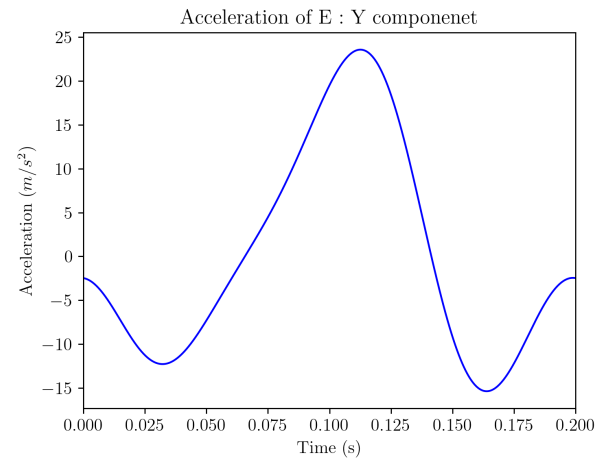
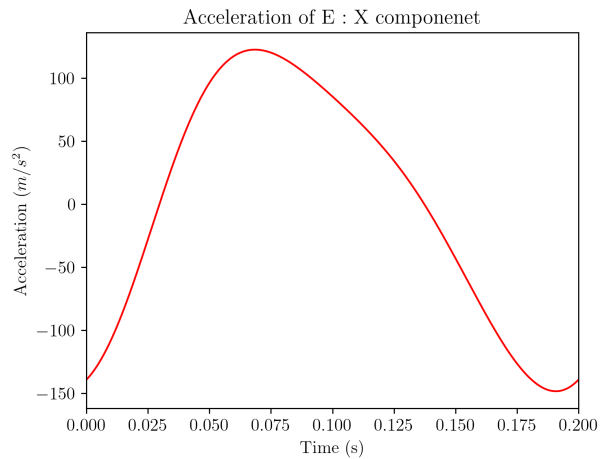
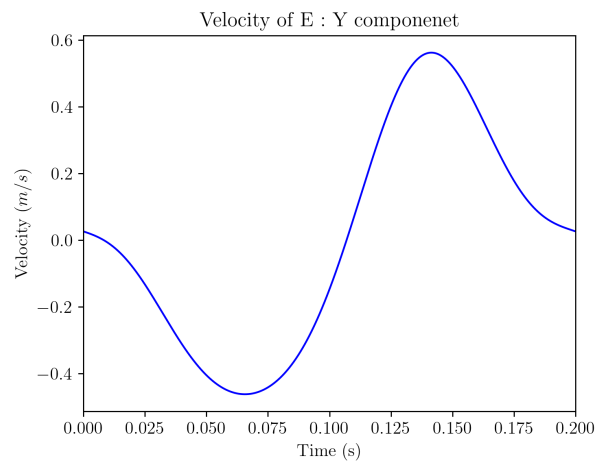
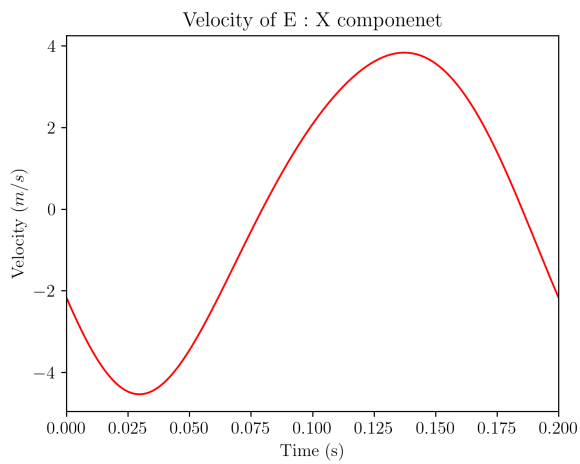
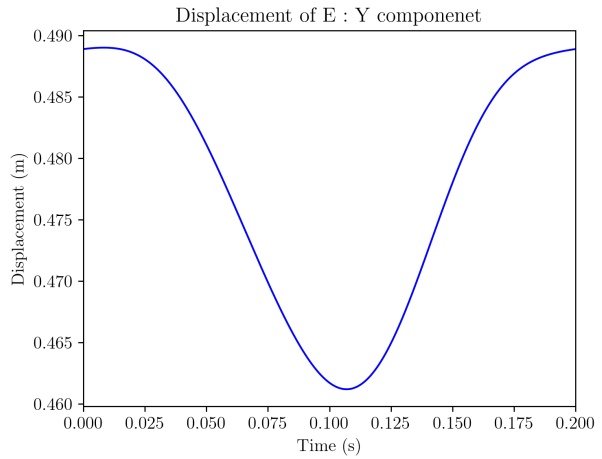
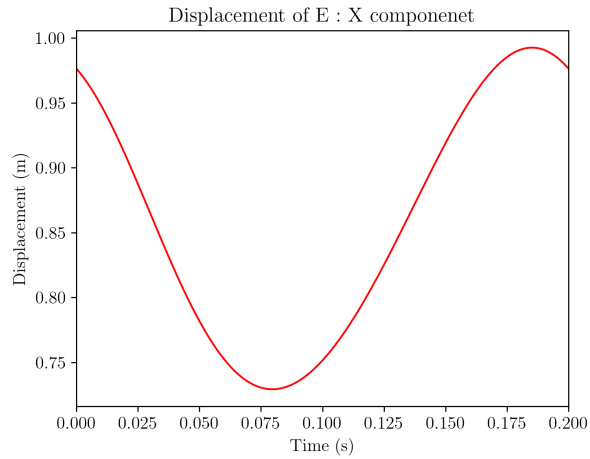


