Project 5

Omkar Mulekar
AERO 4630-002
Aerospace Structural Dynamics

24 April 2019

Part 1a: Clamped Composite Beam

The vibrational response to composite beam of length $L = 3$m,$W = 0.1$m, and $h = 0.1$m to an impulse traction of $T = 10^5$Pa to on the right face a $x = L$ was simulated and assessed. The left end $x = 0$ of the beam is clamped. The beam has three layers, each $1/3W$ thick. The top and bottom layers are steel and the middle layer is copper. A plot of the vertical deflection of the end of the beam is shown in Figure 1.

The natural frequency is 0.0021869 rad/s, and the amplitude is 0.55 m.

Part 1b: Thicker copper layer

The middle copper layer was changed to a thickness of $3/5W$, and the top and bottom steel layers were each changed to $1/5W$. A plot of the vertical deflection of the end of the beam is shown in Figure 2.

The overall frequency and amplitude did not change.

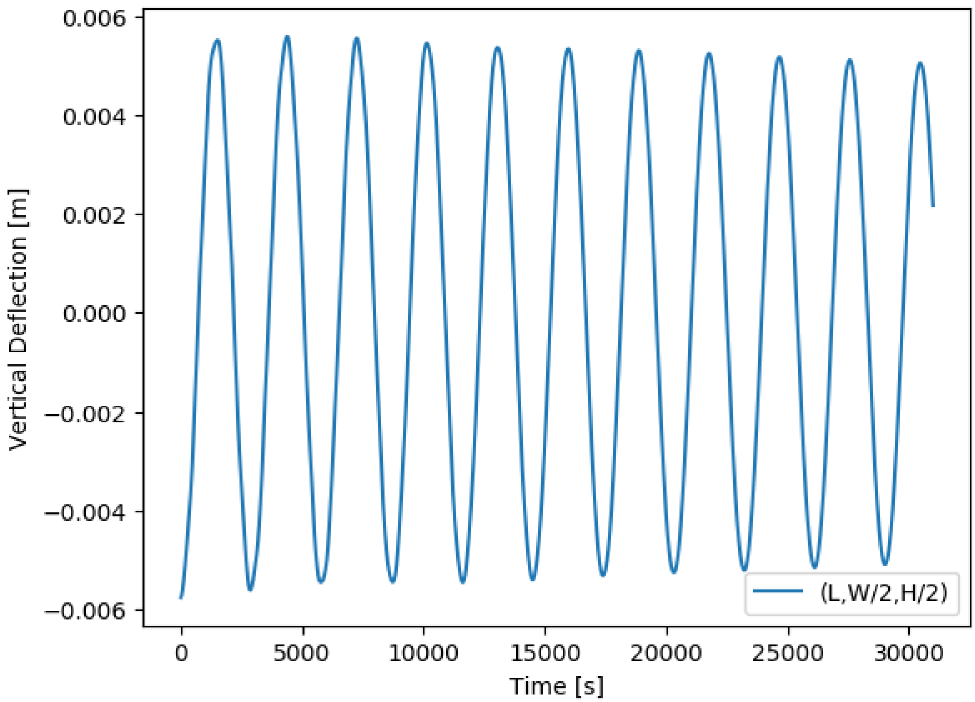

Figure 1: 1a: Displacements for problem 1a

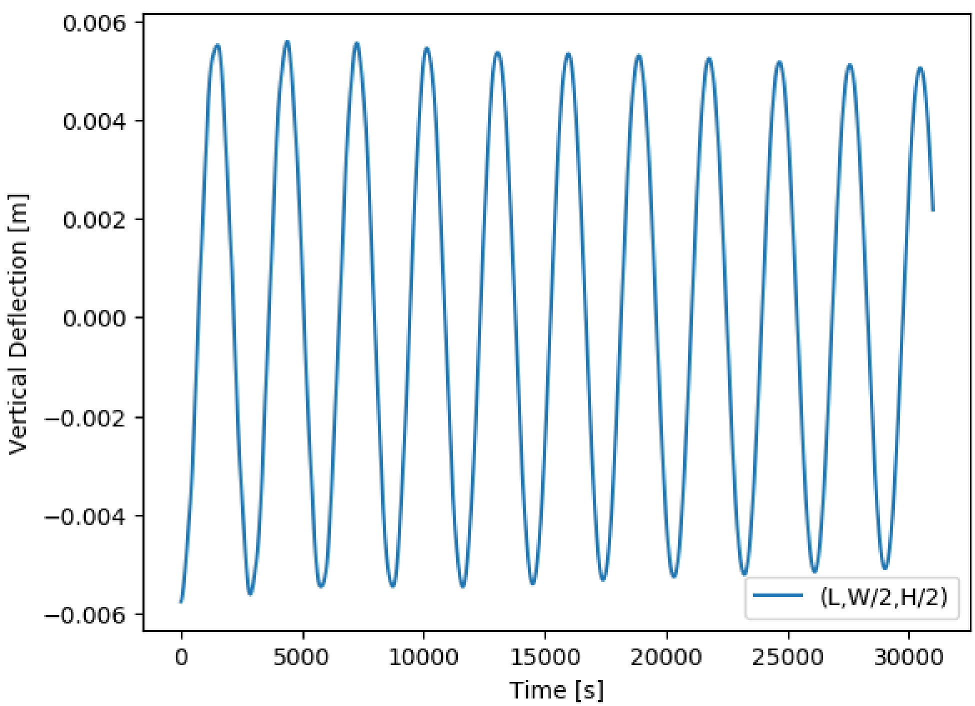

Figure 2: 1b: Displacements for problem 1b.

Part 2a: Composite Plate Clamped on All Ends

A thin plate of length $L = 1\text{m}$, width $W = 1m$, and height $H = 0.01\text{m}$ is clamped on all sides. A middle patch is made of copper, and the rest of the plate is steel. Figure 3 shows the displacements of several locations on the plate.

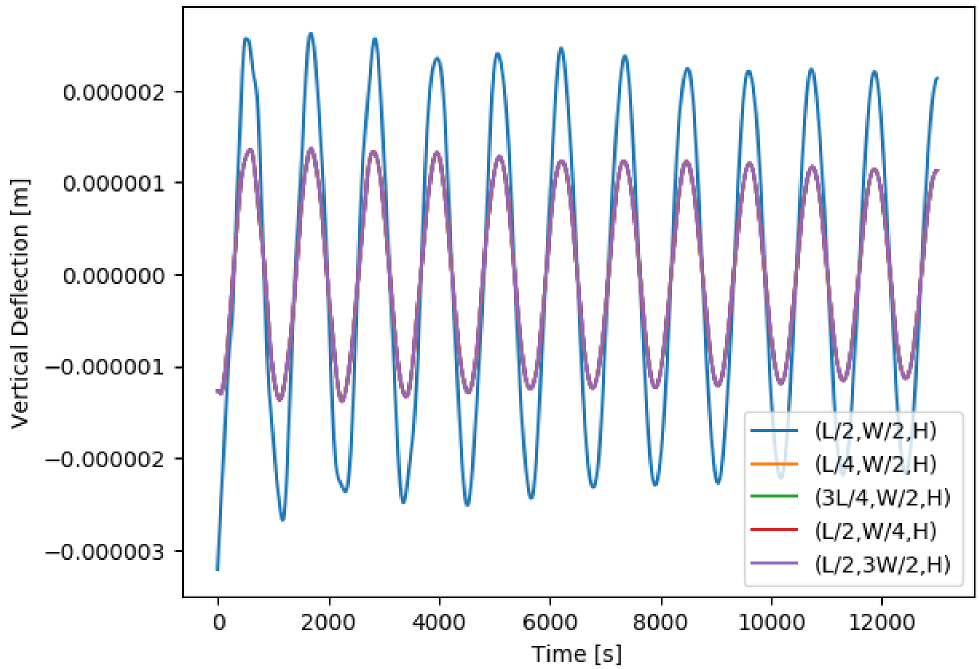

Figure 3: Displacements for problem 2a

The natural frequency is 0.0053565 rad/s.

Part 2a: Changing Copper Size

Part 2b-i

See Figure 4.

The frequencies of oscillations are 0.0053565 rad/s. The natural frequency is 0.0052711 rad/s. The overall frequency and amplitudes decreased.

Part 2b-ii

See Figure 4.

