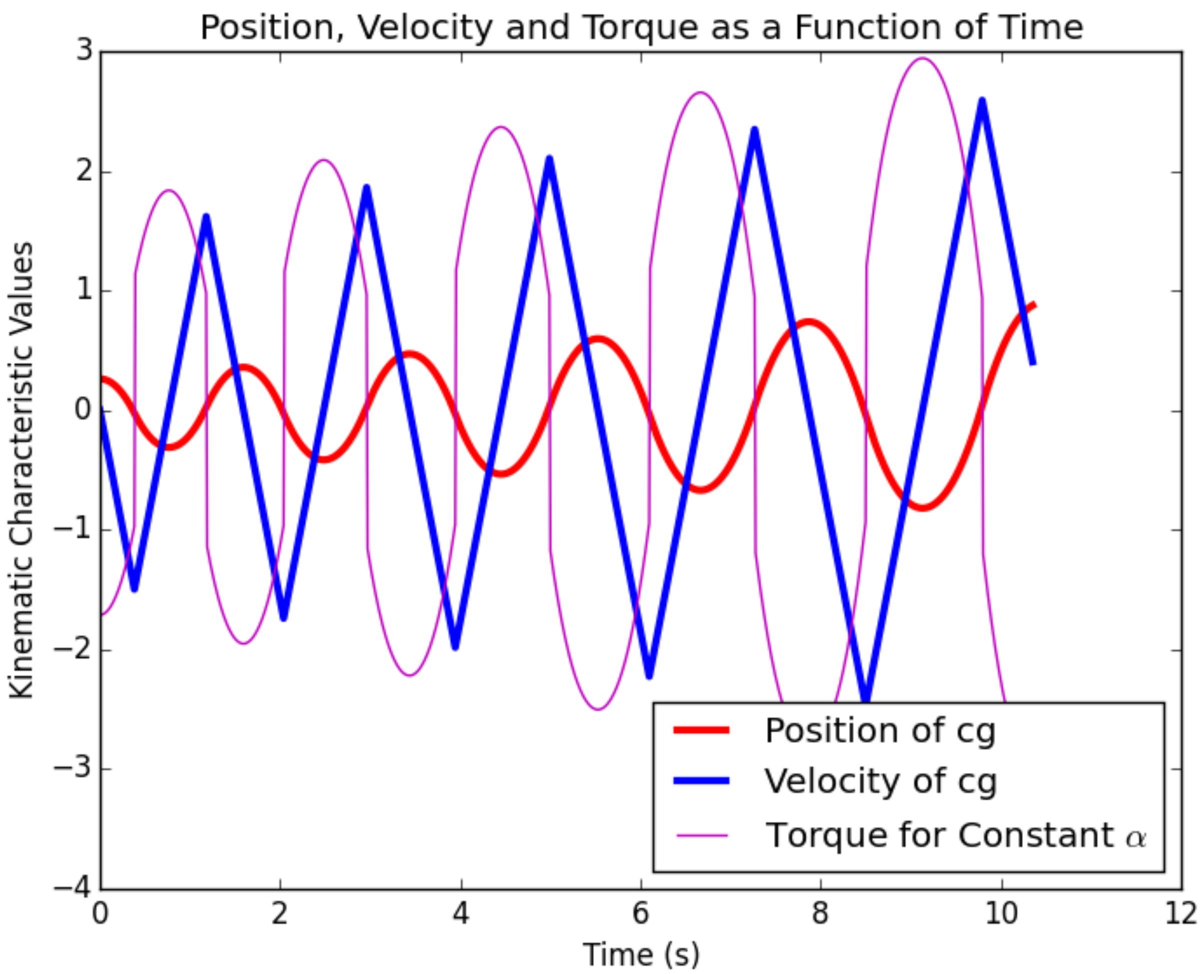
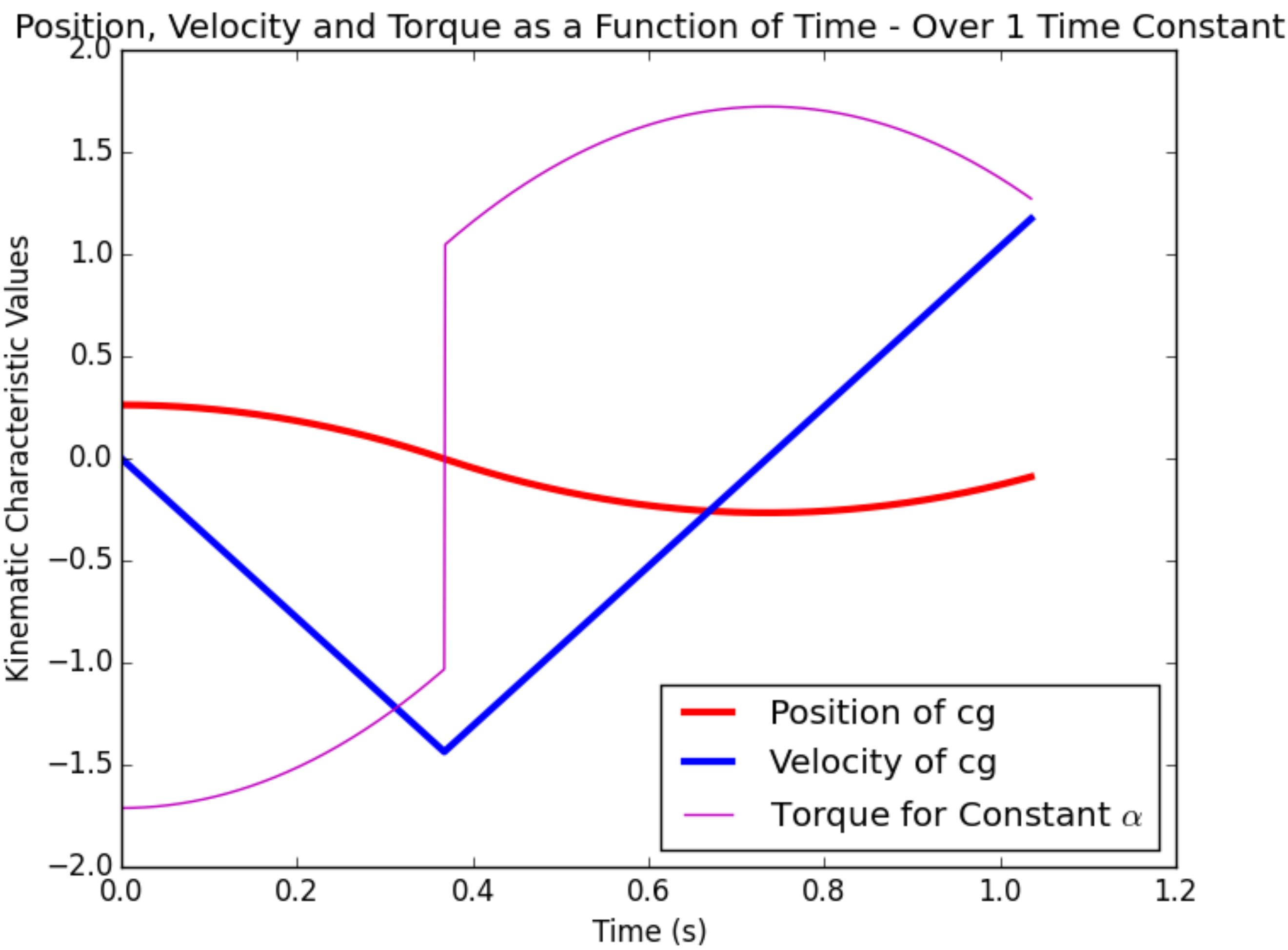
```python
#!/usr/bin/env python

import numpy as np, Kinematic_Characteristics as kc
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

gravity = 9.81

# function to calculate instantaneous rotational characteristic
def atorquei(alpha, length_to_cg, mass, theta):
  return mass*length_to_cg*(alpha-gravity*np.sin(theta))
def thetatt(torque, length_to_cg, mass, theta):
  return torque/(length_to_cg*mass)+gravity*np.sin(theta)
def thetat(theta_tt, current_time, previous_time, theta_t):
  return theta_tt*(current_time-previous_time) + theta_t
def theta(theta_tt, theta_t, theta_i, current_time, previous_time):
  print (theta_tt*(current_time-previous_time)**2/2 + theta_t *(current_time - previ
ous_time),
      theta_i)
# print (
#     (theta_tt*(current_time**2-previous_time**2)/2, theta_tt),
#     (theta_t *(current_time - previous_time), theta_t),
#     theta_i, (current_time, previous_time)
#      )
  return (
    theta_tt*(current_time-previous_time)**2/2 +
    theta_t *(current_time - previous_time) +
    theta_i
    )
# calulate dynamic characteristics with constant acceleration calculating motor
# - torque
def update_rotational( alpha, mass, iterations, initial_time, final_time, initial_po
sition,
    length_to_cg):
  # set up initial conditions
  time_limit = final_time-initial_time
  direction = 1
  theta_i, theta_t, theta_tt, torque = initial_position, 0, -alpha, 0
  step_size = time_limit/float(iterations)
