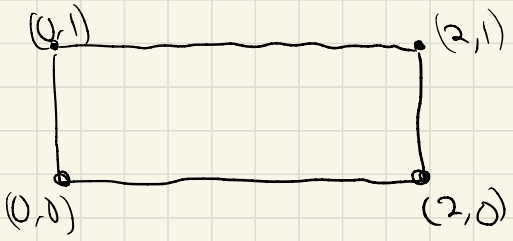
Juan Meriles
ME 280 B
Assignment 3

Problem 1

4 node quad plane strain



a) Write shape functions

$$N_1 = \frac{1}{2}(2-x_1)(1-x_2) = \frac{1}{2}(2-x_1+x_1x_2-2x_2)$$

$$N_2 = \frac{1}{2}(x_1)(1-x_2) = \frac{1}{2}(x_1-x_1x_2)$$

$$N_3 = \frac{1}{2}x_1x_2$$

$$N_4 = \frac{1}{2}(2-x_1)(x_2) = \frac{1}{2}(2x_2-x_1x_2)$$

b) Determine Components of F

$$F = \frac{\partial x}{\partial X} \qquad u = \begin{bmatrix} N_1 & & & N_4 \\ & N_1 & \cdots & & N_4 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ \vdots \\ u_{41} \\ u_{42} \end{bmatrix}$$

$$U = x - X$$

$$\rightarrow F = \frac{\partial U}{\partial X} - I$$

given $u_{11} = u_{12} = 0 \quad u_{21} = \bar{v} \quad u_{22} = 0$

$u_{31} = \bar{u} \quad u_{32} = -\bar{v} \quad u_{41} = 0 \quad u_{42} = -\bar{v}$

$$u = \begin{bmatrix} 0 \\ 0 \\ \bar{u} \\ 0 \\ \bar{v} \\ -\bar{v} \\ 0 \\ -\bar{v} \end{bmatrix}$$

$u_1 = N_2 \bar{v} + N_3 \bar{u}$

$u_2 = N_3 \bar{v} - N_4 \bar{v}$

$$\frac{du_1}{dx_1} = \frac{1}{2}(1 - x_2)\bar{u} + \frac{1}{2}x_2 \bar{u} = \frac{1}{2}\bar{u}$$

$$\frac{du_1}{dx_2} = -\frac{1}{2}x_1 \bar{u} + \frac{1}{2}x_1 \bar{u} = 0$$

$$\frac{dv_2}{dx_1} = -\frac{1}{2}x_2 \bar{v} - \left(-\frac{1}{2}x_2\right)\bar{v} = 0$$

$$\frac{dv_2}{dx_2} = -\frac{1}{2}x_1 \bar{v} - \left(\frac{1}{2}(2 - x_1)\bar{v}\right)$$

$$= -\frac{1}{2}x_1 \bar{v} - \left(\bar{v} - \frac{1}{2}x_1 \bar{v}\right) = -\bar{v}$$

$$F = \frac{du}{dx} + \Sigma = \begin{bmatrix} \frac{1}{2}\bar{u} + 1 & 0 \\ 0 & 1 - \bar{v} \end{bmatrix}$$

c) Calculate Jacobian of F

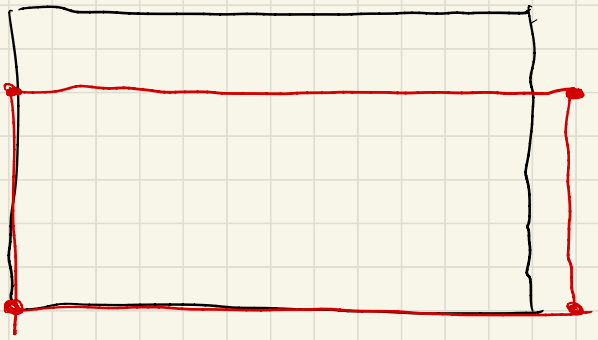
Jacobian = $\det F = \frac{1}{2}\bar{u} - \frac{1}{2}\bar{u}\bar{v} + \frac{1}{2}\bar{u} - \bar{v}$

