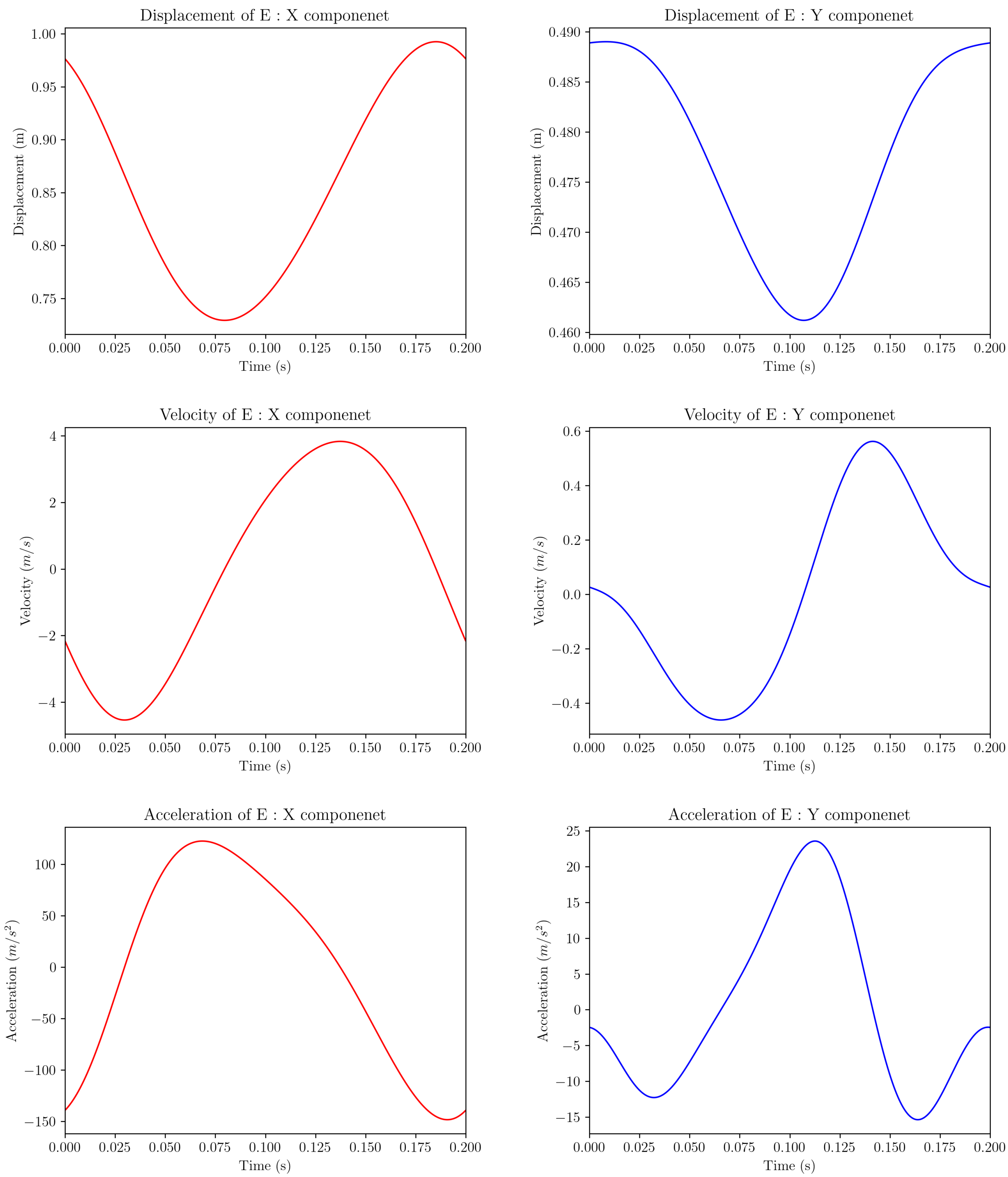
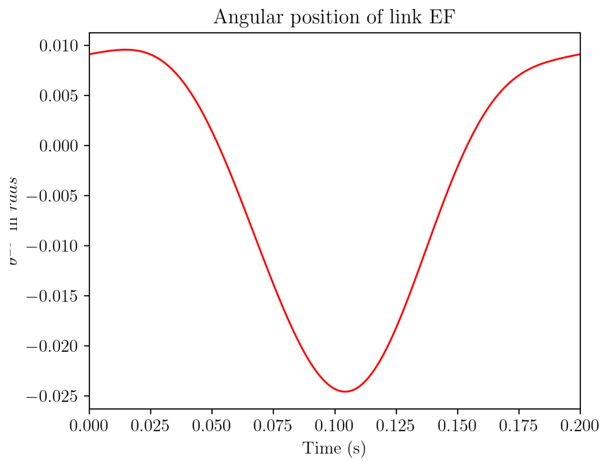
Multibody Dynamics and its Applications : Endsemester Jul-Nov 23'

