)')
plt.plot(time,displacements1,label='(0.5L,0.25H,W)')
plt.plot(time,displacements2,label='(0.5L,0.5H,W)')
plt.plot(time,displacements3,label='(0.5L,0.75H,W)')
plt.plot(time,displacements4,label='(0.5L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_a/1a_ii_disps.png',bbox_inches='tight')


nat_freq = [0]*len(widths)

displacements0 = savgol_filter(displacements0, 51, 4)
u_np = np.array(displacements0)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[2] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period

displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[4] = 2*math.pi /period

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

Appendix 1b-i: Code for Problem 1b

```
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
from scipy.signal import savgol_filter

#==============================================================
#  Dimensional parameters
#==============================================================
length = 2.0
W = 2.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==============================================================
#  Dimensionless parameters
#==============================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#==================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

boundary_left = 'near(x[0],0)'
bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

tol = 1E-14

#==================================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
   && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init, u_init, bc_left)

#==================================================================
# 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 = Expression(('near(x[0],l) ? (t <= t_i ? A : 0.0) :
    0.0','0.0','0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
    t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps
```

```
for n in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    u = Function(V)
    solve(a == L, u, bc_left)

    u_grab = u(l_nd,widths[0]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(l_nd,widths[1]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(l_nd,widths[2]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(l_nd,widths[3]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(l_nd,widths[4]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]

    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length,t)
    #   stress = lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
       + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
    #   index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(L,0,W)')
plt.plot(time,displacements1,label='(L,0.25H,W)')
plt.plot(time,displacements2,label='(L,0.5H,W)')
plt.plot(time,displacements3,label='(L,0.75H,W)')
plt.plot(time,displacements4,label='(L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_b/1a_i_disps.png',bbox_inches='tight')


nat_freq = [0]*len(widths)

displacements0 = savgol_filter(displacements0, 51, 4)
u_np = np.array(displacements0)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[2] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period

displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[4] = 2*math.pi /period

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

Appendix 1b-ii: Code for Problem 1b

```
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
from scipy.signal import savgol_filter

#==========================================================================
#  Dimensional parameters
#==========================================================================
length = 2.0
W = 2.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==========================================================================
#  Dimensionless parameters
#==========================================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#==============================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

boundary_left = 'near(x[0],0)'
bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

tol = 1E-14

#==============================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
   && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init, u_init, bc_left)

#==================================================================
# 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 = Expression(('near(x[0], l) ? (t <= t_i ? A : 0.0) :
    0.0', '0.0', '0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
    t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps
```

```
for n in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    u = Function(V)
    solve(a == L, u, bc_left)

    u_grab = u(0.5*l_nd,widths[0]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(0.5*l_nd,widths[1]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(0.5*l_nd,widths[2]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(0.5*l_nd,widths[3]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(0.5*l_nd,widths[4]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]

    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length,t)
    #   stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
    #   index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(0.5L,0,W)')
plt.plot(time,displacements1,label='(0.5L,0.25H,W)')
plt.plot(time,displacements2,label='(0.5L,0.5H,W)')
plt.plot(time,displacements3,label='(0.5L,0.75H,W)')
plt.plot(time,displacements4,label='(0.5L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_b/1a_ii_disps.png',bbox_inches='tight')


nat_freq = [0]*len(widths)

displacements0 = savgol_filter(displacements0, 51, 4)
u_np = np.array(displacements0)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[2] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period

displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[4] = 2*math.pi /period

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

Appendix 1c-i: Code for Problem 1c

```
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
from scipy.signal import savgol_filter

#==============================================================
#  Dimensional parameters
#==============================================================
length = 1.0
W = 1.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==============================================================
#  Dimensionless parameters
#==============================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#==========================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

# boundary_left = 'near(x[0],0)'
# bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

# boundary_right = 'near(x[0],l_nd)'
# bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
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)

tol = 1E-14

#==========================================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
    && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right])

#==========================================================================
# 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 = Expression(('near(x[0],l) ? (t <= t_i ? A : 0.0) :
    0.0','0.0','0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
    t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
```

```
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps

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(length,widths[0]*W,H)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(length,widths[1]*W,H)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(length,widths[2]*W,H)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(length,widths[3]*W,H)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(length,widths[4]*W,H)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]

    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length,t)
    #   stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
```

```
        #  index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(L,0,W)')
plt.plot(time,displacements1,label='(L,0.25H,W)')
plt.plot(time,displacements2,label='(L,0.5H,W)')
plt.plot(time,displacements3,label='(L,0.75H,W)')
plt.plot(time,displacements4,label='(L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_c/1c_i_disps.png',bbox_inches='tight')


nat_freq = [0]*len(widths)

displacements0 = savgol_filter(displacements0, 51, 4)
u_np = np.array(displacements0)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[2] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period
```

```
displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[4] = 2*math.pi /period

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

Appendix 1c-ii: Code for Problem 1c