must be ≥ 0 this occurs when $\bar{u} > -2$, $\bar{v} < 1$

or

$\bar{u} < -2$, $\bar{v} > 1$

d) Sketch current config



e) Suppose edge 2→3 has traction $\bar{p}$ per unit length of ref config given as $\bar{p} = p e_1$, find eq nodal forces

$$\int_0^1 [N^e(2, x_2)] \begin{bmatrix} p \\ 0 \end{bmatrix}$$

$N_1(2, x_2) = 0 \quad N_2(2, x_2) = (1 - x_2) \quad N_3(2, x_2) = x_2 \quad N_4(2, x_2) = 0$

$$= \int_0^1 \begin{bmatrix} 0 \\ 0 \\ (1-x_2)p \\ 0 \\ x_2 p \\ 0 \\ 0 \\ 0 \end{bmatrix} dx_2 = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2}p \\ 0 \\ \frac{1}{2}p \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

f) $\bar{F}$ along unit length of current config

$\bar{F} = \rho e_1$ ρ is a constant

along current config

→ the edge on current config has a length $1-v$
when current config has length 1

thus half the total load still goes to each node

$$\begin{bmatrix} 0 \\ 0 \\ \frac{1}{2}(1-v)\rho \\ 0 \\ \frac{1}{2}(1-v)\rho \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Problem 2

a) Verify that linear momentum balance is maintained in ref config

$$\frac{d}{dt}\int_P \rho v \, dv = \int_P \rho b \, dv + \int_{\partial P} t \, da$$

in weighted residual weak form this becomes

$$\int_R \xi \cdot \rho a \, dv + \int_R \frac{\partial \xi}{\partial x} \cdot T \, dv = \int_R \xi \cdot \rho b \, dv + \int_{\Gamma_q} \xi \cdot t \, da$$

in ref config

$$\int_{R_0} \xi \cdot \rho_0 a \, dV + \int_{R_0} \frac{\partial \xi}{\partial x} \cdot T \, dV = \int_{R_0} \xi \cdot \rho_0 b \, dV + \int_{\Gamma_q} \xi \cdot t \, dA$$

$a = 0 \quad b = 0$

$$\int_{R_0} \frac{\partial \xi}{\partial x} \cdot T \, dV = \int_{\Gamma_q} \xi \cdot t \, dA \qquad t = Tn$$

$$= \int_{\Gamma_q \cup \Gamma_u} \xi \cdot Tn \, dA$$

ξ is 0 on dericlet boundary

$$= \int_{\Gamma_q} \xi \cdot t \, dA$$

lin momentum balance satisfied

b) Calculate components for mass matrix M

$$N_1 = (1 - x_1)(1 - x_2)$$
$$N_2 = (x_1)(1 - x_2)$$
$$N_3 = x_1 x_2$$
$$N_4 = (1 - x_1) x_2$$

$$[M^e] = \int_{\Omega_0^e} [N^e]^T \rho_0 [N^e] \, dV$$

$$[N^e] = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 \end{bmatrix}$$

$$[M^e] = \rho_0 \int_{-\frac{a}{2}}^{\frac{a}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} [N^e]^T [N^e] \, dX_2 \, dX_1$$ done in pyramo

$$\{F\} \int_0^1 [N(1, x_2)]^T \begin{Bmatrix} 10^4 \\ 0 \\ 0 \end{Bmatrix}$$

$$= \int_0^1 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 10^4 x_2 + (1 - x_2) \\ 0 \\ 0 \\ 10^4 x_2 + (x_2) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} dx_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{10^4 +}{6} \\ 0 \\ 0 \\ \frac{10^4 +}{2} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$10^4 + \int_0^1 (x_2 - x_2^2)$$

$$10^4 + \left(\frac{x_2^2}{2} - \frac{x_2^3}{3} \right) \Big|_0^1 = \frac{10^4 +}{6}$$

