chengcharles_final_project

December 8, 2021

1 Final Project: Angry Birds Space

Author: Charles Cheng

Date: 12/8/21

1.0.1 1 General Functions

[1]:
```
import numpy as np
import sympy as sym
from sympy import symbols, sin, cos, lambdify, solve, Eq, Matrix, simplify,␣
↪Function, integrate, shape
from sympy.abc import t
import math
from itertools import combinations

def rk4_integrate(f, funcs, xt, dt):

    k1 = f(xt,funcs)
    k2 = f(xt+dt*k1/2.,funcs)
    k3 = f(xt+dt*k2/2.,funcs)
    k4 = f(xt+dt*k3,funcs)
    new_xt = xt + (1/6.) * dt * (k1+2.0*k2+2.0*k3+k4)

    return new_xt

def euler_equations(L, funcs, D, t):

    N = len(funcs)//3
    q = Matrix([funcs[0:N]])
    qdot = Matrix([funcs[N:2*N]])

    J = Matrix([L])
    dJdq = J.jacobian(q)
    dJdqdot = J.jacobian(qdot)
    d_dJdqdot_dt = dJdqdot.diff(t)

    D_mat = Matrix([D])
```

```
    dDdqdot = D_mat.jacobian(qdot).T

    lhs = (d_dJdqdot_dt-dJdq).T
    EL_eqn = sym.Eq(lhs,dDdqdot)

    return EL_eqn

def solve_eqns(eqn, q_solv):

    q = Matrix(q_solv)
    soln = solve(eqn, q, dict=True)

    accl_exprs = []
    for sol in soln:
        for v in q:
            accl_sol = sol[v]
            accl_exprs.append(accl_sol)

    return accl_exprs

def dyn(s, func_list):

    N = len(s)//2
    func_eval = np.array([func(s) for func in func_list])

    return np.concatenate((s[N:], func_eval), axis=None)

def g_tr(theta,p):

    x, y, z = (p[0], p[1], p[2])

    return Matrix([[cos(theta), -sin(theta), 0, x],
                  [sin(theta), cos(theta), 0, y],
                  [0, 0, 1, z],
                  [0, 0, 0, 1]])

def inverse(g):

    R = g[0:3,0:3]
    p = g[0:3,3]

    RT = R.T
    RTp = RT*p

    g_inv = Matrix([[RT[0,0], RT[0,1], RT[0,2], -RTp[0]],
                    [RT[1,0], RT[1,1], RT[1,2], -RTp[1]],
                    [RT[2,0], RT[2,1], RT[2,2], -RTp[2]],
```

```
                    [0, 0, 0, 1]])

    return g_inv

def Vb(g):

    gg = inverse(g)*g.diff(t)

    Vb = Matrix([gg[0,3], gg[1,3], gg[2,3], gg[2,1], gg[0,2], gg[1,0]])

    return Vb

def hat_operation(r):

    r1, r2, r3 = r
    rhat = Matrix([[0, -r3, r2],[r3, 0, -r1],[-r2, r1, 0]])

    return rhat

def impact_condition(s, phi_func, threshold = 1e-1):

    for i, phi in enumerate(phi_func):
        if abs(phi(*s)) < threshold:
            return (True, i)

    return (False, None)

def animate(q, circles, T=10):

    from plotly.offline import init_notebook_mode, iplot
    from IPython.display import display, HTML
    import plotly.graph_objects as go

    def configure_plotly_browser_state():
        import IPython
        display(IPython.core.display.HTML('''
            <script src="/static/components/requirejs/require.js"></script>
            <script>
              requirejs.config({
                paths: {
                  base: '/static/base',
                  plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
                },
              });
            </script>
            '''))
    configure_plotly_browser_state()
```

```python
    init_notebook_mode(connected=False)

    xx1=q[0]
    yy1=q[1]
    xx2=q[3]
    yy2=q[4]
    xx3=q[6]
    yy3=q[7]
    N = len(q[0])

    mass1_corner1 = np.zeros((2,N))
    mass1_corner2 = np.zeros((2,N))
    mass1_corner3 = np.zeros((2,N))
    mass1_corner4 = np.zeros((2,N))
    mass2_corner1 = np.zeros((2,N))
    mass2_corner2 = np.zeros((2,N))
    mass2_corner3 = np.zeros((2,N))
    mass2_corner4 = np.zeros((2,N))
    mass3_corner1 = np.zeros((2,N))
    mass3_corner2 = np.zeros((2,N))
    mass3_corner3 = np.zeros((2,N))
    mass3_corner4 = np.zeros((2,N))
    mass1_eye1 = np.zeros((2,N))
    mass1_eye2 = np.zeros((2,N))
    mass1_mouth1 = np.zeros((2,N))
    mass1_mouth2 = np.zeros((2,N))
    mass1_eyebrow1 = np.zeros((2,N))
    mass1_eyebrow2 = np.zeros((2,N))
    mass1_eyebrow3 = np.zeros((2,N))
    mass1_eyebrow4 = np.zeros((2,N))
    mass2_eye1 = np.zeros((2,N))
    mass2_eye2 = np.zeros((2,N))
    mass2_mouth = np.zeros((2,N))
    mass3_eye1 = np.zeros((2,N))
    mass3_eye2 = np.zeros((2,N))
    mass3_mouth = np.zeros((2,N))
    for i in range(N):
        ### Transformations Between World and Body Frame ###
        t_01 = np.matmul(np.array([[1,0,xx1[i]],[0,1,yy1[i]],[0,0,1]]),np.array([[cos(q[2][i]),-sin(q[2][i]),0],[sin(q[2][i]),cos(q[2][i]),0],[0,0,1]]))
        t_02 = np.matmul(np.array([[1,0,xx2[i]],[0,1,yy2[i]],[0,0,1]]),np.array([[cos(q[5][i]),-sin(q[5][i]),0],[sin(q[5][i]),cos(q[5][i]),0],[0,0,1]]))
        t_03 = np.matmul(np.array([[1,0,xx3[i]],[0,1,yy3[i]],[0,0,1]]),np.array([[cos(q[8][i]),-sin(q[8][i]),0],[sin(q[8][i]),cos(q[8][i]),0],[0,0,1]]))
        ### End of Comment ###
        mass1_corner1[:,i] = t_01.dot([l1, w1, 1])[0:2]
        mass1_corner2[:,i] = t_01.dot([-l1, w1, 1])[0:2]
```

```
        mass1_corner3[:,i] = t_01.dot([-l1, -w1, 1])[0:2]
        mass1_corner4[:,i] = t_01.dot([l1, -w1, 1])[0:2]
        mass2_corner1[:,i] = t_02.dot([l2, w2, 1])[0:2]
        mass2_corner2[:,i] = t_02.dot([-l2, w2, 1])[0:2]
        mass2_corner3[:,i] = t_02.dot([-l2, -w2, 1])[0:2]
        mass2_corner4[:,i] = t_02.dot([l2, -w2, 1])[0:2]
        mass3_corner1[:,i] = t_03.dot([l3, w3, 1])[0:2]
        mass3_corner2[:,i] = t_03.dot([-l3, w3, 1])[0:2]
        mass3_corner3[:,i] = t_03.dot([-l3, -w3, 1])[0:2]
        mass3_corner4[:,i] = t_03.dot([l3, -w3, 1])[0:2]
        mass1_eye1[:,i] = t_01.dot([0.35*l1, 0.25*w1, 1])[0:2]
        mass1_eye2[:,i] = t_01.dot([-0.35*l1, 0.25*w1, 1])[0:2]
        mass1_mouth1[:,i] = t_01.dot([-0.5*l1, -0.40*w1, 1])[0:2]
        mass1_mouth2[:,i] = t_01.dot([0.5*l1, -0.40*w1, 1])[0:2]
        mass1_eyebrow1[:,i] = t_01.dot([0.2*l1, 0.50*w1, 1])[0:2]
        mass1_eyebrow2[:,i] = t_01.dot([0.7*l1, 0.75*w1, 1])[0:2]
        mass1_eyebrow3[:,i] = t_01.dot([-0.2*l1, 0.50*w1, 1])[0:2]
        mass1_eyebrow4[:,i] = t_01.dot([-0.7*l1, 0.75*w1, 1])[0:2]
        mass2_eye1[:,i] = t_02.dot([0.35*l2, 0.27*w2, 1])[0:2]
        mass2_eye2[:,i] = t_02.dot([-0.35*l2, 0.27*w2, 1])[0:2]
        mass2_mouth[:,i] = t_02.dot([0, -0.40*w2, 1])[0:2]
        mass3_eye1[:,i] = t_03.dot([0.35*l3, 0.27*w3, 1])[0:2]
        mass3_eye2[:,i] = t_03.dot([-0.35*l3, 0.27*w3, 1])[0:2]
        mass3_mouth[:,i] = t_03.dot([0, -0.40*w3, 1])[0:2]

    # Using these to specify axis limits
    xm = -9
    xM = 13.5
    ym = -11.5
    yM = 6

    data=[
        dict(name='Bird COM'),
        dict(name='Bird Side 1'),
        dict(name='Bird Side 2'),
        dict(name='Bird Side 3'),
        dict(name='Bird Side 4'),
        dict(name='Bird Trajectory'),
        dict(name='Bird Eyes'),
        dict(name='Bird Mouth'),
        dict(name='Bird Eyebrow 1'),
        dict(name='Bird Eyebrow 2'),
        dict(name='Pig 1 COM'),
        dict(name='Pig 1 Side 1'),
        dict(name='Pig 1 Side 2'),
        dict(name='Pig 1 Side 3'),
        dict(name='Pig 1 Side 4'),
```

```
        dict(name='Pig 1 Eyes'),
        dict(name='Pig 1 Mouth'),
        dict(name='Pig 2 COM'),
        dict(name='Pig 2 Side 1'),
        dict(name='Pig 2 Side 2'),
        dict(name='Pig 2 Side 3'),
        dict(name='Pig 2 Side 4'),
        dict(name='Pig 2 Eyes'),
        dict(name='Pig 2 Mouth'),
        dict(name='Planet 1',
             x=[point[0] for point in circles[0]],
             y=[point[1] for point in circles[0]],
             mode="lines",
             line=dict(width=3, color="black")),
        dict(name='Planet 1 Fill',
             x=[x1_cen],
             y=[y1_cen],
             mode="markers",
             marker=dict(size=160, color="darkviolet")),
        dict(name='Grav 1 Field',
             x=[point[0] for point in circles[1]],
             y=[point[1] for point in circles[1]],
             mode="markers",
             marker=dict(size=3, color="blue")),
        dict(name='Planet 2',
             x=[point[0] for point in circles[2]],
             y=[point[1] for point in circles[2]],
             mode="lines",
             line=dict(width=3, color="black")),
