

In [1]:

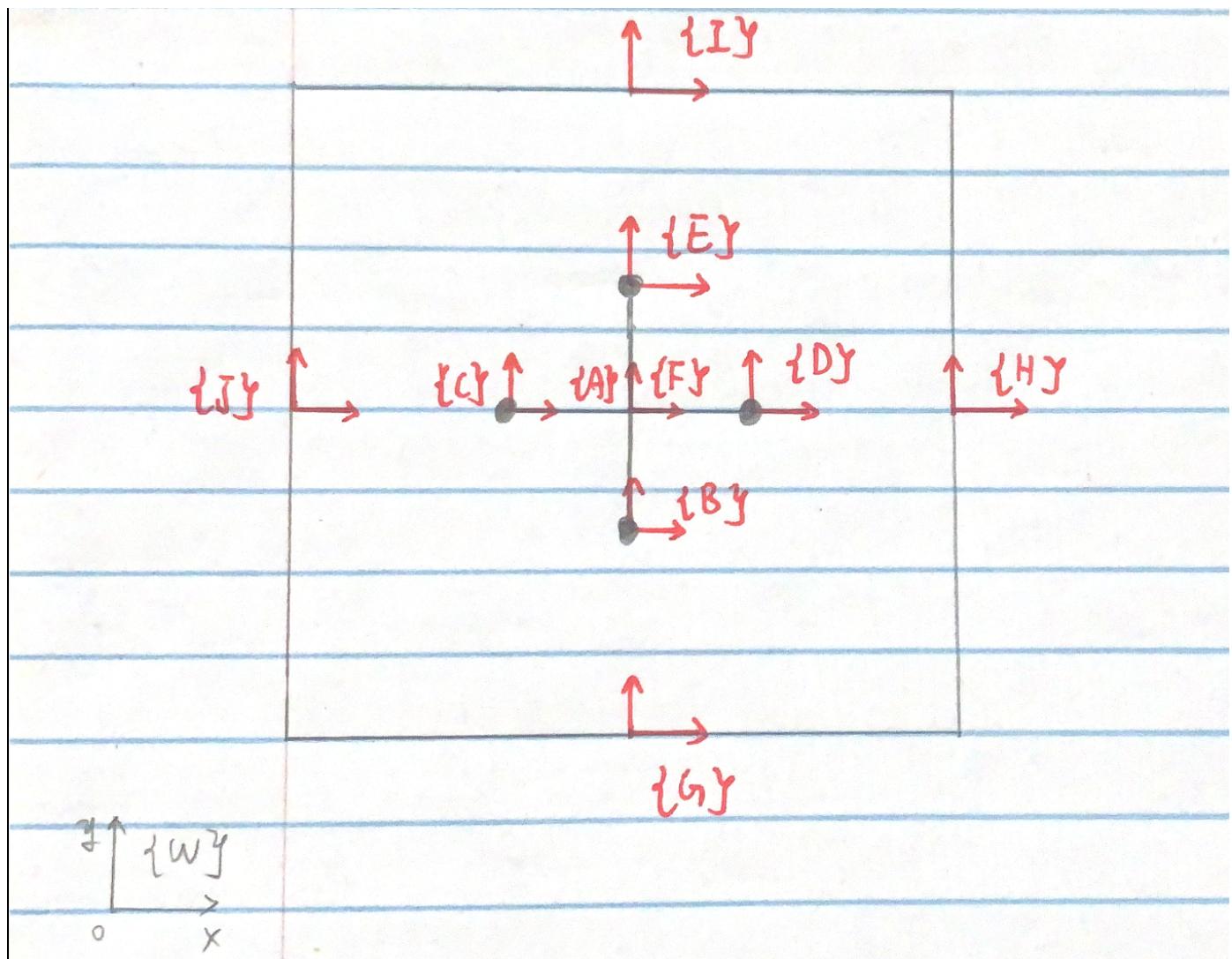
```
1 import sympy as sym
2 from sympy.abc import t
3 from sympy import symbols, Function, solve, Matrix, Eq, pi, cos, sin, transpose, Invers
4 import numpy as np
5 from numpy import pi
6
7 m = 1
8 g = 9.81
9 # define x1, y1, and theta1 as x_Box, y_Box, theta_Box
10 x1 = Function(r'x_1')(t)
11 y1 = Function(r'y_1')(t)
12 theta1 = Function(r'theta_1')(t)
13 # define x2, y2, and theta2 as x_jack, y_jack, theta_jack
14 x2 = Function(r'x_2')(t)
15 y2 = Function(r'y_2')(t)
16 theta2 = Function(r'theta_2')(t)
17 q = Matrix([x1, y1, theta1, x2, y2, theta2])
18 qdot = q.diff(t)
19 qddot = qdot.diff(t)
```

In [2]:

```

1 # define a function returns SE(3) matrices given rotation angle
2 # and 2D translation vector
3 def SE3(x, y, theta):
4     transform = Matrix([[cos(theta), -sin(theta), 0, x], [sin(theta), cos(theta), 0, y],
5                         [0, 0, 1, 0], [0, 0, 0, 1]])
6     return(transform)
7
8 # define a function for "hat" operation
9 def omghat(omg):
10    omghat = Matrix([[0, -omg[2], omg[1]],
11                     [omg[2], 0, -omg[1]],
12                     [-omg[2], omg[1], 0]])
13    return omghat
14
15 # define a function for "unhat" omg operation
16 def omgunhat(omghat):
17    omgunhat = Matrix([[omghat[2,1], omghat[0,2], omghat[1,0]]])
18    return(omgunhat)
19
20 # define a function for "unhat" V operation
21 def V_unhat(V_hat):
22    omghat = Matrix([[V_hat[0,0], V_hat[0,1], V_hat[0,2]],
23                     [V_hat[1,0], V_hat[1,1], V_hat[1,2]],
24                     [V_hat[2,0], V_hat[2,1], V_hat[2,2]]])
25    omg_unhat = omgunhat(omghat)
26    V_unhat = Matrix([[V_hat[0,3]],
27                      [V_hat[1,3]],
28                      [V_hat[2,3]],
29                      [omg_unhat[0]],
30                      [omg_unhat[1]],
31                      [omg_unhat[2]]])
32    return(V_unhat)
33
34 # define a function for matrix inverse of SE(3)
35 def inverse_SE3(g):
36    R = Matrix([[g[0, 0], g[0, 1], g[0, 2]], [g[1, 0], g[1, 1], g[1, 2]],
37                [g[2, 0], g[2, 1], g[2, 2]])]
38    R_t = transpose(R)
39    p = Matrix([[g[0,3]], [g[1,3]], [g[2,3]]])
40    p_t = -R_t*p
41    g_inverse = Matrix([[R_t[0,0], R_t[0,1], R_t[0,2], p_t[0,0]],
42                        [R_t[1,0], R_t[1,1], R_t[1,2], p_t[1,0]],
43                        [R_t[2,0], R_t[2,1], R_t[2,2], p_t[2,0]],
44                        [0,0,0,1]])
45    return(g_inverse)

```



In this system, frame{A} is the center of the Jack. The frame{F} is the center of the box. I assume in the initial condition the Jack is in the center of the box. So frame{A} and frame{F} are coincident temporarily.

In [3]:

```

1 # define Inertia matrix, assume rotational inertia of Jack is J = 1
2 I = Matrix([[m,0,0,0,0,0],
3             [0,m,0,0,0,0],
4             [0,0,m,0,0,0],
5             [0,0,0,1,0,0],
6             [0,0,0,0,1,0],
7             [0,0,0,0,0,1]
8           ])
9 # get transformation matrixes for jack
10 gwa = SE3(x2, y2, theta2)
11
12 gab = SE3(0, -0.5, 0)
13 gac = SE3(-0.5, 0, 0)
14
15 gad = SE3(0.5, 0, 0)
16 gae = SE3(0, 0.5, 0)

```

In [4]:

```
1 #calculate KE for {B} frame
2 gwb = gwa*gab
3 gwb_inv = inverse_SE3(gwb)
4 gwb_dot = gwb.diff(t)
5 Vb_hat = gwb_inv*gwb_dot
6 Vb_unhat = V_unhat(Vb_hat)
7 KE_b = 0.5*transpose(Vb_unhat)*I*Vb_unhat
8 #calculate PE for {B} frame
9 PE_b = Matrix([m*g*gwb[1,3]])
10
11 #calculate KE for {C} frame
12 gwc = gwa*gac
13 gwc_inv = inverse_SE3(gwc)
14 gwc_dot = gwc.diff(t)
15 Vc_hat = gwc_inv*gwc_dot
16 Vc_unhat = V_unhat(Vc_hat)
17 KE_c = 0.5*transpose(Vc_unhat)*I*Vc_unhat
18 #calculate PE for {C} frame
19 PE_c = Matrix([m*g*gwc[1,3]])
20
21 #calculate KE for {D} frame
22 gwd = gwa*gad
23 gwd_inv = inverse_SE3(gwd)
24 gwd_dot = gwd.diff(t)
25 Vd_hat = gwd_inv*gwd_dot
26 Vd_unhat = V_unhat(Vd_hat)
27 KE_d = 0.5*transpose(Vd_unhat)*I*Vd_unhat
28 #calculate PE for {D} frame
29 PE_d = Matrix([m*g*gwd[1,3]])
30
31 #calculate KE for {E} frame
32 gwe = gwa*gae
33 gwe_inv = inverse_SE3(gwe)
34 gwe_dot = gwe.diff(t)
35 Ve_hat = gwe_inv*gwe_dot
36 Ve_unhat = V_unhat(Ve_hat)
37 KE_e = 0.5*transpose(Ve_unhat)*I*Ve_unhat
38 #calculate PE for {E} frame
39 PE_e = Matrix([m*g*gwe[1,3]])
```

In [5]:

```
1 # define Inertia matrix, assume rotational inertia of Box is J = 1
2 I = Matrix([[m,0,0,0,0,0],
3             [0,m,0,0,0,0],
4             [0,0,m,0,0,0],
5             [0,0,0,1,0,0],
6             [0,0,0,0,1,0],
7             [0,0,0,0,0,1]
8           ])
9 # get transformation matrixes for Box
10 gwf = SE3(x1, y1, theta1)
11
12 gfg = SE3(0, -2, 0)
13 gfh = SE3(2, 0, 0)
14
15 gfi = SE3(0, 2, 0)
16 gfj = SE3(-2, 0, 0)
17
18 gfw = inverse_SE3(gwf)
```

In [6]:

```

1 #calculate KE for {G} frame
2 gwg = gwf*gfg
3 gwg_inv = inverse_SE3(gwg)
4 gwg_dot = gwg.diff(t)
5 Vg_hat = gwg_inv*gwg_dot
6 Vg_unhat = V_unhat(Vg_hat)
7 KE_g = 0.5*transpose(Vg_unhat)*I*Vg_unhat
8 #calculate PE for {G} frame
9 PE_g = Matrix([m*g*gwg[1,3]])
10
11 #calculate KE for {H} frame
12 gwh = gwf*gfh
13 gwh_inv = inverse_SE3(gwh)
14 gwh_dot = gwh.diff(t)
15 Vh_hat = gwh_inv*gwh_dot
16 Vh_unhat = V_unhat(Vh_hat)
17 KE_h = 0.5*transpose(Vh_unhat)*I*Vh_unhat
18 #calculate PE for {H} frame
19 PE_h = Matrix([m*g*gwh[1,3]])
20
21 #calculate KE for {I} frame
22 gwi = gwf*gfi
23 gwi_inv = inverse_SE3(gwi)
24 gwi_dot = gwi.diff(t)
25 Vi_hat = gwi_inv*gwi_dot
26 Vi_unhat = V_unhat(Vi_hat)
27 KE_i = 0.5*transpose(Vi_unhat)*I*Vi_unhat
28 #calculate PE for {I} frame
29 PE_i = Matrix([m*g*gwi[1,3]])
30
31 #calculate KE for {J} frame
32 gwj = gwf*gfj
33 gwj_inv = inverse_SE3(gwj)
34 gwj_dot = gwj.diff(t)
35 Vj_hat = gwj_inv*gwj_dot
36 Vj_unhat = V_unhat(Vj_hat)
37 KE_j = 0.5*transpose(Vj_unhat)*I*Vj_unhat
38 #calculate PE for {J} frame
39 PE_j = Matrix([m*g*gwj[1,3]])