$$10^4 + \int_0^1 \left(\frac{x_2^2}{2} \right) = \frac{10^4 +}{2}$$

$$\int_{V_e} [N^e]^T \beta_0 [b_{n+1}] \, dV$$

$$= \int_0^1 \int_0^1 [N^e I]^T \beta_0 \begin{bmatrix} 0 \\ -10^{-3} \times x_2 + \\ 0 \end{bmatrix} dX_1 dX_2$$

done in Python

$$[R^e_{n+1}] = \int_{\Omega^e_0} [B^e]^T \langle P_{n+1} \rangle dV$$

$$\underset{9\times(3\times nen)}{[B^e]} = \left[[B^e_1], [B^e_2] \cdots [B^e_{nen}] \right]$$

$$[B^e_i] = \begin{bmatrix} N^e_{i,1} & & N^e_{i,2} & & N^e_{i,3} \\ & N^e_{i,2} & & N^e_{i,3} & & N^e_{i,1} \\ N^e_{i,3} & & N^e_{i,1} & & N^e_{i,2} \end{bmatrix}$$

$$\langle P_{n+1} \rangle = \begin{bmatrix} P_{11} \\ P_{22} \\ P_{33} \\ P_{12} \\ P_{23} \\ P_{31} \\ P_{13} \\ P_{21} \\ P_{32} \end{bmatrix}$$

def gradient $F = \dfrac{\partial x}{\partial X}$

$$= \begin{bmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} & \frac{\partial x_1}{\partial X_3} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} & \frac{\partial x_2}{\partial X_3} \\ \frac{\partial x_3}{\partial X_1} & \frac{\partial x_3}{\partial X_2} & \frac{\partial x_3}{\partial X_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial x(X_1,X_2,X_3)}{\partial X_1} & \frac{\partial x(X_1,X_2,X_3)}{\partial X_2} & 0 \\ \frac{\partial x_2(X_1,X_2,X_3)}{\partial X_1} & \frac{\partial x_2(X_1,X_2,X_3)}{\partial X_2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$U = [N]\langle U^e \rangle$$

$3\times 12 \quad 12\times 1 \to 3\times 1$

$$C = F^T F \qquad x = X + U$$

$$E = \frac{1}{2}(C - I)$$

$$S = \lambda (\mathrm{tr}\, E) I + 2\mu E$$

$$P = FS$$

$$F = \begin{bmatrix} \frac{\partial U_1}{\partial X_1} & \frac{\partial U_1}{\partial X_2} & 0 \\ \frac{\partial U_2}{\partial X_1} & \frac{\partial U_2}{\partial X_2} & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}$$

$$u^e = \begin{bmatrix} 0 \\ 0 \\ 0 \\ u_{21} \\ u_{22} \\ 0 \\ u_{31} \\ u_{32} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$U_1 = N_2 u_{21} + N_3 u_{31} = (X_1)(1-X_2)\, u_{21} + X_1 X_2 u_{31}$$

$$U_2 = N_2 u_{22} + N_3 u_{32} = (X_1)(1-X_2)\, u_{22} + X_1 X_2 u_{32}$$

$$U_3 = N_3 u_{23} + N_3 u_{33} \quad (X_1)(1-X_2)\, u_{23} + X_1 X_2 u_{33}$$

$$\frac{\partial U_1}{\partial X_1} = (1-X_2)\, u_{21} + X_2 u_{31}, \quad \frac{\partial U_1}{\partial X_2} = -X_1 u_{21} + X_1 u_{31}$$

$$\frac{\partial U_2}{\partial X_1} = (1-X_2) u_{22} + X_2 u_{32}, \quad \frac{\partial U_2}{\partial X_2} = (-X_1)\, u_{22} + X_1 u_{32}$$