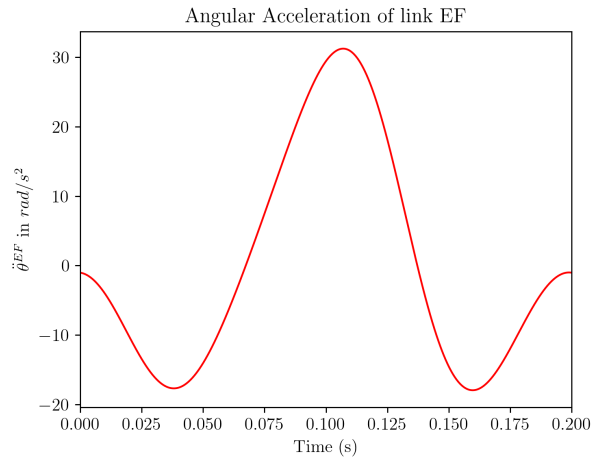
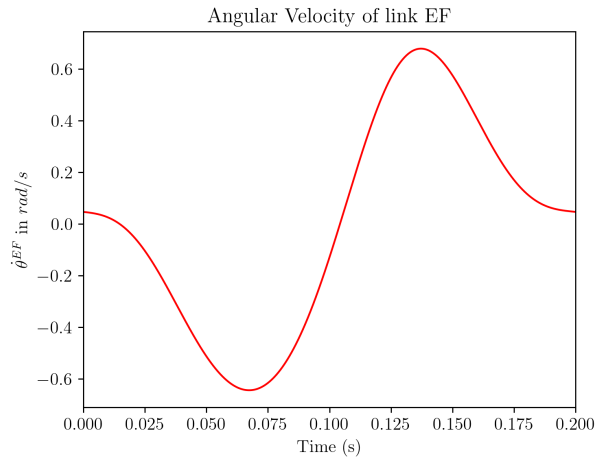
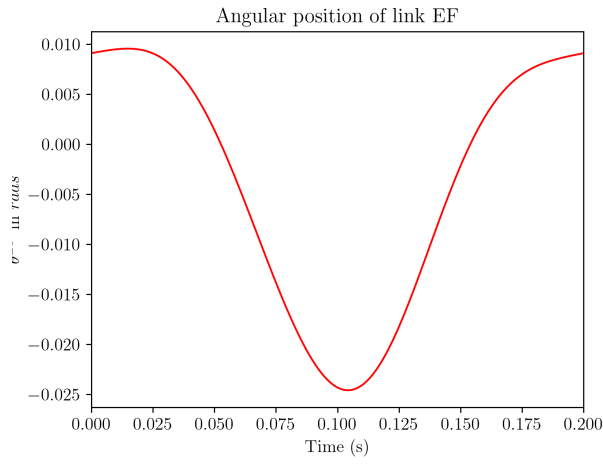
Multibody Dynamics and its Applications : Endsemester Jul-Nov 23'

