```python
# -*- coding: utf-8 -*-
"""
Plot and animation context setup/initialization for Rigid Body class

Created on Tue Nov 11 16:05:05 2014

@author: whymranderson.blogspot.tw
"""
import numpy as np
print('RB animation and draw functions called')

from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d  #<- uncomment this, you need this

class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)

polynum = [3,4,5,6,3]    # indexing vertices of a closed rectangle

def CK(rotvec):
    '''Cayley-Klein parameters. Important: the built rotation matrix is with its couterclock
    This is basically the Rodriguez rotation formula in a matric form.
    '''
    amp = np.sqrt(np.dot(rotvec,rotvec))
    if amp == 0:
        ret = np.eye(3)
    else:
        axis = rotvec/amp
        phi = amp % (2*np.pi)
        a = np.cos(phi/2)
        b,c,d = axis*np.sin(phi/2)
        ret =  np.array([[a*a+b*b-c*c-d*d, 2*(b*c-a*d), 2*(b*d+a*c)],
                         [2*(b*c+a*d), a*a+c*c-b*b-d*d, 2*(c*d-a*b)],
                         [2*(b*d-a*c), 2*(c*d+a*b), a*a+d*d-b*b-c*c]])
    return ret
```

```
def get111norm(xyzaxes):
    '''Return array of body (1,1,1) unit vector as a function of time N in size (N,3)
    '''
    vec = np.sum(xyzaxes,axis=2)
    vecnorm = np.array([vec[i,:]/np.linalg.norm(vec[i,:]) for i in range(len(vec[:,0]))])
    return vecnorm

def CalDiffInDeg(cordvec):
    '''Calculate and return angle difference between A and B method's 111 unit vectors'''
    DCM111norm = get111norm(cordvec[:,:,:3])
    Has111norm = get111norm(cordvec[:,:,7:10])
    print(np.shape(DCM111norm),np.shape(Has111norm))
    diffindeg = np.degrees([np.arccos(np.dot(DCM111norm[i,:],Has111norm[i,:])) for i in rang
#    print(np.shape(diffindeg))
    return diffindeg

def CalDiffInDegBC(cordvec):
    '''Calculate and return angle difference between B and C method's 111 unit vectors'''
    dDCM111norm = get111norm(cordvec[:,:,10:13])
    Has111norm = get111norm(cordvec[:,:,7:10])
    print(np.shape(dDCM111norm),np.shape(Has111norm))
    BCdiffindeg = np.degrees([np.arccos(np.dot(dDCM111norm[i,:],Has111norm[i,:])) for i in r
    print(np.shape(BCdiffindeg))
    return BCdiffindeg

def getPlotAnimationLines(RBO):
    # no use. '''Construct and return the line animation data sequence in a rigid body obje
    RBO.AlineCMxAxis = np.hstack([RBO.cordvec[:,:,2]/2,(RBO.cordvec[:,:,0]/2+RBO.cordvec[:,
    RBO.AlineCMyAxis = np.hstack([RBO.cordvec[:,:,2]/2,(RBO.cordvec[:,:,1]/2+RBO.cordvec[:,
    RBO.AlineCMzAxis = np.hstack([np.zeros((RBO.N+1,3)),RBO.cordvec[:,:,2]])
    RBO.SquareEdge3 = np.hstack([RBO.cordvec[:,:,3],RBO.cordvec[:,:,4]])
    RBO.SquareEdge4 = np.hstack([RBO.cordvec[:,:,4],RBO.cordvec[:,:,5]])
    RBO.SquareEdge5 = np.hstack([RBO.cordvec[:,:,5],RBO.cordvec[:,:,6]])
    RBO.SquareEdge6 = np.hstack([RBO.cordvec[:,:,6],RBO.cordvec[:,:,3]])
    RBO.BlineCMxAxis = np.hstack([RBO.cordvec[:,:,9]/2,(RBO.cordvec[:,:,7]/2+RBO.cordvec[:,
    RBO.BlineCMyAxis = np.hstack([RBO.cordvec[:,:,9]/2,(RBO.cordvec[:,:,8]/2+RBO.cordvec[:,
    RBO.BlineCMzAxis = np.hstack([np.zeros((RBO.N+1,3)),RBO.cordvec[:,:,9]])
    RBO.lineLData = np.hstack([np.zeros((RBO.N+1,3)),RBO.L_plot])
    RBO.w_bnormData = np.hstack([np.zeros((RBO.N+1,3)),RBO.w_lab_norm])
    RBO.ClineCMxAxis = np.hstack([RBO.cordvec[:,:,12]/2,(RBO.cordvec[:,:,10]/2+RBO.cordvec[
    RBO.ClineCMyAxis = np.hstack([RBO.cordvec[:,:,12]/2,(RBO.cordvec[:,:,11]/2+RBO.cordvec[
    RBO.ClineCMzAxis = np.hstack([np.zeros((RBO.N+1,3)),RBO.cordvec[:,:,12]])


sphere_color = 'lightgray'#'#FFDDDD'
sphere_alpha = 0.1
```

```
frame_color = 'darkgray'
frame_alpha = 0.4
frame_width = 1
sphere_grid = 25
wirestride = 4 # this affects the drawing speed, if less than 4

def plot_lucid_sphere(axes):
    '''plot the lucid sphere'''
