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

import numpy as np, Kinematic_Characteristics as kc

# function to calculate instantaneous rotational characteristic
def torquei(alpha, length_to_cg, mass, theta):
  return alpha*mass/length_to_cg-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):
  return theta_tt*(current_time-previous_time)
def theta(theta_tt, theta_t, theta_i, current_time, previous_time):
  return (
    theta_tt*(current_time**2-previous_time**2)/2 +
    theta_t *(current_time   -previous_time) +
    theta_i
    )

# calculate the instantaneous radial characteristics
def rtt(torque, length_of_driver, mass, mech_adv, theta_t, r, theta_i):
  return (theta_t**2/r +
    gravity*np.cos(theta) +
    torque*length_of_driver*mech_adv/mass
    )
def rt(r_tt, current_time, previous_time):
  return r_tt*(current_time-previous_time)
def r(r_tt, r_t, r_i, current_time, previous_time):
  return (
    r_tt*(current_time**2-previou_time**2)/2 +
    r_t *(current_time - previous_time) +
    r_i
    )

# calulate dynamic characteristics with constant acceleration calculating motor
# - torque
def update_rotational( alpha, mass, iterations, time_limit, initial_position,):
  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([])
  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(alpha, theta_t, theta_i, current_time, previous_time)
    theta_t = thetat(alpha, current_time, previous_time)
    motor_torque  = atorquei(alpha, length_to_cg, mass, theta)
    # append values to arrays
    position = np.concatenate((position, np.array([theta_i])))
    velocity = np.concatenate((velocity, np.array([theta_t])))
    motor_torque = np.concatenate((motor_torque, np.array([torque])))
  return position, velocity, motor_torque

# calculate the radial dynamic chararteristics
def update_radial():
  time = np.arange(0, (time_limit + step_size), step_size)
  positition = np.array([])
  velocity = np.array([])
  motor_torque = np.array([])
  for i in range(len(time)):
    current_time = time[i]
    previous_time = time[i-1]
    if i == 0:
        previous_time = current_time
    r_i  = r(r_tt, r_t, r_i, current_time, previous_time)
    r_t  = rt(r_tt, current_time, previous_time)
    motor_torque = rtorque(acceleration, length_of_driver, mass, mech_adv, theta_t,
r, theta_i)
    position = np.concatenate((position, np.array([r_i])))
    velocity = np.concatenate((velocity, np.array([r_t])))
    motor_torque = np.concatenate((motor_torque, np.array([torque])))
```

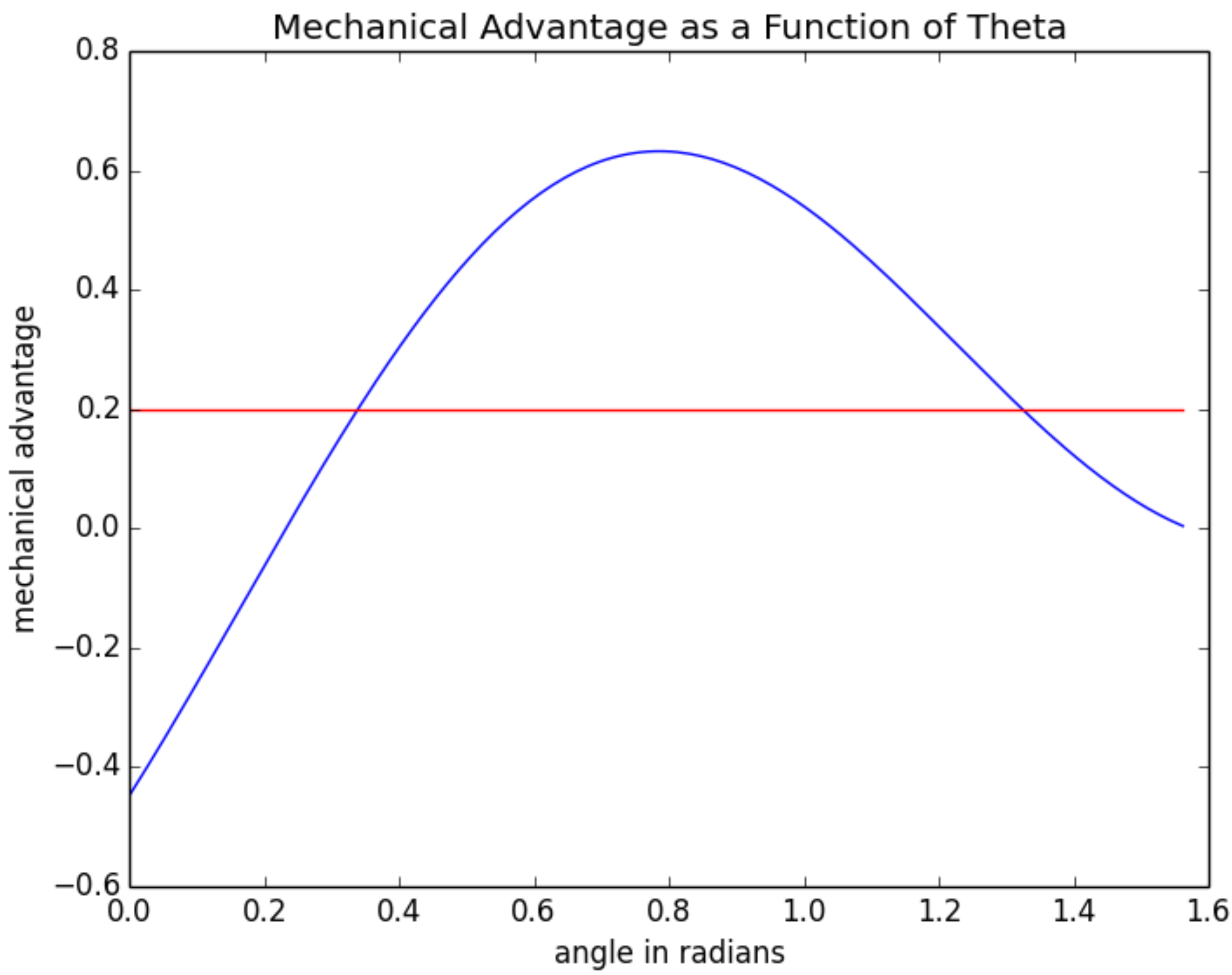
```
  return position, velocity, motor_torque