Girish Madhavan V ME20B072

Question 1

Plots :





Grashof Mechanism Check :

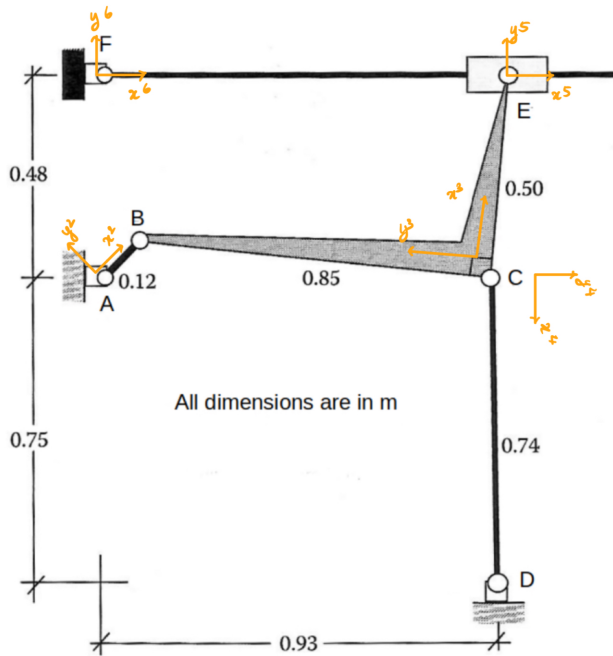


Figure 1:

$AB = 0.12\text{m}$ (Shortest Link)