        dict(name='Planet 2 Fill',
             x=[x2_cen],
             y=[y2_cen],
             mode="markers",
             marker=dict(size=95, color="steelblue")),
        dict(name='Grav 2 Field',
             x=[point[0] for point in circles[3]],
             y=[point[1] for point in circles[3]],
             mode="markers",
             marker=dict(size=3, color="blue")),
        ]

    layout=dict(autosize=False, width=1000, height=1000,
                xaxis=dict(range=[xm, xM], autorange=False,␣
↪zeroline=False,dtick=1),
                yaxis=dict(range=[ym, yM], autorange=False,␣
↪zeroline=False,scaleanchor = "x",dtick=1),
                title='Simulation',
```

```
                hovermode='closest',
                updatemenus= [{'type': 'buttons',
                               'buttons': [{'label': 'Play','method': 'animate',
                                            'args': [None, {'frame':␣
↪{'duration': T, 'redraw': False}}]},
                                           {'args': [[None], {'frame':␣
↪{'duration': T, 'redraw': False}, 'mode': 'immediate',
                                            'transition': {'duration':␣
↪0}}],'label': 'Pause','method': 'animate'}
                                          ]
                              }]
               )

   frames=[dict(data=[
                        go.Scatter(
                             x=[xx1[k]],
                             y=[yy1[k]],
                             mode="markers",
                             marker=dict(color="red", size=27)),
                        dict(x=[mass1_corner1[0][k],mass1_corner2[0][k]],
                             y=[mass1_corner1[1][k],mass1_corner2[1][k]],
                             mode='lines',
                             line=dict(color='red', width=1),
                             ),
                        dict(x=[mass1_corner2[0][k],mass1_corner3[0][k]],
                             y=[mass1_corner2[1][k],mass1_corner3[1][k]],
                             mode='lines',
                             line=dict(color='red', width=1),
                             ),
                        dict(x=[mass1_corner3[0][k],mass1_corner4[0][k]],
                             y=[mass1_corner3[1][k],mass1_corner4[1][k]],
                             mode='lines',
                             line=dict(color='red', width=1),
                             ),
                        dict(x=[mass1_corner4[0][k],mass1_corner1[0][k]],
                             y=[mass1_corner4[1][k],mass1_corner1[1][k]],
                             mode='lines',
                             line=dict(color='red', width=1),
                             ),
                        go.Scatter(
                             x=xx1[0:k:8],
                             y=yy1[0:k:8],
                             mode="markers",
                             marker=dict(color="red", size=2)),
                        go.Scatter(
                             x=[mass1_eye1[0][k], mass1_eye2[0][k]],
                             y=[mass1_eye1[1][k], mass1_eye2[1][k]],
```

```
                            mode="markers",
                            marker=dict(color="black", size=5)),
                        dict(x=[mass1_mouth1[0][k],mass1_mouth2[0][k]],
                            y=[mass1_mouth1[1][k],mass1_mouth2[1][k]],
                            mode='lines',
                            line=dict(color='black', width=3),
                            ),
                        dict(x=[mass1_eyebrow1[0][k],mass1_eyebrow2[0][k]],
                            y=[mass1_eyebrow1[1][k],mass1_eyebrow2[1][k]],
                            mode='lines',
                            line=dict(color='black', width=3),
                            ),
                        dict(x=[mass1_eyebrow3[0][k],mass1_eyebrow4[0][k]],
                            y=[mass1_eyebrow3[1][k],mass1_eyebrow4[1][k]],
                            mode='lines',
                            line=dict(color='black', width=3),
                            ),
                        go.Scatter(
                            x=[xx2[k]],
                            y=[yy2[k]],
                            mode="markers",
                            marker=dict(color="limegreen", size=27)),
                        dict(x=[mass2_corner1[0][k],mass2_corner2[0][k]],
                            y=[mass2_corner1[1][k],mass2_corner2[1][k]],
                            mode='lines',
                            line=dict(color='limegreen', width=1),
                            ),
                        dict(x=[mass2_corner2[0][k],mass2_corner3[0][k]],
                            y=[mass2_corner2[1][k],mass2_corner3[1][k]],
                            mode='lines',
                            line=dict(color='limegreen', width=1),
                            ),
                        dict(x=[mass2_corner3[0][k],mass2_corner4[0][k]],
                            y=[mass2_corner3[1][k],mass2_corner4[1][k]],
                            mode='lines',
                            line=dict(color='limegreen', width=1),
                            ),
                        dict(x=[mass2_corner4[0][k],mass2_corner1[0][k]],
                            y=[mass2_corner4[1][k],mass2_corner1[1][k]],
                            mode='lines',
                            line=dict(color='limegreen', width=1),
                            ),
                        go.Scatter(
                            x=[mass2_eye1[0][k], mass2_eye2[0][k]],
                            y=[mass2_eye1[1][k], mass2_eye2[1][k]],
                            mode="markers",
                            marker=dict(color="black", size=5)),
```

```
                        go.Scatter(
                            x=[mass2_mouth[0][k]],
                            y=[mass2_mouth[1][k]],
                            mode="markers",
                            marker=dict(color="black", size=12)),
                        go.Scatter(
                            x=[xx3[k]],
                            y=[yy3[k]],
                            mode="markers",
                            marker=dict(color="limegreen", size=27)),
                        dict(x=[mass3_corner1[0][k],mass3_corner2[0][k]],
                            y=[mass3_corner1[1][k],mass3_corner2[1][k]],
                            mode='lines',
                            line=dict(color='limegreen', width=1),
                            ),
                        dict(x=[mass3_corner2[0][k],mass3_corner3[0][k]],
                            y=[mass3_corner2[1][k],mass3_corner3[1][k]],
                            mode='lines',
                            line=dict(color='limegreen', width=1),
                            ),
                        dict(x=[mass3_corner3[0][k],mass3_corner4[0][k]],
                            y=[mass3_corner3[1][k],mass3_corner4[1][k]],
                            mode='lines',
                            line=dict(color='limegreen', width=1),
                            ),
                        dict(x=[mass3_corner4[0][k],mass3_corner1[0][k]],
                            y=[mass3_corner4[1][k],mass3_corner1[1][k]],
                            mode='lines',
                            line=dict(color='limegreen', width=1),
                            ),
                        go.Scatter(
                            x=[mass3_eye1[0][k], mass3_eye2[0][k]],
                            y=[mass3_eye1[1][k], mass3_eye2[1][k]],
                            mode="markers",
                            marker=dict(color="black", size=5)),
                        go.Scatter(
                            x=[mass3_mouth[0][k]],
                            y=[mass3_mouth[1][k]],
                            mode="markers",
                            marker=dict(color="black", size=12)),
                       ]) for k in range(N)]

figure1=dict(data=data, layout=layout, frames=frames)
iplot(figure1)
```