```
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
from scipy.signal import savgol_filter

#=====================================================================
#  Dimensional parameters
#=====================================================================
length = 1.0
W = 1.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#=====================================================================
#  Dimensionless parameters
#=====================================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#===================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

# boundary_left = 'near(x[0],0)'
# bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

# boundary_right = 'near(x[0],l_nd)'
# bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
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)

tol = 1E-14

#===================================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
    && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left, bc_right])

#==========================================================================
# 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 = Expression(('near(x[0],l) ? (t <= t_i ? A : 0.0) :
    0.0','0.0','0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
    t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
```

```
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps

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(0.5*length,widths[0]*W,H)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(0.5*length,widths[1]*W,H)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(0.5*length,widths[2]*W,H)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(0.5*length,widths[3]*W,H)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(0.5*length,widths[4]*W,H)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]

    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length, t)
    #   stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj, t)
    #  index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(0.5L,0,W)')
plt.plot(time,displacements1,label='(0.5L,0.25H,W)')
plt.plot(time,displacements2,label='(0.5L,0.5H,W)')
plt.plot(time,displacements3,label='(0.5L,0.75H,W)')
plt.plot(time,displacements4,label='(0.5L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_c/1c_ii_disps.png',bbox_inches='tight')


nat_freq = [0]*len(widths)

displacements0 = savgol_filter(displacements0, 51, 4)
u_np = np.array(displacements0)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[2] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period
```

```
displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[4] = 2*math.pi /period

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

Appendix 1d-i: Code for Problem 1d

```
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
from scipy.signal import savgol_filter

#==============================================================================
#  Dimensional parameters
#==============================================================================
length = 1.0
W = 1.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==============================================================================
#  Dimensionless parameters
#==============================================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#==========================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

# boundary_left = 'near(x[0],0)'
# bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

# boundary_right = 'near(x[0],l_nd)'
# bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
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)

def boundary_front(x,on_boundary):
    return on_boundary and near(x[1],0,tol)

def boundary_back(x,on_boundary):
    return on_boundary and near(x[1],w_nd,tol)

bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)
bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
bc_front = DirichletBC(V,Constant((0,0,0)),boundary_front)
bc_back = DirichletBC(V,Constant((0,0,0)),boundary_back)

tol = 1E-14

#==========================================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
   && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right,bc_front,bc_back])

#==========================================================================
# 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 = Expression(('near(x[0],l) ? (t <= t_i ? A : 0.0) :
   0.0','0.0','0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
   t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps

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,bc_front,bc_back])

    u_grab = u(length,widths[0]*W,H)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(length,widths[1]*W,H)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(length,widths[2]*W,H)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(length,widths[3]*W,H)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(length,widths[4]*W,H)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]
    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length,t)
    #   stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
    #   index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(L,0,W)')
plt.plot(time,displacements1,label='(L,0.25H,W)')
plt.plot(time,displacements2,label='(L,0.5H,W)')
plt.plot(time,displacements3,label='(L,0.75H,W)')
plt.plot(time,displacements4,label='(L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_d/1d_i_disps.png',bbox_inches='tight')


widths = [0.25,0.5,0.75,1]
nat_freq = [0]*len(widths)

# displacements0 = savgol_filter(displacements0, 51, 4)
# u_np = np.array(displacements0)
# min_args = argrelextrema(u_np,np.greater)
# period = (time[min_args[0][1]] - time[min_args[0][0]])
# nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
```

```
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[2] = 2*math.pi /period

displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period


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

Appendix 1d-ii: Code for Problem 1d

```
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
from scipy.signal import savgol_filter

#==============================================================
#  Dimensional parameters
#==============================================================
length = 1.0
W = 1.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==============================================================
#  Dimensionless parameters
#==============================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#==========================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