Done in python

e) Newmark method

Implicit Nemark done by first solving for F_{eff}

$$F_{eff} = F_{n+1} + M\left\{(\hat{u}_n + \hat{v}_n \Delta t_n)\frac{1}{\beta \Delta t_n^2} + \left(1 - \frac{2\beta}{2\beta}\right)\hat{a}_n\right\}$$

we then use newton Raphson algorithm to solve nonlinear set of eqns

$$\frac{1}{\beta \Delta t_n^2} M \hat{u}_{n+1} + R(\hat{u}_{n+1}) = F_{eff}$$

if $G(\hat{u}_{n+1}) = \frac{1}{\beta \Delta t_n^2} M \hat{u}_{n+1} + R(\hat{u}_{n+1}) - F_{eff} = 0$

this can be solved by updating $u_{n+1}^{j+1} = u_{n+1}^{j} - (DG)^{-1} G(u_{n+1}^{j})$

this can be repeated until residual is under a threshold

DG is estimated by using perturbations of u

$$DG \approx \frac{D(u_{n+1}^{j} + w) - D(u_{n+1}^{j})}{w}$$

implementation on next page

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt

lam = 4 * 10**10
miu = 2.7 * 10**10
rho = 2.7 * 10**3

Xref = np.array([[0,0,0,1,0,0,1,1,0,0,1,0]])

X1 = sym.Symbol('X1')
X2 = sym.Symbol('X2')
X3 = sym.Symbol('X3')
u11 = sym.Symbol('u11')
u21 = sym.Symbol('u21')
u31 = sym.Symbol('u31')
u41 = sym.Symbol('u41')
u12 = sym.Symbol('u12')
u22 = sym.Symbol('u22')
u32 = sym.Symbol('u32')
u42 = sym.Symbol('u42')
u13 = sym.Symbol('u13')
u23 = sym.Symbol('u23')
u33 = sym.Symbol('u33')
u43 = sym.Symbol('u43')
t = sym.Symbol('t')

N1 = (1-X1)*(1-X2)
N2 = X1*(1-X2)
N3 = X1*X2
N4 = (1-X1)*X2
N = [N1,N2,N3,N4]

Ne = np.array([[N1,0,0,N2,0,0,N3,0,0,N4,0,0],
               [0,N1,0,0,N2,0,0,N3,0,0,N4,0],
               [0,0,N1,0,0,N2,0,0,N3,0,0,N4]])

Be = []
for i in range(len(N)):
    Be.append([[sym.diff(N[i],X1),0,0],
               [0,sym.diff(N[i],X2),0],
               [0,0,sym.diff(N[i],X3)],
               [sym.diff(N[i],X2),0,0],
               [0,sym.diff(N[i],X3),0],
               [0,0,sym.diff(N[i],X1)],
               [sym.diff(N[i],X3),0,0],
               [0,sym.diff(N[i],X1),0],
               [0,0,sym.diff(N[i],X2)]])

Be= np.array(Be)
Be = np.hstack(Be)