    u = np.linspace(-np.pi, 0, sphere_grid)
    v = np.linspace(0, np.pi, sphere_grid)
    x = np.outer(np.cos(u), np.sin(v))
    y = np.outer(np.sin(u), np.sin(v))
    z = np.outer(np.ones(np.size(u)), np.cos(v))
    axes.plot_surface(x, y, z, rstride=2, cstride=2,
                           color=sphere_color, linewidth=0,
                           alpha=sphere_alpha)
    axes.plot_surface(-x, -y, z, rstride=2, cstride=2,
                           color=sphere_color, linewidth=0,
                           alpha=sphere_alpha)

    # wireframe
    axes.plot_wireframe(x, y, z, rstride=wirestride, cstride=wirestride,
                             color=frame_color,
                             alpha=frame_alpha)
    axes.plot_wireframe(-x, -y, z, rstride=wirestride, cstride=wirestride,
                             color=frame_color,
                             alpha=frame_alpha)
#    # equator
#    axes.plot(1.0 * np.cos(u), 1.0 * np.sin(u),
#                   zs=0, zdir='z', lw=frame_width,
#                   color=frame_color)
#    axes.plot(1.0 * np.cos(u), 1.0 * np.sin(u),
#                   zs=0, zdir='x', lw=frame_width,
#                   color=frame_color)


def plot_support_bar(axes):
    '''plot the support bar and disk'''
    support_disk_radius = 0.2
    support_bar_length = 0.8
    axes.plot([0,0],[0,0],[-support_bar_length,0],'g-',lw=5)
    ud = np.linspace(-np.pi, 0, sphere_grid)
    vd = np.linspace(0, np.pi, sphere_grid)
    xd = support_disk_radius*np.outer(np.cos(ud), np.sin(vd))
    yd = support_disk_radius*np.outer(np.sin(ud), np.sin(vd))
    zd = np.zeros((np.size(ud),np.size(vd)))-support_bar_length
```

```
    axes.plot_surface(xd, yd, zd, rstride=2, cstride=2,
                           color='r', linewidth=0,
                           alpha=1)
    axes.plot_surface(-xd, -yd, zd, rstride=2, cstride=2,
                           color='r', linewidth=0,
                           alpha=1)


def plot_xy_plane(axes):
    '''plot the xy plane'''
    u = np.linspace(-np.pi, 0, sphere_grid)
    v = np.linspace(0, np.pi, sphere_grid)
    x = np.outer(np.cos(u), np.sin(v))
    y = np.outer(np.sin(u), np.sin(v))
    z = np.zeros((np.size(u),np.size(v)))
    axes.plot_surface(x, y, z, rstride=2, cstride=2,
                           color=sphere_color, linewidth=0,
                           alpha=sphere_alpha)
    axes.plot_surface(-x, -y, z, rstride=2, cstride=2,
                           color=sphere_color, linewidth=0,
                           alpha=sphere_alpha)

    # wireframe
    axes.plot_wireframe(x, y, z, rstride=2, cstride=2,
                             color='green',
                             alpha=0.2)#frame_alpha)
    axes.plot_wireframe(-x, -y, z, rstride=2, cstride=2,
                             color='green',
                             alpha=0.2)#frame_alpha)

def update_line_multiple_segments(x,*arg):
    '''3D Animation function. Take multiple segments line data.'''
    arglen = len(arg)
    RBO = arg[0]
    cubeaf = RBO.construct_cube_4wires_method(x,RBO.display_ABC_cube)
    for i in np.arange(1,5):
        line3DObject = arg[i]
        for linei,lineid in zip(line3DObject,line3DObject):
            linei.set_data(cubeaf[0,4*(i-1):4*i],
                           cubeaf[1,4*(i-1):4*i])
            linei.set_3d_properties(cubeaf[2,4*(i-1):4*i])
    for i in np.arange(5,arglen,2):
        line3DObject = arg[i]
        line3DData = arg[i+1]
        for linei,lineid in zip(line3DObject,line3DObject):
            linei.set_data([line3DData[x,0],line3DData[x,3]],
```

```
                        [line3DData[x,1],line3DData[x,4]])
            linei.set_3d_properties([line3DData[x,2],line3DData[x,5]])

def update_line_one_segments(x,*arg):
    '''3D Animation function. Take one segment line data.'''