# boundary_left = 'near(x[0],0)'
# bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

# boundary_right = 'near(x[0],l_nd)'
# bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
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)

def boundary_front(x,on_boundary):
    return on_boundary and near(x[1],0,tol)

def boundary_back(x,on_boundary):
    return on_boundary and near(x[1],w_nd,tol)

bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)
bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
bc_front = DirichletBC(V,Constant((0,0,0)),boundary_front)
bc_back = DirichletBC(V,Constant((0,0,0)),boundary_back)

tol = 1E-14

#==========================================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
   && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right,bc_front,bc_back])

#==============================================================
# 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 = Expression(('near(x[0],l) ? (t <= t_i ? A : 0.0) :
   0.0','0.0','0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
   t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps

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,bc_front,bc_back])

    u_grab = u(0.5*length,widths[0]*W,H)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(0.5*length,widths[1]*W,H)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(0.5*length,widths[2]*W,H)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(0.5*length,widths[3]*W,H)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(0.5*length,widths[4]*W,H)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]
    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length,t)
    #   stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
    #   index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(0.5L,0,W)')
plt.plot(time,displacements1,label='(0.5L,0.25H,W)')
plt.plot(time,displacements2,label='(0.5L,0.5H,W)')
plt.plot(time,displacements3,label='(0.5L,0.75H,W)')
plt.plot(time,displacements4,label='(0.5L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_d/1d_ii_disps.png',bbox_inches='tight')


nat_freq = [0]*len(widths)

displacements0 = savgol_filter(displacements0, 51, 4)
u_np = np.array(displacements0)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
```

```
nat_freq[2] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period

displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[4] = 2*math.pi /period

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

Appendix 1d2-i: Code for Problem 1d

```
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
from scipy.signal import savgol_filter

#==============================================================
#  Dimensional parameters
#==============================================================
length = 2.0
W = 2.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==============================================================
#  Dimensionless parameters
#==============================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#==============================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

# boundary_left = 'near(x[0],0)'
# bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

# boundary_right = 'near(x[0],l_nd)'
# bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
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)

def boundary_front(x,on_boundary):
    return on_boundary and near(x[1],0,tol)

def boundary_back(x,on_boundary):
    return on_boundary and near(x[1],w_nd,tol)

bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)
bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
bc_front = DirichletBC(V,Constant((0,0,0)),boundary_front)
bc_back = DirichletBC(V,Constant((0,0,0)),boundary_back)

tol = 1E-14

#==============================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
   && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right,bc_front,bc_back])

#==================================================================
# 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 = Expression(('near(x[0],l) ? (t <= t_i ? A : 0.0) :
   0.0','0.0','0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
   t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps

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,bc_front,bc_back])

    u_grab = u(l_nd,widths[0]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(l_nd,widths[1]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(l_nd,widths[2]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(l_nd,widths[3]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(l_nd,widths[4]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]
    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length, t)
    #   stress = lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj, t)
    #   index += 1

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

plt.figure(1)
plt.plot(time,displacements0,label='(L,0,W)')
plt.plot(time,displacements1,label='(L,0.25H,W)')
plt.plot(time,displacements2,label='(L,0.5H,W)')
plt.plot(time,displacements3,label='(L,0.75H,W)')
plt.plot(time,displacements4,label='(L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_d2/1d2_i_disps.png',bbox_inches='tight')


widths = [0.25,0.5,0.75,1]
nat_freq = [0]*len(widths)

# displacements0 = savgol_filter(displacements0, 51, 4)
# u_np = np.array(displacements0)
# min_args = argrelextrema(u_np,np.greater)
# period = (time[min_args[0][1]] - time[min_args[0][0]])
# nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
```

```
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[2] = 2*math.pi /period

displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period

plt.figure(2)
plt.plot(widths,nat_freq,'b-x')
plt.xlabel('Plate Width [m]')
plt.ylabel('Natural Frequency [rad/s]')
plt.savefig('results_d2/1d2_i_natfreq.png')
```