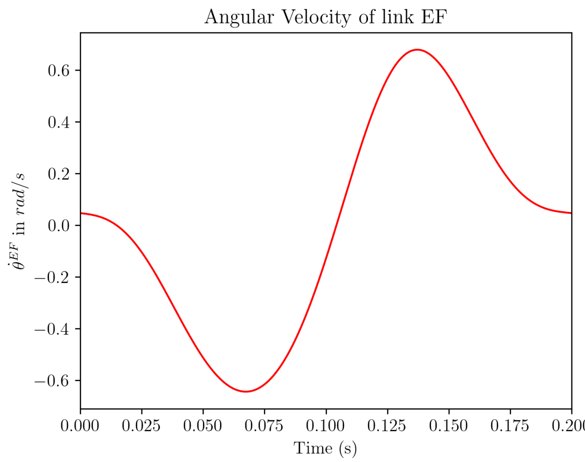
Girish Madhavan V ME20B072

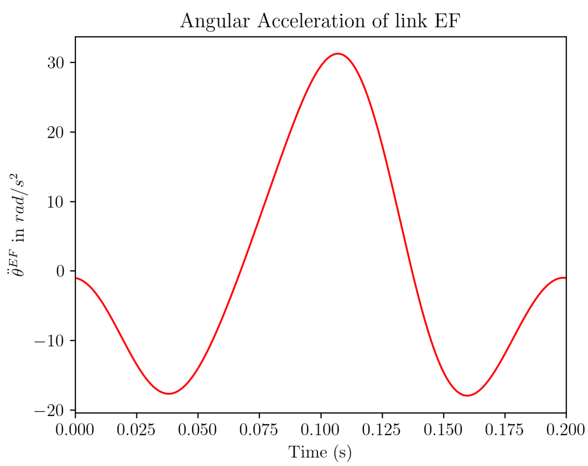
Question 1

Plots :









Grashof Mechanism Check :

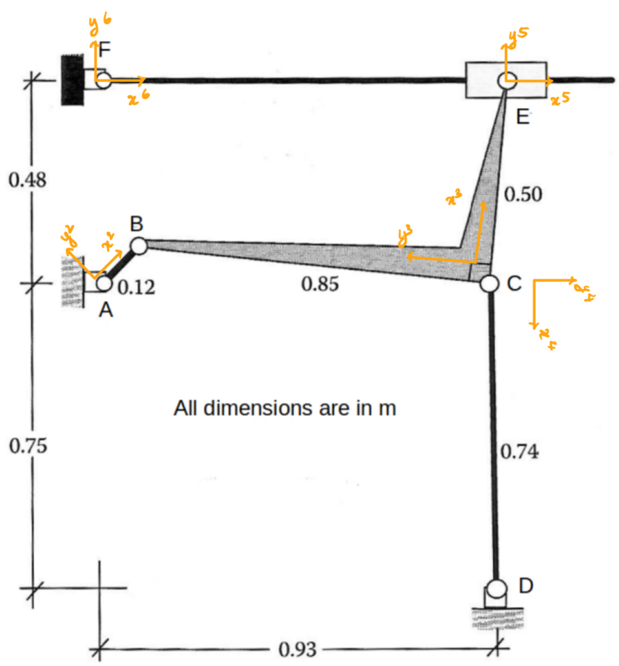


Figure 1:

AB = 0.12m (Shortest Link)
BC = 0.85m
CD = 0.74m
AD = $\sqrt{0.75^2 + 0.93^2}$ = 1.1947m (Longest Link)

For ABCD to be a Grashof Mechanism,
Sum of Lengths of shortest and longest links should be less than the sum of the remaining two link lengths in a four bar mechanism

AB + AD $<$ BC + CD
0.12 + 1.1947 $<$ 0.85 + 0.74
1.3147 $<$ 1.59
Hence ABCD is a Grashof Mechanism

Code :

```
import numpy as np
import numpy.linalg as linal
import matplotlib.pyplot as plt
import os
import sys

sys.path.append(os.path.join('D:\PRML\pyblish','plots'))
import publish

l2  = 0.12
l3x = 0.50
l3y = 0.85
l4  = 0.74

L2  = np.array([[l2 , 0]]).T
L3x = np.array([[l3x, 0]]).T
L3y = np.array([[0  , l3y]]).T
L4  = np.array([[l4 , 0]]).T

def A(theta):
    return np.array([[np.cos(theta), -np.sin(theta)],\
                     [np.sin(theta),  np.cos(theta)] ])

def At(theta) :
    return np.array([[-np.sin(theta), -np.cos(theta)],\
                     [ np.cos(theta), -np.sin(theta)]])

def split_coordinates(q) :
    R2     = q[0:2]
    R3     = q[2:4]
    theta3 = q[4,0]
    R4     = q[5:7]
    theta4 = q[7,0]
    R5     = q[8:10]
    theta5 = q[10,0]
    R6     = q[11:13]
    theta6 = q[13,0]
    theta2 = q[14,0]

    return [R2,R3,R4,R5,R6,theta2,theta3,theta4,theta5,theta6]

def C(q,t) :
    R2,R3,R4,R5,R6,theta2,theta3,theta4,theta5,theta6 = split_coordinates(q)

    A2 = A(theta2)
    A3 = A(theta3)
    A4 = A(theta4)
    A5 = A(theta5)
    A6 = A(theta6)

    row1 = R2
    row2 = R3+A3@L3y-R2-A2@L2
    row3 = R4-R3
    row4 = R4+A4@L4-np.array([[0.93],[-0.75]])
    row5 = R5-R3-A3@L3x
    row6 = R6-np.array([[0],[0.48]])
    row7 = theta5-theta6
    row8 = A6[:,[1]].T@(R5-R6)
    row9 = theta2 + 31.416*t

    return np.row_stack([ row1, row2, row3, row4, row5, row6, row7, row8, row9 ])