```

In [7]:

```

1 #calculate Lagrangian equation
2 L = (KE_b+KE_c+KE_d+KE_e+KE_g+KE_h+KE_i+KE_j)-(PE_b+PE_c+PE_d+PE_e+PE_g+PE_h+PE_i+PE_j)
3 L = sym.simplify(L)

```

In [8]:

```

1 #define external force
2 Force1 = 8*m*g
3 Force2 = 0.5*m*g

```

In [9]:

```

1 # calculate Euler-Lagrangian equation
2 dLdq = L.jacobian(q)
3 dLdqdot = L.jacobian(qdot)
4 lhs = dLdqdot.diff(t) - dLdq
5 rhs = Matrix([Force2, Force1, 0, 0, 0, 0]).T
6 el_eqns = Eq(lhs, rhs)

```

In [10]:

```

1 # numerical evaluation
2 qddot_soln = solve(el_eqns, [qddot[0], qddot[1], qddot[2],
3                               qddot[3], qddot[4], qddot[5]], dict=True)[0]
4 x1ddot_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
5                             qdot[0], qdot[1], qdot[2], qdot[3], qdot[4], qdot[5]],
6                             qddot_soln[qddot[0]])
7 y1ddot_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
8                             qdot[0], qdot[1], qdot[2], qdot[3], qdot[4], qdot[5]],
9                             qddot_soln[qddot[1]])
10 theta1ddot_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
11                             qdot[0], qdot[1], qdot[2], qdot[3], qdot[4], qdot[5]],
12                             qddot_soln[qddot[2]])
13
14 x2ddot_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
15                             qdot[0], qdot[1], qdot[2], qdot[3], qdot[4], qdot[5]],
16                             qddot_soln[qddot[3]])
17 y2ddot_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
18                             qdot[0], qdot[1], qdot[2], qdot[3], qdot[4], qdot[5]],
19                             qddot_soln[qddot[4]])
20 theta2ddot_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
21                             qdot[0], qdot[1], qdot[2], qdot[3], qdot[4], qdot[5]],
22                             qddot_soln[qddot[5]])
23

```

**phi1, phi2, phi3, phi4: b ball impact on the Box**

In [11]:

```

1 #define impact constraint
2 gfb= gfw*gwb
3 phi1 = gfb[1,3]+2
4 phi1 = sym.simplify(phi1)
5
6 phi2 = gfb[1,3]-2
7 phi2 = sym.simplify(phi2)
8
9 phi3 = gfb[0,3]+2
10 phi3 = sym.simplify(phi3)
11
12 phi4 = gfb[0,3]-2
13 phi4 = sym.simplify(phi4)
14 # define dummy variables
15 x1_s, y1_s, theta1_s = symbols(r'x_1, y_1, theta_1')
16 x2_s, y2_s, theta2_s = symbols(r'x_2, y_2, theta_2')
17 x1dot, y1dot, theta1dot = symbols(r'\dot{x_1}, \dot{y_1}, \dot{\theta_1}')
18 x2dot, y2dot, theta2dot = symbols(r'\dot{x_2}, \dot{y_2}, \dot{\theta_2}')
19 x1ddot, y1ddot, theta1ddot = symbols(r'\ddot{x_1}, \ddot{y_1}, \ddot{\theta_1}')
20 x2ddot, y2ddot, theta2ddot = symbols(r'\ddot{x_2}, \ddot{y_2}, \ddot{\theta_2}')
21
22
23 dummy_dict = {q[0]:x1_s, q[1]:y1_s, q[2]:theta1_s, q[3]:x2_s, q[4]:y2_s, q[5]:theta2_s
24     qdot[0]:x1dot, qdot[1]:y1dot, qdot[2]:theta1dot,
25     qdot[3]:x2dot, qdot[4]:y2dot, qdot[5]:theta2dot,
26     qddot[0]:x1ddot, qddot[1]:y1ddot, qddot[2]:theta1ddot,
27     qddot[3]:x2ddot, qddot[4]:y2ddot, qddot[5]:theta2ddot,
28 }
29
30 phi1_Sym = sym.Matrix([phi1.subs(dummy_dict)])
31 phi2_Sym = sym.Matrix([phi2.subs(dummy_dict)])
32 phi3_Sym = sym.Matrix([phi3.subs(dummy_dict)])
33 phi4_Sym = sym.Matrix([phi4.subs(dummy_dict)])
34
35 # compute Hamiltonian
36 H = dLdqdot*qdot-L
37 H = sym.simplify(H)
38
39 #compute expression
40 dLdqdot_Sym = dLdqdot.subs(dummy_dict)
41
42 dphi1dq_Sym = phi1_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
43 dphi2dq_Sym = phi2_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
44 dphi3dq_Sym = phi3_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
45 dphi4dq_Sym = phi4_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
46
47 H_Sym = H.subs(dummy_dict)
48
49 # define dummy symbols for tau+
50 lamb = symbols(r'\lambda')
51 x1dotPlus, y1dotPlus, theta1dotPlus = \
52 symbols(r'\dot{x_1}^+, \dot{y_1}^+, \dot{\theta_1}^+')
53
54 x2dotPlus, y2dotPlus, theta2dotPlus = \
55 symbols(r'\dot{x_2}^+, \dot{y_2}^+, \dot{\theta_2}^+')
56
57 impact_dict = {x1dot:x1dotPlus, y1dot:y1dotPlus, theta1dot:theta1dotPlus,
58                 x2dot:x2dotPlus, y2dot:y2dotPlus, theta2dot:theta2dotPlus}
59

```

```

60 #evaluate the expression at tau+
61 dLdqdot_SymPlus = dLdqdot_Sym.subs(impact_dict)
62 dLdqdot_SymPlus = sym.simplify(dLdqdot_SymPlus)
63
64 dphi1dq_SymPlus = dphi1dq_Sym.subs(impact_dict)
65 dphi1dq_SymPlus = sym.simplify(dphi1dq_SymPlus)
66
67 dphi2dq_SymPlus = dphi2dq_Sym.subs(impact_dict)
68 dphi2dq_SymPlus = sym.simplify(dphi2dq_SymPlus)
69
70 dphi3dq_SymPlus = dphi3dq_Sym.subs(impact_dict)
71 dphi3dq_SymPlus = sym.simplify(dphi3dq_SymPlus)
72
73 dphi4dq_SymPlus = dphi4dq_Sym.subs(impact_dict)
74 dphi4dq_SymPlus = sym.simplify(dphi4dq_SymPlus)
75
76 H_SymPlus = H_Sym.subs(impact_dict)
77 H_SymPlus = sym.simplify(H_SymPlus)
78
79 # define equations
80 lhs = Matrix([dLdqdot_SymPlus[0]-dLdqdot_Sym[0],
81                 dLdqdot_SymPlus[1]-dLdqdot_Sym[1],
82                 dLdqdot_SymPlus[2]-dLdqdot_Sym[2],
83                 dLdqdot_SymPlus[3]-dLdqdot_Sym[3],
84                 dLdqdot_SymPlus[4]-dLdqdot_Sym[4],
85                 dLdqdot_SymPlus[5]-dLdqdot_Sym[5],
86                 H_SymPlus[0]-H_Sym[0]])
87
88 rhs1 = Matrix([lamb*dphi1dq_Sym[0],
89                 lamb*dphi1dq_Sym[1],
90                 lamb*dphi1dq_Sym[2],
91                 lamb*dphi1dq_Sym[3],
92                 lamb*dphi1dq_Sym[4],
93                 lamb*dphi1dq_Sym[5], 0])
94 impact_eqns1 = Eq(lhs, rhs1)
95 impact_eqns1 = sym.simplify(impact_eqns1)
96
97 rhs2 = Matrix([lamb*dphi2dq_Sym[0],
98                 lamb*dphi2dq_Sym[1],
99                 lamb*dphi2dq_Sym[2],
100                lamb*dphi2dq_Sym[3],
101                lamb*dphi2dq_Sym[4],
102                lamb*dphi2dq_Sym[5], 0])
103 impact_eqns2 = Eq(lhs, rhs2)
104 impact_eqns2 = sym.simplify(impact_eqns2)
105
106 rhs3 = Matrix([lamb*dphi3dq_Sym[0],
107                 lamb*dphi3dq_Sym[1],
108                 lamb*dphi3dq_Sym[2],
109                 lamb*dphi3dq_Sym[3],
110                 lamb*dphi3dq_Sym[4],
111                 lamb*dphi3dq_Sym[5], 0])
112 impact_eqns3 = Eq(lhs, rhs3)
113 impact_eqns3 = sym.simplify(impact_eqns3)
114
115 rhs4 = Matrix([lamb*dphi4dq_Sym[0],
116                 lamb*dphi4dq_Sym[1],
117                 lamb*dphi4dq_Sym[2],
118                 lamb*dphi4dq_Sym[3],
119                 lamb*dphi4dq_Sym[4],
120                 lamb*dphi4dq_Sym[5], 0])

```

```

121 impact_eqns4 = Eq(lhs, rhs4)
122 impact_eqns4 = sym.simplify(impact_eqns4)
123
124 #define impact update function
125 def impact_update1(s, impact_eqns, sym_list):
126     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s
127                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],
128                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}
129     new_impact_eqns = impact_eqns.subs(subs_dict)
130     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,
131                                                 x2dotPlus, y2dotPlus, theta2dotPlus,
132                                                 lamb], dict=True)
133     if len(impact_solns) == 1:
134         print("Damn only one solution ...")
135     else:
136         for sol in impact_solns:
137             lamb_sol = sol[lamb]
138             if abs(lamb_sol) < 1e-06:
139                 pass
140             else:
141                 return np.array([
142                     s[0],
143                     s[1],
144                     s[2],
145                     s[3],
146                     s[4],
147                     s[5],
148                     float(sym.N(sol[sym_list[0]]))),
149                     float(sym.N(sol[sym_list[1]]))),
150                     float(sym.N(sol[sym_list[2]]))),
151                     float(sym.N(sol[sym_list[3]]))),
152                     float(sym.N(sol[sym_list[4]]))),
153                     float(sym.N(sol[sym_list[5]]))),
154                 ])
155
156 def impact_update2(s, impact_eqns, sym_list):
157     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s
158                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],
159                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}
160     new_impact_eqns = impact_eqns.subs(subs_dict)
161     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,
162                                                 x2dotPlus, y2dotPlus, theta2dotPlus,
163                                                 lamb], dict=True)
164     if len(impact_solns) == 1:
165         print("Damn only one solution ...")
166     else:
167         for sol in impact_solns:
168             lamb_sol = sol[lamb]
169             if abs(lamb_sol) < 1e-06:
170                 pass
171             else:
172                 return np.array([
173                     s[0],
174                     s[1],
175                     s[2],
176                     s[3],
177                     s[4],
178                     s[5],
179                     float(sym.N(sol[sym_list[0]]))),
180                     float(sym.N(sol[sym_list[1]]))),
181                     float(sym.N(sol[sym_list[2]]))),