    arglen = len(arg)
    for i in np.arange(1,arglen,2):
        line3DObject = arg[i]
        line3DData = arg[i+1]
        for linei,lineid in zip(line3DObject,line3DObject):
            linei.set_data([line3DData[x,0],line3DData[x,3]],
                        [line3DData[x,1],line3DData[x,4]])
            linei.set_3d_properties([line3DData[x,2],line3DData[x,5]])

def InitFirstAnimationFrame(RBO,ax,plt):
    # no use. plot the initial body xyz axes DCM method
    cordvec,Lzppnorm,L_plot,w_lab,w = RBO.cordvec,RBO.Lzppnorm,RBO.L_plot,RBO.w_lab,RBO.w
    #RBO.baxes = [ax.plot(*[ [0+cordvec[0,j,2]/2,(cordvec[0,j,i]+cordvec[0,j,2])/2] for j in
    #plt.setp(RBO.baxes[2],marker='o') #set z axis marker
    # unpacking list to tuple and list comprehension, see below
    # baxes[1]=ax.plot([0,cordvec[0,0,0]],[0,cordvec[0,1,0]],[0,cordvec[0,2,0]]), plot x ax

    # plot the initial body xyz axes from Hasbun's euler angle method
    #RBO.baxes_hasbun = [ax.plot(*[ [0+cordvec[0,j,9]/2,(cordvec[0,j,i]+cordvec[0,j,9])/2] j
    #plt.setp(RBO.baxes_hasbun,linestyle=':') #set z axis marker

    # Color of the rectangle
    ##
    import matplotlib.cm as mplcm
    import matplotlib.colors as colors
    cm = plt.get_cmap('Oranges')
    cNorm  = colors.Normalize(vmin=-1, vmax=3)
    scalarMap = mplcm.ScalarMappable(norm=cNorm, cmap=cm)
    ax.set_color_cycle([scalarMap.to_rgba(i) for i in range(4)])
    ##
    #

    # plot the initial square box
    '''
    RBO.polylines = [ ax.plot(*[ [cordvec[0,j,polynum[i]],
                              cordvec[0,j,polynum[i+1]]] for j in range(3) ]
                        ) for i in range(4) ]
    plt.setp(RBO.polylines,lw=5)
    '''
    #change previous w(t_i-1) for w(t i-1) comparison
    linetrace=ax.plot(cordvec[:,0,2],cordvec[:,1,2],cordvec[:,2,2],'b-',
```

```
                              markersize=1,label='DCM with $\omega(t_i)$')

    #lineLtrance=ax.plot(L_plot[:,0],L_plot[:,1],L_plot[:,2],'k.',markersize=2)
    #HasbunLtrance=ax.plot(Lzppnorm[:,0],Lzppnorm[:,1],Lzppnorm[:,2],'k-',
    #                 markersize=2,label='£\omega(t_{i+1})£ and Hasbun\'s result')
#    ax.legend()
    '''
    RBO.lineL=ax.plot([0,L_plot[0,0]],
                  [0,L_plot[0,1]],
                  [0,L_plot[0,2]],'k-')
    '''
    #HasbunL=ax.plot([0,L_plot[0,0]],
    #               [0,L_plot[0,1]],
    #               [0,L_plot[0,2]],'k-')

    # plot angular velocity vector normalized to w(t0)--
    RBO.w_lab_norm = w_lab/w[0,2]
    #print(w[0,2],np.linalg.norm(w_lab[0,:]))
    '''
    RBO.lineW=ax.plot([0,RBO.w_lab_norm[0,0]],
                  [0,RBO.w_lab_norm[0,1]],
                  [0,RBO.w_lab_norm[0,2]],'g-')
    '''
# ----

def Animation_Setup(RBO,ax,plt,istep):
    '''DrawOption['name of observable']=True attributes will be read and corresponding
    vectors or trails of the observable will be collected. The animation sequence
    data are also contructed here.'''