1.0.2 2 Parameters

[2]:
```
### BODIES PARAMETERS ###
l1,w1,h1,rho1,l2,w2,h2,rho2,l3,w3,h3,rho3 = (0.4,0.4,0.4,10,
                                             0.4,0.4,0.4,10,
                                             0.4,0.4,0.4,10)


### PLANET PARAMETERS ###
grav0,x0_cen,y0_cen,R0,R0_out,grav1,x1_cen,y1_cen,R1,R1_out,grav2,x2_cen,y2_cen,R2,R2_out␣
↪= (0,0,0,0,0,
                                                                                         ␣
↪               800,1,2,2.5,4.5,
                                                                                         ␣
↪               800,9,-7,1.5,3.5)

### DRAG PARAMETERS ###
rho_air0, Cd0, rho_air1, Cd1, rho_air2, Cd2 = (0, 0,
                                               1.225, 0.8,
                                               2.0, 0.8)
```

[3]:
```
# Circles used for animation
circle1 = [(math.cos(2*math.pi/100*x)*R1+x1_cen,math.sin(2*math.pi/
↪100*x)*R1+y1_cen) for x in range(0,100+1)]
circle2 = [(math.cos(2*math.pi/25*x)*R1_out+x1_cen,math.sin(2*math.pi/
↪25*x)*R1_out+y1_cen) for x in range(0,25+1)]
circle3 = [(math.cos(2*math.pi/100*x)*R2+x2_cen,math.sin(2*math.pi/
↪100*x)*R2+y2_cen) for x in range(0,100+1)]
circle4 = [(math.cos(2*math.pi/25*x)*R2_out+x2_cen,math.sin(2*math.pi/
↪25*x)*R2_out+y2_cen) for x in range(0,25+1)]
circles = [circle1, circle2, circle3, circle4]
```