```

```
182         float(sym.N(sol[sym_list[3]])),  
183         float(sym.N(sol[sym_list[4]])),  
184         float(sym.N(sol[sym_list[5]])),  
185     ])  
186  
187 def impact_update3(s, impact_eqns, sym_list):  
188     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s  
189                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],  
190                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}  
191     new_impact_eqns = impact_eqns.subs(subs_dict)  
192     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,  
193                                         x2dotPlus, y2dotPlus, theta2dotPlus,  
194                                         lamb], dict=True)  
195     if len(impact_solns) == 1:  
196         print("Damn only one solution ...")  
197     else:  
198         for sol in impact_solns:  
199             lamb_sol = sol[lamb]  
200             if abs(lamb_sol) < 1e-06:  
201                 pass  
202             else:  
203                 return np.array([  
204                     s[0],  
205                     s[1],  
206                     s[2],  
207                     s[3],  
208                     s[4],  
209                     s[5],  
210                     float(sym.N(sol[sym_list[0]])),  
211                     float(sym.N(sol[sym_list[1]])),  
212                     float(sym.N(sol[sym_list[2]])),  
213                     float(sym.N(sol[sym_list[3]])),  
214                     float(sym.N(sol[sym_list[4]])),  
215                     float(sym.N(sol[sym_list[5]]))),  
216                 ])  
217  
218 def impact_update4(s, impact_eqns, sym_list):  
219     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s  
220                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],  
221                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}  
222     new_impact_eqns = impact_eqns.subs(subs_dict)  
223     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,  
224                                         x2dotPlus, y2dotPlus, theta2dotPlus,  
225                                         lamb], dict=True)  
226     if len(impact_solns) == 1:  
227         print("Damn only one solution ...")  
228     else:  
229         for sol in impact_solns:  
230             lamb_sol = sol[lamb]  
231             if abs(lamb_sol) < 1e-06:  
232                 pass  
233             else:  
234                 return np.array([  
235                     s[0],  
236                     s[1],  
237                     s[2],  
238                     s[3],  
239                     s[4],  
240                     s[5],  
241                     float(sym.N(sol[sym_list[0]])),  
242                     float(sym.N(sol[sym_list[1]]))),
```

```
243     float(sym.N(sol[sym_list[2]])),  
244     float(sym.N(sol[sym_list[3]])),  
245     float(sym.N(sol[sym_list[4]])),  
246     float(sym.N(sol[sym_list[5]])),  
247     ])  
248 s_test1 = np.array([0, 0, 0, 3, 3, 0.2, 0, 0.1, 0, 0, -0.1, 0])  
249 display(impact_update1(s_test1, impact_eqns1, [x1dotPlus, y1dotPlus, theta1dotPlus,  
250                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
251 display(impact_update2(s_test1, impact_eqns2, [x1dotPlus, y1dotPlus, theta1dotPlus,  
252                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
253 display(impact_update3(s_test1, impact_eqns3, [x1dotPlus, y1dotPlus, theta1dotPlus,  
254                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
255 display(impact_update4(s_test1, impact_eqns4, [x1dotPlus, y1dotPlus, theta1dotPlus,  
256                                     x2dotPlus, y2dotPlus, theta2dotPlus]))
```

```
array([ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  3.00000000e+00,  
       3.00000000e+00,  2.00000000e-01,  0.00000000e+00, -1.80528838e-03,  
      -6.31057319e-02,  0.00000000e+00,  1.80528838e-03,  8.09023541e-03])
```

```
array([ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  3.00000000e+00,  
       3.00000000e+00,  2.00000000e-01,  0.00000000e+00, -1.80528838e-03,  
      -6.31057319e-02,  0.00000000e+00,  1.80528838e-03,  8.09023541e-03])
```

Damn only one solution ...

None

Damn only one solution ...

None

## phi5,phi6,phi7,phi8: C ball impact on the Box

In [12]:

```

1 #define impact constraint
2 gfc= gfw*gwc
3 phi5 = gfc[1,3]+2
4 phi5 = sym.simplify(phi5)
5
6 phi6 = gfc[1,3]-2
7 phi6 = sym.simplify(phi6)
8
9 phi7 = gfc[0,3]+2
10 phi7 = sym.simplify(phi7)
11
12 phi8 = gfc[0,3]-2
13 phi8 = sym.simplify(phi8)
14 # define dummy variables
15 x1_s, y1_s, theta1_s = symbols(r'x_1, y_1, theta_1')
16 x2_s, y2_s, theta2_s = symbols(r'x_2, y_2, theta_2')
17 x1dot, y1dot, theta1dot = symbols(r'\dot{x_1}, \dot{y_1}, \dot{\theta_1}')
18 x2dot, y2dot, theta2dot = symbols(r'\dot{x_2}, \dot{y_2}, \dot{\theta_2}')
19 x1ddot, y1ddot, theta1ddot = symbols(r'\ddot{x_1}, \ddot{y_1}, \ddot{\theta_1}')
20 x2ddot, y2ddot, theta2ddot = symbols(r'\ddot{x_2}, \ddot{y_2}, \ddot{\theta_2}')
21
22
23 dummy_dict = {q[0]:x1_s, q[1]:y1_s, q[2]:theta1_s, q[3]:x2_s, q[4]:y2_s, q[5]:theta2_s
24     qdot[0]:x1dot, qdot[1]:y1dot, qdot[2]:theta1dot,
25     qdot[3]:x2dot, qdot[4]:y2dot, qdot[5]:theta2dot,
26     qddot[0]:x1ddot, qddot[1]:y1ddot, qddot[2]:theta1ddot,
27     qddot[3]:x2ddot, qddot[4]:y2ddot, qddot[5]:theta2ddot,
28 }
29
30 phi5_Sym = sym.Matrix([phi5.subs(dummy_dict)])
31 phi6_Sym = sym.Matrix([phi6.subs(dummy_dict)])
32 phi7_Sym = sym.Matrix([phi7.subs(dummy_dict)])
33 phi8_Sym = sym.Matrix([phi8.subs(dummy_dict)])
34
35 # compute Hamiltonian
36 H = dLdqdot*qdot-L
37 H = sym.simplify(H)
38
39 #compute expression
40 dLdqdot_Sym = dLdqdot.subs(dummy_dict)
41
42 dphi5dq_Sym = phi5_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
43 dphi6dq_Sym = phi6_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
44 dphi7dq_Sym = phi7_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
45 dphi8dq_Sym = phi8_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
46
47 H_Sym = H.subs(dummy_dict)
48
49 # define dummy symbols for tau+
50 lamb = symbols(r'\lambda')
51 x1dotPlus, y1dotPlus, theta1dotPlus = \
52 symbols(r'\dot{x_1}^+, \dot{y_1}^+, \dot{\theta_1}^+')
53
54 x2dotPlus, y2dotPlus, theta2dotPlus = \
55 symbols(r'\dot{x_2}^+, \dot{y_2}^+, \dot{\theta_2}^+')
56
57 impact_dict = {x1dot:x1dotPlus, y1dot:y1dotPlus, theta1dot:theta1dotPlus,
58                 x2dot:x2dotPlus, y2dot:y2dotPlus, theta2dot:theta2dotPlus}
59

```

```

60 #evaluate the expression at tau+
61 dLdqdot_SymPlus = dLdqdot_Sym.subs(impact_dict)
62 dLdqdot_SymPlus = sym.simplify(dLdqdot_SymPlus)
63
64 dphi5dq_SymPlus = dphi5dq_Sym.subs(impact_dict)
65 dphi5dq_SymPlus = sym.simplify(dphi5dq_SymPlus)
66
67 dphi6dq_SymPlus = dphi6dq_Sym.subs(impact_dict)
68 dphi6dq_SymPlus = sym.simplify(dphi6dq_SymPlus)
69
70 dphi7dq_SymPlus = dphi7dq_Sym.subs(impact_dict)
71 dphi7dq_SymPlus = sym.simplify(dphi7dq_SymPlus)
72
73 dphi8dq_SymPlus = dphi8dq_Sym.subs(impact_dict)
74 dphi8dq_SymPlus = sym.simplify(dphi8dq_SymPlus)
75
76 H_SymPlus = H_Sym.subs(impact_dict)
77 H_SymPlus = sym.simplify(H_SymPlus)
78
79 # define equations
80 lhs = Matrix([dLdqdot_SymPlus[0]-dLdqdot_Sym[0],
81                 dLdqdot_SymPlus[1]-dLdqdot_Sym[1],
82                 dLdqdot_SymPlus[2]-dLdqdot_Sym[2],
83                 dLdqdot_SymPlus[3]-dLdqdot_Sym[3],
84                 dLdqdot_SymPlus[4]-dLdqdot_Sym[4],
85                 dLdqdot_SymPlus[5]-dLdqdot_Sym[5],
86                 H_SymPlus[0]-H_Sym[0]])
87
88 rhs5 = Matrix([lamb*dphi5dq_Sym[0],
89                 lamb*dphi5dq_Sym[1],
90                 lamb*dphi5dq_Sym[2],
91                 lamb*dphi5dq_Sym[3],
92                 lamb*dphi5dq_Sym[4],
93                 lamb*dphi5dq_Sym[5], 0])
94 impact_eqns5 = Eq(lhs, rhs5)
95 impact_eqns5 = sym.simplify(impact_eqns5)
96
97 rhs6 = Matrix([lamb*dphi6dq_Sym[0],
98                 lamb*dphi6dq_Sym[1],
99                 lamb*dphi6dq_Sym[2],
100                lamb*dphi6dq_Sym[3],
101                lamb*dphi6dq_Sym[4],
102                lamb*dphi6dq_Sym[5], 0])
103 impact_eqns6 = Eq(lhs, rhs6)
104 impact_eqns6 = sym.simplify(impact_eqns6)
105
106 rhs7 = Matrix([lamb*dphi7dq_Sym[0],
107                 lamb*dphi7dq_Sym[1],
108                 lamb*dphi7dq_Sym[2],
109                 lamb*dphi7dq_Sym[3],
110                 lamb*dphi7dq_Sym[4],
111                 lamb*dphi7dq_Sym[5], 0])
112 impact_eqns7 = Eq(lhs, rhs7)
113 impact_eqns7 = sym.simplify(impact_eqns7)
114
115 rhs8 = Matrix([lamb*dphi8dq_Sym[0],
116                 lamb*dphi8dq_Sym[1],
117                 lamb*dphi8dq_Sym[2],
118                 lamb*dphi8dq_Sym[3],
119                 lamb*dphi8dq_Sym[4],
120                 lamb*dphi8dq_Sym[5], 0])

```

```

121 impact_eqns8 = Eq(lhs, rhs8)
122 impact_eqns8 = sym.simplify(impact_eqns8)
123
124 #define impact update function
125 def impact_update5(s, impact_eqns, sym_list):
126     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s
127                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],
128                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}
129     new_impact_eqns = impact_eqns.subs(subs_dict)
130     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,
131                                                 x2dotPlus, y2dotPlus, theta2dotPlus,
132                                                 lamb], dict=True)
133     if len(impact_solns) == 1:
134         print("Damn only one solution ...")
135     else:
136         for sol in impact_solns:
137             lamb_sol = sol[lamb]
138             if abs(lamb_sol) < 1e-06:
139                 pass
140             else:
141                 return np.array([
142                     s[0],
143                     s[1],
144                     s[2],
145                     s[3],
146                     s[4],
147                     s[5],
148                     float(sym.N(sol[sym_list[0]])),
149                     float(sym.N(sol[sym_list[1]])),
150                     float(sym.N(sol[sym_list[2]])),
151                     float(sym.N(sol[sym_list[3]])),
152                     float(sym.N(sol[sym_list[4]])),
153                     float(sym.N(sol[sym_list[5]])),
154                 ])
155
156 def impact_update6(s, impact_eqns, sym_list):
157     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s
158                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],
159                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}
160     new_impact_eqns = impact_eqns.subs(subs_dict)
161     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,
162                                                 x2dotPlus, y2dotPlus, theta2dotPlus,
163                                                 lamb], dict=True)
164     if len(impact_solns) == 1:
165         print("Damn only one solution ...")
166     else:
167         for sol in impact_solns:
168             lamb_sol = sol[lamb]
169             if abs(lamb_sol) < 1e-06:
170                 pass
171             else:
172                 return np.array([
173                     s[0],
174                     s[1],
175                     s[2],
176                     s[3],
177                     s[4],
178                     s[5],
179                     float(sym.N(sol[sym_list[0]])),
180                     float(sym.N(sol[sym_list[1]])),
181                     float(sym.N(sol[sym_list[2]])),