$BC = 0.85\text{m}$

$CD = 0.74\text{m}$

$AD = \sqrt{0.75^2 + 0.93^2} = 1.1947\text{m}$ (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 :

```
1 import numpy as np
2 import numpy.linalg as linal
3 import matplotlib.pyplot as plt
4 import os
5 import sys
6
7 sys.path.append(os.path.join('D:\PRML\pyblish', 'plots'))
8 import publish
9
10 l2 = 0.12
11 l3x = 0.50
12 l3y = 0.85
13 l4 = 0.74
14
15 L2 = np.array([[l2, 0]]).T
16 L3x = np.array([[l3x, 0]]).T
17 L3y = np.array([[0, l3y]]).T
18 L4 = np.array([[l4, 0]]).T
19
20 def A(theta):
21     return np.array([[np.cos(theta), -np.sin(theta)],\
22                     [np.sin(theta), np.cos(theta)]] )
23
24 def At(theta) :
25     return np.array([[ -np.sin(theta), -np.cos(theta)],\
26                     [ np.cos(theta), -np.sin(theta)]] )
27
28 def split_coordinates(q) :
29     R2 = q[0:2]
30     R3 = q[2:4]
31     theta3 = q[4,0]
32     R4 = q[5:7]
33     theta4 = q[7,0]
34     R5 = q[8:10]
35     theta5 = q[10,0]
36     R6 = q[11:13]
37     theta6 = q[13,0]
38     theta2 = q[14,0]
39
40     return [R2,R3,R4,R5,R6,theta2,theta3,theta4,theta5,theta6]
41
42 def C(q,t) :
43     R2,R3,R4,R5,R6,theta2,theta3,theta4,theta5,theta6 = split_coordinates(q)
44
45     A2 = A(theta2)
46     A3 = A(theta3)
47     A4 = A(theta4)
48     A5 = A(theta5)
49     A6 = A(theta6)
50
51     row1 = R2
52     row2 = R3+A3@L3y-R2-A2@L2
53     row3 = R4-R3
54     row4 = R4+A4@L4-np.array([[0.93],[-0.75]])
55     row5 = R5-R3-A3@L3x
56     row6 = R6-np.array([[0],[0.48]])
57     row7 = theta5-theta6
58     row8 = A6[:,[1]].T@(R5-R6)
59     row9 = theta2 + 31.416*t
60
61     return np.row_stack([ row1, row2, row3, row4, row5, row6, row7, row8, row9 ])
```

```

63 def Cq(q):
64     I = np.eye(2)
65     Z = 0*I
66     Zv = Z[:,0]
67
68     R2,R3,R4,R5,R6,theta2,theta3,theta4,theta5,theta6 = split_coordinates(q)
69
70     A2 = At(theta2)@L2
71     A3 = At(theta3)
72     A4 = At(theta4)@L4
73     A6 = A(theta6)
74
75     A3x = A3@L3x
76     A3y = A3@L3y
77     A62 = A6[:,[1]].T
78     A61 = A6[:,[0]].T
79
80     row1 = np.column_stack([ I, Z, Zv, Z, Zv, Z, Zv, Z, Zv, Zv])
81     row2 = np.column_stack([-I, I, A3y, Z, Zv, Z, Zv, Z, Zv, -A2])
82     row3 = np.column_stack([ Z,-I, Zv, I, Zv, Z, Zv, Z, Zv, Zv])
83     row4 = np.column_stack([ Z, Z, Zv, I, A4, Z, Zv, Z, Zv, Zv])
84     row5 = np.column_stack([ Z,-I,-A3x, Z, Zv, I, Zv, Z, Zv, Zv])
85     row6 = np.column_stack([ Z, Z, Zv, Z, Zv, Z, Zv, I, Zv, Zv])
86     row7 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
87     row8 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 0, A62, 0, -A62, -A61@(R5-R6), 0])
88     row9 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
89
90     return np.row_stack([ row1, row2, row3, row4, row5, row6, row7, row8, row9 ])
91
92 def CqQ(q,qd):
93     I = np.eye(2)
94     Z = 0*I
95     Zv = Z[:,0]
96     Zr1 = np.zeros((1,15))
97     Zr2 = np.zeros((2,15))
98
99     R2 ,R3 ,R4 , R5, R6, theta2, theta3, theta4, theta5, theta6
100 = split_coordinates(q)
101 R2d,R3d,R4d,R5d,R6d,theta2d,theta3d,theta4d,theta5d,theta6d
102 = split_coordinates(qd)
103
104 A2 = A(theta2)@L2
105 A3 = A(theta3)
106 A4 = A(theta4)@L4
107 A6 = A(theta6)
108
109 A3x= A3@L3x
110 A3y= A3@L3y
111 A62 = A6[:,[1]].T
112 A61 = A6[:,[0]].T
113
114 row1 = Zr2
115 row2 = np.column_stack([ Z, Z, -theta3d*A3y, Z, Zv, Z, Zv, Z, Zv, theta2d*A2])
116 row3 = Zr2
117 row4 = np.column_stack([ Z, Z, Zv, Z, -theta4d*A4, Z, Zv, Z, Zv, Zv])
118 row5 = np.column_stack([ Z, Z, theta3d*A3x, Z, Zv, Z, Zv, Z, Zv, Zv])
119 row6 = Zr2
120 row7 = Zr1
121 row8 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 0, -theta6d*A61, 0, \
122 theta6d*A61, -A61@(R5d-R6d)-theta6d*A62@(R5-R6), 0])
123 row9 = Zr1
124
125 return np.row_stack([ row1, row2, row3, row4, row5, row6, row7, row8, row9 ])
126

```

```

127 # Initial Coordinates :
128 R2 = np.array([[0],[0]])
129 R3 = np.array([[0.969927],[-0.011077]])
130 R4 = R3
131 R5 = np.array([[0.976444],[0.488879]])
132 R6 = np.array([[0],[0.48]])
133
134 theta2 = 0
135 theta3 = np.deg2rad(89.25325)
136 theta4 = np.deg2rad(360-93.09298)
137 theta5 = np.deg2rad(0.521021)
138 theta6 = theta5
139
140 q1 = np.row_stack([R2,R3,theta3,R4,theta4,R5,theta5,R6,theta6,theta2])
141 q1
142
143 q1 = np.row_stack([R2,R3,theta3,R4,theta4,R5,theta5,R6,theta6,theta2])
144 time = np.arange(0,0.2,0.0001)
145 Ct = np.array([[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,300*2*np.pi/60]]).T
146 dispEx = []
147 dispEy = []
148 velEx = []
149 velEy = []
150 accEX = []
151 accEY = []
152 angularposEF = []
153 angularvelEF = []
154 angularaccEF = []
155
156 for t in time :
157
158     while True :
159
160         L = Cq(q1)
161         R = -C(q1,t)
162
163         dq1 = linalg.inv(L)@R
164         q1 = q1 + dq1
165         if ( np.abs(dq1) < 1e-5 ).all() :
166             break
167
168         dispEx.append(q1[8,0])
169         dispEy.append(q1[9,0])
170         angularposEF.append(q1[10,0])
171
172     # Velocity
173     q2 = -linalg.inv(Cq(q1))@Ct
174
175     velEx.append(q2[8,0])
176     velEy.append(q2[9,0])
177     angularvelEF.append(q2[10,0])
178
179     # Acceleration
180     q2_dot = -linalg.inv(Cq(q1))@CqQ(q1,q2)@q2
181
182     accEX.append(q2_dot[8,0])
183     accEY.append(q2_dot[9,0])
184     angularaccEF.append(q2_dot[13,0])
185
186 plt.plot(time,dispEx,color='red')
187 plt.title("Displacement of E : X componenet")
188 plt.xlabel('Time (s)')
189 plt.ylabel('Displacement (m)')
190 plt.xlim(0,0.2)