```

```
def Cq(q):
    I  = np.eye(2)
    Z  = 0*I
    Zv = Z[:,0]

    R2,R3,R4,R5,R6,theta2,theta3,theta4,theta5,theta6 = split_coordinates(q)

    A2 = At(theta2)@L2
    A3 = At(theta3)
    A4 = At(theta4)@L4
    A6 =  A(theta6)

    A3x = A3@L3x
    A3y = A3@L3y
    A62 = A6[:,[1]].T
    A61 = A6[:,[0]].T

    row1 = np.column_stack([ I, Z,  Zv, Z, Zv, Z, Zv, Z, Zv, Zv])
    row2 = np.column_stack([-I, I, A3y, Z, Zv, Z, Zv, Z, Zv,-A2])
    row3 = np.column_stack([ Z,-I,  Zv, I, Zv, Z, Zv, Z, Zv, Zv])
    row4 = np.column_stack([ Z, Z,  Zv, I, A4, Z, Zv, Z, Zv, Zv])
    row5 = np.column_stack([ Z,-I,-A3x, Z, Zv, I, Zv, Z, Zv, Zv])
    row6 = np.column_stack([ Z, Z,  Zv, Z, Zv, Z, Zv, I, Zv, Zv])
    row7 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 0,   0, 0,    1, 0, 0, -1, 0])
    row8 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 0, A62, 0, -A62, -A61@(R5-R6), 0])
    row9 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 0,   0, 0,    0, 0, 0,  0, 1])

    return np.row_stack([ row1, row2, row3, row4, row5, row6, row7, row8, row9 ])

def CqQ(q,qd):
    I  = np.eye(2)
    Z  = 0*I
    Zv = Z[:,0]
    Zr1 = np.zeros((1,15))
    Zr2 = np.zeros((2,15))

    R2 ,R3 ,R4 , R5, R6, theta2, theta3, theta4, theta5, theta6
    = split_coordinates(q)
    R2d,R3d,R4d,R5d,R6d,theta2d,theta3d,theta4d,theta5d,theta6d
    = split_coordinates(qd)

    A2 = A(theta2)@L2
    A3 = A(theta3)
    A4 = A(theta4)@L4
    A6 = A(theta6)

    A3x= A3@L3x
    A3y= A3@L3y
    A62 = A6[:,[1]].T
    A61 = A6[:,[0]].T

    row1 = Zr2
    row2 = np.column_stack([ Z, Z, -theta3d*A3y, Z, Zv, Z, Zv, Z, Zv, theta2d*A2])
    row3 = Zr2
    row4 = np.column_stack([ Z, Z, Zv, Z, -theta4d*A4, Z, Zv, Z, Zv, Zv])
    row5 = np.column_stack([ Z, Z,  theta3d*A3x, Z, Zv, Z, Zv, Z, Zv, Zv])
    row6 = Zr2
    row7 = Zr1
    row8 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 0, -theta6d*A61, 0, \
    theta6d*A61, -A61@(R5d-R6d)-theta6d*A62@(R5-R6), 0])
    row9 = Zr1

    return np.row_stack([ row1, row2, row3, row4, row5, row6, row7, row8, row9 ])
```

```python
# Initial Coordinates :
R2 = np.array([[0],[0]])
R3 = np.array([[0.969927],[-0.011077]])
R4 = R3
R5 = np.array([[0.976444],[0.488879]])
R6 = np.array([[0],[0.48]])

theta2 = 0
theta3 = np.deg2rad(89.25325)
theta4 = np.deg2rad(360-93.09298)
theta5 = np.deg2rad(0.521021)
theta6 = theta5

q1 = np.row_stack([R2,R3,theta3,R4,theta4,R5,theta5,R6,theta6,theta2])
q1

q1   = np.row_stack([R2,R3,theta3,R4,theta4,R5,theta5,R6,theta6,theta2])
time = np.arange(0,0.2,0.0001)
Ct   = np.array([[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,300*2*np.pi/60]]).T
dispEx = []
dispEy = []
velEx  = []
velEy  = []
accEX  = []
accEY  = []
angularposEF = []
angularvelEF = []
angularaccEF = []

for t in time :

    while True :

        L = Cq(q1)
        R = -C(q1,t)

        dq1 = linal.inv(L)@R
        q1 = q1 + dq1
        if ( np.abs(dq1) < 1e-5 ).all() :
            break

    dispEx.append(q1[8,0])
    dispEy.append(q1[9,0])
    angularposEF.append(q1[10,0])

    # Velocity
    q2 = -linal.inv(Cq(q1))@Ct

    velEx.append(q2[8,0])
    velEy.append(q2[9,0])
    angularvelEF.append(q2[10,0])

    # Acceleration
    q2_dot = -linal.inv(Cq(q1))@CqQ(q1,q2)@q2

    accEX.append(q2_dot[8,0])
    accEY.append(q2_dot[9,0])
    angularaccEF.append(q2_dot[13,0])

plt.plot(time,dispEx,color='red')
plt.title("Displacement of E : X componenet")
plt.xlabel('Time (s)')
plt.ylabel('Displacement (m)')
plt.xlim(0,0.2)
```