1.0.3 3 Equations of Motion

[42]:
```
def equations_of_motion(q, param_dict):

    _locals = locals()
    _locals['KE'] = 0
    _locals['V'] = 0
    _locals['x'] = []
    _locals['v'] = []
    _locals['a'] = []
    _locals['func_list'] = []
    N_bodies = len(q)//3

    for config_var in q:
```

```
        exec("{q0}=Function(r'{q0}')(t)\nx.append({q0})\n{q0}dot={q0}.
↪diff(t)\nv.append({q0}dot)\n{q0}ddot={q0}dot.diff(t)\na.append({q0}ddot)".
↪format(q0=config_var),globals(),_locals)
    i = q[0][1]
    exec("x{N}b,y{N}b,z{N}b=symbols(r'x{N}_b,y{N}_b,z{N}_b')\n".
↪format(N=i),globals(),_locals)
    ␣
↪exec("m{N}=8*param_dict['l{N}']*param_dict['w{N}']*param_dict['h{N}']*param_dict['rho{N}']"
↪format(N=i),globals(),_locals)
    exec("r{N}=Matrix([x{N}b,y{N}b,z{N}b])".format(N=i),globals(),_locals)
    exec("r{N}hat=hat_operation(r{N})".format(N=i),globals(),_locals)
    exec("integrand{N}=param_dict['rho{N}']*r{N}hat.T*r{N}hat".
↪format(N=i),globals(),_locals)
    ␣
↪exec("I{N}=integrate(integrate(integrate(integrand{N},(x{N}b,-param_dict['l{N}'],param_dict
↪format(N=i),globals(),_locals)
    exec("I6{N}=Matrix([[m{N}*sym.eye(3),sym.zeros(3)],[sym.zeros(3),I{N}]])".
↪format(N=i),globals(),_locals)
    ### Transformations Between World and Body Frame ###
    exec("g_0{N}=g_tr(theta{N},[x{N},y{N},0])".format(N=i),globals(),_locals)
    ### End of Comment ###
    exec("COM{N}=g_0{N}*Matrix([0,0,0,1])".format(N=i),globals(),_locals)
    exec("KE+=(0.5*(Vb(g_0{N})).T*I6{N}*Vb(g_0{N}))[0]".
↪format(N=i),globals(),_locals)
    ␣
↪exec("V+=-param_dict['grav']*m{N}*((COM{N}[0]-param_dict['x_cen'])**2+(COM{N}[1]-param_dict
↪5)".format(N=i),globals(),_locals)

    _locals['xv'] = _locals['x']+_locals['v']
    _locals['xva'] = _locals['xv']+_locals['a']
    _locals['L'] = simplify(_locals['KE'] - _locals['V'])
    exec("F_drag=-0.
↪5*param_dict['rho_air']*param_dict['Cd']*param_dict['w{N}']*param_dict['l{N}']*(x{N}dot**2+
↪5)".format(N=i),globals(),_locals)
    _locals['EL_eqn'] = simplify(euler_equations(_locals['L'], _locals['xva'],␣
↪_locals['F_drag'], symbols(r't')))
    exec("".join(config_var+'ddot_sol,' for config_var in␣
↪q)+"=solve_eqns(EL_eqn,a)",globals(),_locals)

    for config_var in q:
        exec("{q0}ddot_func=lambdify([xv],{q0}ddot_sol)\nfunc_list.
↪append({q0}ddot_func)".format(q0=config_var),globals(),_locals)

    return _locals['func_list']

space_EOM = equations_of_motion(["x1","y1","theta1"],
```

```
                                {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":
↪grav0,"x_cen":x0_cen,"y_cen":y0_cen,"rho_air":rho_air0,"Cd":Cd0})

planet1_EOM = equations_of_motion(["x1","y1","theta1"],
                                {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":
↪grav1,"x_cen":x1_cen,"y_cen":y1_cen,"rho_air":rho_air1,"Cd":Cd1})

planet2_EOM = equations_of_motion(["x1","y1","theta1"],
                                {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":
↪grav2,"x_cen":x2_cen,"y_cen":y2_cen,"rho_air":rho_air2,"Cd":Cd2})

EOM_list = [space_EOM, planet1_EOM, planet2_EOM]
```