ue = np.array([[0,0,0,u21,u22,0,u31,u32,0,0,0,0]])
U = Ne @ ue.T
```

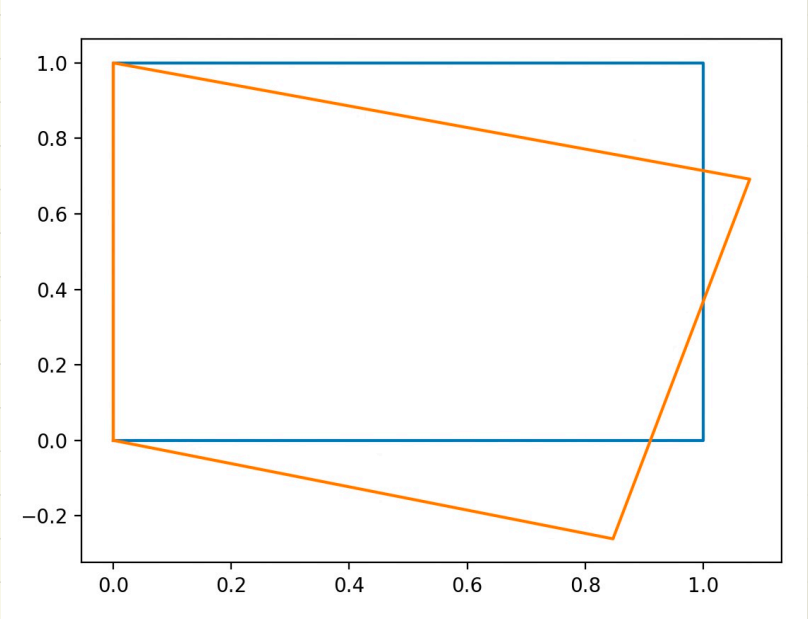
```
F = np.array([[sym.diff(U[0],X1)[0],sym.diff(U[0],X2)[0],0],
              [sym.diff(U[1],X1)[0],sym.diff(U[1],X2)[0],0],
              [0,0,0]])
F = F + np.eye(3)
C = F.T @ F
E = 1/2*(C-np.eye(3))
S = lam*(np.trace(E))*np.eye(3)+2*miu*E
P = F@S
P = np.array([[P[0][0],P[1][1],P[2][2],P[0][1],P[1][2],P[2][0],P[0][2],P[1][0],P[2][1]]])
P = P.T
M = rho * Ne.T @ Ne


R = Be.T @ P

#2nd order Gauss quadrature
e = [-.57735,.57735]
n = [-.57735,.57735]

R_int = []
M_int = np.zeros((np.shape(M)[0],np.shape(M)[1]))

F_t = np.array([[0,0,0,(10**8)*t/6,0,0,(10**8)*t/2,0,0,0,0,0]])
F_b = rho*Ne.T@np.array([[0],[(-10**7)*X1*X2*t],[0]])
F = F_t.T+F_b
F_int = []

for i in range(len(R)):
    temp1 = 0
    for j in range(len(n)):
        for k in range(len(e)):
            if R[i]!=0:
                temp1 = temp1+R[i][0].subs(X1,(e[j]/2+1/2)).subs(X2,(n[k]/2+1/2))
    R_int.append(1/4*temp1)

for l in range(np.shape(M)[0]):
    for p in range(np.shape(M)[1]):
        temp2 = 0
        for j in range(len(n)):
            for k in range(len(e)):
                if M[l][p]!=0:
                    temp2 = temp2+M[l][p].subs(X1,(e[j]/2+1/2)).subs(X2,(n[k]/2+1/2))

        M_int[l][p] = 1/4*temp2

for i in range(len(F)):
    temp3 = 0
    for j in range(len(n)):
        for k in range(len(e)):
            if F[i]!=0:
                temp3 = temp3+F[i][0].subs(X1,(e[j]/2+1/2)).subs(X2,(n[k]/2+1/2))

    F_int.append(1/4*temp3)
```

```
#To approximate derivative

def approxD(current_u,func):
    eps = np.finfo(float).eps
    w = (10**3)*eps

    DA_o = func(current_u[0],current_u[1],\
                current_u[2],current_u[3])

    DA = []
    for i in range(len(ue[0])):
        pert = np.zeros(len(ue[0]))
        pert[i] = w
        u_cur_tot = np.array([0,0,0,current_u[0],current_u[1],0,current_u[2],current_u[3],0,0,0,0])
        u_pert = u_cur_tot+pert

        DA_col = func(u_pert[3],u_pert[4],\
                      u_pert[6],u_pert[7])
        DA_col = (np.array([DA_col]) - np.array([DA_o]))/w
        DA.append(DA_col[0])

    DA = np.hstack(DA)
    freeDofs = [0,0,0,1,1,0,1,1,0,0,0,0]
    for i in range(len(freeDofs)-1,-1,-1):
        if freeDofs[i] == 0:
            DA = np.delete(DA,i,0)
            DA = np.delete(DA,i,1)
    return DA