```

```

191 plt.savefig('dispEx.png',dpi=260)
192 plt.show()
193
194 plt.plot(time,dispEy,color='blue')
195 plt.title("Displacement of E : Y componenet")
196 plt.xlabel('Time (s)')
197 plt.ylabel('Displacement (m)')
198 plt.xlim(0,0.2)
199 plt.savefig('dispEy.png',dpi=260)
200 plt.show()
201
202 plt.plot(time,velEx,color='red')
203 plt.title("Velocity of E : X componenet")
204 plt.xlabel('Time (s)')
205 plt.ylabel('Velocity ($m/s$)')
206 plt.xlim(0,0.2)
207 plt.savefig('velEx.png',dpi=260)
208 plt.show()
209
210 plt.plot(time,velEy,color='blue')
211 plt.title("Velocity of E : Y componenet")
212 plt.xlabel('Time (s)')
213 plt.ylabel('Velocity ($m/s$)')
214 plt.xlim(0,0.2)
215 plt.savefig('velEy.png',dpi=260)
216 plt.show()
217
218 plt.plot(time,accEX,color='red')
219 plt.title("Acceleration of E : X componenet")
220 plt.xlabel('Time (s)')
221 plt.ylabel('Acceleration ($m/s^2$)')
222 plt.xlim(0,0.2)
223 plt.savefig('accEx.png',dpi=260)
224 plt.show()
225
226 plt.plot(time,accEY,color='blue')
227 plt.title("Acceleration of E : Y componenet")
228 plt.xlabel('Time (s)')
229 plt.ylabel('Acceleration ($m/s^2$)')
230 plt.xlim(0,0.2)
231 plt.savefig('accEy.png',dpi=260)
232 plt.show()
233
234 plt.plot(time,angularposEF,color='red')
235 plt.xlabel('Time (s)')
236 plt.ylabel(r'$\theta^{EF}$ in $rads$')
237 plt.title("Angular position of link EF")
238 plt.xlim(0,0.2)
239 plt.savefig('angularposEF.png',dpi=260)
240
241 plt.plot(time,angularvelEF,color='red')
242 plt.xlabel('Time (s)')
243 plt.ylabel(r'$\dot{\theta}^{EF}$ in $rad/s$')
244 plt.title("Angular Velocity of link EF")
245 plt.xlim(0,0.2)
246 plt.savefig('angularvelEF.png',dpi=260)
247
248 plt.plot(time,angularaccEF,color='red')
249 plt.xlabel('Time (s)')
250 plt.ylabel(r'$\ddot{\theta}^{EF}$ in $rad/s^2$')
251 plt.title("Angular Acceleration of link EF")
252 plt.xlim(0,0.2)
253 plt.savefig('angularaccEF.png',dpi=260)

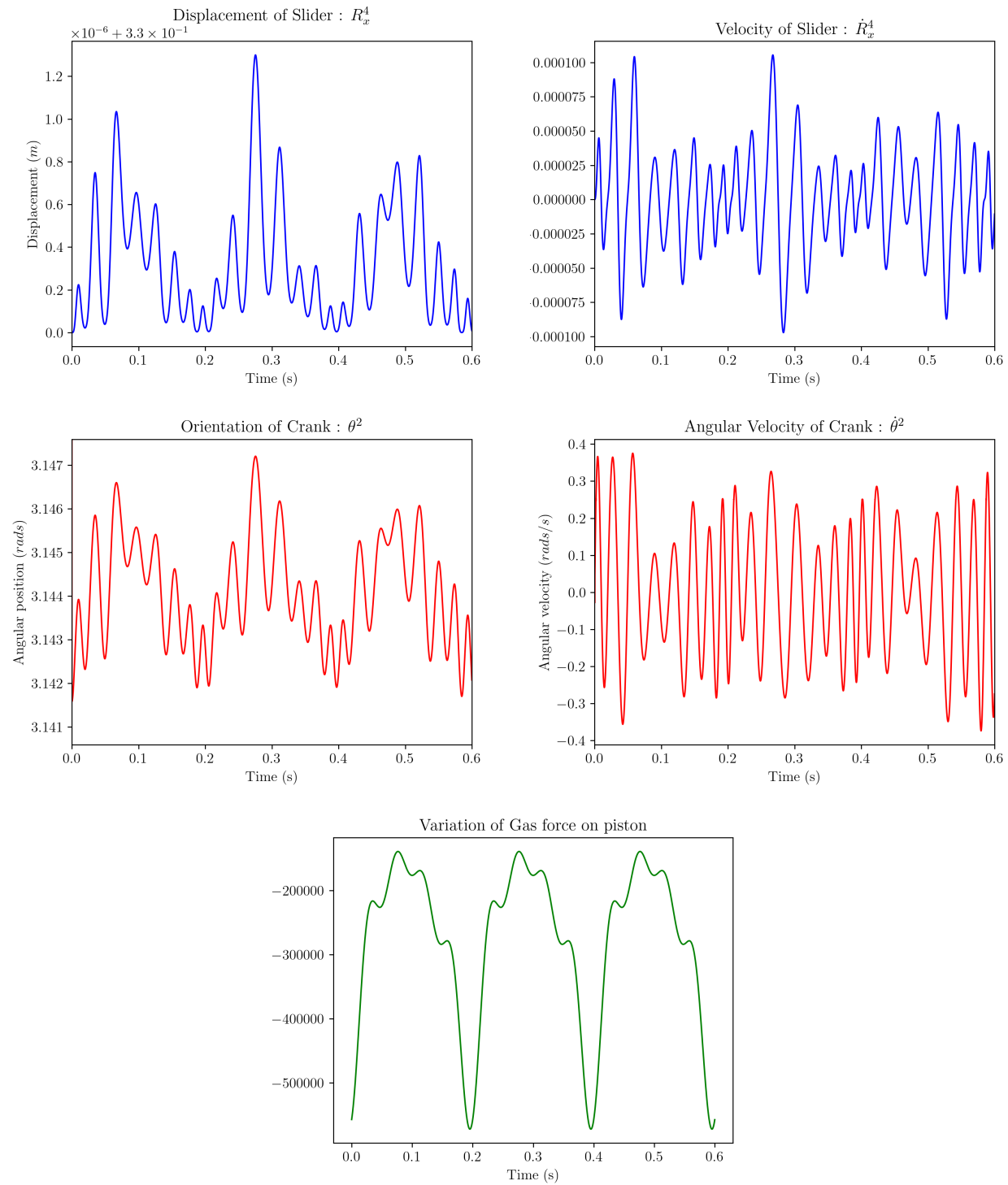
```