    if RBO.DrawOption['xy_plane'] == True:
        plot_xy_plane(ax)

    # plot the initial body xyz axes DCM method
    cordvec = RBO.cordvec
    if RBO.DrawOption['A_axes'] == True:
        RBO.baxes =[ax.plot(*[ [cordvec[istep,j,2]/2,(cordvec[istep,j,i]/4+cordvec[istep,j,2
        # explaination for the above command: unpacking list to tuple and list comprehension
        # baxes[1]=ax.plot([0,cordvec[0,0,0]],[0,cordvec[0,1,0]],[0,cordvec[0,2,0]]), plot x
        plt.setp(RBO.baxes[2],marker='o') #set z axis marker
        RBO.AlineCMxAxis = np.hstack([RBO.cordvec[:,:,2]/2,(RBO.cordvec[:,:,0]/4+RBO.cordvec
        RBO.AlineCMyAxis = np.hstack([RBO.cordvec[:,:,2]/2,(RBO.cordvec[:,:,1]/4+RBO.cordvec
        RBO.AlineCMzAxis = np.hstack([np.zeros((RBO.N+1,3)),RBO.cordvec[:,:,2]])
        RBO.linesarg = RBO.linesarg + (RBO.baxes[0],RBO.AlineCMxAxis,
                                       RBO.baxes[1],RBO.AlineCMyAxis,
                                        RBO.baxes[2],RBO.AlineCMzAxis)
```

```
    # plot the initial body xyz axes from Hasbun's euler angle method
    if RBO.DrawOption['B_axes'] == True:
        RBO.baxes_hasbun = [ax.plot(*[ [0+cordvec[0,j,9]/2,(cordvec[0,j,i]+cordvec[0,j,9])/2
        plt.setp(RBO.baxes_hasbun[2],linestyle='-',marker='o')
        RBO.BlineCMxAxis = np.hstack([RBO.cordvec[:,:,9]/2,(RBO.cordvec[:,:,7]/2+RBO.cordvec
        RBO.BlineCMyAxis = np.hstack([RBO.cordvec[:,:,9]/2,(RBO.cordvec[:,:,8]/2+RBO.cordvec
        RBO.BlineCMzAxis = np.hstack([np.zeros((RBO.N+1,3)),RBO.cordvec[:,:,9]])
        RBO.linesarg = RBO.linesarg + (RBO.baxes_hasbun[0],RBO.BlineCMxAxis,
            RBO.baxes_hasbun[1],RBO.BlineCMyAxis,
            RBO.baxes_hasbun[2],RBO.BlineCMzAxis)

    if RBO.DrawOption['C_axes'] == True:
        RBO.baxes_C =[ax.plot(*[ [cordvec[0,j,12]/2,(cordvec[0,j,i]/2+cordvec[0,j,9])/2] for
#        RBO.baxes_C= [ax.plot(*[ [0+cordvec[0,j,12]/2,(cordvec[0,j,i]+cordvec[0,j,9])/2] fo
        plt.setp(RBO.baxes_C,linestyle='--')
        RBO.ClineCMxAxis = np.hstack([RBO.cordvec[:,:,12]/2,(RBO.cordvec[:,:,10]/2+RBO.cordv
        RBO.ClineCMyAxis = np.hstack([RBO.cordvec[:,:,12]/2,(RBO.cordvec[:,:,11]/2+RBO.cordv
        RBO.ClineCMzAxis = np.hstack([np.zeros((RBO.N+1,3)),RBO.cordvec[:,:,12]])
        RBO.linesarg = RBO.linesarg + (RBO.baxes_C[0],RBO.ClineCMxAxis,
            RBO.baxes_C[1],RBO.ClineCMyAxis,
            RBO.baxes_C[2],RBO.ClineCMzAxis)

    # Color of the rectangle
    ##
    import matplotlib.cm as mplcm
    import matplotlib.colors as colors
    cm = plt.get_cmap('Oranges')
    cNorm  = colors.Normalize(vmin=-1, vmax=3)
    scalarMap = mplcm.ScalarMappable(norm=cNorm, cmap=cm)
    ax.set_color_cycle([scalarMap.to_rgba(i) for i in range(4)])
    ##
    #

    # plot the initial square box
    if RBO.DrawOption['A_square'] == True:
        RBO.polylines = [ ax.plot(*[ [cordvec[istep,j,polynum[i]],
                                      cordvec[istep,j,polynum[i+1]]] for j in range(3) ]
                                      ) for i in range(4) ]
        plt.setp(RBO.polylines,lw=5)
        RBO.SquareEdge3 = np.hstack([RBO.cordvec[:,:,3],RBO.cordvec[:,:,4]])