```

```
182         float(sym.N(sol[sym_list[3]])),  
183         float(sym.N(sol[sym_list[4]])),  
184         float(sym.N(sol[sym_list[5]])),  
185     ])  
186  
187 def impact_update7(s, impact_eqns, sym_list):  
188     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s  
189                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],  
190                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}  
191     new_impact_eqns = impact_eqns.subs(subs_dict)  
192     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,  
193                                         x2dotPlus, y2dotPlus, theta2dotPlus,  
194                                         lamb], dict=True)  
195     if len(impact_solns) == 1:  
196         print("Damn only one solution ...")  
197     else:  
198         for sol in impact_solns:  
199             lamb_sol = sol[lamb]  
200             if abs(lamb_sol) < 1e-06:  
201                 pass  
202             else:  
203                 return np.array([  
204                     s[0],  
205                     s[1],  
206                     s[2],  
207                     s[3],  
208                     s[4],  
209                     s[5],  
210                     float(sym.N(sol[sym_list[0]])),  
211                     float(sym.N(sol[sym_list[1]])),  
212                     float(sym.N(sol[sym_list[2]])),  
213                     float(sym.N(sol[sym_list[3]])),  
214                     float(sym.N(sol[sym_list[4]])),  
215                     float(sym.N(sol[sym_list[5]]))),  
216                 ])  
217  
218 def impact_update8(s, impact_eqns, sym_list):  
219     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s  
220                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],  
221                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}  
222     new_impact_eqns = impact_eqns.subs(subs_dict)  
223     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,  
224                                         x2dotPlus, y2dotPlus, theta2dotPlus,  
225                                         lamb], dict=True)  
226     if len(impact_solns) == 1:  
227         print("Damn only one solution ...")  
228     else:  
229         for sol in impact_solns:  
230             lamb_sol = sol[lamb]  
231             if abs(lamb_sol) < 1e-06:  
232                 pass  
233             else:  
234                 return np.array([  
235                     s[0],  
236                     s[1],  
237                     s[2],  
238                     s[3],  
239                     s[4],  
240                     s[5],  
241                     float(sym.N(sol[sym_list[0]])),  
242                     float(sym.N(sol[sym_list[1]]))),
```

```
243     float(sym.N(sol[sym_list[2]])),  
244     float(sym.N(sol[sym_list[3]])),  
245     float(sym.N(sol[sym_list[4]])),  
246     float(sym.N(sol[sym_list[5]])),  
247     ])  
248 s_test1 = np.array([0, 0, 0, 3, 0.2, 0, 0.1, 0, 0, -0.1, 0])  
249 display(impact_update5(s_test1, impact_eqns5, [x1dotPlus, y1dotPlus, theta1dotPlus,  
250                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
251 display(impact_update6(s_test1, impact_eqns6, [x1dotPlus, y1dotPlus, theta1dotPlus,  
252                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
253 display(impact_update7(s_test1, impact_eqns7, [x1dotPlus, y1dotPlus, theta1dotPlus,  
254                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
255 display(impact_update8(s_test1, impact_eqns8, [x1dotPlus, y1dotPlus, theta1dotPlus,  
256                                     x2dotPlus, y2dotPlus, theta2dotPlus]))
```

```
array([ 0.          ,  0.          ,  0.          ,  3.          ,  3.          ,  
       0.2         ,  0.          , -0.01587174, -0.05816684,  0.          ,  
      0.01587174, -0.04542481])  
  
array([ 0.          ,  0.          ,  0.          ,  3.          ,  3.          ,  
       0.2         ,  0.          , -0.01587174, -0.05816684,  0.          ,  
      0.01587174, -0.04542481])
```

Damn only one solution ...

None

Damn only one solution ...

None

## phi9,phi10,phi11,phi12: d ball impact on the Box

In [13]:

```

1 #define impact constraint
2 gfd= gfw*gwd
3 phi9 = gfd[1,3]+2
4 phi9 = sym.simplify(phi9)
5
6 phi10 = gfd[1,3]-2
7 phi10 = sym.simplify(phi10)
8
9 phi11 = gfd[0,3]+2
10 phi11 = sym.simplify(phi11)
11
12 phi12 = gfd[0,3]-2
13 phi12 = sym.simplify(phi12)
14 # define dummy variables
15 x1_s, y1_s, theta1_s = symbols(r'x_1, y_1, theta_1')
16 x2_s, y2_s, theta2_s = symbols(r'x_2, y_2, theta_2')
17 x1dot, y1dot, theta1dot = symbols(r'\dot{x_1}, \dot{y_1}, \dot{\theta_1}')
18 x2dot, y2dot, theta2dot = symbols(r'\dot{x_2}, \dot{y_2}, \dot{\theta_2}')
19 x1ddot, y1ddot, theta1ddot = symbols(r'\ddot{x_1}, \ddot{y_1}, \ddot{\theta_1}')
20 x2ddot, y2ddot, theta2ddot = symbols(r'\ddot{x_2}, \ddot{y_2}, \ddot{\theta_2}')
21
22
23 dummy_dict = {q[0]:x1_s, q[1]:y1_s, q[2]:theta1_s, q[3]:x2_s, q[4]:y2_s, q[5]:theta2_s
24     qdot[0]:x1dot, qdot[1]:y1dot, qdot[2]:theta1dot,
25     qdot[3]:x2dot, qdot[4]:y2dot, qdot[5]:theta2dot,
26     qddot[0]:x1ddot, qddot[1]:y1ddot, qddot[2]:theta1ddot,
27     qddot[3]:x2ddot, qddot[4]:y2ddot, qddot[5]:theta2ddot,
28 }
29
30 phi9_Sym = sym.Matrix([phi9.subs(dummy_dict)])
31 phi10_Sym = sym.Matrix([phi10.subs(dummy_dict)])
32 phi11_Sym = sym.Matrix([phi11.subs(dummy_dict)])
33 phi12_Sym = sym.Matrix([phi12.subs(dummy_dict)])
34
35 # compute Hamiltonian
36 H = dLdqdot*qdot-L
37 H = sym.simplify(H)
38
39 #compute expression
40 dLdqdot_Sym = dLdqdot.subs(dummy_dict)
41
42 dphi9dq_Sym = phi9_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
43 dphi10dq_Sym = phi10_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
44 dphi11dq_Sym = phi11_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
45 dphi12dq_Sym = phi12_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
46
47 H_Sym = H.subs(dummy_dict)
48
49 # define dummy symbols for tau+
50 lamb = symbols(r'\lambda')
51 x1dotPlus, y1dotPlus, theta1dotPlus = \
52 symbols(r'\dot{x_1}^+, \dot{y_1}^+, \dot{\theta_1}^+')
53
54 x2dotPlus, y2dotPlus, theta2dotPlus = \
55 symbols(r'\dot{x_2}^+, \dot{y_2}^+, \dot{\theta_2}^+')
56
57 impact_dict = {x1dot:x1dotPlus, y1dot:y1dotPlus, theta1dot:theta1dotPlus,
58                 x2dot:x2dotPlus, y2dot:y2dotPlus, theta2dot:theta2dotPlus}
59

```

```

60 #evaluate the expression at tau+
61 dLdqdot_SymPlus = dLdqdot_Sym.subs(impact_dict)
62 dLdqdot_SymPlus = sym.simplify(dLdqdot_SymPlus)
63
64 dphi9dq_SymPlus = dphi9dq_Sym.subs(impact_dict)
65 dphi9dq_SymPlus = sym.simplify(dphi9dq_SymPlus)
66
67 dphi10dq_SymPlus = dphi10dq_Sym.subs(impact_dict)
68 dphi10dq_SymPlus = sym.simplify(dphi10dq_SymPlus)
69
70 dphi11dq_SymPlus = dphi11dq_Sym.subs(impact_dict)
71 dphi11dq_SymPlus = sym.simplify(dphi11dq_SymPlus)
72
73 dphi12dq_SymPlus = dphi12dq_Sym.subs(impact_dict)
74 dphi12dq_SymPlus = sym.simplify(dphi12dq_SymPlus)
75
76 H_SymPlus = H_Sym.subs(impact_dict)
77 H_SymPlus = sym.simplify(H_SymPlus)
78
79 # define equations
80 lhs = Matrix([dLdqdot_SymPlus[0]-dLdqdot_Sym[0],
81                 dLdqdot_SymPlus[1]-dLdqdot_Sym[1],
82                 dLdqdot_SymPlus[2]-dLdqdot_Sym[2],
83                 dLdqdot_SymPlus[3]-dLdqdot_Sym[3],
84                 dLdqdot_SymPlus[4]-dLdqdot_Sym[4],
85                 dLdqdot_SymPlus[5]-dLdqdot_Sym[5],
86                 H_SymPlus[0]-H_Sym[0]])
87
88 rhs9 = Matrix([lamb*dphi9dq_Sym[0],
89                 lamb*dphi9dq_Sym[1],
90                 lamb*dphi9dq_Sym[2],
91                 lamb*dphi9dq_Sym[3],
92                 lamb*dphi9dq_Sym[4],
93                 lamb*dphi9dq_Sym[5], 0])
94 impact_eqns9 = Eq(lhs, rhs9)
95 impact_eqns9 = sym.simplify(impact_eqns9)
96
97 rhs10 = Matrix([lamb*dphi10dq_Sym[0],
98                 lamb*dphi10dq_Sym[1],
99                 lamb*dphi10dq_Sym[2],
100                lamb*dphi10dq_Sym[3],
101                lamb*dphi10dq_Sym[4],
102                lamb*dphi10dq_Sym[5], 0])
103 impact_eqns10 = Eq(lhs, rhs10)
104 impact_eqns10 = sym.simplify(impact_eqns10)
105
106 rhs11 = Matrix([lamb*dphi11dq_Sym[0],
107                 lamb*dphi11dq_Sym[1],
108                 lamb*dphi11dq_Sym[2],
109                 lamb*dphi11dq_Sym[3],
110                 lamb*dphi11dq_Sym[4],
111                 lamb*dphi11dq_Sym[5], 0])
112 impact_eqns11 = Eq(lhs, rhs11)
113 impact_eqns11 = sym.simplify(impact_eqns11)
114
115 rhs12 = Matrix([lamb*dphi12dq_Sym[0],
116                 lamb*dphi12dq_Sym[1],
117                 lamb*dphi12dq_Sym[2],
118                 lamb*dphi12dq_Sym[3],
119                 lamb*dphi12dq_Sym[4],
120                 lamb*dphi12dq_Sym[5], 0])

```

```

121 impact_eqns12 = Eq(lhs, rhs12)
122 impact_eqns12 = sym.simplify(impact_eqns12)
123
124 #define impact update function
125 def impact_update9(s, impact_eqns, sym_list):
126     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s
127                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],
128                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}
129     new_impact_eqns = impact_eqns.subs(subs_dict)
130     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,
131                                                 x2dotPlus, y2dotPlus, theta2dotPlus,
132                                                 lamb], dict=True)
133     if len(impact_solns) == 1:
134         print("Damn only one solution ...")
135     else:
136         for sol in impact_solns:
137             lamb_sol = sol[lamb]
138             if abs(lamb_sol) < 1e-06:
139                 pass
140             else:
141                 return np.array([
142                     s[0],
143                     s[1],
144                     s[2],
145                     s[3],
146                     s[4],
147                     s[5],
148                     float(sym.N(sol[sym_list[0]])),
149                     float(sym.N(sol[sym_list[1]])),
150                     float(sym.N(sol[sym_list[2]])),
151                     float(sym.N(sol[sym_list[3]])),
152                     float(sym.N(sol[sym_list[4]])),
153                     float(sym.N(sol[sym_list[5]])),
154                 ])
155
156 def impact_update10(s, impact_eqns, sym_list):
157     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s
158                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],
159                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}
160     new_impact_eqns = impact_eqns.subs(subs_dict)
161     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,
162                                                 x2dotPlus, y2dotPlus, theta2dotPlus,
163                                                 lamb], dict=True)
164     if len(impact_solns) == 1:
165         print("Damn only one solution ...")
166     else:
167         for sol in impact_solns:
168             lamb_sol = sol[lamb]
169             if abs(lamb_sol) < 1e-06:
170                 pass
171             else:
172                 return np.array([
173                     s[0],
174                     s[1],
175                     s[2],
176                     s[3],
177                     s[4],
178                     s[5],
179                     float(sym.N(sol[sym_list[0]])),
180                     float(sym.N(sol[sym_list[1]])),
181                     float(sym.N(sol[sym_list[2]])),