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 :

```
1 import numpy as np
2 import numpy.linalg as linalg
3 import matplotlib.pyplot as plt
4 import os
5 import sys
6
7 # Latex Plotting Plugin
8 sys.path.append(os.path.join('D:\PRML\pyblish', 'plots'))
9 import publish
10
11 R = 0.11
12 r = 0.0275
13 L = 0.44
14 l = 0.12
15
16 g = 9.81
17
18 L12 = np.array([[r],[0]])
19 L2j = np.array([[R-r],[0]])
20 Lj3 = np.array([[l],[0]])
21 L34 = np.array([[L-l],[0]])
22
23 m2 = 89.5
24 m3 = 30.1
25 m4 = 24.3
26 J2 = m2*R**2/12
27 J3 = m3*L**2/12
28 J4 = 0
29
30 M = np.diag([ m2, m2, J2, m3, m3, J3, m4, 0, m4 ])
31
32 def Qe(t) :
33     PI = np.pi
34     val = 108.22 + 57.54*np.cos(10*PI*t) - 18.73*np.sin(10*PI*t) + 22.23*np.cos(20*PI*t) -
35     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*
36     t) - 6.55*np.sin(40*PI*t)
37     val = val*PI*0.09**2*100000
38     return np.array([[0],[-m2*g],[0],[0],[-m3*g],[0],[-m4*g],[0],[-val]])
39
40 def Force(t) :
41     PI = np.pi
42     val = 108.22 + 57.54*np.cos(10*PI*t) - 18.73*np.sin(10*PI*t) + 22.23*np.cos(20*PI*t) -
43     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*
44     t) - 6.55*np.sin(40*PI*t)
45     val = val*PI*0.09**2*100000
46     return -val
47
48 def A(theta):
49     return np.array([[np.cos(theta), -np.sin(theta)],\
50                     [np.sin(theta), np.cos(theta)] ])
51
52 def At(theta) :
53     return np.array([[ -np.sin(theta), -np.cos(theta)],\
54                     [ np.cos(theta), -np.sin(theta)]] )
55
56 # Order of variables : Rx2 Ry2 theta2 Rx3 Ry3 theta3 Ry4 theta4 Rx4
57 def split_coordinates(q) :
58     R2 = q[0:2]
59     R3 = q[3:5]
60     R4 = q[[8,6]]
61     theta2 = q[2,0]
62     theta3 = q[5,0]
```

```

59     theta4 = q[7,0]
60
61     return [R2,R3,R4,theta2,theta3,theta4]
62
63 def C(q):
64     I = np.eye(2)
65     Z = 0*I
66     Zv= Z[:,0]
67
68     R2,R3,R4,theta2,theta3,theta4 = split_coordinates(q)
69
70     A2 = A(theta2)
71     A21 = A2@L12
72     A2j = A2@L2j
73     A3 = A(theta3)
74     Aj3 = A3@Lj3
75     A34 = A3@L34
76
77     row1 = R2 - A21
78     row2 = R3 - Aj3 - R2 - A2j
79     row3 = R4 - R3 - A34
80     row4 = theta4
81     row5 = R4[1,0]
82
83     return np.row_stack([ row1, row2, row3, row4, row5 ])
84
85 def Cq(q) :
86     I = np.eye(2)
87     Z = 0*I
88     Zv= Z[:,0]
89
90     R2,R3,R4,theta2,theta3,theta4 = split_coordinates(q)
91
92     A2 = At(theta2)
93     A21 = A2@L12
94     A2j = A2@L2j
95
96     A3 = At(theta3)
97     Aj3 = A3@Lj3
98     A34 = A3@L34
99
100    row1 = np.column_stack([ I, -A21, Z, Zv, Zv, Zv, Zv ])
101    row2 = np.column_stack([-I, -A2j, I,-Aj3, Zv, Zv, Zv ])
102    row3 = np.column_stack([ Z, Zv,-I,-A34, I[:,1], Zv, I[:,0]])
103    row4 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 1, 0])
104    row5 = np.column_stack([ 0, 0, 0, 0, 0, 0, 0, 1, 0])
105
106    return np.row_stack([ row1, row2, row3, row4, row5 ])
107
108 def CqQ(q,qd) :
109     I = np.eye(2)
110     Z = 0*I
111     Zv= Z[:,0]
112
113     R2,R3,R4,theta2,theta3,theta4 = split_coordinates(q)
114     R2d, R3d, R4d, theta2d, theta3d, theta4d = split_coordinates(qd)
115
116     A2 = A(theta2)
117     A21 = A2@L12
118     A2j = A2@L2j
119     A3 = A(theta3)
120     Aj3 = A3@Lj3
121     A34 = A3@L34
122