        RBO.SquareEdge4 = np.hstack([RBO.cordvec[:,:,4],RBO.cordvec[:,:,5]])
        RBO.SquareEdge5 = np.hstack([RBO.cordvec[:,:,5],RBO.cordvec[:,:,6]])
        RBO.SquareEdge6 = np.hstack([RBO.cordvec[:,:,6],RBO.cordvec[:,:,3]])
        RBO.linesarg = RBO.linesarg+ (RBO.polylines[0],RBO.SquareEdge3,
```

```
                                    RBO.polylines[1],RBO.SquareEdge4,
                                    RBO.polylines[2],RBO.SquareEdge5,
                                    RBO.polylines[3],RBO.SquareEdge6)

# plot the initial cube
if RBO.DrawOption['A_cube'] == True:
    RBO.display_ABC_cube = 0
    testout = RBO.construct_cube_4wires_method(istep,RBO.display_ABC_cube)
    cubeinst1 = ax.plot(*testout[:,0:4],c='black',linewidth=2)
    cubeinst2 = ax.plot(*testout[:,4:8],c='gray',linewidth=2)
    cubeinst3 = ax.plot(*testout[:,8:12],c='lightgray',linewidth=2)
    cubeinst4 = ax.plot(*testout[:,12:16],c='darkgray',linewidth=2)
    #RBPlot.plot_body_space_cone(ax4,b.cordvec[iframe,:,2],b.w_lab_norm[iframe,:],b.L_p
    RBO.cube_4_lines = (cubeinst1,cubeinst2,cubeinst3,cubeinst4)

if RBO.DrawOption['B_cube'] == True:
    RBO.display_ABC_cube = 7

    testout = RBO.construct_cube_4wires_method(istep,RBO.display_ABC_cube)
    cubeinst1 = ax.plot(*testout[:,0:4],c='black',linewidth=2)
    cubeinst2 = ax.plot(*testout[:,4:8],c='gray',linewidth=2)
    cubeinst3 = ax.plot(*testout[:,8:12],c='lightgray',linewidth=2)
    cubeinst4 = ax.plot(*testout[:,12:16],c='darkgray',linewidth=2)
    #RBPlot.plot_body_space_cone(ax4,b.cordvec[iframe,:,2],b.w_lab_norm[iframe,:],b.L_p
    RBO.cube_4_lines = (cubeinst1,cubeinst2,cubeinst3,cubeinst4)

if RBO.DrawOption['C_cube'] == True:
    RBO.display_ABC_cube = 10




#change previous w(t_i-1) for w(t i-1) comparison
trailsamplerate = 1
if RBO.DrawOption['A_z_axis_trace'] == True:
    zlineAtrace=ax.plot(cordvec[::trailsamplerate,0,2],
                        cordvec[::trailsamplerate,1,2],
                        cordvec[::trailsamplerate,2,2],'b-',
    linewidth=2,label='A method with $\omega(t_{i+1})$ and B method')
if RBO.DrawOption['A_Angular Momentum Trace'] == True:
    lineLtrance=ax.plot(RBO.L_plot[::trailsamplerate,0],
                        RBO.L_plot[::trailsamplerate,1],
                        RBO.L_plot[::trailsamplerate,2],
                        'k-',markersize=2)
if RBO.DrawOption['B_z_axis_trace'] == True:
    Lzppnorm = RBO.Lzppnorm
```

```
        HasbunZtrace=ax.plot(Lzppnorm[:,0],Lzppnorm[:,1],Lzppnorm[:,2],'k-',
                     linewidth=2,label='$\omega(t_{i+1})$ and Hasbun\'s result')
    if RBO.DrawOption['B_Angular Momentum Trace'] == True:
        L_Bnorm = RBO.L_Bnorm
        L_Btraceline=ax.plot(RBO.L_Bnorm[:,0],RBO.L_Bnorm[:,1],RBO.L_Bnorm[:,2],'k.',
                     markersize=1)
    if RBO.DrawOption['C_z_axis_trace'] == True:
        zlineCtrace=ax.plot(cordvec[::trailsamplerate,0,12],
                            cordvec[::trailsamplerate,1,12],
                            cordvec[::trailsamplerate,2,12],'g-',
                          linewidth=1,label='C method')