#Do newmark method
beta = .25
gamma = .5

u_hist = []
v_hist = []
a_hist = []
F_hist = []
R_hist = []

time = 0
dt = 1
tf = 1
steps = np.arange(dt,tf+.000000001,dt)
u_hist.append(np.zeros(12))
v_hist.append(np.zeros(12))
a_hist.append(np.zeros(12))
F_hist.append(np.zeros(12))
R_hist.append(np.zeros(12))

F_temp = []
k=0
func2 = sym.lambdify([t],F_int)
func3 = sym.lambdify([t,u11,u12,u13,u21,u22,u23,u31,u32,u33,u41,u42,u43],R_int)
```

```
for n in steps:
    print(k)
    F_temp = func2(n)
    # for i in range(len(F_int)):
    #     if F_int[i]!=0:
    #         F_temp.append(F_int[i].subs(t,n))
    #     else:
    #         F_temp.append(0)
    F_hist.append(F_temp)

    F_eff = F_hist[k+1]+M_int @ ((np.array(u_hist[k])+np.array(v_hist[k])*dt)*1/(beta*dt**2)-

    def Substitute1D(vect,time,u_hist):
        vect_eval = []
        for i in range(len(vect)):
            if vect[i] !=0:
                temp = vect[i].subs(u11,u_hist[time][0])
                temp = temp.subs(u12,u_hist[time][1])
                temp = temp.subs(u13,u_hist[time][2])
                temp = temp.subs(u21,u_hist[time][3])
                temp = temp.subs(u22,u_hist[time][4])
                temp = temp.subs(u23,u_hist[time][5])
                temp = temp.subs(u31,u_hist[time][6])
                temp = temp.subs(u32,u_hist[time][7])
                temp = temp.subs(u33,u_hist[time][8])
                temp = temp.subs(u41,u_hist[time][9])
                temp = temp.subs(u42,u_hist[time][10])
                temp = temp.subs(u43,u_hist[time][11])
                temp = temp.subs(t,time)
                vect_eval.append(temp)
            else:
                vect_eval.append(0)
        return vect_eval


    R_hist.append(Substitute1D(R_int,0,u_hist))


    A = (1/(beta*dt**2))*M_int @ ue.T + np.array([R_int]).T-np.array([F_eff]).T
    func = sym.lambdify([u21,u22,u31,u32],A)

    def func1(u,func):
        vec = func(u[0],u[1],u[2],u[3])
        return np.array([vec[4],vec[5],vec[7],vec[8]]).T[0]


    #Implement Newton Raphson formulation
    DA_o = func(u_hist[0][3],u_hist[0][4],\
                u_hist[0][6],u_hist[0][7])
    #u_it_n = np.array([u_hist[0][3],u_hist[0][4],u_hist[0][6],u_hist[0][7]])
    u_it_n = np.array([0,0,0,0])
    f_it_n = np.array([DA_o[3][0],DA_o[4][0],DA_o[6][0],DA_o[7][0]])
```

```
        error = 10
        i = 0
        while error>10**-5:
            #print(i)
            current_u = u_it_n
            DA = approxD(current_u,func)
            u_it_n1 = u_it_n - np.linalg.solve(DA,f_it_n)
            #print(u_it_n1)
            new_f = func(u_it_n1[0],u_it_n1[1],\
                    u_it_n1[2],u_it_n1[3])
            f_it_n = np.array([new_f[3][0],new_f[4][0],new_f[6][0],new_f[7][0]])
            error= np.linalg.norm(f_it_n,2)
            u_it_n = u_it_n1
            i = i+1

        u_hist.append(np.array([u_hist[0][0],u_hist[0][0],u_hist[0][0],u_it_n1[0],u_it_n1[1],u_hist[0][0],\
                                u_it_n1[2],u_it_n1[3],u_hist[0][0],u_hist[0][0],u_hist[0][0],u_hist[0][0]]))
        anew = (1/(beta*(dt**2)))*(u_hist[k+1]-u_hist[k]-v_hist[k]*dt)-((1-2*beta)/(2*beta))*a_hist[k]
        a_hist.append(anew)
        vnew = v_hist[k]+((1-gamma)*a_hist[k]+gamma*a_hist[k+1])*dt
        v_hist.append(vnew)
        k = k+1