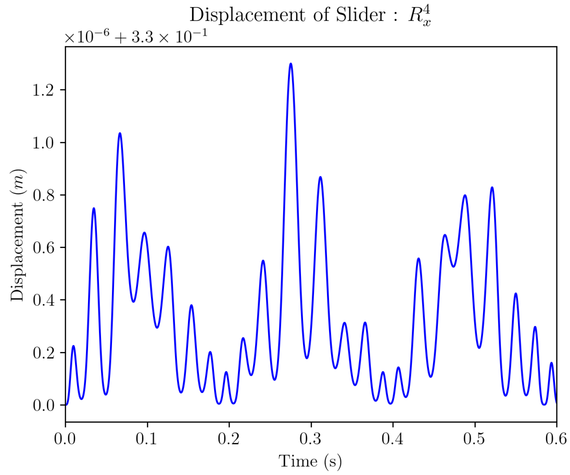
```
plt.savefig('dispEx.png',dpi=260)
plt.show()

plt.plot(time,dispEy,color='blue')
plt.title("Displacement of E : Y componenet")
plt.xlabel('Time (s)')
plt.ylabel('Displacement (m)')
plt.xlim(0,0.2)
plt.savefig('dispEy.png',dpi=260)
plt.show()

plt.plot(time,velEx,color='red')
plt.title("Velocity of E : X componenet")
plt.xlabel('Time (s)')
plt.ylabel('Velocity ($m/s$)')
plt.xlim(0,0.2)
plt.savefig('velEx.png',dpi=260)
plt.show()

plt.plot(time,velEy,color='blue')
plt.title("Velocity of E : Y componenet")
plt.xlabel('Time (s)')
plt.ylabel('Velocity ($m/s$)')
plt.xlim(0,0.2)
plt.savefig('velEy.png',dpi=260)
plt.show()

plt.plot(time,accEX,color='red')
plt.title("Acceleration of E : X componenet")
plt.xlabel('Time (s)')
plt.ylabel('Acceleration ($m/s^2$)')
plt.xlim(0,0.2)
plt.savefig('accEx.png',dpi=260)
plt.show()

plt.plot(time,accEY,color='blue')
plt.title("Acceleration of E : Y componenet")
plt.xlabel('Time (s)')
plt.ylabel('Acceleration ($m/s^2$)')
plt.xlim(0,0.2)
plt.savefig('accEy.png',dpi=260)
plt.show()

plt.plot(time,angularposEF,color='red')
plt.xlabel('Time (s)')
plt.ylabel(r'$\theta^{EF}$ in $rads$')
plt.title("Angular position of link EF")
plt.xlim(0,0.2)
plt.savefig('angularposEF.png',dpi=260)

plt.plot(time,angularvelEF,color='red')
plt.xlabel('Time (s)')
plt.ylabel(r'$\dot \theta^{EF}$ in $rad/s$')
plt.title("Angular Velocity of link EF")
plt.xlim(0,0.2)
plt.savefig('angularvelEF.png',dpi=260)

plt.plot(time,angularaccEF,color='red')
plt.xlabel('Time (s)')
plt.ylabel(r'$\ddot \theta^{EF}$ in $rad/s^2$')
plt.title("Angular Acceleration of link EF")
plt.xlim(0,0.2)
plt.savefig('angularaccEF.png',dpi=260)
```

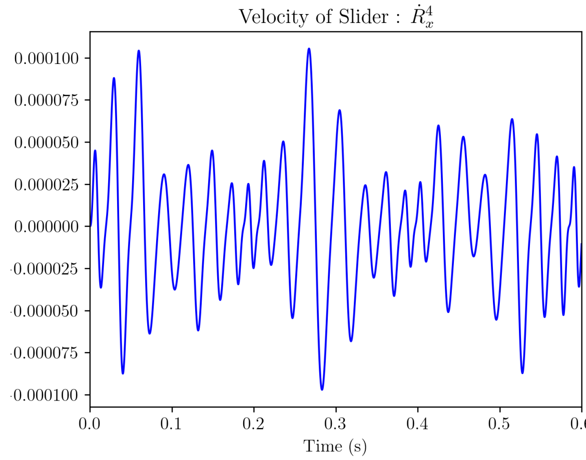
Justification :