#    ax.legend()
    if RBO.DrawOption['A_Angular Momentum Vec'] == True:
        L_plot = RBO.L_plot

        RBO.lineL=ax.plot([0,L_plot[istep,0]],
                          [0,L_plot[istep,1]],
                          [0,L_plot[istep,2]],'k-',marker='o')
#        print RBO.lineL
#        fanArrow3d=Arrow3D([0,L_plot[istep,0]],
#                          [0,L_plot[istep,1]],
#                          [0,L_plot[istep,2]],arrowstyle="-|>", color="b")
#        #RBO.lineL=ax.add_artist(fanArrow3d)
#        print fanArrow3d
        RBO.lineLAData = np.hstack([np.zeros((RBO.N+1,3)),RBO.L_plot])
        RBO.linesarg = RBO.linesarg + (RBO.lineL,RBO.lineLAData)

    if RBO.DrawOption['B_Angular Momentum Vec'] == True:
        RBO.lineL=ax.plot([0,L_Bnorm[0,0]],
                          [0,L_Bnorm[0,1]],
                          [0,L_Bnorm[0,2]],'r:',marker='o')
        #shape of L_Bnorm is (N,3), one vector less than N+1
        RBO.lineLBData = np.hstack([np.zeros((RBO.N+1,3)),RBO.L_Bnorm])
        RBO.linesarg = RBO.linesarg + (RBO.lineL,RBO.lineLBData)

    #RBO.HasbunL=ax.plot([0,L_plot[0,0]],
    #               [0,L_plot[0,1]],
     #              [0,L_plot[0,2]],'k-')

    # plot angular velocity vector normalized to w(t0)--
    #print(w[0,2],np.linalg.norm(w_lab[0,:]))
    if RBO.DrawOption['A_Angular Velocity Vec'] == True:
        RBO.w_lab_norm = RBO.w_lab/RBO.w[0,2]
        RBO.lineW=ax.plot([0,RBO.w_lab_norm[istep,0]],
                          [0,RBO.w_lab_norm[istep,1]],
```

```
                        [0,RBO.w_lab_norm[istep,2]],'g-',marker='o')
        RBO.w_bnormData = np.hstack([np.zeros((RBO.N+1,3)),RBO.w_lab_norm])
        RBO.linesarg = RBO.linesarg + (RBO.lineW,RBO.w_bnormData)

    if RBO.DrawOption['B_Angular Velocity Vector (normalized to t0 value)'] == True:
        RBO.w_lab_norm_hasbun = RBO.w_lab_hasbun/RBO.w[0,2]
        lineW=ax.plot([0,RBO.w_lab_norm_hasbun[istep,0]],
                        [0,RBO.w_lab_norm_hasbun[istep,1]],
                        [0,RBO.w_lab_norm_hasbun[istep,2]],'m-',marker='o')
        w_lab_hasbun_normData = np.hstack([np.zeros((RBO.N+1,3)),RBO.w_lab_norm_hasbun])
        RBO.linesarg = RBO.linesarg + (lineW,w_lab_hasbun_normData)

    if RBO.DrawOption['B_contact_force_vector'] == True:
        draw_contact_force_vectors_diagram(RBO,istep,ax)

    if RBO.DrawOption['Torque_vector_A_method'] == True:
        RBO.Tau_lab_A_method
        start_of_vec = RBO.L_plot
        end_of_vec = RBO.Tau_lab_A_method + RBO.L_plot
        lineTau=ax.plot([start_of_vec[0,0],end_of_vec[0,0]],
                        [start_of_vec[0,1],end_of_vec[0,1]],
                        [start_of_vec[0,2],end_of_vec[0,2]],'m-',marker='o')
        tau_lab_A_method = np.hstack([start_of_vec,end_of_vec])
        RBO.linesarg = RBO.linesarg + (lineTau,tau_lab_A_method)

def mplot3d_animation(RBO,iframe):
    '''Setup 3D window and prepare animation sequence from the solved
    rigid body object. Plot a lucid sphere and static istep-th keyframe as the
    initial frame as it is needed for mplot3D manager. Then call mplot3d
    funcanimation manager.'''