1.0.4 4 Impacts

[5]:
```
def impact_eqns(q, param_dict):

    _locals = locals()
    _locals['KE'] = 0
    _locals['V'] = 0
    _locals['x'] = []
    _locals['v'] = []
    _locals['phi_eqns'] = []
    _locals['lambda_list'] = []
    _locals['qdotplus'] = []
    N_bodies = len(q)//3

    for config_var in q:
        exec("{q0}=symbols(r'{q0}')\nx.
↪append({q0})\n{q0}dot=symbols(r'{q0}dot')\nv.append({q0}dot)".
↪format(q0=config_var),globals(),_locals)
        exec("{q0}t=Function(r'{q0}')(t)\n{q0}tdot={q0}t.diff(t)".
↪format(q0=config_var),globals(),_locals)
    for i in range(1,N_bodies+1):
        exec("x{N}b,y{N}b,z{N}b=symbols(r'x{N}_b,y{N}_b,z{N}_b')\n".
↪format(N=i),globals(),_locals)
        ␣
↪exec("m{N}=8*param_dict['l{N}']*param_dict['w{N}']*param_dict['h{N}']*param_dict['rho{N}']"
↪format(N=i),globals(),_locals)
        exec("r{N}=Matrix([x{N}b,y{N}b,z{N}b])".format(N=i),globals(),_locals)
        exec("r{N}hat=hat_operation(r{N})".format(N=i),globals(),_locals)
        exec("integrand{N}=param_dict['rho{N}']*r{N}hat.T*r{N}hat".
↪format(N=i),globals(),_locals)
        ␣
↪exec("I{N}=integrate(integrate(integrate(integrand{N},(x{N}b,-param_dict['l{N}'],param_dict
↪format(N=i),globals(),_locals)
```

```
        exec("I6{N}=Matrix([[m{N}*sym.eye(3),sym.zeros(3)],[sym.
↪zeros(3),I{N}]])".format(N=i),globals(),_locals)
        ### Transformations Between World and Body Frame ###
        exec("g_0{N}=g_tr(theta{N}t,[x{N}t,y{N}t,0])".
↪format(N=i),globals(),_locals)
        ### End of Comment ###
        exec("COM{N}=g_0{N}*Matrix([0,0,0,1])".format(N=i),globals(),_locals)
        exec("KE+=(0.5*(Vb(g_0{N})).T*I6{N}*Vb(g_0{N}))[0]".
↪format(N=i),globals(),_locals)
        ␣
↪exec("V+=-param_dict['grav']*m{N}*((COM{N}[0]-param_dict['x_cen'])**2+(COM{N}[1]-param_dict
↪5)".format(N=i),globals(),_locals)
        exec("g_0{N}=g_tr(theta{N},[x{N},y{N},0])".
↪format(N=i),globals(),_locals)
        ␣
↪exec("r{N}_c1=g_0{N}*Matrix([param_dict['l{N}'],param_dict['w{N}'],0,1])".
↪format(N=i),globals(),_locals)
        ␣
↪exec("r{N}_c2=g_0{N}*Matrix([-param_dict['l{N}'],param_dict['w{N}'],0,1])".
↪format(N=i),globals(),_locals)
        ␣
↪exec("r{N}_c3=g_0{N}*Matrix([-param_dict['l{N}'],-param_dict['w{N}'],0,1])".
↪format(N=i),globals(),_locals)
        ␣
↪exec("r{N}_c4=g_0{N}*Matrix([param_dict['l{N}'],-param_dict['w{N}'],0,1])".
↪format(N=i),globals(),_locals)
        if param_dict['R'] != 0:
            ␣
↪exec("phi_0{N}1=param_dict['R']**2-(r{N}_c1[0]-param_dict['x_cen'])**2-(r{N}_c1[1]-param_di
↪append(phi_0{N}1)".format(N=i),globals(),_locals)
            ␣
↪exec("phi_0{N}2=param_dict['R']**2-(r{N}_c2[0]-param_dict['x_cen'])**2-(r{N}_c2[1]-param_di
↪append(phi_0{N}2)".format(N=i),globals(),_locals)
            ␣
↪exec("phi_0{N}3=param_dict['R']**2-(r{N}_c3[0]-param_dict['x_cen'])**2-(r{N}_c3[1]-param_di
↪append(phi_0{N}3)".format(N=i),globals(),_locals)
            ␣
↪exec("phi_0{N}4=param_dict['R']**2-(r{N}_c4[0]-param_dict['x_cen'])**2-(r{N}_c4[1]-param_di
↪append(phi_0{N}4)".format(N=i),globals(),_locals)

    exec("body_pairs=combinations(list(range(1,{N1})),2)".
↪format(N1=i+1),globals(),_locals)
    for pair in _locals['body_pairs']:
        for corner in range(1,5):
            for side in [(1,2),(2,3),(3,4),(4,1)]:
```

```
                exec("phi_{b1}{b2}{c}{p1}=(((r{b2}_c{p1}-r{b1}_c{c}).
↪T*(r{b2}_c{p1}-r{b1}_c{c}))[0])**0.5+(((r{b1}_c{c}-r{b2}_c{p2}).
↪T*(r{b1}_c{c}-r{b2}_c{p2}))[0])**0.5-(((r{b2}_c{p1}-r{b2}_c{p2}).
↪T*(r{b2}_c{p1}-r{b2}_c{p2}))[0])**0.5\nphi_eqns.append(phi_{b1}{b2}{c}{p1})".
↪format(b1=pair[0],b2=pair[1],c=corner,p1=side[0],p2=side[1]),globals(),_locals)

    for config_var in q:
        exec("KE=KE.subs({{{q0}t:{q0},{q0}tdot:{q0}dot}})\nV=V.subs({{{q0}t:
↪{q0},{q0}tdot:{q0}dot}})".format(q0=config_var),globals(),_locals)

    xv = _locals['x'] +_locals['v']
    phi_funcs = [lambdify(xv, func) for func in _locals['phi_eqns']]

    L = simplify(_locals['KE'] - _locals['V'])
    q = Matrix(_locals['x'])
    qdot = Matrix(_locals['v'])
    J = Matrix([L])
    phi_matrix = Matrix(_locals['phi_eqns'])

    dLdqdot = simplify(J.jacobian(qdot))
    dphidq = phi_matrix.jacobian(q)
    H = simplify((dLdqdot*qdot - J)[0])

    for q in _locals['x']:
        exec("{q0}dot_plus=symbols(r'{q0}dot^+')\nqdotplus.
↪append({q0}dot_plus)".format(q0=q),globals(),_locals)
    for i in range(0,len(_locals['phi_eqns'])):
        exec("lambda{N}=symbols(r'\lambda_{N}')\nlambda_list.append(lambda{N})".
↪format(N=i+1),globals(),_locals)

    dLdqdot_impact = []
    for exprs in dLdqdot:
        _locals['exprs_plus']=exprs
        for q in _locals['x']:
            exec("exprs_plus=exprs_plus.subs({q0}dot,{q0}dot_plus)".
↪format(q0=q,exp=exprs),globals(),_locals)
        dLdqdot_impact.append(_locals['exprs_plus']-exprs)

    _locals['H_plus']=H
    for q in _locals['x']:
        exec("H_plus=H_plus.subs({q0}dot,{q0}dot_plus)".
↪format(q0=q,exp=exprs),globals(),_locals)
    dLdqdot_impact.append(_locals['H_plus'] - H)
    lhs = Matrix(dLdqdot_impact)

    lambda_matrix = Matrix([_locals['lambda_list']])
```

```
    dphidq_impact = (lambda_matrix*dphidq).T
    rhs = dphidq_impact.row_insert(shape(dphidq_impact)[0],Matrix([0]))
    impact_eqns = Eq(lhs, rhs)

    return impact_eqns, phi_funcs, xv, _locals['qdotplus'],␣
 ↪_locals['lambda_list']

planet0_impact_eqn, planet0_phi_funcs, planet0_qqdot, planet0_qdotplus,␣
 ↪planet0_lambda = impact_eqns(["x1","y1","theta1",
                                                                          ␣
 ↪                       "x2","y2","theta2",
                                                                          ␣
 ↪                       "x3","y3","theta3"],
                                                                          ␣
 ↪                      {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":
 ↪grav0,"x_cen":x0_cen,"y_cen":y0_cen,"R":R0,
                                                                          ␣
 ↪                       "l2":l2,"w2":w2,"h2":h2,"rho2":rho2,
                                                                          ␣
 ↪                       "l3":l3,"w3":w3,"h3":h3,"rho3":rho3})

planet1_impact_eqn, planet1_phi_funcs, planet1_qqdot, planet1_qdotplus,␣
 ↪planet1_lambda = impact_eqns(["x1","y1","theta1",
                                                                          ␣
 ↪                       "x2","y2","theta2",
                                                                          ␣
 ↪                       "x3","y3","theta3"],
                                                                          ␣
 ↪                      {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":
 ↪grav1,"x_cen":x1_cen,"y_cen":y1_cen,"R":R1,
                                                                          ␣
 ↪                       "l2":l2,"w2":w2,"h2":h2,"rho2":rho2,
                                                                          ␣
 ↪                       "l3":l3,"w3":w3,"h3":h3,"rho3":rho3})

planet2_impact_eqn, planet2_phi_funcs, planet2_qqdot, planet2_qdotplus,␣
 ↪planet2_lambda = impact_eqns(["x1","y1","theta1",
                                                                          ␣
 ↪                       "x2","y2","theta2",
                                                                          ␣
 ↪                       "x3","y3","theta3"],
                                                                          ␣
 ↪                      {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":
 ↪grav2,"x_cen":x2_cen,"y_cen":y2_cen,"R":R2,
                                                                          ␣
 ↪                       "l2":l2,"w2":w2,"h2":h2,"rho2":rho2,
```

```
                                                                              ␣
↪                       "l3":l3,"w3":w3,"h3":h3,"rho3":rho3})

phi_funcs = [planet0_phi_funcs, planet1_phi_funcs, planet2_phi_funcs]
```

```
[6]: def impact_update(s, num_lambda, planet, threshold = 1e-10, alpha=1.0):

    _locals = locals()

    for i in range(0,len(s)):
        exec("planet{N}_impact_eqn=planet{N}_impact_eqn.
↪subs(planet{N}_qqdot[{idx}],s[{idx}])".
↪format(N=planet,idx=i),globals(),_locals)
    exec("impact_eqns_num=planet{N}_impact_eqn".
↪format(N=planet),globals(),_locals)
    exec("N=planet{N}_phi_funcs".format(N=planet),globals(),_locals)
    for i in range(0,len(_locals['N'])):
        if i != num_lambda:
            exec("impact_eqns_num=impact_eqns_num.
↪subs(planet{N}_lambda[{idx}],0)".format(N=planet,idx=i),globals(),_locals)
    exec("impact_eqns_num=simplify(impact_eqns_num)",globals(),_locals)
  ␣
↪exec("impact_sol=solve_eqns(impact_eqns_num,planet{N}_qdotplus+[planet{N}_lambda[{idx}]])".
↪format(N=planet,idx=num_lambda),globals(),_locals)
    impact_sol = _locals['impact_sol']

    num_vars = len(s)//2+1
    index_sol = None
    for i in range(len(impact_sol)//num_vars):
        if abs(impact_sol[(i+1)*num_vars-1]) > threshold:
            index_sol = i*num_vars
            break
    _locals['lambda_sol'] = impact_sol[index_sol+num_vars-1]
    exec("impact_eqns_new=impact_eqns_num.
↪subs(planet{N}_lambda[{idx}],alpha*lambda_sol)".
↪format(N=planet,idx=num_lambda),globals(),_locals)
    impact_eqns_new = _locals['impact_eqns_new']
    lhs = impact_eqns_new.lhs
    lhs = lhs.row_del(num_vars-1)
    rhs = impact_eqns_new.rhs
    rhs = rhs.row_del(num_vars-1)
    impact_eqns_new = Eq(lhs,rhs)
    _locals['impact_eqns_new'] = impact_eqns_new
    exec("impact_sol=solve_eqns(impact_eqns_new, planet{N}_qdotplus)".
↪format(N=planet),globals(),_locals)
```

```
    return np.array([*s[0:len(s)//2], *_locals['impact_sol']], dtype = float)
```