    pos_x_ref = [Xref[0][0],Xref[0][3],Xref[0][6],Xref[0][9],Xref[0][0]]
    pos_y_ref = [Xref[0][1],Xref[0][4],Xref[0][7],Xref[0][10],Xref[0][1]]

    pos_x_def = [Xref[0][0],Xref[0][3]+u_hist[-1][3],Xref[0][6]+u_hist[-1][6],Xref[0][9],Xref[0][0]]
    pos_y_def = [Xref[0][1],Xref[0][4]+u_hist[-1][4],Xref[0][7]+u_hist[-1][7],Xref[0][10],Xref[0][1]]

    plt.plot(pos_x_ref,pos_y_ref)
    plt.plot(pos_x_def,pos_y_def)
```

f) Deformed shape



g)

dt=1

```
In [743]: u_hist[-1]
Out[743]:
array([ 0.        ,  0.        ,  0.        , -0.15296539, -0.26082419,
        0.        ,  0.07889803, -0.30796178,  0.        ,  0.        ,
        0.        ,  0.        ])

In [744]: v_hist[-1]
Out[744]:
array([ 0.        ,  0.        ,  0.        , -0.30593077, -0.52164837,
        0.        ,  0.15779606, -0.61592356,  0.        ,  0.        ,
        0.        ,  0.        ])
```

dt=0.1

```
In [779]: u_hist[-1]
Out[779]:
array([ 0.        ,  0.        ,  0.        , -0.15297052, -0.26083038,
        0.        ,  0.07889896, -0.30796901,  0.        ,  0.        ,
        0.        ,  0.        ])

In [780]: v_hist[-1]
Out[780]:
array([ 0.        ,  0.        ,  0.        , -0.10125236,  0.00954857,
        0.        , -0.08088117, -0.03455514,  0.        ,  0.        ,
        0.        ,  0.        ])
```

dt=.01

```
In [784]: u_hist[-1]
Out[784]:
array([ 0.        ,  0.        ,  0.        , -0.15293557, -0.26079099,
        0.        ,  0.07889436, -0.30792498,  0.        ,  0.        ,
        0.        ,  0.        ])

In [785]: v_hist[-1]
Out[785]:
array([ 0.        ,  0.        ,  0.        , -0.39783897, -0.47582028,
        0.        ,  0.05265401, -0.56967437,  0.        ,  0.        ,
        0.        ,  0.        ])
```

dt=0.001

```
In [781]: u_hist1000[-1]
Out[781]:
array([ 0.        ,  0.        ,  0.        , -0.15290716, -0.26076053,
        0.        ,  0.07889162, -0.30788877,  0.        ,  0.        ,
        0.        ,  0.        ])

In [782]: v_hist1000[-1]
Out[782]:
array([ 0.        ,  0.        ,  0.        , -0.17406737, -0.23878483,
        0.        ,  0.03480034, -0.29950978,  0.        ,  0.        ,
        0.        ,  0.        ])
```

As can be seen, the displacement is the same for all time steps, but the velocity changes. This is likely because

Instantaneous velocity is averaged out when the time increment is too large.

Problem 3

a) the diagonal approx to the mass matrix assumes mass is distributed among the nodes based on area as this is a symmetric element

$$M = \frac{1}{4} \rho_0 I_{12\times 12}$$

b) The explicit newmark algorithm first calculates $\hat{u}_{n+1}$ using $\beta = 0$ and $\gamma = 0.5$

$$\hat{u}_{n+1} = \hat{u}_n + \hat{v}_n \Delta t_n + \frac{1}{2} \hat{a}_n \Delta t_n^2$$

we then solve for $\hat{a}_{n+1}$

$$M \hat{a}_{n+1} = F_{n+1} - R\left(\hat{u}_n + \hat{v}_n \Delta t_n + \frac{1}{2} \hat{a}_n \Delta t_n^2\right)$$

finally $\hat{v}_{n+1}$ is solved for

$$\hat{v}_{n+1} = \hat{v}_n + \left[(1-\gamma)\hat{a}_n + \gamma \hat{a}_{n+1}\right] \Delta t_n$$

Implementation on next page