    import matplotlib.pyplot as plt
    fig2 = plt.figure(2,figsize=(6,6))
    import mpl_toolkits.mplot3d as p3
    ax = p3.Axes3D(fig2)
    ax.set_axis_off()
    ax.view_init(elev=RBO.view_overlook_angle, azim=RBO.view_azim_angle)
    factor = 0.6
    ax.set_xlim3d([-1*factor, 1*factor])
    ax.set_ylim3d([-1*factor, 1*factor])
    ax.set_zlim3d([-1*factor, 1*factor])
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_zticks([])
    #ax.autoscale_view(tight=True, scalex=True, scaley=True, scalez=True)
    plot_lucid_sphere(ax)
    plot_support_bar(ax)
```

```
    #AnimationFrameSetUp(RBO,ax,plt,iframe)
    Animation_Setup(RBO,ax,plt,iframe)

    #uncomment here to save z motion for drawing gyro logo
    #np.save('outfile', b.cordvec[::20,:20,2])
    import matplotlib.animation as animation

    if RBO.DrawOption['A_cube'] | RBO.DrawOption['B_cube'] | RBO.DrawOption['C_cube'] == Fal
        line_ani = animation.FuncAnimation(fig2, update_line_one_segments,
                                list(range(1,RBO.N,RBO.framerate)),
                                fargs=(RBO,)+ RBO.linesarg,
                                       interval=10, blit=False,repeat=True)
    else:
        line_ani = animation.FuncAnimation(fig2, update_line_multiple_segments,
                                list(range(1,RBO.N,RBO.framerate)),
                                fargs=(RBO,)+ RBO.cube_4_lines + RBO.linesarg,
                                       interval=30, blit=False,repeat=True)
    return line_ani



def draw_contact_force_vectors_diagram(RBO,istep,ax):
    '''animate the vectors of forces acting on gyro's center of mass'''
    max_amplitude = np.amax(np.linalg.norm(RBO.F_cp[1:-2,:],axis=1))
    #exclude first and last value of F_cp due to boundary condition on derivative
    print max_amplitude
    #plot contact force
    line_F_cp=ax.plot([0,RBO.F_cp[istep,0]],
                      [0,RBO.F_cp[istep,1]],
                      [0,RBO.F_cp[istep,2]],'b-',marker='o')
    F_cp_normData = np.hstack([np.zeros((len(RBO.F_cp[:,0]),3)),RBO.F_cp/max_amplitude])
    RBO.linesarg = RBO.linesarg + (line_F_cp,F_cp_normData)
    #plot contact force vertical dash line
    line_F_cp_v=ax.plot([RBO.F_cp[istep,0],RBO.F_cp[istep,0]],
                        [RBO.F_cp[istep,1],RBO.F_cp[istep,1]],
                        [RBO.F_cp[istep,2],0                ],'b:',marker='o')
    F_cp_v_normData = np.hstack([RBO.F_cp/max_amplitude,
                              np.hstack([ RBO.F_cp[:,:-1]/max_amplitude,np.zeros((len(RBO
                              ])
    RBO.linesarg = RBO.linesarg + (line_F_cp_v,F_cp_v_normData)

    #plot contact force horizontal dash line
    line_F_cp_h=ax.plot([0,RBO.F_cp[istep,0]],
                        [0,RBO.F_cp[istep,1]],
                        [0,0],'b:',marker='o')
```

```
    F_cp_h_normData = np.hstack([np.zeros((len(RBO.F_cp[:,0]),3)),
                                np.hstack([ RBO.F_cp[:,:-1]/max_amplitude,np.zeros((len(RBO.
                                 ])
    RBO.linesarg = RBO.linesarg + (line_F_cp_h,F_cp_h_normData)
    '''
    #plot contact force along radial direction
    line_F_contact_radial=ax.plot([0,RBO.F_contact_radial[istep,0]],
                                  [0,RBO.F_contact_radial[istep,1]],
                                  [0,RBO.F_contact_radial[istep,2]],'r-',marker='x')
    F_contact_radial_normData = np.hstack([np.zeros((len(RBO.F_contact_radial[:,0]),3)),RBO.
    RBO.linesarg = RBO.linesarg + (line_F_contact_radial,F_contact_radial_normData)

    #plot contact force tangent to CM trajectory
    line_F_contact_tan=ax.plot([0,RBO.F_contact_tan[istep,0]],
                               [0,RBO.F_contact_tan[istep,1]],
                               [0,RBO.F_contact_tan[istep,2]],'r-',marker='d')
    F_contact_tan_normData = np.hstack([np.zeros((len(RBO.F_contact_tan[:,0]),3)),RBO.F_con
    RBO.linesarg = RBO.linesarg + (line_F_contact_tan,F_contact_tan_normData)
    '''
    #plot gravity force vector
    line_F=ax.plot([RBO.cordvec[istep,0,2]*0.09*np.sqrt(12)/2,RBO.cordvec[istep,0,2]*0.09*np
                   [RBO.cordvec[istep,1,2]*0.09*np.sqrt(12)/2,RBO.cordvec[istep,1,2]*0.09*np
                   [RBO.cordvec[istep,2,2]*0.09*np.sqrt(12)/2,RBO.cordvec[istep,2,2]*0.09*np
    F_normData = np.hstack([RBO.cordvec[:,:,2]*0.09*np.sqrt(12)/2,RBO.F/max_amplitude+RBO.co
    RBO.linesarg = RBO.linesarg + (line_F,F_normData)

    #plot contact force trail's projection onto horizontal plane
    if RBO.DrawOption['CF_trail'] == True:
        line_F_cp_h_trail = ax.plot(RBO.F_cp[1::3,0]/max_amplitude,
                                    RBO.F_cp[1::3,1]/max_amplitude,
                                    RBO.F_cp[1::3,2]/max_amplitude,'r:',marker='.')