1.0.5 5 Simulation

[7]:
```
def simulate(f, funcs, phi_funcs, x0, tspan, dt, integrate, thres1, thres2, a=1.
↪0):

    N = int((max(tspan)-min(tspan))/dt)
    x = np.copy(x0)
    tvec = np.linspace(min(tspan),max(tspan),N)
    xtraj = np.zeros((len(x0),N))
    xtraj[:,0] = x
    for i in range(1,N):

        print(tvec[i],end='\r')

        ### IMPACT: PLANET 1 ###
        if (xtraj[:,i-1][0]-x1_cen)**2 + (xtraj[:,i-1][1]-y1_cen)**2 <=␣
↪R1_out**2 or (xtraj[:,i-1][3]-x1_cen)**2 + (xtraj[:,i-1][4]-y1_cen)**2 <=␣
↪R1_out**2 or (xtraj[:,i-1][6]-x1_cen)**2 + (xtraj[:,i-1][7]-y1_cen)**2 <=␣
↪R1_out**2:
            impact, num_lambda = impact_condition(xtraj[:
↪,i-1],phi_funcs[1],threshold=thres1)
            if not impact:
                impact, num_lambda = impact_condition(xtraj[:
↪,i-1],phi_funcs[1],threshold=thres2)
            if impact:
                x = impact_update(xtraj[:,i-1], num_lambda, 1, threshold=1e-6,␣
↪alpha=a)

        ### IMPACT: PLANET 2 ###
        if (xtraj[:,i-1][0]-x2_cen)**2 + (xtraj[:,i-1][1]-y2_cen)**2 <=␣
↪R2_out**2 or (xtraj[:,i-1][3]-x2_cen)**2 + (xtraj[:,i-1][4]-y2_cen)**2 <=␣
↪R2_out**2 or (xtraj[:,i-1][6]-x2_cen)**2 + (xtraj[:,i-1][7]-y2_cen)**2 <=␣
↪R2_out**2:
            impact, num_lambda = impact_condition(xtraj[:
↪,i-1],phi_funcs[2],threshold=thres1)
            if not impact:
                impact, num_lambda = impact_condition(xtraj[:
↪,i-1],phi_funcs[2],threshold=thres2)
            if impact:
                x = impact_update(xtraj[:,i-1], num_lambda, 2, threshold=1e-6,␣
↪alpha=a)

        ### IMPACT: SPACE ###
```

```
        impact, num_lambda = impact_condition(xtraj[:
↪,i-1],phi_funcs[0],threshold=thres1)
        if not impact:
            impact, num_lambda = impact_condition(xtraj[:
↪,i-1],phi_funcs[0],threshold=thres2)
        if impact:
            x = impact_update(xtraj[:,i-1], num_lambda, 0, threshold=1e-6,␣
↪alpha=1.0)

        ### BODY 1 ###
        body1_indices = [0,1,2,9,10,11]
        if (xtraj[:,i-1][0]-x1_cen)**2 + (xtraj[:,i-1][1]-y1_cen)**2 <=␣
↪R1_out**2:
            xtraj[body1_indices,i]=integrate(f,funcs[1],x[body1_indices],dt)
        elif (xtraj[:,i-1][0]-x2_cen)**2 + (xtraj[:,i-1][1]-y2_cen)**2 <=␣
↪R2_out**2:
            xtraj[body1_indices,i]=integrate(f,funcs[2],x[body1_indices],dt)
        else:
            xtraj[body1_indices,i]=integrate(f,funcs[0],x[body1_indices],dt)

        ### BODY 2 ###
        body2_indices = [3,4,5,12,13,14]
        if (xtraj[:,i-1][3]-x1_cen)**2 + (xtraj[:,i-1][4]-y1_cen)**2 <=␣
↪R1_out**2:
            xtraj[body2_indices,i]=integrate(f,funcs[1],x[body2_indices],dt)
        elif (xtraj[:,i-1][3]-x2_cen)**2 + (xtraj[:,i-1][4]-y2_cen)**2 <=␣
↪R2_out**2:
            xtraj[body2_indices,i]=integrate(f,funcs[2],x[body2_indices],dt)
        else:
            xtraj[body2_indices,i]=integrate(f,funcs[0],x[body2_indices],dt)

        ### BODY 3 ###
        body3_indices = [6,7,8,15,16,17]
        if (xtraj[:,i-1][6]-x1_cen)**2 + (xtraj[:,i-1][7]-y1_cen)**2 <=␣
↪R1_out**2:
            xtraj[body3_indices,i]=integrate(f,funcs[1],x[body3_indices],dt)
        elif (xtraj[:,i-1][6]-x2_cen)**2 + (xtraj[:,i-1][7]-y2_cen)**2 <=␣
↪R2_out**2:
            xtraj[body3_indices,i]=integrate(f,funcs[2],x[body3_indices],dt)
        else:
            xtraj[body3_indices,i]=integrate(f,funcs[0],x[body3_indices],dt)

        x = np.copy(xtraj[:,i])
    return xtraj
```

1.0.6 6 Tests

[8]:
```
### Missed Both Pigs ###
s0 = np.array([-8, 6.3, 0,
               6.5, -1, np.pi/6,
               5, -3, np.pi/4,
               19.3, 0, 1.2,
               0, 0, 0,
               0, 0, 0])
tspan = [0, 2.5]
dt = 0.001
traj1 = simulate(dyn, EOM_list, phi_funcs, s0, tspan, dt, rk4_integrate, 5e-2,␣
 ↪8e-2, a=1.0)
```

2.5989995998399368755

[9]:
```
### Two Pigs Hit ###
s0 = np.array([-8, 6.3, 0,
               6.5, -1, np.pi/6,
               5, -3, np.pi/4,
               18.7, 0, 1.2,
               0, 0, 0,
               0, 0, 0])
tspan = [0, 2.5]
dt = 0.0006
traj2 = simulate(dyn, EOM_list, phi_funcs, s0, tspan, dt, rk4_integrate, 5e-2,␣
 ↪8e-2, a=0.95)
```

2.5993997599039615465

[10]:
```
### One Pig Hit ###
s0 = np.array([-8, 6.3, 0,
               6.5, -1, np.pi/6,
               5, -3, np.pi/4,
               18.4, 0, 1.2,
               0, 0, 0,
               0, 0, 0])
tspan = [0, 2.5]
dt = 0.0006
traj3 = simulate(dyn, EOM_list, phi_funcs, s0, tspan, dt, rk4_integrate, 5e-2,␣
 ↪8e-2, a=1.0)
```