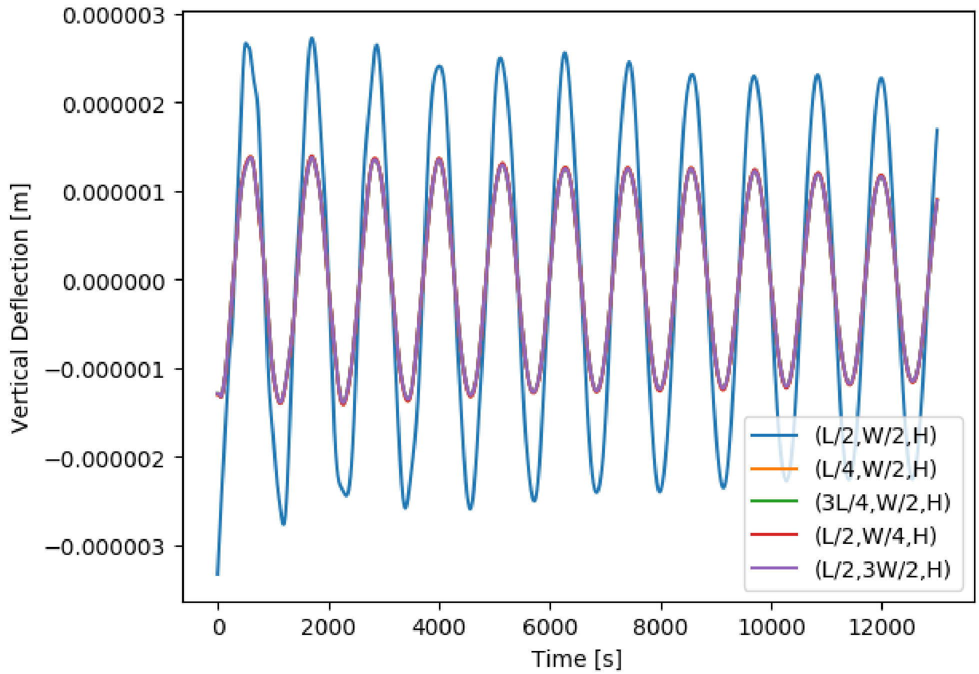

Figure 4: Displacements for problem 2b-i

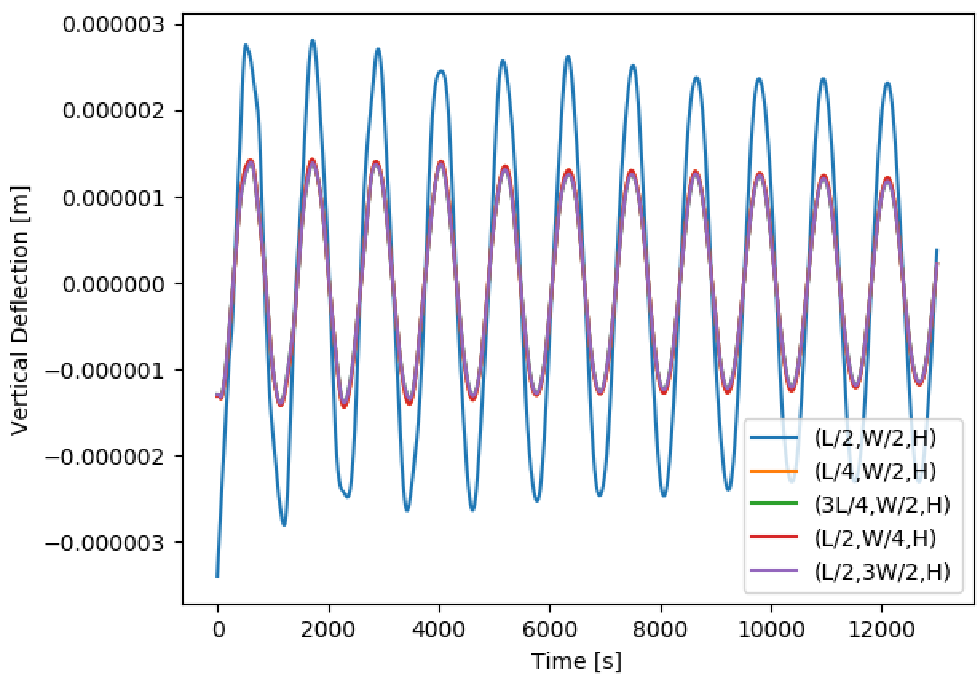

Figure 5: Displacements for problem 2b-ii

The frequencies of oscillations are 0.0053565 rad/s. The natural frequency is 0.0052013 rad/s. The overall frequency and amplitudes decreased.