  time = np.arange(0, (time_limit + step_size), step_size)
  position = np.array([])
  velocity = np.array([])
  motor_torque = np.array([])
  posi = np.array([])
  switch = False
  for i in range(len(time)):
    current_time = time[i]
    previous_time = time[i-1]
    if i == 0:
      previous_time = current_time
    theta_i = theta(theta_tt, theta_t, theta_i, current_time, previous_time)
    theta_t = thetat(theta_tt, current_time, previous_time, theta_t)
    torque  = atorquei(theta_tt, length_to_cg, mass, theta_i)
    j = i-1
    if i > 0:
      if (
          ((position[j] < 0 and position[j-1] > 0) or
            (position[j] > 0 and position[j-1] < 0)) and
          switch == False):
        switch = True
        theta_tt = -1*thetatt(torque, length_to_cg, mass, theta_i)
      else:
        switch = False
        theta_tt = thetatt(torque, length_to_cg, mass, theta_i)
    else:
      theta_tt = thetatt(torque, length_to_cg, mass, theta_i)

#   print theta_tt
    # append values to arrays
    position = np.hstack((position, np.array([theta_i])))
    velocity = np.hstack((velocity, np.array([theta_t])))
    motor_torque = np.hstack((motor_torque, np.array([torque])))
  return time, position, velocity, motor_torque
```

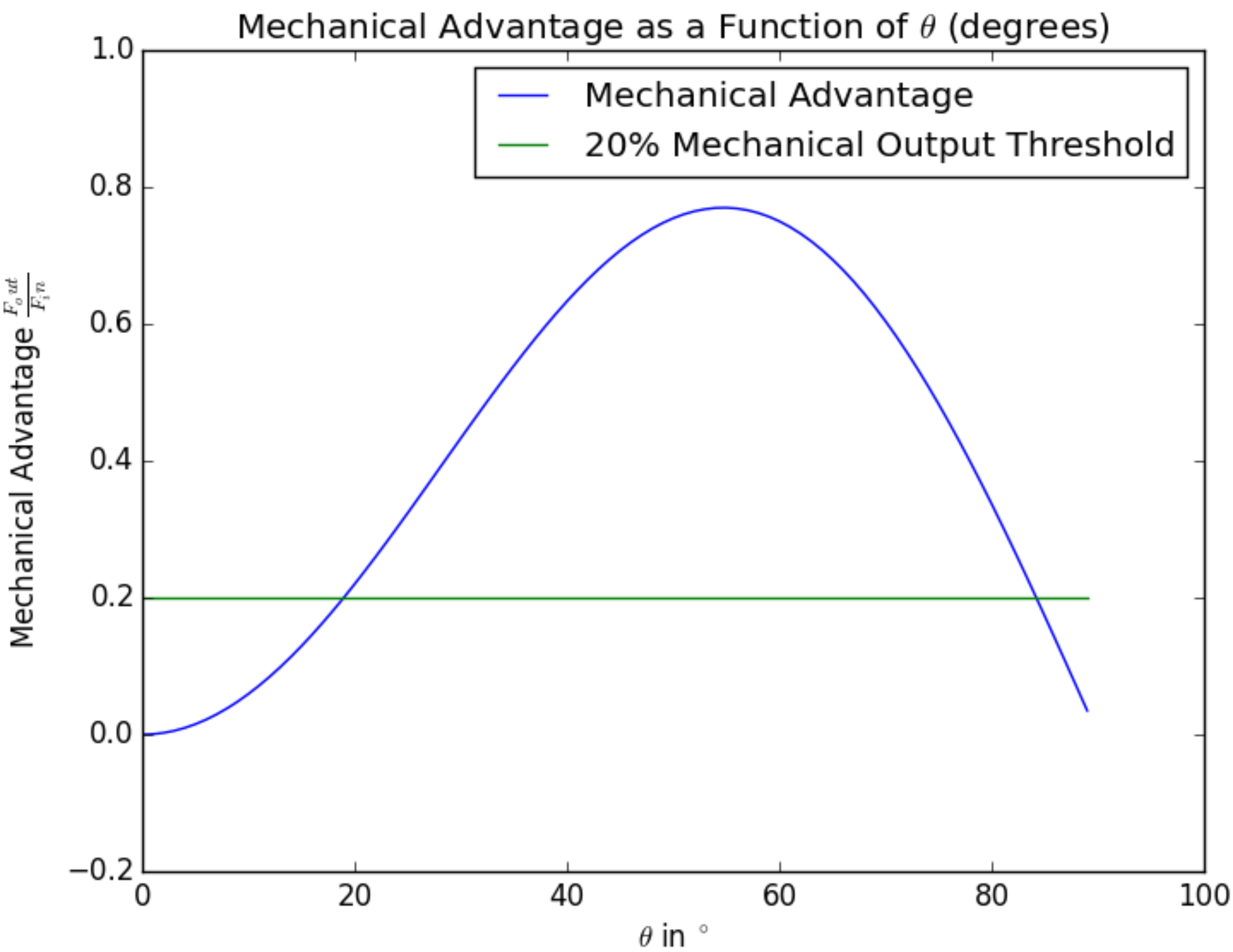
```python
# get the calculate the necessary values
def get_requirement():
  # calc values
  length_of_leg = .5
  dx = 2*length_of_leg*np.sin(15.0*np.pi/180)
  vx = .5
  time_of_swing = dx/vx
  dt0 = (15.0-0.0)*np.pi/180.0
  dt1 = (0.0-15.0)*np.pi/180.0