```

```

123     row1 = np.column_stack([ Z, theta2d*A21, Z,          Zv, Zv, Zv, Zv])
124     row2 = np.column_stack([ Z, theta2d*A2j, Z, theta3d*Aj3, Zv, Zv, Zv])
125     row3 = np.column_stack([ Z,          Zv, Z, theta3d*A34, Zv, Zv, Zv])
126     row4 = np.zeros((2,9))
127
128     return np.row_stack([ row1, row2, row3, row4 ])
129
130 def Bi(Cq):
131     Cqd = Cq[:,0:8]
132     Cqi = Cq[:,[8]]
133     I    = np.eye(1)
134     return np.row_stack([ -linal.inv(Cqd)@Cqi, I])
135
136 def G(Cq,Qd1):
137     Cqd = Cq[:,0:8]
138     return np.row_stack([linal.inv(Cqd)@Qd1, 0])
139
140 def Qd(q,q_dot,Cqt=np.zeros((8,9)),Ctt=np.zeros((8,1))):
141     A1 = -CqQ(q,q_dot)@q_dot
142     A2 = -2*Cqt@q_dot
143     A3 = -Ctt
144     return A1+A2+A3
145
146 R2 = np.array([[ -r ],[0]])
147 R3 = np.array([[1-R],[0]])
148 R4 = np.array([[0.33],[0]])
149
150 theta2 = np.deg2rad(180)
151 theta3 = np.deg2rad(0)
152 theta4 = np.deg2rad(0)
153
154 q1 = np.row_stack([R2,theta2,R3,theta3,R4[1,0],theta4,R4[0,0]])
155 q2i = 0
156 h = 0.00004
157
158 sliderpos = []
159 slidervel = []
160 angularpos = []
161 angularvel = []
162
163 T = np.arange(0,0.6,h)
164
165 for time in T :
166
167     while True :
168
169         LHS = np.row_stack([Cq(q1),[0,0,0,0,0,0,0,0,1]])
170         RHS = np.row_stack([-C(q1),0])
171
172         dq1 = linal.inv(LHS)@RHS
173         q1 = q1 + dq1
174         if ( np.abs(dq1) < 1e-5 ).all() :
175             break
176
177     Cq1 = Cq(q1)
178     Bi1 = Bi(Cq1)
179
180     RHS = np.row_stack([np.zeros((8,1)),q2i])
181     q2 = linal.inv(LHS)@RHS
182
183     Qd1 = Qd(q1,q2)
184     G1 = G(Cq1,Qd1)
185     Mi = Bi1.T@M@Bi1
186