Appendix 1a: Code for Problem 1a

```
"""
Python script for Part 1a of Project 5

Author:             Omkar Mulekar
Due Date:           April 24, 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
from scipy.signal import savgol_filter
import csv

#==========================================================================
#  Dimensional parameters
#==========================================================================
length = 3.0
W = 0.1
H = 0.1

E_l = 200e9
nu_l = 0.3
mu_l = (E_l)/(2*(1+nu_l))
rho_l = 7960
lambda_l = (nu_l*E_l)/((1+nu_l)*(1-2*nu_l))

E_r = 100e9
nu_r = 0.3
mu_r = (E_r)/(2*(1+nu_r))
rho_r = 8960
lambda_r = (nu_r*E_r)/((1+nu_r)*(1-2*nu_r))


traction_applied = -1e5
```

```
#==============================================================================
#   Dimensionless parameters
#==============================================================================
youngs = (mu_l*(3.0*lambda_l+2.0*mu_l))/(lambda_l+mu_l)
bar_speed = math.sqrt(youngs/rho_l)

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

t_char = W/bar_speed
t = 0
t_i = 0.5
dt = 1
num_steps = 31000

mu_l_nd = mu_l/youngs
lambda_l_nd = lambda_l/youngs
mu_r_nd = mu_r/youngs
lambda_r_nd = lambda_r/youngs

#mu_nd = Expression('x[0]<=l_nd ? mu_l_nd:mu_r_nd', l_nd=l_nd,
    mu_l_nd=mu_l_nd, mu_r_nd=mu_r_nd, degree=1)
#lambda_nd = Expression('x[0]<=l_nd ? lambda_l_nd:lambda_r_nd',
    l_nd=l_nd, lambda_l_nd=lambda_l_nd, lambda_r_nd=lambda_r_nd,
    degree=1)

traction_nd = traction_applied/youngs

#==============================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,6,6)
S = FunctionSpace(mesh,'P',1)
V = VectorFunctionSpace(mesh,'P',1)

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

mu_nd = interpolate(Expression('x[1]<(2/3)*w && x[1]>(1/3)*w ?
    mu_r_nd:mu_l_nd',w=w_nd,mu_l_nd=mu_l_nd,mu_r_nd=mu_r_nd,degree
    =1),S)
lambda_nd = interpolate(Expression('x[1]<(2/3)*w && x[1]>(1/3)*w ?
    lambda_r_nd:lambda_l_nd',w=w_nd,lambda_l_nd=lambda_l_nd,
    lambda_r_nd=lambda_r_nd,degree=1),S)
```

```
rho_nd = interpolate(Expression('x[1]<(2/3)*w && x[1]>(1/3)*w ?
    rho_r/rho_l:1.0',w=w_nd,rho_l=rho_l,rho_r=rho_r,degree=1),S)

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', 'near(x[0],l)? A : 0.0' ,'0.0'),
    degree=1, l=l_nd, w=w_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/rho_nd)*inner(sigma(u),epsilon(v))*dx \
    + dot(u,v)*dx \
    - (dt*dt/rho_nd)*dot(f,v)*dx \
    - (dt*dt/rho_nd)*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
time = [0] * num_steps
u_grab = [0] * num_steps

for i in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    solve(a == L, u, bc_left)
    u_grab[i] = u(l_nd,w_nd/2,h_nd/2)[1] * W

    # if(abs(t-index)<0.01):
    #   print("Writing output files...")
    #   xdmffile_u.write(u*length,t)
    #   stress =  sigma(u)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
    #   index += 1
    time[i] = t
    t+=dt
    u_n_1.assign(u_n)
    u_n.assign(u)

np.savetxt('results_1/p1a_u.txt', np.c_[time,u_grab])
plt.figure(1)
plt.plot(time,u_grab,label='(L,W/2,H/2)')
plt.xlabel('Time [s]')
```

```
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_1/1a_disps.png',bbox_inches='tight')
```