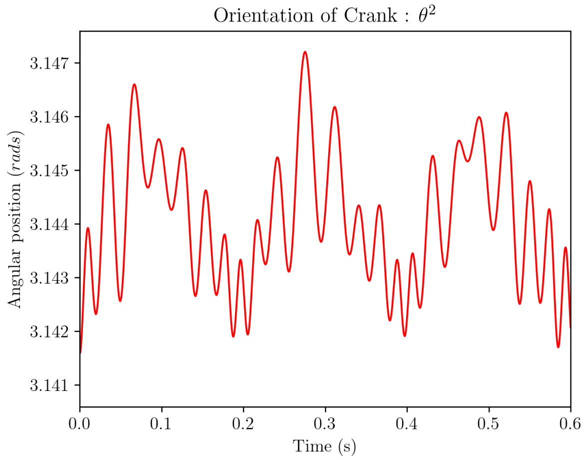
The computational results can be verified using the logic, whenever a function's derivative vanishes the corresponding function value at that point would have reached an extremum (maxima or minima). Using this from the plots for EF we can see that the Angular acceleration is zero at four points and the corresponding angular velocities are part of the local extremum. Similarly for the Angular position of link EF, we can observe two extremum and the Angular velocity vanishes at these points in the plot. In the same manner When AB starts rotating clockwise, through intuition one can conclude that the X and Y coordinates of E must decrease before increasing. Once we have proven that the displacement plot is correct, we can verify the other plots using the derivative of displacement plots. As the plots obtained through numerical methods align with our intuition, this could be considered as a valid proof for correctness of numerical methods.

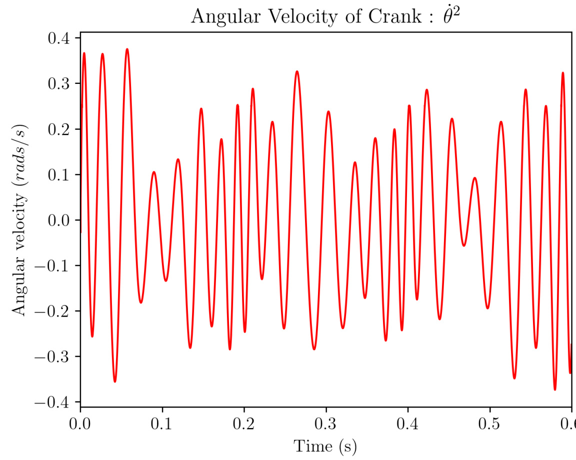
Question 2

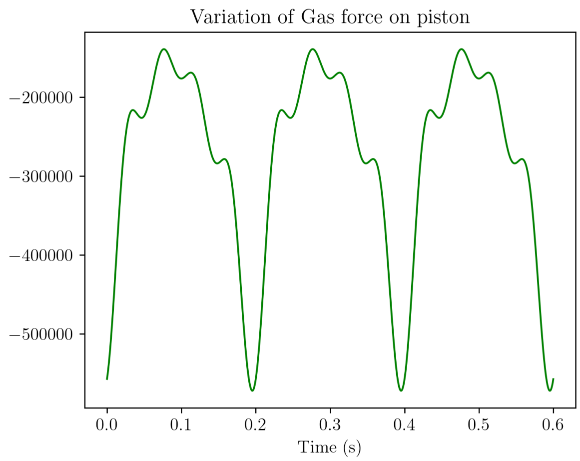
Plots :