```

```
182         float(sym.N(sol[sym_list[3]])),  
183         float(sym.N(sol[sym_list[4]])),  
184         float(sym.N(sol[sym_list[5]])),  
185     ])  
186  
187 def impact_update11(s, impact_eqns, sym_list):  
188     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s  
189                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],  
190                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}  
191     new_impact_eqns = impact_eqns.subs(subs_dict)  
192     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,  
193                                         x2dotPlus, y2dotPlus, theta2dotPlus,  
194                                         lamb], dict=True)  
195     if len(impact_solns) == 1:  
196         print("Damn only one solution ...")  
197     else:  
198         for sol in impact_solns:  
199             lamb_sol = sol[lamb]  
200             if abs(lamb_sol) < 1e-06:  
201                 pass  
202             else:  
203                 return np.array([  
204                     s[0],  
205                     s[1],  
206                     s[2],  
207                     s[3],  
208                     s[4],  
209                     s[5],  
210                     float(sym.N(sol[sym_list[0]])),  
211                     float(sym.N(sol[sym_list[1]])),  
212                     float(sym.N(sol[sym_list[2]])),  
213                     float(sym.N(sol[sym_list[3]])),  
214                     float(sym.N(sol[sym_list[4]])),  
215                     float(sym.N(sol[sym_list[5]]))),  
216                 ])  
217  
218 def impact_update12(s, impact_eqns, sym_list):  
219     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s  
220                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],  
221                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}  
222     new_impact_eqns = impact_eqns.subs(subs_dict)  
223     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,  
224                                         x2dotPlus, y2dotPlus, theta2dotPlus,  
225                                         lamb], dict=True)  
226     if len(impact_solns) == 1:  
227         print("Damn only one solution ...")  
228     else:  
229         for sol in impact_solns:  
230             lamb_sol = sol[lamb]  
231             if abs(lamb_sol) < 1e-06:  
232                 pass  
233             else:  
234                 return np.array([  
235                     s[0],  
236                     s[1],  
237                     s[2],  
238                     s[3],  
239                     s[4],  
240                     s[5],  
241                     float(sym.N(sol[sym_list[0]])),  
242                     float(sym.N(sol[sym_list[1]]))),
```

```
243     float(sym.N(sol[sym_list[2]])),  
244     float(sym.N(sol[sym_list[3]])),  
245     float(sym.N(sol[sym_list[4]])),  
246     float(sym.N(sol[sym_list[5]])),  
247     ])  
248 s_test1 = np.array([0, 0, 0, 3, 0.2, 0, 0.1, 0, 0, -0.1, 0])  
249 display(impact_update9(s_test1, impact_eqns9, [x1dotPlus, y1dotPlus, theta1dotPlus,  
250                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
251 display(impact_update10(s_test1, impact_eqns10, [x1dotPlus, y1dotPlus, theta1dotPlus,  
252                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
253 display(impact_update11(s_test1, impact_eqns11, [x1dotPlus, y1dotPlus, theta1dotPlus,  
254                                     x2dotPlus, y2dotPlus, theta2dotPlus]))  
255 display(impact_update12(s_test1, impact_eqns12, [x1dotPlus, y1dotPlus, theta1dotPlus,  
256                                     x2dotPlus, y2dotPlus, theta2dotPlus]))
```

```
array([ 0.          ,  0.          ,  0.          ,  3.          ,  3.          ,  
       0.2         ,  0.          ,  0.0135728 , -0.06032676,  0.          ,  
      -0.0135728 ,  0.03388176])
```

```
array([ 0.          ,  0.          ,  0.          ,  3.          ,  3.          ,  
       0.2         ,  0.          ,  0.0135728 , -0.06032676,  0.          ,  
      -0.0135728 ,  0.03388176])
```

Damn only one solution ...

None

Damn only one solution ...

None

## phi13,phi14,phi15,phi16: e ball impact on the Box

In [14]:

```

1 #define impact constraint
2 gfe= gfw*gwe
3 phi13 = gfe[1,3]+2
4 phi13 = sym.simplify(phi13)
5
6 phi14 = gfe[1,3]-2
7 phi14 = sym.simplify(phi14)
8
9 phi15 = gfe[0,3]+2
10 phi15 = sym.simplify(phi15)
11
12 phi16 = gfe[0,3]-2
13 phi16 = sym.simplify(phi16)
14 # define dummy variables
15 x1_s, y1_s, theta1_s = symbols(r'x_1, y_1, theta_1')
16 x2_s, y2_s, theta2_s = symbols(r'x_2, y_2, theta_2')
17 x1dot, y1dot, theta1dot = symbols(r'\dot{x_1}, \dot{y_1}, \dot{\theta_1}')
18 x2dot, y2dot, theta2dot = symbols(r'\dot{x_2}, \dot{y_2}, \dot{\theta_2}')
19 x1ddot, y1ddot, theta1ddot = symbols(r'\ddot{x_1}, \ddot{y_1}, \ddot{\theta_1}')
20 x2ddot, y2ddot, theta2ddot = symbols(r'\ddot{x_2}, \ddot{y_2}, \ddot{\theta_2}')
21
22
23 dummy_dict = {q[0]:x1_s, q[1]:y1_s, q[2]:theta1_s, q[3]:x2_s, q[4]:y2_s, q[5]:theta2_s
24     qdot[0]:x1dot, qdot[1]:y1dot, qdot[2]:theta1dot,
25     qdot[3]:x2dot, qdot[4]:y2dot, qdot[5]:theta2dot,
26     qddot[0]:x1ddot, qddot[1]:y1ddot, qddot[2]:theta1ddot,
27     qddot[3]:x2ddot, qddot[4]:y2ddot, qddot[5]:theta2ddot,
28 }
29
30 phi13_Sym = sym.Matrix([phi13.subs(dummy_dict)])
31 phi14_Sym = sym.Matrix([phi14.subs(dummy_dict)])
32 phi15_Sym = sym.Matrix([phi15.subs(dummy_dict)])
33 phi16_Sym = sym.Matrix([phi16.subs(dummy_dict)])
34
35 # compute Hamiltonian
36 H = dLdqdot*qdot-L
37 H = sym.simplify(H)
38
39 #compute expression
40 dLdqdot_Sym = dLdqdot.subs(dummy_dict)
41
42 dphi13dq_Sym = phi13_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
43 dphi14dq_Sym = phi14_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
44 dphi15dq_Sym = phi15_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
45 dphi16dq_Sym = phi16_Sym.jacobian([x1_s, y1_s, theta1_s, x2_s, y2_s, theta2_s])
46
47 H_Sym = H.subs(dummy_dict)
48
49 # define dummy symbols for tau+
50 lamb = symbols(r'\lambda')
51 x1dotPlus, y1dotPlus, theta1dotPlus = \
52 symbols(r'\dot{x_1}^+, \dot{y_1}^+, \dot{\theta_1}^+')
53
54 x2dotPlus, y2dotPlus, theta2dotPlus = \
55 symbols(r'\dot{x_2}^+, \dot{y_2}^+, \dot{\theta_2}^+')
56
57 impact_dict = {x1dot:x1dotPlus, y1dot:y1dotPlus, theta1dot:theta1dotPlus,
58                 x2dot:x2dotPlus, y2dot:y2dotPlus, theta2dot:theta2dotPlus}
59

```

```

60 #evaluate the expression at tau+
61 dLdqdot_SymPlus = dLdqdot_Sym.subs(impact_dict)
62 dLdqdot_SymPlus = sym.simplify(dLdqdot_SymPlus)
63
64 dphi13dq_SymPlus = dphi13dq_Sym.subs(impact_dict)
65 dphi13dq_SymPlus = sym.simplify(dphi13dq_SymPlus)
66
67 dphi14dq_SymPlus = dphi14dq_Sym.subs(impact_dict)
68 dphi14dq_SymPlus = sym.simplify(dphi14dq_SymPlus)
69
70 dphi15dq_SymPlus = dphi15dq_Sym.subs(impact_dict)
71 dphi15dq_SymPlus = sym.simplify(dphi15dq_SymPlus)
72
73 dphi16dq_SymPlus = dphi16dq_Sym.subs(impact_dict)
74 dphi16dq_SymPlus = sym.simplify(dphi16dq_SymPlus)
75
76 H_SymPlus = H_Sym.subs(impact_dict)
77 H_SymPlus = sym.simplify(H_SymPlus)
78
79 # define equations
80 lhs = Matrix([dLdqdot_SymPlus[0]-dLdqdot_Sym[0],
81                 dLdqdot_SymPlus[1]-dLdqdot_Sym[1],
82                 dLdqdot_SymPlus[2]-dLdqdot_Sym[2],
83                 dLdqdot_SymPlus[3]-dLdqdot_Sym[3],
84                 dLdqdot_SymPlus[4]-dLdqdot_Sym[4],
85                 dLdqdot_SymPlus[5]-dLdqdot_Sym[5],
86                 H_SymPlus[0]-H_Sym[0]])
87
88 rhs13 = Matrix([lamb*dphi13dq_Sym[0],
89                  lamb*dphi13dq_Sym[1],
90                  lamb*dphi13dq_Sym[2],
91                  lamb*dphi13dq_Sym[3],
92                  lamb*dphi13dq_Sym[4],
93                  lamb*dphi13dq_Sym[5], 0])
94 impact_eqns13 = Eq(lhs, rhs13)
95 impact_eqns13 = sym.simplify(impact_eqns13)
96
97 rhs14 = Matrix([lamb*dphi14dq_Sym[0],
98                  lamb*dphi14dq_Sym[1],
99                  lamb*dphi14dq_Sym[2],
100                 lamb*dphi14dq_Sym[3],
101                 lamb*dphi14dq_Sym[4],
102                 lamb*dphi14dq_Sym[5], 0])
103 impact_eqns14 = Eq(lhs, rhs14)
104 impact_eqns14 = sym.simplify(impact_eqns14)
105
106 rhs15 = Matrix([lamb*dphi15dq_Sym[0],
107                  lamb*dphi15dq_Sym[1],
108                  lamb*dphi15dq_Sym[2],
109                  lamb*dphi15dq_Sym[3],
110                  lamb*dphi15dq_Sym[4],
111                  lamb*dphi15dq_Sym[5], 0])
112 impact_eqns15 = Eq(lhs, rhs15)
113 impact_eqns15 = sym.simplify(impact_eqns15)
114
115 rhs16 = Matrix([lamb*dphi16dq_Sym[0],
116                  lamb*dphi16dq_Sym[1],
117                  lamb*dphi16dq_Sym[2],
118                  lamb*dphi16dq_Sym[3],
119                  lamb*dphi16dq_Sym[4],
120                  lamb*dphi16dq_Sym[5], 0])

```

```

121 impact_eqns16 = Eq(lhs, rhs16)
122 impact_eqns16 = sym.simplify(impact_eqns16)
123
124 #define impact update function
125 def impact_update13(s, impact_eqns, sym_list):
126     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s
127                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],
128                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}
129     new_impact_eqns = impact_eqns.subs(subs_dict)
130     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,
131                                                 x2dotPlus, y2dotPlus, theta2dotPlus,
132                                                 lamb], dict=True)
133     if len(impact_solns) == 1:
134         print("Damn only one solution ...")
135     else:
136         for sol in impact_solns:
137             lamb_sol = sol[lamb]
138             if abs(lamb_sol) < 1e-06:
139                 pass
140             else:
141                 return np.array([
142                     s[0],
143                     s[1],
144                     s[2],
145                     s[3],
146                     s[4],
147                     s[5],
148                     float(sym.N(sol[sym_list[0]])),
149                     float(sym.N(sol[sym_list[1]])),
150                     float(sym.N(sol[sym_list[2]])),
151                     float(sym.N(sol[sym_list[3]])),
152                     float(sym.N(sol[sym_list[4]])),
153                     float(sym.N(sol[sym_list[5]])),
154                 ])
155
156 def impact_update14(s, impact_eqns, sym_list):
157     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s
158                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],
159                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}
160     new_impact_eqns = impact_eqns.subs(subs_dict)
161     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,
162                                                 x2dotPlus, y2dotPlus, theta2dotPlus,
163                                                 lamb], dict=True)
164     if len(impact_solns) == 1:
165         print("Damn only one solution ...")
166     else:
167         for sol in impact_solns:
168             lamb_sol = sol[lamb]
169             if abs(lamb_sol) < 1e-06:
170                 pass
171             else:
172                 return np.array([
173                     s[0],
174                     s[1],
175                     s[2],
176                     s[3],
177                     s[4],
178                     s[5],
179                     float(sym.N(sol[sym_list[0]])),
180                     float(sym.N(sol[sym_list[1]])),
181                     float(sym.N(sol[sym_list[2]])),