Appendix 1b: Code for Problem ba

```
"""
Python script for Part 1b of Project 5

Author:         Omkar Mulekar
Due Date:       April 24, 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
from scipy.signal import savgol_filter
import csv

#==========================================================================
#  Dimensional parameters
#==========================================================================
length = 3.0
W = 0.1
H = 0.1

E_l = 200e9
nu_l = 0.3
mu_l = (E_l)/(2*(1+nu_l))
rho_l = 7960
lambda_l = (nu_l*E_l)/((1+nu_l)*(1-2*nu_l))

E_r = 100e9
nu_r = 0.3
mu_r = (E_r)/(2*(1+nu_r))
rho_r = 8960
lambda_r = (nu_r*E_r)/((1+nu_r)*(1-2*nu_r))


traction_applied = -1e5
```

```
#==================================================================
#  Dimensionless parameters
#==================================================================
youngs = (mu_l*(3.0*lambda_l+2.0*mu_l))/(lambda_l+mu_l)
bar_speed = math.sqrt(youngs/rho_l)

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

t_char = W/bar_speed
t = 0
t_i = 0.5
dt = 1
num_steps = 31000

mu_l_nd = mu_l/youngs
lambda_l_nd = lambda_l/youngs
mu_r_nd = mu_r/youngs
lambda_r_nd = lambda_r/youngs

#mu_nd = Expression('x[0]<=l_nd ? mu_l_nd:mu_r_nd', l_nd=l_nd,
    mu_l_nd=mu_l_nd, mu_r_nd=mu_r_nd, degree=1)
#lambda_nd = Expression('x[0]<=l_nd ? lambda_l_nd:lambda_r_nd',
    l_nd=l_nd, lambda_l_nd=lambda_l_nd, lambda_r_nd=lambda_r_nd,
    degree=1)

traction_nd = traction_applied/youngs

#==================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,6,6)
S = FunctionSpace(mesh, 'P',1)
V = VectorFunctionSpace(mesh, 'P',1)

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

mu_nd = interpolate(Expression('x[1]<(4/5)*w && x[1]>(1/5)*w ?
    mu_r_nd:mu_l_nd',w=w_nd, mu_l_nd=mu_l_nd, mu_r_nd=mu_r_nd, degree
    =1),S)
lambda_nd = interpolate(Expression('x[1]<(4/5)*w && x[1]>(1/5)*w ?
     lambda_r_nd:lambda_l_nd',w=w_nd, lambda_l_nd=lambda_l_nd,
    lambda_r_nd=lambda_r_nd, degree=1),S)
```

```
rho_nd = interpolate(Expression('x[1]<(4/5)*w && x[1]>(1/5)*w ?
    rho_r/rho_l:1.0',w=w_nd,rho_l=rho_l,rho_r=rho_r,degree=1),S)
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', 'near(x[0],l)? A : 0.0' ,'0.0'),
    degree=1, l=l_nd, w=w_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/rho_nd)*inner(sigma(u),epsilon(v))*dx \
    + dot(u,v)*dx \
    - (dt*dt/rho_nd)*dot(f,v)*dx \
    - (dt*dt/rho_nd)*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
time = [0] * num_steps
u_grab = [0] * num_steps

for i in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    solve(a == L, u, bc_left)
    u_grab[i] = u(l_nd,w_nd/2,h_nd/2)[1] * W

    # if(abs(t-index)<0.01):
    #   print("Writing output files...")
    #   xdmffile_u.write(u*length, t)
    #   stress =  sigma(u)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj, t)
    #   index += 1
    time[i] = t
    t+=dt
    u_n_1.assign(u_n)
    u_n.assign(u)

np.savetxt('results_1/p1b_u.txt', np.c_[time,u_grab])
plt.figure(1)
plt.plot(time,u_grab,label='(L,W/2,H/2)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
```

```
plt.legend(loc='best')
plt.savefig('results_1/1b_disps.png',bbox_inches='tight')
```