Code :

```
import numpy as np
import numpy.linalg as linal
import matplotlib.pyplot as plt
import os
import sys

# Latex Plotting Plugin
sys.path.append(os.path.join('D:\PRML\pyblish','plots'))
import publish

R = 0.11
r = 0.0275
L = 0.44
l = 0.12

g = 9.81

L12 = np.array([[r],[0]])
L2j = np.array([[R-r],[0]])
Lj3 = np.array([[l],[0]])
L34 = np.array([[L-l],[0]])

m2 = 89.5
m3 = 30.1
m4 = 24.3
J2 = m2*R**2/12
J3 = m3*L**2/12
J4 = 0

M = np.diag([ m2, m2, J2, m3, m3, J3, m4, 0, m4 ])

def Qe(t) :
    PI = np.pi
    val = 108.22 + 57.54*np.cos(10*PI*t) - 18.73*np.sin(10*PI*t) + 22.23*np.cos(20*PI*t) -
    6.97*np.sin(20*PI*t) + 17.28*np.cos(30*PI*t) - 7.03*np.sin(30*PI*t) + 13.66*np.cos(40*PI*
    t) - 6.55*np.sin(40*PI*t)
    val = val*PI*0.09**2*100000
    return np.array([[0],[-m2*g],[0],[0],[-m3*g],[0],[-m4*g],[0],[-val]])

def Force(t) :
    PI = np.pi
    val = 108.22 + 57.54*np.cos(10*PI*t) - 18.73*np.sin(10*PI*t) + 22.23*np.cos(20*PI*t) -
    6.97*np.sin(20*PI*t) + 17.28*np.cos(30*PI*t) - 7.03*np.sin(30*PI*t) + 13.66*np.cos(40*PI*
    t) - 6.55*np.sin(40*PI*t)
    val = val*PI*0.09**2*100000
    return -val

def A(theta):
    return np.array([[np.cos(theta), -np.sin(theta)],\
                     [np.sin(theta),  np.cos(theta)] ])

def At(theta) :
    return np.array([[-np.sin(theta), -np.cos(theta)],\
                     [ np.cos(theta), -np.sin(theta)]])

# Order of variables : Rx2 Ry2 theta2 Rx3 Ry3 theta3 Ry4 theta4 Rx4
def split_coordinates(q) :
    R2 = q[0:2]
    R3 = q[3:5]
    R4 = q[[8,6]]
    theta2 = q[2,0]
    theta3 = q[5,0]
```

```
    theta4 = q[7,0]

    return [R2,R3,R4,theta2,theta3,theta4]

def C(q):
    I = np.eye(2)
    Z = 0*I
    Zv= Z[:,0]

    R2,R3,R4,theta2,theta3,theta4 = split_coordinates(q)

    A2 = A(theta2)
    A21 = A2@L12
    A2j = A2@L2j
    A3 = A(theta3)
    Aj3 = A3@Lj3
    A34 = A3@L34

    row1 = R2 - A21
    row2 = R3 - Aj3 - R2 - A2j
    row3 = R4 - R3 - A34
    row4 = theta4
    row5 = R4[1,0]

    return np.row_stack([ row1, row2, row3, row4, row5 ])

def Cq(q) :
    I = np.eye(2)
    Z = 0*I
    Zv= Z[:,0]

    R2,R3,R4,theta2,theta3,theta4 = split_coordinates(q)

    A2  = At(theta2)
    A21 = A2@L12
    A2j = A2@L2j

    A3  = At(theta3)
    Aj3 = A3@Lj3
    A34 = A3@L34

    row1 = np.column_stack([ I, -A21, Z,  Zv, Zv, Zv, Zv ])
    row2 = np.column_stack([-I, -A2j, I,-Aj3, Zv, Zv, Zv ])
    row3 = np.column_stack([ Z,   Zv,-I,-A34, I[:,1], Zv, I[:,0]])
    row4 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 1, 0])
    row5 = np.column_stack([ 0, 0, 0, 0, 0, 0, 1, 0, 0])

    return np.row_stack([ row1, row2, row3, row4, row5 ])

def CqQ(q,qd) :
    I = np.eye(2)
    Z = 0*I
    Zv= Z[:,0]

    R2,R3,R4,theta2,theta3,theta4 = split_coordinates(q)
    R2d, R3d, R4d, theta2d, theta3d, theta4d = split_coordinates(qd)

    A2  = A(theta2)
    A21 = A2@L12
    A2j = A2@L2j
    A3  = A(theta3)
    Aj3 = A3@Lj3
    A34 = A3@L34