2.5993997599039615465

[11]:
```
### From Below Planet ###
s0 = np.array([-8, 6.3, 0,
               6.5, -1, np.pi/6,
               5, -3, np.pi/4,
               13, -16, 1.2,
```

```
                0, 0, 0,
                0, 0, 0])
tspan = [0, 2.5]
dt = 0.0006
traj4 = simulate(dyn, EOM_list, phi_funcs, s0, tspan, dt, rk4_integrate, 6e-2,␣
↪9e-2, a=1.0)
```

2.5993997599039615465

[12]:
```
### Split ###
s0 = np.array([-3, -11.3, np.pi/4,
                3.9, -2.9, -np.pi/6,
                5.25, -4.25, -np.pi/3,
                16.2, 16.7, 5.7,
                0, 0, 0,
                0, 0, 0])
tspan = [0, 1.5]
dt = 0.001
traj5 = simulate(dyn, EOM_list, phi_funcs, s0, tspan, dt, rk4_integrate, 6e-2,␣
↪9e-2, a=0.95)
```

1.5989993328885924665

[13]:
```
### Test w/ Alpha=0.85 ###
s0 = np.array([-8, 5.8, 0,
                3.9, -2.9, -np.pi/6,
                5.25, -4.25, -np.pi/3,
                7.5, 0, 2.5,
                0, 0, 0,
                0, 0, 0])
tspan = [0, 1.5]
dt = 0.001
traj6 = simulate(dyn, EOM_list, phi_funcs, s0, tspan, dt, rk4_integrate, 12e-2,␣
↪15e-2, a=0.85)
```

1.5989993328885924665

[14]:
```
### Test w/ Alpha=1.0 ###
s0 = np.array([-8, 5.8, 0,
                3.9, -2.9, -np.pi/6,
                5.25, -4.25, -np.pi/3,
                7.5, 0, 2.5,
                0, 0, 0,
                0, 0, 0])
tspan = [0, 1.5]
dt = 0.001
traj7 = simulate(dyn, EOM_list, phi_funcs, s0, tspan, dt, rk4_integrate, 12e-2,␣
↪15e-2, a=1.0)
```

```
1.5989993328885924665
```

[25]:
```
### Multiple Body Impacts ###
s0 = np.array([-8, 6.3, 0,
               -8, 4.7, np.pi/6,
               -8, -2, np.pi/4,
               9, 0, 5,
               7, -6, 4,
               9, 0, 4])
tspan = [0, 2.5]
dt = 0.001
traj8 = simulate(dyn, EOM_list, phi_funcs, s0, tspan, dt, rk4_integrate, 6e-2,␣
 ↪9e-2, a=1.0)
```

```
2.5989995998399368755
```