```

```
182         float(sym.N(sol[sym_list[3]])),  
183         float(sym.N(sol[sym_list[4]])),  
184         float(sym.N(sol[sym_list[5]])),  
185     ])  
186  
187 def impact_update15(s, impact_eqns, sym_list):  
188     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s  
189                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],  
190                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}  
191     new_impact_eqns = impact_eqns.subs(subs_dict)  
192     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,  
193                                         x2dotPlus, y2dotPlus, theta2dotPlus,  
194                                         lamb], dict=True)  
195     if len(impact_solns) == 1:  
196         print("Damn only one solution ...")  
197     else:  
198         for sol in impact_solns:  
199             lamb_sol = sol[lamb]  
200             if abs(lamb_sol) < 1e-06:  
201                 pass  
202             else:  
203                 return np.array([  
204                     s[0],  
205                     s[1],  
206                     s[2],  
207                     s[3],  
208                     s[4],  
209                     s[5],  
210                     float(sym.N(sol[sym_list[0]])),  
211                     float(sym.N(sol[sym_list[1]])),  
212                     float(sym.N(sol[sym_list[2]])),  
213                     float(sym.N(sol[sym_list[3]])),  
214                     float(sym.N(sol[sym_list[4]])),  
215                     float(sym.N(sol[sym_list[5]]))),  
216                 ])  
217  
218 def impact_update16(s, impact_eqns, sym_list):  
219     subs_dict = {x1_s:s[0], y1_s:s[1], theta1_s:s[2], x2_s:s[3], y2_s:s[4], theta2_s:s  
220                 x1dot:s[6], y1dot:s[7], theta1dot:s[8],  
221                 x2dot:s[9], y2dot:s[10], theta2dot:s[11]}  
222     new_impact_eqns = impact_eqns.subs(subs_dict)  
223     impact_solns = sym.solve(new_impact_eqns, [x1dotPlus, y1dotPlus, theta1dotPlus,  
224                                         x2dotPlus, y2dotPlus, theta2dotPlus,  
225                                         lamb], dict=True)  
226     if len(impact_solns) == 1:  
227         print("Damn only one solution ...")  
228     else:  
229         for sol in impact_solns:  
230             lamb_sol = sol[lamb]  
231             if abs(lamb_sol) < 1e-06:  
232                 pass  
233             else:  
234                 return np.array([  
235                     s[0],  
236                     s[1],  
237                     s[2],  
238                     s[3],  
239                     s[4],  
240                     s[5],  
241                     float(sym.N(sol[sym_list[0]])),  
242                     float(sym.N(sol[sym_list[1]]))),
```

```
243     float(sym.N(sol[sym_list[2]])),  
244     float(sym.N(sol[sym_list[3]])),  
245     float(sym.N(sol[sym_list[4]])),  
246     float(sym.N(sol[sym_list[5]])),  
247     ])  
248 s_test1 = np.array([0, 0, 0, 3, 3, 0.2, 0, 0.1, 0, 0, -0.1, 0])  
249 display(impact_update13(s_test1, impact_eqns13, [x1dotPlus, y1dotPlus, theta1dotPlus,  
250                                         x2dotPlus, y2dotPlus, theta2dotPlus])  
251 display(impact_update14(s_test1, impact_eqns14, [x1dotPlus, y1dotPlus, theta1dotPlus,  
252                                         x2dotPlus, y2dotPlus, theta2dotPlus])  
253 display(impact_update15(s_test1, impact_eqns15, [x1dotPlus, y1dotPlus, theta1dotPlus,  
254                                         x2dotPlus, y2dotPlus, theta2dotPlus])  
255 display(impact_update16(s_test1, impact_eqns16, [x1dotPlus, y1dotPlus, theta1dotPlus,  
256                                         x2dotPlus, y2dotPlus, theta2dotPlus])
```

```
array([ 0.          ,  0.          ,  0.          ,  3.          ,  3.          ,  
       0.2         ,  0.          , -0.00838153, -0.06287571,  0.          ,  
      0.00838153, -0.00861283])  
  
array([ 0.          ,  0.          ,  0.          ,  3.          ,  3.          ,  
       0.2         ,  0.          , -0.00838153, -0.06287571,  0.          ,  
      0.00838153, -0.00861283])
```

Damn only one solution ...

None

Damn only one solution ...

None

In [15]:

```

1 # define impact condition function
2 phi1_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
3                           qdot[0], qdot[1], qdot[2],
4                           qdot[3], qdot[4], qdot[5]], phi1)
5 def impact_condition_1(s, threshold=1e-1):
6     phi1_val = phi1_func(*s)
7     if phi1_val > -threshold and phi1_val < threshold:
8         return True
9     return False
10
11 phi2_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
12                           qdot[0], qdot[1], qdot[2],
13                           qdot[3], qdot[4], qdot[5]], phi2)
14 def impact_condition_2(s, threshold=1e-1):
15     phi2_val = phi2_func(*s)
16     if phi2_val > -threshold and phi2_val < threshold:
17         return True
18     return False
19
20 phi3_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
21                           qdot[0], qdot[1], qdot[2],
22                           qdot[3], qdot[4], qdot[5]], phi3)
23 def impact_condition_3(s, threshold=1e-1):
24     phi3_val = phi3_func(*s)
25     if phi3_val > -threshold and phi3_val < threshold:
26         return True
27     return False
28
29 phi4_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
30                           qdot[0], qdot[1], qdot[2],
31                           qdot[3], qdot[4], qdot[5]], phi4)
32 def impact_condition_4(s, threshold=1e-1):
33     phi4_val = phi4_func(*s)
34     if phi4_val > -threshold and phi4_val < threshold:
35         return True
36     return False
37
38 phi5_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
39                           qdot[0], qdot[1], qdot[2],
40                           qdot[3], qdot[4], qdot[5]], phi5)
41 def impact_condition_5(s, threshold=1e-1):
42     phi5_val = phi5_func(*s)
43     if phi5_val > -threshold and phi5_val < threshold:
44         return True
45     return False
46
47 phi6_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
48                           qdot[0], qdot[1], qdot[2],
49                           qdot[3], qdot[4], qdot[5]], phi6)
50 def impact_condition_6(s, threshold=1e-1):
51     phi6_val = phi6_func(*s)
52     if phi6_val > -threshold and phi6_val < threshold:
53         return True
54     return False
55
56 phi7_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
57                           qdot[0], qdot[1], qdot[2],
58                           qdot[3], qdot[4], qdot[5]], phi7)
59 def impact_condition_7(s, threshold=1e-1):

```

```
60     phi7_val = phi7_func(*s)
61     if phi7_val > -threshold and phi7_val < threshold:
62         return True
63     return False
64
65 phi8_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
66                           qdot[0], qdot[1], qdot[2],
67                           qdot[3], qdot[4], qdot[5]], phi8)
68 def impact_condition_8(s, threshold=1e-1):
69     phi8_val = phi8_func(*s)
70     if phi8_val > -threshold and phi8_val < threshold:
71         return True
72     return False
73
74 phi9_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
75                           qdot[0], qdot[1], qdot[2],
76                           qdot[3], qdot[4], qdot[5]], phi9)
77 def impact_condition_9(s, threshold=1e-1):
78     phi9_val = phi9_func(*s)
79     if phi9_val > -threshold and phi9_val < threshold:
80         return True
81     return False
82
83 phi10_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
84                           qdot[0], qdot[1], qdot[2],
85                           qdot[3], qdot[4], qdot[5]], phi10)
86 def impact_condition_10(s, threshold=1e-1):
87     phi10_val = phi10_func(*s)
88     if phi10_val > -threshold and phi10_val < threshold:
89         return True
90     return False
91
92 phi11_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
93                           qdot[0], qdot[1], qdot[2],
94                           qdot[3], qdot[4], qdot[5]], phi11)
95 def impact_condition_11(s, threshold=1e-1):
96     phi11_val = phi11_func(*s)
97     if phi11_val > -threshold and phi11_val < threshold:
98         return True
99     return False
100
101 phi12_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
102                           qdot[0], qdot[1], qdot[2],
103                           qdot[3], qdot[4], qdot[5]], phi12)
104 def impact_condition_12(s, threshold=1e-1):
105     phi12_val = phi12_func(*s)
106     if phi12_val > -threshold and phi12_val < threshold:
107         return True
108     return False
109
110 phi13_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
111                           qdot[0], qdot[1], qdot[2],
112                           qdot[3], qdot[4], qdot[5]], phi13)
113 def impact_condition_13(s, threshold=1e-1):
114     phi13_val = phi13_func(*s)
115     if phi13_val > -threshold and phi13_val < threshold:
116         return True
117     return False
118
119 phi14_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
120                           qdot[0], qdot[1], qdot[2],
```

```
121     qdot[3], qdot[4], qdot[5]], phi14)
122 def impact_condition_14(s, threshold=1e-1):
123     phi14_val = phi14_func(*s)
124     if phi14_val > -threshold and phi14_val < threshold:
125         return True
126     return False
127
128 phi15_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
129                             qdot[0], qdot[1], qdot[2],
130                             qdot[3], qdot[4], qdot[5]], phi15)
131 def impact_condition_15(s, threshold=1e-1):
132     phi15_val = phi15_func(*s)
133     if phi15_val > -threshold and phi15_val < threshold:
134         return True
135     return False
136
137 phi16_func = sym.lambdify([q[0], q[1], q[2], q[3], q[4], q[5],
138                             qdot[0], qdot[1], qdot[2],
139                             qdot[3], qdot[4], qdot[5]], phi16)
140 def impact_condition_16(s, threshold=1e-1):
141     phi16_val = phi16_func(*s)
142     if phi16_val > -threshold and phi16_val < threshold:
143         return True
144     return False
```

In [16]:

```

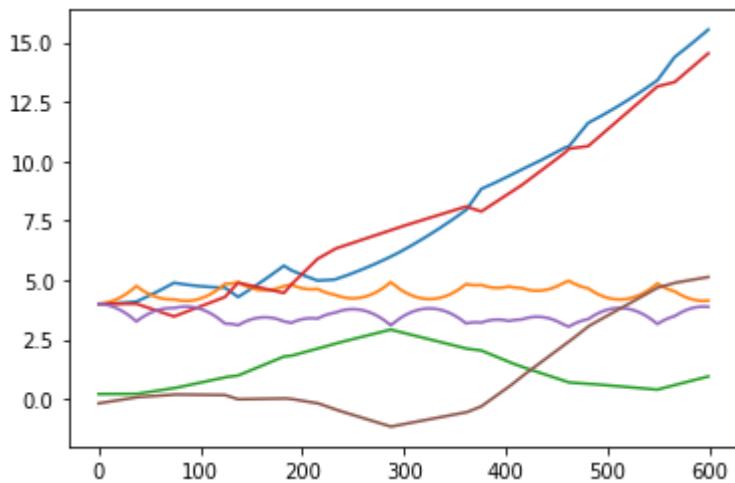
1 # define dynamics for the particle
2 def dyn(s):
3     return np.array([
4         s[6],
5         s[7],
6         s[8],
7         s[9],
8         s[10],
9         s[11],
10        x1ddot_func(*s),
11        y1ddot_func(*s),
12        theta1ddot_func(*s),
13        x2ddot_func(*s),
14        y2ddot_func(*s),
15        theta2ddot_func(*s)
16    ])
17
18 def integrate(f, xt, dt):
19     k1 = dt * f(xt)
20     k2 = dt * f(xt+k1/2.)
21     k3 = dt * f(xt+k2/2.)
22     k4 = dt * f(xt+k3)
23     new_xt = xt + (1/6.) * (k1+2.0*k2+2.0*k3+k4)
24     return new_xt
25
26 def simulate_with_impact(f, x0, tspan, dt, integrate):
27     N = int (( max(tspan)-min(tspan))/dt)
28     x = np.copy(x0)
29     tvec = np.linspace(min(tspan),max(tspan),N)
30     xtraj = np.zeros((len(x0),N))
31     for i in range (N):
32         if impact_condition_1(x) is True:
33             x = impact_update1(x, impact_eqns1, [x1dotPlus, y1dotPlus, theta1dotPlus,
34                                                 x2dotPlus, y2dotPlus, theta2dotPlus])
35             xtraj[:,i]=integrate(f,x,dt)
36         elif impact_condition_2(x) is True:
37             x = impact_update2(x, impact_eqns2, [x1dotPlus, y1dotPlus, theta1dotPlus,
38                                                 x2dotPlus, y2dotPlus, theta2dotPlus])
39             xtraj[:,i]=integrate(f,x,dt)
40         elif impact_condition_3(x) is True:
41             x = impact_update3(x, impact_eqns3, [x1dotPlus, y1dotPlus, theta1dotPlus,
42                                                 x2dotPlus, y2dotPlus, theta2dotPlus])
43             xtraj[:,i]=integrate(f,x,dt)
44         elif impact_condition_4(x) is True:
45             x = impact_update4(x, impact_eqns4, [x1dotPlus, y1dotPlus, theta1dotPlus,
46                                                 x2dotPlus, y2dotPlus, theta2dotPlus])
47             xtraj[:,i]=integrate(f,x,dt)
48         elif impact_condition_5(x) is True:
49             x = impact_update5(x, impact_eqns5, [x1dotPlus, y1dotPlus, theta1dotPlus,
50                                                 x2dotPlus, y2dotPlus, theta2dotPlus])
51             xtraj[:,i]=integrate(f,x,dt)
52         elif impact_condition_6(x) is True:
53             x = impact_update6(x, impact_eqns6, [x1dotPlus, y1dotPlus, theta1dotPlus,
54                                                 x2dotPlus, y2dotPlus, theta2dotPlus])
55             xtraj[:,i]=integrate(f,x,dt)
56         elif impact_condition_7(x) is True:
57             x = impact_update7(x, impact_eqns7, [x1dotPlus, y1dotPlus, theta1dotPlus,
58                                                 x2dotPlus, y2dotPlus, theta2dotPlus])
59             xtraj[:,i]=integrate(f,x,dt)