```

```
    row1 = np.column_stack([ Z, theta2d*A21, Z,           Zv, Zv, Zv, Zv])
    row2 = np.column_stack([ Z, theta2d*A2j, Z, theta3d*Aj3, Zv, Zv, Zv])
    row3 = np.column_stack([ Z,           Zv, Z, theta3d*A34, Zv, Zv, Zv])
    row4 = np.zeros((2,9))

    return np.row_stack([ row1, row2, row3, row4 ])

def Bi(Cq):
    Cqd = Cq[:,0:8]
    Cqi = Cq[:,[8]]
    I   = np.eye(1)
    return np.row_stack([ -linal.inv(Cqd)@Cqi, I])

def G(Cq,Qd1):
    Cqd = Cq[:,0:8]
    return np.row_stack([linal.inv(Cqd)@Qd1, 0])

def Qd(q,q_dot,Cqt=np.zeros((8,9)),Ctt=np.zeros((8,1))):
    A1 = -CqQ(q,q_dot)@q_dot
    A2 = -2*Cqt@q_dot
    A3 = -Ctt
    return A1+A2+A3

R2 = np.array([[-r],[0]])
R3 = np.array([[l-R],[0]])
R4 = np.array([[0.33],[0]])

theta2 = np.deg2rad(180)
theta3 = np.deg2rad(0)
theta4 = np.deg2rad(0)

q1  = np.row_stack([R2,theta2,R3,theta3,R4[1,0],theta4,R4[0,0]])
q2i = 0
h   = 0.00004

sliderpos  = []
slidervel  = []
angularpos = []
angularvel = []

T    = np.arange(0,0.6,h)

for time in T :

    while True :

        LHS = np.row_stack([Cq(q1),[0,0,0,0,0,0,0,0,1]])
        RHS = np.row_stack([-C(q1),0])

        dq1 = linal.inv(LHS)@RHS
        q1 = q1 + dq1
        if ( np.abs(dq1) < 1e-5 ).all() :
            break

    Cq1 = Cq(q1)
    Bi1 = Bi(Cq1)

    RHS  = np.row_stack([np.zeros((8,1)),q2i])
    q2   = linal.inv(LHS)@RHS

    Qd1 = Qd(q1,q2)
    G1  = G(Cq1,Qd1)
    Mi  = Bi1.T@M@Bi1

```

```python
    q2i_dot = ( linal.inv(Mi)@Bi1.T@Qe(time) - linal.inv(Mi)@Bi1.T@M@G1 )[0,0]
    q2i    = q2i + q2i_dot*h
    q1i    = q1[8,0] + q2i*h
    q1     = np.row_stack([q1[0:8],q1i])

    print(time*100/0.6)
    sliderpos.append(q1i)
    slidervel.append(q2i)
    angularpos.append(q1[2,0])
    angularvel.append(q2[2,0])

plt.plot(T,sliderpos,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Displacement ($m$)')
plt.title('Displacement of Slider : $R_x^4$')
plt.savefig('Rx4.png',dpi=260)
plt.xlim(0,0.6)
plt.show()

plt.plot(T,slidervel,color='blue')
plt.xlabel('Time (s)')
plt.ylabel('Velocity ($m/s$)')
plt.title('Velocity of Slider : $\dot R_x^4$')
plt.savefig('Rx4d.png',dpi=260)
plt.xlim(0,0.6)
plt.show()

plt.plot(T,np.array(angularpos)+104*np.pi,color='red')
plt.xlabel('Time (s)')
plt.ylabel('Angular position ($rads$)')
plt.title(r'Orientation of Crank : $\theta^2$')
plt.xlim(0,0.6)
plt.ylim(np.pi -0.001, np.pi + 0.006)
plt.savefig('theta2.png',dpi=260)
plt.show()

plt.plot(T,angularvel,color='red')
plt.xlabel('Time (s)')
plt.ylabel('Angular velocity ($rads/s$)')
plt.title(r'Angular Velocity of Crank : $\dot \theta^2$')
plt.savefig('theta2d.png',dpi=260)
plt.xlim(0,0.6)
plt.show()
Qev = np.vectorize(Force)
plt.plot(T,Qev(T))
```

Justification :

Our Initial orientation of crank says that it is in a fully retracted configuration and the Gas pressure acting on the system prevents the piston from moving around. This is because the piston needs to move in positive X direction to reach other positions, as it is already in an extremum. But from the Force Plot we can see that the Force is always directed along negative X direction. Secondly the magnitude of the force is of order 10^5 but that of the weights of the bodies are of 10^2. Hence when the crank tries to move downwards due to gravity, the Gas pressure prevents this and easily restores it to its starting position due to the huge difference in order of magnitudes. Whenever the crank overshoots this initial position and moves upwards, the Gas pressure will again prevent it from doing so. From this we can conclude that our system will be undergoing oscillations about the given initial state, as this intuition is consistent with the plots obtained through computation we can confirm the correctness of this method.