  alpha0 = 2*(dt0-dt1)/( time_of_swing)**2
  alpha1 = 2*dt1/( time_of_swing/2)**2
  iterations = 1000
  # mass of the leg 1.0292kg
  mass_of_leg = 1.0292
  # declare position vectors
  #determine necessary rotational acceleration
  rotational_chars = update_rotational( alpha0, mass_of_leg, iterations,
      0.0, 20*time_of_swing, dt0, .258  )

  print max(rotational_chars[3]), min(rotational_chars[3])

  print "alpha is: " + str(alpha0)
  print "time of swing: " +str(time_of_swing)
  print("range of rotation is from: "+ str(dt0)+
      " to: " + str(dt1))
  print "the mass of the leg is: " + str(mass_of_leg)
  print ("the max and min positions: " + str(max(rotational_chars[1])) + ", " +
      str(min(rotational_chars[1])) )
  print ("the max and min velocity: " + str(max(rotational_chars[2])) + ", " +
      str(min(rotational_chars[2])) )
  print ("the max and min acceleration: " + str(max(rotational_chars[3])) + ", " +
      str(min(rotational_chars[3])) )

  #plot some fancies
  posi = plt.plot(
      rotational_chars[0], rotational_chars[1], color='r',
      linewidth=3, label=r'Position of cg'# position
      )
  velo = plt.plot(
      rotational_chars[0], rotational_chars[2], color='b',
      linewidth=3, label=r'Velocity of cg'# velocity
      )
  acce = plt.plot(
      rotational_chars[0], rotational_chars[3], color='m',
      label=r'Torque for Constant $\alpha$'# acceleration
      )
  plt.legend(loc=4)
  plt.xlabel('Time (s)')
  plt.ylabel(r'Kinematic Characteristic Values')
  plt.title('Position, Velocity and Torque as a Function of Time')
