```
'''
File name      : hw7_prob8.py
Author         : Jared Hansen
Date created   : 11/26/2019
Python version : 3.7.3

DESCRIPTION: The purpose of this script is to solve the planetary landing
             problem posed in question 8 in homework 7. Something in the
             formulation is off, since the plot obviously is wrong. It
             satisfies the constraints of x_1 = a, x_{N+1} = b, and starts at
             the correct x and u, but it somehow manages to go negative (e.g.
             crashing far into the ground before coming back up)

'''



#=============================================================================
#==== IMPORT STATEMENTS
#=============================================================================
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as npla



#=============================================================================
#==== DEFINING CONSTANTS, GIVEN INFORMATION
#=============================================================================
# Dimensions of individual x and u vectors, number of time points.
n = 4
m = 2
T = 500
# Given values: A, B, g, N, a, b, rho1, rho2
A = np.matrix([[1, 0, 0.1303, 0     ],
               [0, 1, 0,      0.1303],
               [0, 0, 1,      0     ],
               [0, 0, 0,      1     ]])
B = np.matrix([[0.0085, 0     ],
               [0     , 0.0085],
               [0.1303, 0     ],
               [0     , 0.1303]])
g = np.array([0, -0.0832, 0, -1.2779])#.reshape(4,1)
a = np.array([1000, 1500, -25, 0])#.reshape(4,1)
b = np.array([0, 0, 0, 0])#.reshape(4,1)
rho1 = 4;
rho2 = 12;

# Defining variables for the convex solver. We introduce a new variable, gamma,
# to make the problem convex.
x = cp.Variable((n, T))
u = cp.Variable((m, T))
gamma = cp.Variable((1,T))


# Initialize the cost to 0, make constraints an empty list, then populate.
cost = 0
constr = []
for t in range(T-2):
    # Incrementing the cost function for each successive u vector
    cost += cp.sum((gamma[:,t])**2)
    # Defining the new constraints...basically these constraints are enforcing
    # that the states are connected, and that the magnitude of our thrust
    # vector is within the bounds of rho1 and rho2
    constr += [x[:,t+1] == A@x[:,t] + B@u[:,t] + g,
               cp.norm(u[:,t], 'inf') <= gamma[:,t],
               gamma[:,t] <= rho2,
```

```
            -gamma[:,t] <= -rho1,
            #gamma[:,t] >= rho1,
            #np.array([0,0,0,0]) <= -x[:,t],
            #np.array([0,0]) <= -u[:,t]
            ]
    '''
    # This is where the problem lies: there is nothing telling the algorithm
    # that we can't have negative thrust or negative position values. When I
    # try and implement these constraints the solver says that the objective
    # value of the solution is infinite, so this doesn't work with this solver.
    constr += [ -x[:,t] <= -np.array([0,0,0,0]),
             -u[:,t] <= np.array([0,0])  ]
    '''
# Adding the boundary constraints (must start at "a" and end at "b")
constr += [x[:,499] == b, x[:,0] == a]
# Define the problem with our completed cost and constraints list.
problem = cp.Problem(cp.Minimize(cost), constr)
# Have the solver determine optimal values for x and u.
problem.solve(solver=cp.ECOS)


#============================================================================
#==== PLOTTING (INCORRECT) RESULTS
#============================================================================
# Store the x's and u's in a nicer format (as 2-dimensional numpy arrays).
x_mat = x.value
u_mat = u.value
# Grab range and altitude for plotting.
x_range = x_mat[0,:]
x_altitude = x_mat[1,:]
# Grab the thrusts for plotting.
u_xThrust = u_mat[0,:]
u_yThrust = u_mat[1,:]

# Plot of range versus altitude
plt.plot(x_range, x_altitude)
plt.title("Plot of Range vs Altitude (incorrect)")
plt.xlabel('range (should only go to 0, not negative)')
plt.ylabel('altitude (should only go to 0, not negative)')

# Plot of xThrust versus yThrust
plt.plot(u_xThrust, u_yThrust)
plt.title('Plot of xThrust versus yThrust')
plt.xlabel('xThrust (horizontal thrust)')
plt.ylabel('yThrust (vertical thrust)')
```