```

```
60 elif impact_condition_8(x) is True:
61     x = impact_update8(x, impact_eqns8, [x1dotPlus, y1dotPlus, theta1dotPlus,
62                                         x2dotPlus, y2dotPlus, theta2dotPlus])
63     xtraj[:,i]=integrate(f,x,dt)
64 elif impact_condition_9(x) is True:
65     x = impact_update9(x, impact_eqns9, [x1dotPlus, y1dotPlus, theta1dotPlus,
66                                         x2dotPlus, y2dotPlus, theta2dotPlus])
67     xtraj[:,i]=integrate(f,x,dt)
68 elif impact_condition_10(x) is True:
69     x = impact_update10(x, impact_eqns10, [x1dotPlus, y1dotPlus, theta1dotPlus,
70                                         x2dotPlus, y2dotPlus, theta2dotPlus])
71     xtraj[:,i]=integrate(f,x,dt)
72 elif impact_condition_11(x) is True:
73     x = impact_update11(x, impact_eqns11, [x1dotPlus, y1dotPlus, theta1dotPlus,
74                                         x2dotPlus, y2dotPlus, theta2dotPlus])
75     xtraj[:,i]=integrate(f,x,dt)
76 elif impact_condition_12(x) is True:
77     x = impact_update12(x, impact_eqns12, [x1dotPlus, y1dotPlus, theta1dotPlus,
78                                         x2dotPlus, y2dotPlus, theta2dotPlus])
79     xtraj[:,i]=integrate(f,x,dt)
80 elif impact_condition_13(x) is True:
81     x = impact_update13(x, impact_eqns13, [x1dotPlus, y1dotPlus, theta1dotPlus,
82                                         x2dotPlus, y2dotPlus, theta2dotPlus])
83     xtraj[:,i]=integrate(f,x,dt)
84 elif impact_condition_14(x) is True:
85     x = impact_update14(x, impact_eqns14, [x1dotPlus, y1dotPlus, theta1dotPlus,
86                                         x2dotPlus, y2dotPlus, theta2dotPlus])
87     xtraj[:,i]=integrate(f,x,dt)
88 elif impact_condition_15(x) is True:
89     x = impact_update15(x, impact_eqns15, [x1dotPlus, y1dotPlus, theta1dotPlus,
90                                         x2dotPlus, y2dotPlus, theta2dotPlus])
91     xtraj[:,i]=integrate(f,x,dt)
92 elif impact_condition_16(x) is True:
93     x = impact_update16(x, impact_eqns16, [x1dotPlus, y1dotPlus, theta1dotPlus,
94                                         x2dotPlus, y2dotPlus, theta2dotPlus])
95     xtraj[:,i]=integrate(f,x,dt)
96 else :
97     xtraj[:,i]=integrate(f,x,dt)
98 x = np.copy(xtraj[:,i])
99 return xtraj
```

In [20]:

```
1 # simulate
2 s0 = np.array([4,4,0.2,4,4,-0.2,0,0.1,0,0,-0.1,0.7])
3 traj = simulate_with_impact(dyn, s0, tspan=[0,6], dt=0.01, integrate=integrate)
4 # plot
5 import matplotlib.pyplot as plt
6 plt.plot(np.arange(traj.shape[1]), traj[0:6].T)
7 plt.show()
```



In [21]:

```
1 def animate_biped(q_array,T=10):
2     """
3         Function to generate web-based animation of double-pendulum system
4
5     Parameters:
6     =====
7     q_array:
8         trajectory of x, y, theta1 and theta2, should be a NumPy array with
9         shape of (2,N)
10    T:
11        length/seconds of animation duration
12
13    Returns: None
14    """
15
16    #####
17    # Imports required for animation.
18    from plotly.offline import init_notebook_mode, iplot
19    from IPython.display import display, HTML
20    import plotly.graph_objects as go
21
22    #####
23    # Browser configuration.
24    def configure_plotly_browser_state():
25        import IPython
26        display(IPython.core.display.HTML('''
27            <script src="/static/components/requirejs/require.js"></script>
28            <script>
29                requirejs.config({
30                    paths: {
31                        base: '/static/base',
32                        plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
33                        },
34                    });
35            </script>
36        '''))
37        configure_plotly_browser_state()
38        init_notebook_mode(connected=False)
39
40    #####
41    N = len(q_array[0]) # Need this for specifying Length of simulation
42
43
44    #####
45    # Define arrays containing data for frame axes
46    # In each frame, the x and y axis are always fixed
47    x_axis = np.array([0.3, 0.0])
48    y_axis = np.array([0.0, 0.3])
49
50    x_1 = np.array([2,2])
51    x_2 = np.array([2,-2])
52    x_3 = np.array([-2,-2])
53    x_4 = np.array([-2,2])
54
55    x_origin = np.array([0, 0])
56    y_origin = np.array([0, 0])
57    # Use homogeneous transformation to transfer these two axes/points
58    # back to the fixed frame
59    # Jack frame
```

```
60 frame_a_x_axis = np.zeros((2,N))
61 frame_a_y_axis = np.zeros((2,N))
62 frame_a_x_origin = np.zeros((2,N))
63 frame_a_y_origin = np.zeros((2,N))
64
65 frame_b_x_origin = np.zeros((2,N))
66 frame_b_y_origin = np.zeros((2,N))
67 frame_b_x_axis = np.zeros((2,N))
68 frame_b_y_axis = np.zeros((2,N))
69
70 frame_c_x_origin = np.zeros((2,N))
71 frame_c_y_origin = np.zeros((2,N))
72 frame_c_x_axis = np.zeros((2,N))
73 frame_c_y_axis = np.zeros((2,N))
74
75 frame_d_x_origin = np.zeros((2,N))
76 frame_d_y_origin = np.zeros((2,N))
77 frame_d_x_axis = np.zeros((2,N))
78 frame_d_y_axis = np.zeros((2,N))
79
80 frame_e_x_origin = np.zeros((2,N))
81 frame_e_y_origin = np.zeros((2,N))
82 frame_e_x_axis = np.zeros((2,N))
83 frame_e_y_axis = np.zeros((2,N))
84
85 # Box frame
86 frame_f_x_origin = np.zeros((2,N))
87 frame_f_y_origin = np.zeros((2,N))
88 frame_f_x_axis = np.zeros((2,N))
89 frame_f_y_axis = np.zeros((2,N))
90
91 frame_f_x1 = np.zeros((2,N))
92 frame_f_x2 = np.zeros((2,N))
93 frame_f_x3 = np.zeros((2,N))
94 frame_f_x4 = np.zeros((2,N))
95
96 frame_g_x_origin = np.zeros((2,N))
97 frame_g_y_origin = np.zeros((2,N))
98 frame_g_x_axis = np.zeros((2,N))
99 frame_g_y_axis = np.zeros((2,N))
100
101 frame_h_x_origin = np.zeros((2,N))
102 frame_h_y_origin = np.zeros((2,N))
103 frame_h_x_axis = np.zeros((2,N))
104 frame_h_y_axis = np.zeros((2,N))
105
106 frame_i_x_origin = np.zeros((2,N))
107 frame_i_y_origin = np.zeros((2,N))
108 frame_i_x_axis = np.zeros((2,N))
109 frame_i_y_axis = np.zeros((2,N))
110
111 frame_j_x_origin = np.zeros((2,N))
112 frame_j_y_origin = np.zeros((2,N))
113 frame_j_x_axis = np.zeros((2,N))
114 frame_j_y_axis = np.zeros((2,N))
115 for i in range(N): # iteration through each time step
116     # evaluate Jack homogeneous transformation
117     t_wa = np.array([[np.cos(q_array[5][i]), -np.sin(q_array[5][i]), q_array[3][i]
118                      [np.sin(q_array[5][i]), np.cos(q_array[5][i]), q_array[4][i]
119                      [0, 0, 1]])
120     t_ab = np.array([[1, 0, 0],
```

```

121                 [0,  1, -0.5],
122                 [0,  0,  1]])
123     t_ac = np.array([[1,  0, -0.5],
124                     [0,  1,  0],
125                     [0,  0,  1]])
126     t_ad = np.array([[1,  0,  0.5],
127                     [0,  1,  0],
128                     [0,  0,  1]])
129     t_ae = np.array([[1,  0,  0],
130                     [0,  1,  0.5],
131                     [0,  0,  1]])
132 # evaluate Box homogeneous transformation
133 t_wf = np.array([[np.cos(q_array[2][i]), -np.sin(q_array[2][i]), q_array[0][i]
134                  [np.sin(q_array[2][i]), np.cos(q_array[2][i]), q_array[1][i]
135                  [0,  0,  1]])
136 t_fg = np.array([[1,  0,  0],
137                  [0,  1, -3],
138                  [0,  0,  1]])
139 t_fh = np.array([[1,  0,  3],
140                  [0,  1,  0],
141                  [0,  0,  1]])
142 t_fi = np.array([[1,  0,  0],
143                  [0,  1,  3],
144                  [0,  0,  1]])
145 t_fj = np.array([[1,  0, -3],
146                  [0,  1,  0],
147                  [0,  0,  1]])
148 # transfer the x and y axes in body frame back to fixed frame at
149 # the current time step
150 frame_a_x_axis[:,i] = t_wa.dot([x_axis[0], x_axis[1], 1])[0:2]
151 frame_a_y_axis[:,i] = t_wa.dot([y_axis[0], y_axis[1], 1])[0:2]
152 frame_a_x_origin[:,i] = t_wa.dot([x_origin[0], x_origin[1], 1])[0:2]
153 frame_a_y_origin[:,i] = t_wa.dot([y_origin[0], y_origin[1], 1])[0:2]
154
155 t_ab = t_wa.dot(t_ab)
156 frame_b_x_axis[:,i] = t_ab.dot([x_axis[0], x_axis[1], 1])[0:2]
157 frame_b_y_axis[:,i] = t_ab.dot([y_axis[0], y_axis[1], 1])[0:2]
158 frame_b_x_origin[:,i] = t_ab.dot([x_origin[0], x_origin[1], 1])[0:2]
159 frame_b_y_origin[:,i] = t_ab.dot([y_origin[0], y_origin[1], 1])[0:2]
160
161 t_ac = t_wa.dot(t_ac)
162 frame_c_x_axis[:,i] = t_ac.dot([x_axis[0], x_axis[1], 1])[0:2]
163 frame_c_y_axis[:,i] = t_ac.dot([y_axis[0], y_axis[1], 1])[0:2]
164 frame_c_x_origin[:,i] = t_ac.dot([x_origin[0], x_origin[1], 1])[0:2]
165 frame_c_y_origin[:,i] = t_ac.dot([y_origin[0], y_origin[1], 1])[0:2]
166
167 t_ad = t_wa.dot(t_ad)
168 frame_d_x_axis[:,i] = t_ad.dot([x_axis[0], x_axis[1], 1])[0:2]
169 frame_d_y_axis[:,i] = t_ad.dot([y_axis[0], y_axis[1], 1])[0:2]
170 frame_d_x_origin[:,i] = t_ad.dot([x_origin[0], x_origin[1], 1])[0:2]
171 frame_d_y_origin[:,i] = t_ad.dot([y_origin[0], y_origin[1], 1])[0:2]
172
173 t_ae = t_wa.dot(t_ae)
174 frame_e_x_axis[:,i] = t_ae.dot([x_axis[0], x_axis[1], 1])[0:2]
175 frame_e_y_axis[:,i] = t_ae.dot([y_axis[0], y_axis[1], 1])[0:2]
176 frame_e_x_origin[:,i] = t_ae.dot([x_origin[0], x_origin[1], 1])[0:2]
177 frame_e_y_origin[:,i] = t_ae.dot([y_origin[0], y_origin[1], 1])[0:2]
178
179 # Box frame
180 frame_f_x_axis[:,i] = t_wf.dot([x_axis[0], x_axis[1], 1])[0:2]
181 frame_f_y_axis[:,i] = t_wf.dot([y_axis[0], y_axis[1], 1])[0:2]