Appendix 1d2-ii: Code for Problem 1d

```
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
from scipy.signal import savgol_filter

#==================================================================
#  Dimensional parameters
#==================================================================
length = 2.0
W = 2.0
H = 0.001

nx = 20
ny = 20
nz = 6

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))

F = -100;

traction_applied = F/((1-.98)*(.51-.49))


#==================================================================
#  Dimensionless parameters
#==================================================================
youngs = (mu*(3.0*lambda_+2.0*mu))/(lambda_+mu)
bar_speed = math.sqrt(youngs/rho)
```

```
l_nd = length/length
w_nd = W/length
h_nd = H/length

t_char = length/bar_speed
t = 0
t_i = 0.5
dt = 0.1
num_steps = 2000

mu_nd = mu/youngs
lambda_nd = lambda_/youngs

traction_nd = traction_applied/youngs

#===========================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),nx,ny,nz)
V = VectorFunctionSpace(mesh,'P',1)

# boundary_left = 'near(x[0],0)'
# bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)

# boundary_right = 'near(x[0],l_nd)'
# bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
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)

def boundary_front(x,on_boundary):
    return on_boundary and near(x[1],0,tol)

def boundary_back(x,on_boundary):
    return on_boundary and near(x[1],w_nd,tol)

bc_left = DirichletBC(V,Constant((0,0,0)),boundary_left)
bc_right = DirichletBC(V,Constant((0,0,0)),boundary_right)
bc_front = DirichletBC(V,Constant((0,0,0)),boundary_front)
bc_back = DirichletBC(V,Constant((0,0,0)),boundary_back)

tol = 1E-14

#===========================================================================
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', '0.0' ,'(x[0] >= 0.98*l && x[0] <= l)
   && (x[1] >= 0.49*w && x[1] <= 0.51*w) && near(x[2],h)? A : 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("First solving the initial cantelever problem")
u_init = Function(V)
solve(a_init==L_init,u_init,[bc_left,bc_right,bc_front,bc_back])

#==================================================================
# 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 = Expression(('near(x[0],l) ? (t <= t_i ? A : 0.0) :
   0.0','0.0','0.0'), degree=1, l=l_nd, A=traction_nd, t=t, t_i=
   t_i)
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)
Q = TensorFunctionSpace(mesh, "Lagrange", 1)
stress_proj = Function(Q)

index = 0
displacements0 = [0] * num_steps;
displacements1 = [0] * num_steps;
displacements2 = [0] * num_steps;
displacements3 = [0] * num_steps;
displacements4 = [0] * num_steps;
widths = [0,0.25,0.5,0.75,1]
u_store = [0] * num_steps
time = [0] * num_steps

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,bc_front,bc_back])

    u_grab = u(0.5*l_nd,widths[0]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements0[n] = u_store[n]

    u_grab = u(0.5*l_nd,widths[1]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements1[n] = u_store[n]

    u_grab = u(0.5*l_nd,widths[2]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements2[n] = u_store[n]

    u_grab = u(0.5*l_nd,widths[3]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements3[n] = u_store[n]

    u_grab = u(0.5*l_nd,widths[4]*w_nd,h_nd)
    u_store[n] = u_grab[2]
    displacements4[n] = u_store[n]
    # if(abs(t-index)<0.01):
    #   print("Calculating stress...")
    #   xdmffile_u.write(u*length, t)
    #   stress =  lambda_*nabla_div(u)*Identity(d) + mu*(epsilon(u)
        + epsilon(u).T)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj, t)
    #   index += 1

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

plt.figure(1)
plt.plot(time,displacements0, label='(0.5L,0,W)')
plt.plot(time,displacements1, label='(0.5L,0.25H,W)')
plt.plot(time,displacements2, label='(0.5L,0.5H,W)')
plt.plot(time,displacements3, label='(0.5L,0.75H,W)')
plt.plot(time,displacements4, label='(0.5L,H,W)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_d2/1d2_ii_disps.png',bbox_inches='tight')


nat_freq = [0]*len(widths)

displacements0 = savgol_filter(displacements0, 51, 4)
u_np = np.array(displacements0)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[0] = 2*math.pi /period

displacements1 = savgol_filter(displacements1, 51, 4)
u_np = np.array(displacements1)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[1] = 2*math.pi /period

displacements2 = savgol_filter(displacements2, 51, 4)
u_np = np.array(displacements2)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
```

```
nat_freq[2] = 2*math.pi /period

displacements3 = savgol_filter(displacements3, 51, 4)
u_np = np.array(displacements3)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[3] = 2*math.pi /period

displacements4 = savgol_filter(displacements4, 51, 4)
u_np = np.array(displacements4)
min_args = argrelextrema(u_np,np.greater)
period = (time[min_args[0][1]] - time[min_args[0][0]])
nat_freq[4] = 2*math.pi /period

plt.figure(2)
plt.plot(widths,nat_freq,'b-x')
plt.xlabel('Plate Width [m]')
plt.ylabel('Natural Frequency [rad/s]')
plt.savefig('results_d2/1d2_ii_natfreq.png')
```