  plt.show()
  return 0


if __name__ == "__main__":
  get_requirement()
```



Position, Velocity and Torque as a Function of Time
Kinematic Characteristic Values
Time (s)
3
2
1
0
−1
−2
−3
−4
0
2
4
6
8
10
12
Position of cg
Velocity of cg
Torque for Constant $\alpha$


Position, Velocity and Torque as a Function of Time - Over 1 Time Constant
Kinematic Characteristic Values
Time (s)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Position of cg
Velocity of cg
Torque for Constant $\alpha$

```
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def mechanical_advantage( theta):
  return np.cos(np.pi/2.0-2*theta)*np.cos(theta)

if __name__ == "__main__":
  theta = np.arange(0,90)
  ma = mechanical_advantage(np.pi/2-theta*np.pi/180.0)
  thresh = np.array([ .2 for i in range(len(theta))])
  # plot mechanical advantage
  ma_line = plt.plot( theta, ma, label="Mechanical Advantage")
  thresh_line = plt.plot( theta, thresh, label="20% Mechanical Output Threshold")
  # get the pretties
  plt.title(r'Mechanical Advantage as a Function of $\theta$ (degrees)')
  plt.ylabel(r'Mechanical Advantage $\frac{F_out}{F_in}$')
  plt.xlabel(r'$\theta$ in $^\circ$')
  # set up axes
  ax = plt.gca()
  ax.set_xlim([0,100])
  ax.set_ylim([-.2,1])
  plt.legend()
  plt.show()
```


Mechanical Advantage as a Function of $\theta$ (degrees)
Mechanical Advantage
20% Mechanical Output Threshold
Mechanical Advantage $\frac{F_out}{F_in}$
$\theta$ in $^\circ$
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
20
40
60
80
100