```

```

187     q2i_dot = ( linalg.inv(Mi)@Bi1.T@Qe(time) - linalg.inv(Mi)@Bi1.T@M@G1 )[0,0]
188     q2i      = q2i + q2i_dot*h
189     q1i      = q1[8,0] + q2i*h
190     q1       = np.row_stack([q1[0:8],q1i])
191
192     print(time*100/0.6)
193     sliderpos.append(q1i)
194     slidervel.append(q2i)
195     angularpos.append(q1[2,0])
196     angularvel.append(q2[2,0])
197
198 plt.plot(T,sliderpos,color='blue')
199 plt.xlabel('Time (s)')
200 plt.ylabel('Displacement ($m$)')
201 plt.title('Displacement of Slider : $R_x^4$')
202 plt.savefig('Rx4.png',dpi=260)
203 plt.xlim(0,0.6)
204 plt.show()
205
206 plt.plot(T,slidervel,color='blue')
207 plt.xlabel('Time (s)')
208 plt.ylabel('Velocity ($m/s$)')
209 plt.title('Velocity of Slider : $\dot{R}_x^4$')
210 plt.savefig('Rx4d.png',dpi=260)
211 plt.xlim(0,0.6)
212 plt.show()
213
214 plt.plot(T,np.array(angularpos)+104*np.pi,color='red')
215 plt.xlabel('Time (s)')
216 plt.ylabel('Angular position ($rads$)')
217 plt.title(r'Orientation of Crank : $\theta^2$')
218 plt.xlim(0,0.6)
219 plt.ylim(np.pi -0.001, np.pi + 0.006)
220 plt.savefig('theta2.png',dpi=260)
221 plt.show()
222
223 plt.plot(T,angularvel,color='red')
224 plt.xlabel('Time (s)')
225 plt.ylabel('Angular velocity ($rads/s$)')
226 plt.title(r'Angular Velocity of Crank : $\dot{\theta}^2$')
227 plt.savefig('theta2d.png',dpi=260)
228 plt.xlim(0,0.6)
229 plt.show()
230 Qev = np.vectorize(Force)
231 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.