Appendix 2b: Code for Problem 2b

```
"""
Python script for Part 2a of Project 5

Author:         Omkar Mulekar
Due Date:       April 24, 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
from scipy.signal import savgol_filter
import csv

#==========================================================================
#  Dimensional parameters
#==========================================================================
length = 1
W = 1
H = 0.01

E_l = 200e9
nu_l = 0.3
mu_l = (E_l)/(2*(1+nu_l))
rho_l = 7960
lambda_l = (nu_l*E_l)/((1+nu_l)*(1-2*nu_l))

E_r = 100e9
nu_r = 0.3
mu_r = (E_r)/(2*(1+nu_r))
rho_r = 8960
lambda_r = (nu_r*E_r)/((1+nu_r)*(1-2*nu_r))


traction_applied = -1e5
```

```
#==========================================================================
#  Dimensionless parameters
#==========================================================================
youngs = (mu_l*(3.0*lambda_l+2.0*mu_l))/(lambda_l+mu_l)
bar_speed = math.sqrt(youngs/rho_l)

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

t_char = H/bar_speed
t = 0
t_i = 0.5
dt = 1
num_steps = 13000

mu_l_nd = mu_l/youngs
lambda_l_nd = lambda_l/youngs
mu_r_nd = mu_r/youngs
lambda_r_nd = lambda_r/youngs

#mu_nd = Expression('x[0]<=l_nd ? mu_l_nd:mu_r_nd',l_nd=l_nd,
    mu_l_nd=mu_l_nd,mu_r_nd=mu_r_nd,degree=1)
#lambda_nd = Expression('x[0]<=l_nd ? lambda_l_nd:lambda_r_nd',
    l_nd=l_nd,lambda_l_nd=lambda_l_nd,lambda_r_nd=lambda_r_nd,
    degree=1)

traction_nd = traction_applied/youngs

#==========================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,20,3)
S = FunctionSpace(mesh,'P',1)
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)

# boundary_front = 'near(x[1],0)'
# bc_front = DirichletBC(V,Constant((0,0,0)),boundary_front)

# boundary_back = 'near(x[1],w_nd)'
```

```
# bc_back = DirichletBC(V,Constant((0,0,0)),boundary_back)

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)

mu_nd = interpolate(Expression('x[0]>0.4*l && x[0]<0.6*l && x
    [1]>0.4*w && x[1]<0.6*w ? mu_r_nd:mu_l_nd',l=l_nd,w=w_nd,
    mu_l_nd=mu_l_nd,mu_r_nd=mu_r_nd,degree=1),S)
lambda_nd = interpolate(Expression('x[0]>0.4*l && x[0]<0.6*l && x
    [1]>0.4*w && x[1]<0.6*w ? lambda_r_nd:lambda_l_nd',l=l_nd,w=
    w_nd,lambda_l_nd=lambda_l_nd,lambda_r_nd=lambda_r_nd,degree=1),
    S)
rho_nd = interpolate(Expression('x[0]>0.4*l && x[0]<0.6*l && x
    [1]>0.4*w && x[1]<0.6*w ? rho_r/rho_l:1.0',l=l_nd,w=w_nd,rho_l=
    rho_l,rho_r=rho_r,degree=1),S)

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]>0.48*l && x[0]<0.51*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/rho_nd)*inner(sigma(u),epsilon(v))*dx \
    + dot(u,v)*dx \
    - (dt*dt/rho_nd)*dot(f,v)*dx \
    - (dt*dt/rho_nd)*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
time = [0] * num_steps
u_grab1 = [0] * num_steps
u_grab2 = [0] * num_steps
u_grab3 = [0] * num_steps
u_grab4 = [0] * num_steps
u_grab5 = [0] * num_steps

for i in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    solve(a == L, u, [bc_left, bc_right, bc_front, bc_back])
    u_grab1[i] = u(l_nd/2,w_nd/2,h_nd)[2] * H
    u_grab2[i] = u(l_nd/4,w_nd/2,h_nd)[2] * H
    u_grab3[i] = u(3*l_nd/4,w_nd/2,h_nd)[2] * H
    u_grab4[i] = u(l_nd/2,w_nd/4,h_nd)[2] * H
    u_grab5[i] = u(l_nd/2,3*w_nd/4,h_nd)[2] * H

    # if(abs(t-index)<0.01):
    #   print("Writing output files...")
    #   xdmffile_u.write(u*length, t)
    #   stress =  sigma(u)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj, t)
    #   index += 1
    time[i] = t
    t+=dt
    u_n_1.assign(u_n)
    u_n.assign(u)

# np.savetxt('results_2/p2a_u.txt', np.c_[time, u_grab])
plt.figure(1)
plt.plot(time,u_grab1,label='(L/2,W/2,H)')
plt.plot(time,u_grab2,label='(L/4,W/2,H)')
plt.plot(time,u_grab3,label='(3L/4,W/2,H)')
plt.plot(time,u_grab4,label='(L/2,W/4,H)')
plt.plot(time,u_grab5,label='(L/2,3W/2,H)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_2/2a_disps.png',bbox_inches='tight')
```