1.0.7 7 Animate

[26]:
```
select_traj = traj8
skip = len(select_traj[0])//500
sim_traj = np.array([select_traj[0][::skip],select_traj[1][::
 ↪skip],select_traj[2][::skip],
                    select_traj[3][::skip],select_traj[4][::
 ↪skip],select_traj[5][::skip],
                    select_traj[6][::skip],select_traj[7][::
 ↪skip],select_traj[8][::skip]])

animate(sim_traj, circles, T=10)
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
```

```python
def equations_of_motion(q, param_dict):

    _locals = locals()
    _locals['KE'] = 0
    _locals['V'] = 0
    _locals['x'] = []
    _locals['v'] = []
    _locals['a'] = []
    _locals['func_list'] = []
    N_bodies = len(q)//3

    for config_var in q:
        exec("{q0}=Function(r'{q0}')(t)\nx.append({q0})\n{q0}dot={q0}.diff(t)\nv.append({q0}dot)\n{q0}ddot={q0}dot.diff(t)\na.append({q0}ddot)".format(q0=config_var),globals(),_locals)
    i = q[0][1]
    exec("x{N}b,y{N}b,z{N}b=symbols(r'x{N}_b,y{N}_b,z{N}_b')\n".format(N=i),globals(),_locals)
    exec("m{N}=8*param_dict['l{N}']*param_dict['w{N}']*param_dict['h{N}']*param_dict['rho{N}']".format(N=i),globals(),_locals)
    exec("r{N}=Matrix([x{N}b,y{N}b,z{N}b])".format(N=i),globals(),_locals)
    exec("r{N}hat=hat_operation(r{N})".format(N=i),globals(),_locals)
    exec("integrand{N}=param_dict['rho{N}']*r{N}hat.T*r{N}hat".format(N=i),globals(),_locals)
    exec("I{N}=integrate(integrate(integrate(integrand{N},(x{N}b,-param_dict['l{N}'],param_dict['l{N}'])),(y{N}b,-param_dict['w{N}'],param_dict['w{N}'])),(z{N}b,-param_dict['h{N}'],param_dict['h{N}']))".format(N=i),globals(),_locals)
    exec("I6{N}=Matrix([[m{N}*sym.eye(3),sym.zeros(3)],[sym.zeros(3),I{N}]])".format(N=i),globals(),_locals)
    ### Transformations Between World and Body Frame ###
    exec("g_0{N}=g_tr(theta{N},[x{N},y{N},0])".format(N=i),globals(),_locals)
    ### End of Comment ###
    exec("COM{N}=g_0{N}*Matrix([0,0,0,1])".format(N=i),globals(),_locals)
    exec("KE+=(0.5*(Vb(g_0{N})).T*I6{N}*Vb(g_0{N}))[0]".format(N=i),globals(),_locals)
    exec("V+=-param_dict['grav']*m{N}*((COM{N}[0]-param_dict['x_cen'])**2+(COM{N}[1]-param_dict['y_cen'])**2)**(-0.5)".format(N=i),globals(),_locals)

    _locals['xv'] = _locals['x']+_locals['v']
    _locals['xva'] = _locals['xv']+_locals['a']
    _locals['L'] = simplify(_locals['KE'] - _locals['V'])
    exec("F_drag=-0.5*param_dict['rho_air']*param_dict['Cd']*param_dict['w{N}']*param_dict['l{N}']*(x{N}dot**2+y{N}dot**2)**(1.5)".format(N=i),globals(),_locals)
    _locals['EL_eqn'] = simplify(euler_equations(_locals['L'], _locals['xva'], _locals['F_drag'], symbols(r't')))
    exec("".join(config_var+'ddot_sol,' for config_var in q)+"=solve_eqns(EL_eqn,a)",globals(),_locals)

    for config_var in q:
        exec("{q0}ddot_func=lambdify([xv],{q0}ddot_sol)\nfunc_list.append({q0}ddot_func)".format(q0=config_var),globals(),_locals)

    return _locals['func_list']

space_EOM = equations_of_motion(["x1","y1","theta1"],
                                {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":grav0,"x_cen":x0_cen,"y_cen":y0_cen,"rho_air":rho_air0,"Cd":Cd0})

planet1_EOM = equations_of_motion(["x1","y1","theta1"],
                                {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":grav1,"x_cen":x1_cen,"y_cen":y1_cen,"rho_air":rho_air1,"Cd":Cd1})

planet2_EOM = equations_of_motion(["x1","y1","theta1"],
                                {"l1":l1,"w1":w1,"h1":h1,"rho1":rho1,"grav":grav2,"x_cen":x2_cen,"y_cen":y2_cen,"rho_air":rho_air2,"Cd":Cd2})

EOM_list = [space_EOM, planet1_EOM, planet2_EOM]
```

```python
def impact_eqns(q, param_dict):

    _locals = locals()
    _locals['KE'] = 0
    _locals['V'] = 0
    _locals['x'] = []
    _locals['v'] = []
    _locals['phi_eqns'] = []
    _locals['lambda_list'] = []
    _locals['qdotplus'] = []
    N_bodies = len(q)//3

    for config_var in q:
        exec("{q0}=symbols(r'{q0}')\nx.append({q0})\n{q0}dot=symbols(r'{q0}dot')\nv.append({q0}dot)".format(q0=config_var),globals(),_locals)
        exec("{q0}t=Function(r'{q0}')(t)\n{q0}tdot={q0}t.diff(t)".format(q0=config_var),globals(),_locals)
    for i in range(1,N_bodies+1):
        exec("x{N}b,y{N}b,z{N}b=symbols(r'x{N}_b,y{N}_b,z{N}_b')\n".format(N=i),globals(),_locals)
        exec("m{N}=8*param_dict['l{N}']*param_dict['w{N}']*param_dict['h{N}']*param_dict['rho{N}']".format(N=i),globals(),_locals)
        exec("r{N}=Matrix([x{N}b,y{N}b,z{N}b])".format(N=i),globals(),_locals)
        exec("r{N}hat=hat_operation(r{N})".format(N=i),globals(),_locals)
        exec("integrand{N}=param_dict['rho{N}']*r{N}hat.T*r{N}hat".format(N=i),globals(),_locals)
        exec("I{N}=integrate(integrate(integrate(integrand{N},(x{N}b,-param_dict['l{N}'],param_dict['l{N}'])), (y{N}b,-param_dict['w{N}'],param_dict['w{N}'])),(z{N}b,-param_dict['h{N}'],param_dict['h{N}']))".format(N=i),globals(),_locals)
        exec("I6{N}=Matrix([[m{N}*sym.eye(3),sym.zeros(3)],[sym.zeros(3),I{N}]])".format(N=i),globals(),_locals)
        ### Transformations Between World and Body Frame ###
        exec("g_0{N}=g_tr(theta{N}t,[x{N}t,y{N}t,0])".format(N=i),globals(),_locals)
        ### End of Comment ###
        exec("COM{N}=g_0{N}*Matrix([0,0,0,1])".format(N=i),globals(),_locals)
        exec("KE+=(0.5*(Vb(g_0{N})).T*I6{N}*Vb(g_0{N}))[0]".format(N=i),globals(),_locals)
        exec("V+=-param_dict['grav']*m{N}*((COM{N}[0]-param_dict['x_cen'])**2+(COM{N}[1]-param_dict['y_cen'])**2)**(-0.5)".format(N=i),globals(),_locals)
        exec("g_0{N}=g_tr(theta{N},[x{N},y{N},0])".format(N=i),globals(),_locals)
        exec("r{N}_c1=g_0{N}*Matrix([param_dict['l{N}'],param_dict['w{N}'],0,1])".format(N=i),globals(),_locals)
        exec("r{N}_c2=g_0{N}*Matrix([-param_dict['l{N}'],param_dict['w{N}'],0,1])".format(N=i),globals(),_locals)
        exec("r{N}_c3=g_0{N}*Matrix([-param_dict['l{N}'],-param_dict['w{N}'],0,1])".format(N=i),globals(),_locals)
        exec("r{N}_c4=g_0{N}*Matrix([param_dict['l{N}'],-param_dict['w{N}'],0,1])".format(N=i),globals(),_locals)
        if param_dict['R'] != 0:
            exec("phi_0{N}1=param_dict['R']**2-(r{N}_c1[0]-param_dict['x_cen'])**2-(r{N}_c1[1]-param_dict['y_cen'])**2\nphi_eqns.append(phi_0{N}1)".format(N=i),globals(),_locals)
            exec("phi_0{N}2=param_dict['R']**2-(r{N}_c2[0]-param_dict['x_cen'])**2-(r{N}_c2[1]-param_dict['y_cen'])**2\nphi_eqns.append(phi_0{N}2)".format(N=i),globals(),_locals)
            exec("phi_0{N}3=param_dict['R']**2-(r{N}_c3[0]-param_dict['x_cen'])**2-(r{N}_c3[1]-param_dict['y_cen'])**2\nphi_eqns.append(phi_0{N}3)".format(N=i),globals(),_locals)
            exec("phi_0{N}4=param_dict['R']**2-(r{N}_c4[0]-param_dict['x_cen'])**2-(r{N}_c4[1]-param_dict['y_cen'])**2\nphi_eqns.append(phi_0{N}4)".format(N=i),globals(),_locals)
```

```
def impact_update(s, num_lambda, planet, threshold = 1e-10, alpha=1.0):

    _locals = locals()

    for i in range(0,len(s)):
        exec("planet{N}_impact_eqn=planet{N}_impact_eqn.subs(planet{N}_qqdot[{idx}],s[{idx}])".format(N=planet,idx=i),globals(),_locals)
    exec("impact_eqns_num=planet{N}_impact_eqn".format(N=planet),globals(),_locals)
    exec("N=planet{N}_phi_funcs".format(N=planet),globals(),_locals)
    for i in range(0,len(_locals['N'])):
        if i != num_lambda:
            exec("impact_eqns_num=impact_eqns_num.subs(planet{N}_lambda[{idx}],0)".format(N=planet,idx=i),globals(),_locals)
    exec("impact_eqns_num=simplify(impact_eqns_num)",globals(),_locals)
    exec("impact_sol=solve_eqns(impact_eqns_num,planet{N}_qdotplus+[planet{N}_lambda[{idx}]])".format(N=planet,idx=num_lambda),globals(),_locals)
    impact_sol = _locals['impact_sol']

    num_vars = len(s)//2+1
    index_sol = None
    for i in range(len(impact_sol)//num_vars):
        if abs(impact_sol[(i+1)*num_vars-1]) > threshold:
            index_sol = i*num_vars
            break
    _locals['lambda_sol'] = impact_sol[index_sol+num_vars-1]
    exec("impact_eqns_new=impact_eqns_num.subs(planet{N}_lambda[{idx}],alpha*lambda_sol)".format(N=planet,idx=num_lambda),globals(),_locals)
    impact_eqns_new = _locals['impact_eqns_new']
    lhs = impact_eqns_new.lhs
    lhs = lhs.row_del(num_vars-1)
    rhs = impact_eqns_new.rhs
    rhs = rhs.row_del(num_vars-1)
    impact_eqns_new = Eq(lhs,rhs)
    _locals['impact_eqns_new'] = impact_eqns_new
    exec("impact_sol=solve_eqns(impact_eqns_new, planet{N}_qdotplus)".format(N=planet),globals(),_locals)

    return np.array([*s[0:len(s)//2], *_locals['impact_sol']], dtype = float)
```