```
#Explicit Newmark
M_int_lump = rho*1/4*np.eye(12)
beta = 0
gamma = .5
dt = 20*10**-6
tf = 1

u_hist_exp = []
v_hist_exp = []
a_hist_exp = []
F_hist_exp = []
R_hist_exp = []

steps = np.arange(dt,tf+.000000001,dt)
u_hist_exp.append(np.zeros(12))
v_hist_exp.append(np.zeros(12))
a_hist_exp.append(np.zeros(12))
F_hist_exp.append(np.zeros(12))
R_hist_exp.append(np.zeros(12))

u_hist_exp.append(np.zeros(12))
v_hist_exp.append(np.zeros(12))
a_hist_exp.append(np.zeros(12))
F_hist_exp.append(np.zeros(12))
R_hist_exp.append(np.zeros(12))

k = 0
func2 = sym.lambdify([t],F_int)

count =0
for n in steps:
    #print(k)
    F_temp = func2(n)

    # for i in range(len(F_int)):
    #     if F_int[i]!=0:
    #         F_temp.append(func2(n))
    #     else:
    #         F_temp.append(0)
    F_hist_exp[k+1]=(np.array(F_temp))
    u_hist_exp[k+1]=(u_hist_exp[k]+v_hist_exp[k]*dt+(1/2)*a_hist_exp[k]*dt**2)
    #print(F_hist_exp)
    Feff = (np.array(F_hist_exp[k+1])-np.array(func3(n,0,0,0,u_hist_exp[k+1][3],u_hist_exp[k+1][4],0,u_hist_exp[k+1][6]
    anew = np.linalg.inv(M_int_lump)@Feff
    anew = np.array([0,0,0,anew[3],anew[4],0,anew[6],anew[7],0,0,0,0])
    a_hist_exp[k+1]=(anew)
    vnew = v_hist_exp[k]+((1-gamma)*a_hist_exp[k]+gamma*a_hist_exp[k+1])*dt
    v_hist_exp[k+1]=(vnew)
    u_hist_exp[k] = u_hist_exp[k+1]
    v_hist_exp[k] = v_hist_exp[k+1]
    a_hist_exp[k] = a_hist_exp[k+1]
    F_hist_exp[k] = F_hist_exp[k+1]
    print(count)
    count = count+1
```

6) 10 ms

```
In [57]: u_hist_exp[-1]
Out[57]:
array([ 0.        ,  0.        ,  0.        , -0.15293871, -0.26079583,
        0.        ,  0.07889321, -0.30792901,  0.        ,  0.        ,
        0.        ,  0.        ])
```

```
In [56]: v_hist_exp[-1]
Out[56]:
array([ 0.        ,  0.        ,  0.        ,  0.02274391, -0.06694081,
        0.        , -0.01861802, -0.08991612,  0.        ,  0.        ,
        0.        ,  0.        ])
```

$\Delta t = 20$ ms

```
In [59]: u_hist_exp[-1]
Out[59]:
array([ 0.        ,  0.        ,  0.        , -0.15291581, -0.26077172,
        0.        ,  0.07889181, -0.30790158,  0.        ,  0.        ,
        0.        ,  0.        ])

In [60]: v_hist_exp[-1]
Out[60]:
array([ 0.        ,  0.        ,  0.        , -0.04689106, -0.09511108,
        0.        , -0.02110073, -0.13220633,  0.        ,  0.        ,
        0.        ,  0.        ])
```

c) The plot is the same as the implicit version