Appendix 2b-i: Code for Problem 2b-i

```
"""
Python script for Part 2b.i of Project 5

Author:          Omkar Mulekar
Due Date:        April 24, 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
from scipy.signal import savgol_filter
import csv

#==========================================================================
#  Dimensional parameters
#==========================================================================
length = 1
W = 1
H = 0.01

E_l = 200e9
nu_l = 0.3
mu_l = (E_l)/(2*(1+nu_l))
rho_l = 7960
lambda_l = (nu_l*E_l)/((1+nu_l)*(1-2*nu_l))

E_r = 100e9
nu_r = 0.3
mu_r = (E_r)/(2*(1+nu_r))
rho_r = 8960
lambda_r = (nu_r*E_r)/((1+nu_r)*(1-2*nu_r))


traction_applied = -1e5
```

```
#==========================================================================
#   Dimensionless parameters
#==========================================================================
youngs = (mu_l*(3.0*lambda_l+2.0*mu_l))/(lambda_l+mu_l)
bar_speed = math.sqrt(youngs/rho_l)

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

t_char = H/bar_speed
t = 0
t_i = 0.5
dt = 1
num_steps = 13000

mu_l_nd = mu_l/youngs
lambda_l_nd = lambda_l/youngs
mu_r_nd = mu_r/youngs
lambda_r_nd = lambda_r/youngs

#mu_nd = Expression('x[0]<=l_nd ? mu_l_nd:mu_r_nd',l_nd=l_nd,
    mu_l_nd=mu_l_nd,mu_r_nd=mu_r_nd,degree=1)
#lambda_nd = Expression('x[0]<=l_nd ? lambda_l_nd:lambda_r_nd',
    l_nd=l_nd,lambda_l_nd=lambda_l_nd,lambda_r_nd=lambda_r_nd,
    degree=1)

traction_nd = traction_applied/youngs

#=========================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,20,3)
S = FunctionSpace(mesh,'P',1)
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)

# boundary_front = 'near(x[1],0)'
# bc_front = DirichletBC(V,Constant((0,0,0)),boundary_front)

# boundary_back = 'near(x[1],w_nd)'
```

```
# bc_back = DirichletBC(V, Constant((0,0,0)),boundary_back)

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)

mu_nd = interpolate(Expression('x[0]>0.35*l && x[0]<0.55*l && x
    [1]>0.35*w && x[1]<0.55*w ? mu_r_nd:mu_l_nd',l=l_nd,w=w_nd,
    mu_l_nd=mu_l_nd,mu_r_nd=mu_r_nd,degree=1),S)
lambda_nd = interpolate(Expression('x[0]>0.35*l && x[0]<0.55*l &&
    x[1]>0.35*w && x[1]<0.55*w ? lambda_r_nd:lambda_l_nd',l=l_nd,w=
    w_nd,lambda_l_nd=lambda_l_nd,lambda_r_nd=lambda_r_nd,degree=1),
    S)
rho_nd = interpolate(Expression('x[0]>0.35*l && x[0]<0.55*l && x
    [1]>0.35*w && x[1]<0.55*w ? rho_r/rho_l:1.0',l=l_nd,w=w_nd,
    rho_l=rho_l,rho_r=rho_r,degree=1),S)

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]>0.48*l && x[0]<0.51*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/rho_nd)*inner(sigma(u),epsilon(v))*dx \
    + dot(u,v)*dx \
    - (dt*dt/rho_nd)*dot(f,v)*dx \
    - (dt*dt/rho_nd)*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
time = [0] * num_steps
u_grab1 = [0] * num_steps
u_grab2 = [0] * num_steps
u_grab3 = [0] * num_steps
u_grab4 = [0] * num_steps
u_grab5 = [0] * num_steps

for i in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    solve(a == L, u, [bc_left,bc_right,bc_front,bc_back])
    u_grab1[i] = u(l_nd/2,w_nd/2,h_nd)[2] * H
    u_grab2[i] = u(l_nd/4,w_nd/2,h_nd)[2] * H
    u_grab3[i] = u(3*l_nd/4,w_nd/2,h_nd)[2] * H
    u_grab4[i] = u(l_nd/2,w_nd/4,h_nd)[2] * H
    u_grab5[i] = u(l_nd/2,3*w_nd/4,h_nd)[2] * H

    # if(abs(t-index)<0.01):
    #   print("Writing output files...")
    #   xdmffile_u.write(u*length,t)
    #   stress = sigma(u)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj,t)
    #   index += 1
    time[i] = t
    t+=dt
    u_n_1.assign(u_n)
    u_n.assign(u)

# np.savetxt('results_2/p2a_u.txt', np.c_[time,u_grab])
plt.figure(1)
plt.plot(time,u_grab1,label='(L/2,W/2,H)')
plt.plot(time,u_grab2,label='(L/4,W/2,H)')
plt.plot(time,u_grab3,label='(3L/4,W/2,H)')
plt.plot(time,u_grab4,label='(L/2,W/4,H)')
plt.plot(time,u_grab5,label='(L/2,3W/2,H)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_2/2b_i_disps.png',bbox_inches='tight')
```