# get the calculate the necessary values
def get_requirement():
  length_of_leg = .5
  # time of slider is .2s
  time_of_slider = .2
  slider_time_step_size = time_of_slider/100.0
  # time of leg is 2*l*sin(theta)/velocity
  time_of_swing = 2*length_of_leg*np.sin(15.0*np.pi/180)/.5
  swing_time_step_size = time_of_swing/100
  # theta range is -15 to 15
  theta_swing_max_min = np.array([-15.0, 15.0])
  swing_step_size = (15.0+15)/100.0
  # length of the driver is .125 m
  length_of_driver = .1
  # mass of the leg 1.0292kg
  mass_of_leg = 1.0292
  # theta of the driver range is from 20 - 75 degrees
  theta_driver_max_min = np.array([20.0,75.0])
  driver_step_size = (75.0-20.0)/100.0
  # declare position vectors
  pvec1 = np.array([np.amin(theta_swing_max_min), np

  #determine necessary rotational acceleration
  alpha = kc.acceleration_necessary( 0, pvec1, pvec2, time)[0]

  rotational_chars = update_rotational( alpha, mass, iterations, time_limit, initial
_position,)
  print rotational_chars
  return requrements


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


Mechanical Advantage as a Function of Theta
mechanical advantage
angle in radians
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

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

import numpy as np

def cross_product( vec1, vec2):
  return [ (vec1[1]*vec2[2] - vec2[1]*vec1[2]),
          -(vec1[0]*vec2[2] - vec2[0]*vec1[2]),
           (vec1[0]*vec2[1] - vec2[0]*vec1[1])
           ]

def dot_product( vec1, vec2):
  return sum([ vec1[i]*vec2[i] for i in range(len(vec1)) ])

def magnitude(vec):
  val = 0
  for i in vec:
    val = val + i**2
  return val**(1.0/2.0)

def unit( vec):
  '''
  returns unit vector (direction) of the input vector
  '''
  return np.divide(vec, float(np.linalg.norm(vec)))

def moment_from_weight( weight, radius_to_cg):
  '''
  this function accepts 2 vectors of weight and radius of cg to determine
  moment about the cg
  [ Wx, Wy, Wz] x [ rx, ry, rz]
  '''
  return np.cross( weight, radius_to_cg)

def rotation( theta, axis_of_rotation='z'):
  if axis_of_rotation == 'x':
    return np.matrix(
        (1, 0, 0),
        (0, np.cos(theta), -np.sin(theta)),
        (0, np.sin(theta),  np.cos(theta))
        )
  elif(axis_of_rotation == 'y'):
    return np.matrix(
        (np.cos(theta), 0, -np.sin(theta)),
        (0, 1, 0),
        (np.sin(theta), 0,  np.cos(theta))
        )
  elif(axis_of_rotation == 'z'):
    return np.matrix(
        (np.cos(theta), -np.sin(theta), 0),
        (np.sin(theta),  np.cos(theta), 0),
        (0, 0, 1)
        )
  else:
    return 0

def acceleration_necessary( resistance_vec, pvec1, pvec2, time):
  '''
  this function uses equations of motion to determine the necessary acceleration
  to get the leg to a certain point at a specific time
  currently this function will be just ideal acceleration with no resistances
  '''
  return np.subtract(
      np.divide(np.subtract(pvec2, pvec1), np.divide(time**2, 2)),
      resistance_vec)

def output_torque( inertial_moment, radial_acceleration):
  return np.multipy( inertial_moment, radial_acceleration)



def Angular_Momentum_l2_about_hip():
  '''
  this function will use equations of motion in a cylindrical path to determine
```

```
    the angular momentum about of link 2 about the hip
    '''

def angular_momentum_l1_about_hp():
    '''
    this function will use equations of motion in a cylindrical path to determine
    the angular momentum about of link 1 about the hip
    '''
```