```

```

182     frame_f_x_origin[:,i] = t_wf.dot([x_origin[0], x_origin[1], 1])[0:2]
183     frame_f_y_origin[:,i] = t_wf.dot([y_origin[0], y_origin[1], 1])[0:2]
184
185     frame_f_x1[:,i] = t_wf.dot([x_1[0], x_1[1], 1])[0:2]
186     frame_f_x2[:,i] = t_wf.dot([x_2[0], x_2[1], 1])[0:2]
187     frame_f_x3[:,i] = t_wf.dot([x_3[0], x_3[1], 1])[0:2]
188     frame_f_x4[:,i] = t_wf.dot([x_4[0], x_4[1], 1])[0:2]
189
190
191     t_fg = t_wf.dot(t_fg)
192     frame_g_x_axis[:,i] = t_fg.dot([x_axis[0], x_axis[1], 1])[0:2]
193     frame_g_y_axis[:,i] = t_fg.dot([y_axis[0], y_axis[1], 1])[0:2]
194     frame_g_x_origin[:,i] = t_fg.dot([x_origin[0], x_origin[1], 1])[0:2]
195     frame_g_y_origin[:,i] = t_fg.dot([y_origin[0], y_origin[1], 1])[0:2]
196
197     t_fh = t_wf.dot(t_fh)
198     frame_h_x_axis[:,i] = t_fh.dot([x_axis[0], x_axis[1], 1])[0:2]
199     frame_h_y_axis[:,i] = t_fh.dot([y_axis[0], y_axis[1], 1])[0:2]
200     frame_h_x_origin[:,i] = t_fh.dot([x_origin[0], x_origin[1], 1])[0:2]
201     frame_h_y_origin[:,i] = t_fh.dot([y_origin[0], y_origin[1], 1])[0:2]
202
203     t_fi = t_wf.dot(t_fi)
204     frame_i_x_axis[:,i] = t_fi.dot([x_axis[0], x_axis[1], 1])[0:2]
205     frame_i_y_axis[:,i] = t_fi.dot([y_axis[0], y_axis[1], 1])[0:2]
206     frame_i_x_origin[:,i] = t_fi.dot([x_origin[0], x_origin[1], 1])[0:2]
207     frame_i_y_origin[:,i] = t_fi.dot([y_origin[0], y_origin[1], 1])[0:2]
208
209     t_fj = t_wf.dot(t_fj)
210     frame_j_x_axis[:,i] = t_fj.dot([x_axis[0], x_axis[1], 1])[0:2]
211     frame_j_y_axis[:,i] = t_fj.dot([y_axis[0], y_axis[1], 1])[0:2]
212     frame_j_x_origin[:,i] = t_fj.dot([x_origin[0], x_origin[1], 1])[0:2]
213     frame_j_y_origin[:,i] = t_fj.dot([y_origin[0], y_origin[1], 1])[0:2]
214
215 ######
216 # Using these to specify axis limits.
217 xm = -3 #np.min(xx1)-0.5
218 xM = 3 #np.max(xx1)+0.5
219 ym = -3 #np.min(yy1)-2.5
220 yM = 3 #np.max(yy1)+1.5
221
222 #####
223 # Defining data dictionary.
224 # Trajectories are here.
225 data=[
226     # note that except for the trajectory (which you don't need this time),
227     # you don't need to define entries other than "name". The items defined
228     # in this list will be related to the items defined in the "frames" list
229     # later in the same order. Therefore, these entries can be considered as
230     # labels for the components in each animation frame
231     dict(name='World Frame X'),
232     dict(name='World Frame Y'),
233     dict(name='Mass 1'),
234     dict(name='Mass 2'),
235     dict(name='Mass 3'),
236     dict(name='Mass 4'),
237     dict(name='line1'),
238     dict(name='line2'),
239     dict(name='line3'),
240     dict(name='line4'),
241     dict(name='lineF1'),
242     dict(name='lineF2'),

```



```

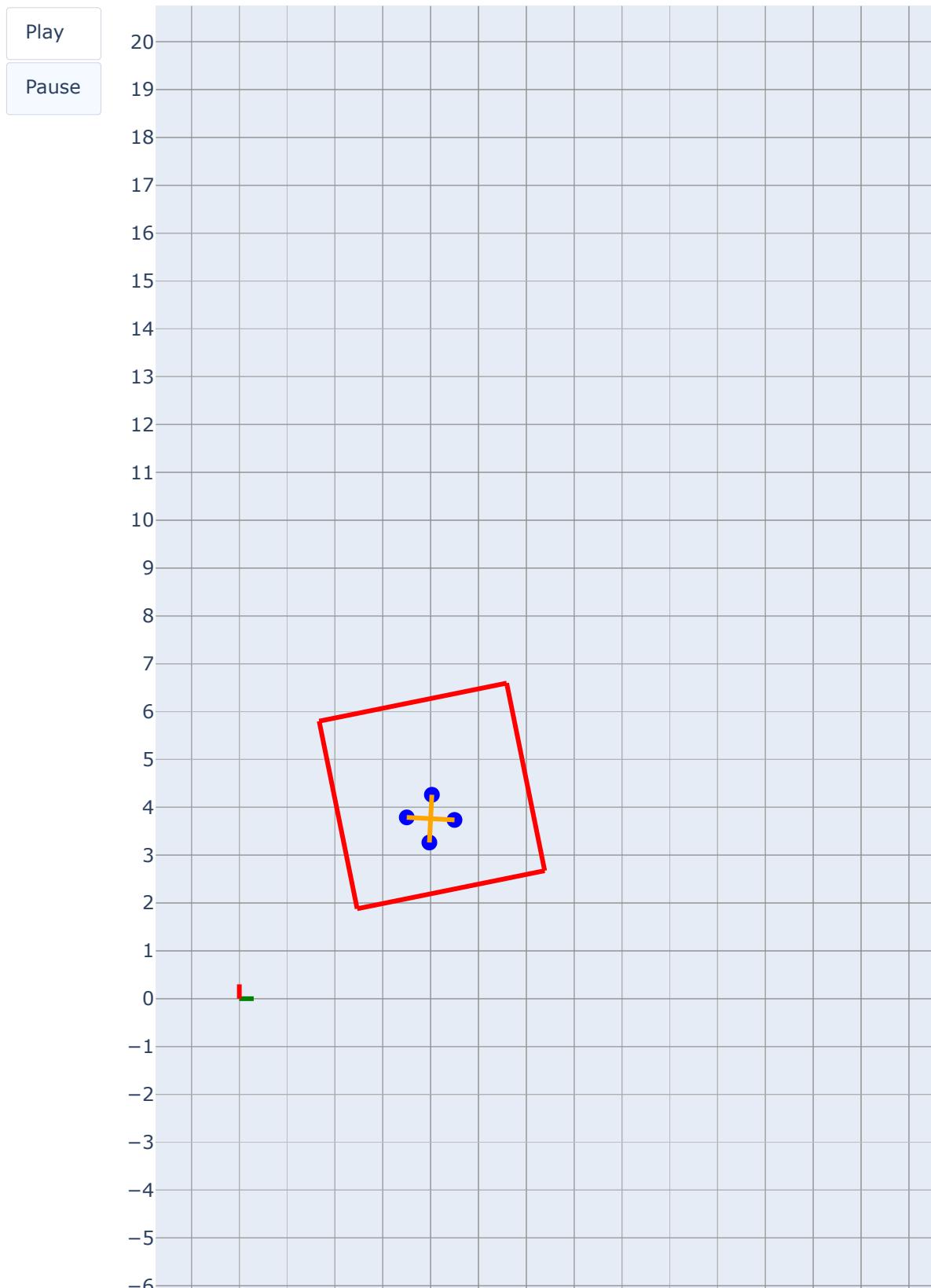
304     marker=dict(color="blue", size=10)),
305     go.Scatter(
306         x=[frame_d_x_origin[0][k]],
307         y=[frame_d_y_origin[1][k]],
308         mode="markers",
309         marker=dict(color="blue", size=10)),
310     go.Scatter(
311         x=[frame_e_x_origin[0][k]],
312         y=[frame_e_y_origin[1][k]],
313         mode="markers",
314         marker=dict(color="blue", size=10)),
315     dict(x=[frame_a_x_origin[0][k], frame_b_x_origin[0][k]],
316          y=[frame_a_y_origin[1][k], frame_b_y_origin[1][k]],
317          mode = 'lines',
318          line=dict(color='orange', width=3),
319      ),
320     dict(x=[frame_a_x_origin[0][k], frame_c_x_origin[0][k]],
321          y=[frame_a_y_origin[1][k], frame_c_y_origin[1][k]],
322          mode = 'lines',
323          line=dict(color='orange', width=3),
324      ),
325     dict(x=[frame_a_x_origin[0][k], frame_d_x_origin[0][k]],
326          y=[frame_a_y_origin[1][k], frame_d_y_origin[1][k]],
327          mode = 'lines',
328          line=dict(color='orange', width=3),
329      ),
330     dict(x=[frame_a_x_origin[0][k], frame_e_x_origin[0][k]],
331          y=[frame_a_y_origin[1][k], frame_e_y_origin[1][k]],
332          mode = 'lines',
333          line=dict(color='orange', width=3),
334      ),
335     dict(x=[frame_f_x1[0][k], frame_f_x2[0][k]],
336          y=[frame_f_x1[1][k], frame_f_x2[1][k]],
337          mode='lines',
338          line=dict(color='red', width=3),
339      ),
340     dict(x=[frame_f_x2[0][k], frame_f_x3[0][k]],
341          y=[frame_f_x2[1][k], frame_f_x3[1][k]],
342          mode='lines',
343          line=dict(color='red', width=3),
344      ),
345     dict(x=[frame_f_x3[0][k], frame_f_x4[0][k]],
346          y=[frame_f_x3[1][k], frame_f_x4[1][k]],
347          mode='lines',
348          line=dict(color='red', width=3),
349      ),
350     dict(x=[frame_f_x4[0][k], frame_f_x1[0][k]],
351          y=[frame_f_x4[1][k], frame_f_x1[1][k]],
352          mode='lines',
353          line=dict(color='red', width=3),
354      ),
355
356     ]) for k in range(N)]
357 ######
358 # Putting it all together and plotting.
359 figure1=dict(data=data, layout=layout, frames=frames)
360 iplot(figure1)
361
362 import numpy as np
363 sim_traj = traj
364 print('shape of trajectory: ', sim_traj.shape)

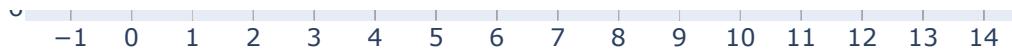
```

```
365 # second, animate!
366 animate_biped(sim_traj, T=10)
```

shape of trajectory: (12, 600)

## Double Pendulum Simulation





In [ ]:

```
1
```