Appendix 2b-ii: Code for Problem 2b-ii

```
"""
Python script for Part 2b.ii of Project 5

Author:         Omkar Mulekar
Due Date:       April 24, 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
from scipy.signal import savgol_filter
import csv

#==========================================================================
#  Dimensional parameters
#==========================================================================
length = 1
W = 1
H = 0.01

E_l = 200e9
nu_l = 0.3
mu_l = (E_l)/(2*(1+nu_l))
rho_l = 7960
lambda_l = (nu_l*E_l)/((1+nu_l)*(1-2*nu_l))

E_r = 100e9
nu_r = 0.3
mu_r = (E_r)/(2*(1+nu_r))
rho_r = 8960
lambda_r = (nu_r*E_r)/((1+nu_r)*(1-2*nu_r))


traction_applied = -1e5
```

```
#=====================================================================
#  Dimensionless parameters
#=====================================================================
youngs = (mu_l*(3.0*lambda_l+2.0*mu_l))/(lambda_l+mu_l)
bar_speed = math.sqrt(youngs/rho_l)

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

t_char = H/bar_speed
t = 0
t_i = 0.5
dt = 1
num_steps = 13000

mu_l_nd = mu_l/youngs
lambda_l_nd = lambda_l/youngs
mu_r_nd = mu_r/youngs
lambda_r_nd = lambda_r/youngs

#mu_nd = Expression('x[0]<=l_nd ? mu_l_nd:mu_r_nd', l_nd=l_nd,
    mu_l_nd=mu_l_nd, mu_r_nd=mu_r_nd, degree=1)
#lambda_nd = Expression('x[0]<=l_nd ? lambda_l_nd:lambda_r_nd',
    l_nd=l_nd, lambda_l_nd=lambda_l_nd, lambda_r_nd=lambda_r_nd,
    degree=1)

traction_nd = traction_applied/youngs

#=====================================================================
mesh = BoxMesh(Point(0,0,0),Point(l_nd,w_nd,h_nd),20,20,3)
S = FunctionSpace(mesh,'P',1)
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)

# boundary_front = 'near(x[1],0)'
# bc_front = DirichletBC(V, Constant((0,0,0)), boundary_front)

# boundary_back = 'near(x[1],w_nd)'
```

```
# bc_back = DirichletBC(V,Constant((0,0,0)),boundary_back)

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)

mu_nd = interpolate(Expression('x[0]>0.3*l && x[0]<0.6*l && x
    [1]>0.3*w && x[1]<0.6*w ? mu_r_nd:mu_l_nd',l=l_nd,w=w_nd,
    mu_l_nd=mu_l_nd,mu_r_nd=mu_r_nd,degree=1),S)
lambda_nd = interpolate(Expression('x[0]>0.3*l && x[0]<0.6*l && x
    [1]>0.3*w && x[1]<0.6*w ? lambda_r_nd:lambda_l_nd',l=l_nd,w=
    w_nd,lambda_l_nd=lambda_l_nd,lambda_r_nd=lambda_r_nd,degree=1),
    S)
rho_nd = interpolate(Expression('x[0]>0.3*l && x[0]<0.6*l && x
    [1]>0.3*w && x[1]<0.6*w ? rho_r/rho_l:1.0',l=l_nd,w=w_nd,rho_l=
    rho_l,rho_r=rho_r,degree=1),S)

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]>0.48*l && x[0]<0.51*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/rho_nd)*inner(sigma(u),epsilon(v))*dx \
    + dot(u,v)*dx \
    - (dt*dt/rho_nd)*dot(f,v)*dx \
    - (dt*dt/rho_nd)*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
time = [0] * num_steps
u_grab1 = [0] * num_steps
u_grab2 = [0] * num_steps
u_grab3 = [0] * num_steps
u_grab4 = [0] * num_steps
u_grab5 = [0] * num_steps

for i in range(num_steps):
    print("time = %.2f" % t)
    T_n.t = t
    solve(a == L, u, [bc_left, bc_right, bc_front, bc_back])
    u_grab1[i] = u(l_nd/2,w_nd/2,h_nd)[2] * H
    u_grab2[i] = u(l_nd/4,w_nd/2,h_nd)[2] * H
    u_grab3[i] = u(3*l_nd/4,w_nd/2,h_nd)[2] * H
    u_grab4[i] = u(l_nd/2,w_nd/4,h_nd)[2] * H
    u_grab5[i] = u(l_nd/2,3*w_nd/4,h_nd)[2] * H

    # if(abs(t-index)<0.01):
    #   print("Writing output files...")
    #   xdmffile_u.write(u*length, t)
    #   stress =  sigma(u)
    #   stress_proj.vector()[:] = project(stress,Q).vector()
    #   xdmffile_s.write(stress_proj, t)
    #   index += 1
    time[i] = t
    t+=dt
    u_n_1.assign(u_n)
    u_n.assign(u)

# np.savetxt('results_2/p2a_u.txt', np.c_[time, u_grab])
plt.figure(1)
plt.plot(time, u_grab1, label='(L/2,W/2,H)')
plt.plot(time, u_grab2, label='(L/4,W/2,H)')
plt.plot(time, u_grab3, label='(3L/4,W/2,H)')
plt.plot(time, u_grab4, label='(L/2,W/4,H)')
plt.plot(time, u_grab5, label='(L/2,3W/2,H)')
plt.xlabel('Time [s]')
plt.ylabel('Vertical Deflection [m]')
plt.legend(loc='best')
plt.savefig('results_2/2b_ii_disps.png',bbox_inches='tight')
```