    #plot average force
    #line_average_force = ax.plot()

def rotmat_from_A_2_B(A,B):
    '''Return the active r.h. rotation matrix constructed from rotation vector :math:`C=A\\t
    rotunit = np.cross(A, B)
    rotunit = rotunit/np.linalg.norm(rotunit)
    thetarot = np.arccos(np.dot(A/np.linalg.norm(A),B/np.linalg.norm(B)))
    rotmat = CK(thetarot*rotunit)
    return rotmat

def plot_body_space_cone(ax,zn,omegavec,Lvec):
    '''
    Plot the body and space cone. A static one-time plot.
```

```
    '''
    # Set up the grid in polar coordinate theta, radius
    cone_theta = np.linspace(0,2*np.pi,90)
    bcone_radius = np.abs(np.linalg.norm(zn))*np.sqrt(1-
    np.square(np.dot(zn,omegavec/np.linalg.norm(omegavec)))
    )
    bcone_len = np.abs(np.linalg.norm(zn))*np.dot(zn,omegavec/np.linalg.norm(omegavec))

    cone_r = np.linspace(0,bcone_radius,15)
    c1T, c1R = np.meshgrid(cone_theta, cone_r)
    # Then calculate X, Y, and Z
    c1X = c1R * np.cos(c1T)
    c1Y = c1R * np.sin(c1T)
    c1Z = np.sqrt(c1X**2 + c1Y**2)/bcone_radius*bcone_len
    cmm,cnn = np.shape(c1Z)
    mat_zn2omega = rotmat_from_A_2_B(np.array([0,0,1]),zn)
    for i in range(cmm):
        for j in range(cnn):
            c1X[i,j],c1Y[i,j],c1Z[i,j] = np.dot(mat_zn2omega,
            np.array([c1X[i,j],c1Y[i,j],c1Z[i,j]]))
    # space cone
    scone_radius = np.abs(np.linalg.norm(omegavec))*np.sqrt(1-
    np.square(np.dot(Lvec/np.linalg.norm(Lvec),omegavec/np.linalg.norm(omegavec)))
    )
    scone_len = np.abs(np.linalg.norm(omegavec))*np.dot(Lvec/np.linalg.norm(Lvec),omegavec/n

    cone_r_s = np.linspace(0,scone_radius,15)
    c2T, c2R = np.meshgrid(cone_theta, cone_r_s)
    # Then calculate X, Y, and Z
    c2X = c2R * np.cos(c2T)
    c2Y = c2R * np.sin(c2T)
    c2Z = np.sqrt(c2X**2 + c2Y**2)/scone_radius*scone_len
    cmm2,cnn2 = np.shape(c2Z)
    mat_omega2L = rotmat_from_A_2_B(np.array([0,0,1]),Lvec)
    for i in range(cmm2):
        for j in range(cnn2):
            c2X[i,j],c2Y[i,j],c2Z[i,j] = np.dot(mat_omega2L,
            np.array([c2X[i,j],c2Y[i,j],c2Z[i,j]]))


    # Set the Z values outside your range to NaNs so they aren't plotted
#    c1Z[c1Z < 0] = np.nan
#    c1Z[c1Z > 2.1] = np.nan
    ax.plot_surface(c1X, c1Y, c1Z,rstride=5, cstride=5,linewidth=0,alpha=0.16,color = 'black
    ax.plot_surface(c2X, c2Y, c2Z,rstride=5, cstride=5,linewidth=0,alpha=0.16,color = 'black
#    ax.set_zlim(0,1)
```

```
def plot_cube_4wires(ax,cube4wires,xframe,zn):
    '''
    No use. Plot the cube from cube vertex data using 4 separate lines each with equal numbe
    '''
    cube_111node_aligning_to_z = 0.18*np.dot(rotmat_from_A_2_B(cube4wires[0,:],zn),
                                     np.transpose(cube4wires))#np.transpose(cube4wires)
    for i in range(16):
            cube_111node_aligning_to_z[:,i] = cube_111node_aligning_to_z[:,i] + zn*0.18*np.s

 #   cubeinst1 = ax.plot(*cube_111node_aligning_to_z[:,0:4],c='b',linewidth=.5)
 #   cubeinst2 = ax.plot(*cube_111node_aligning_to_z[:,4:8],c='b',linewidth=.5)
  #  cubeinst3 = ax.plot(*cube_111node_aligning_to_z[:,8:12],c='b',linewidth=.5)
   # cubeinst4 = ax.plot(*cube_111node_aligning_to_z[:,12:16],c='b',linewidth=.5)
    return cube_111node_aligning_to_z
```