EE16A: Homework 4

Problem 6: Segway Tours

Run the following block of code first to get all the dependencies.

In [1]:
```
# %load gauss_elim.py
from gauss_elim import gauss_elim
```

In [2]:
```
from numpy import zeros, cos, sin, arange, around, hstack
from matplotlib import pyplot as plt
from matplotlib import animation
from matplotlib.patches import Rectangle
import numpy as np
from scipy.interpolate import interp1d
import scipy as sp
```

Dynamics

In [6]:
```
# Dynamics: state to state
A = np.array([[1, 0.05, -.01, 0],
              [0, 0.22, -.17, -.01],
              [0, 0.1, 1.14, 0.10],
              [0, 1.66, 2.85, 1.14]]);
# Control to state
b = np.array([.01, .21, -.03, -0.44])
nr_states = b.shape[0]

# Initial state
state0 = np.array([-0.3853493, 6.1032227, 0.8120005, -14])

# Final (terminal state)
stateFinal = np.array([0, 0, 0, 0])
```

Example of Gaussian Elimination

In [7]:
```
# System of example equations:
# x + y +  5z = 7
# x + 4y - z  = 4
# 3x + y - z  = 4

Q = np.zeros([3,3])

# Matrix construction by specifying column vectors
Q[:,0] = np.array([1, 1, 3]) # coefficients of x
Q[:,1] = np.array([1, 4, 1]) # coefficients of y
Q[:,2] = np.array([5, -1, -1]) # coefficients of z

m = np.array([7, 4, 4])

# Augmented Matrix for system of equations
Q_aug = np.c_[Q, m]

print('Augmented matrix for part f:')
print(Q_aug,'\n')

print('Matrix after Gaussian elimination:')
print(gauss_elim(Q_aug))
```

```
Augmented matrix for part f:
[[ 1.  1.  5.  7.]
 [ 1.  4. -1.  4.]
 [ 3.  1. -1.  4.]]

Matrix after Gaussian elimination:
[[1.   0.   0.   1.35]
 [0.   1.   0.   0.9 ]
 [0.   0.   1.   0.95]]
```

Part (d)

In [91]:
```
# Compute the A^2
A2 = np.dot(A,A)
values = (-1) * np.dot(A2, state0)

coeff_0 = np.dot(A, b).reshape(4, 1)
coeff_1 = b.reshape(4, 1)

variables = np.concatenate((coeff_0, coeff_1), 1).reshape(4, 2)
augmented = np.c_[variables, values]

print("The augmented matrix is:\n", augmented, "\n")
print("After Gaussian Elimination:\n", gauss_elim(augmented))
```

```
The augmented matrix is:
 [[ 0.0208      0.01        0.02243475]
 [ 0.0557      0.21       -0.30785117]
 [-0.0572     -0.03        0.06193476]
 [-0.2385     -0.44        1.38671326]]

After Gaussian Elimination:
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [-0. -0.  1.]
 [ 0.  0.  0.]]
```

Part (e)

In [92]:
```
# Compute the A^3
A3 = np.dot(A2,A)

values = (-1) * np.dot(A3, state0)

coeff_0 = np.dot(A2, b).reshape(4, 1)
coeff_1 = np.dot(A, b).reshape(4, 1)
coeff_2 = b.reshape(4, 1)

variables = np.concatenate((coeff_0, coeff_1, coeff_2), 1).reshape(4, 3
augmented = np.c_[variables, values]

print("The augmented matrix is:\n", augmented, "\n")
print("After Gaussian Elimination:\n", gauss_elim(augmented))
```

```
The augmented matrix is:
 [[ 0.024157    0.0208      0.01        0.00642285]
 [ 0.024363    0.0557      0.21       -0.0921233 ]
 [-0.083488   -0.0572     -0.03        0.17849184]
 [-0.342448   -0.2385     -0.44        1.24633424]]

After Gaussian Elimination:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Part (f)

In [96]:

```
# Compute the A^4
A4 = np.dot(A3,A)

values = (-1) * np.dot(A4, state0)

coeff_0 = np.dot(A3, b).reshape(4, 1)
coeff_1 = np.dot(A2, b).reshape(4, 1)
coeff_2 = np.dot(A, b).reshape(4, 1)
coeff_3 = b.reshape(4, 1)

variables = np.concatenate((coeff_0, coeff_1, coeff_2, coeff_3), 1).res
augmented = np.c_[variables, values]

print("The augmented matrix is:\n", augmented, "\n")
print("After Gaussian Elimination:\n", gauss_elim(augmented))
```

```
The augmented matrix is:
 [[ 2.62100300e-02  2.41570000e-02  2.08000000e-02  1.00000000e-02
   3.17637529e-05]
 [ 2.29773000e-02  2.43630000e-02  5.57000000e-02  2.10000000e-01
  -6.30740802e-02]
 [-1.26984820e-01 -8.34880000e-02 -5.72000000e-02 -3.00000000e-02
   3.18901788e-01]
 [-5.87888940e-01 -3.42448000e-01 -2.38500000e-01 -4.40000000e-01
   1.77659810e+00]]

After Gaussian Elimination:
 [[  1.           0.           0.           0.         -13.24875075]
 [  0.           1.           0.           0.          23.73325125]
 [  0.           0.           1.           0.         -11.57181872]
 [  0.           0.           0.           1.           1.46515973]]
```

Part (g)

Preamble

This function will take care of animating the segway. DO NOT EDIT!

In [97]:

```
# frames per second in simulation
fps = 20
# length of the segway arm/stick
stick_length = 1.

def animate_segway(t, states, controls, length):
    #Animates the segway

    # Set up the figure, the axis, and the plot elements we want to an
    fig = plt.figure()

    # some config
```

```
    segway_width = 0.4
    segway_height = 0.2

    # x coordinate of the segway stick
    segwayStick_x = length * np.add(states[:, 0],sin(states[:, 2]))
    segwayStick_y = length * cos(states[:, 2])

    # set the limits
    xmin = min(around(states[:, 0].min() - segway_width / 2.0, 1), arou
    xmax = max(around(states[:, 0].max() + segway_height / 2.0, 1), arc

    # create the axes
    ax = plt.axes(xlim=(xmin-.2, xmax+.2), ylim=(-length-.1, length+.1

    # display the current time
    time_text = ax.text(0.05, 0.9, 'time', transform=ax.transAxes)

    # display the current control
    control_text = ax.text(0.05, 0.8, 'control', transform=ax.transAxes

    # create rectangle for the segway
    rect = Rectangle([states[0, 0] - segway_width / 2.0, -segway_height
        segway_width, segway_height, fill=True, color='gold', ec='blue
    ax.add_patch(rect)

    # blank line for the stick with o for the ends
    stick_line, = ax.plot([], [], lw=2, marker='o', markersize=6, colo

    # vector for the control (force)
    force_vec = ax.quiver([],[],[],[],angles='xy',scale_units='xy',sca

    # initialization function: plot the background of each frame
    def init():
        time_text.set_text('')
        control_text.set_text('')
        rect.set_xy((0.0, 0.0))
        stick_line.set_data([], [])
        return time_text, rect, stick_line, control_text

    # animation function: update the objects
    def animate(i):
        time_text.set_text('time = {:2.2f}'.format(t[i]))
        control_text.set_text('force = {:2.3f}'.format(controls[i]))
        rect.set_xy((states[i, 0] - segway_width / 2.0, -segway_height
        stick_line.set_data([states[i, 0], segwayStick_x[i]], [0, segwa
        return time_text, rect, stick_line, control_text

    # call the animator function
    anim = animation.FuncAnimation(fig, animate, frames=len(t), init_fu
            interval=1000/fps, blit=False, repeat=False)
    return anim
    #plt.show()
```

Plug in your controller here

In [98]:
```
# If you want to try zero control
# controls = np.zeros((4))

controls = [-13.249, 23.733, -11.572, 1.465]
print(controls)
```

```
[-13.249, 23.733, -11.572, 1.465]
```

Simulation

DO NOT EDIT!

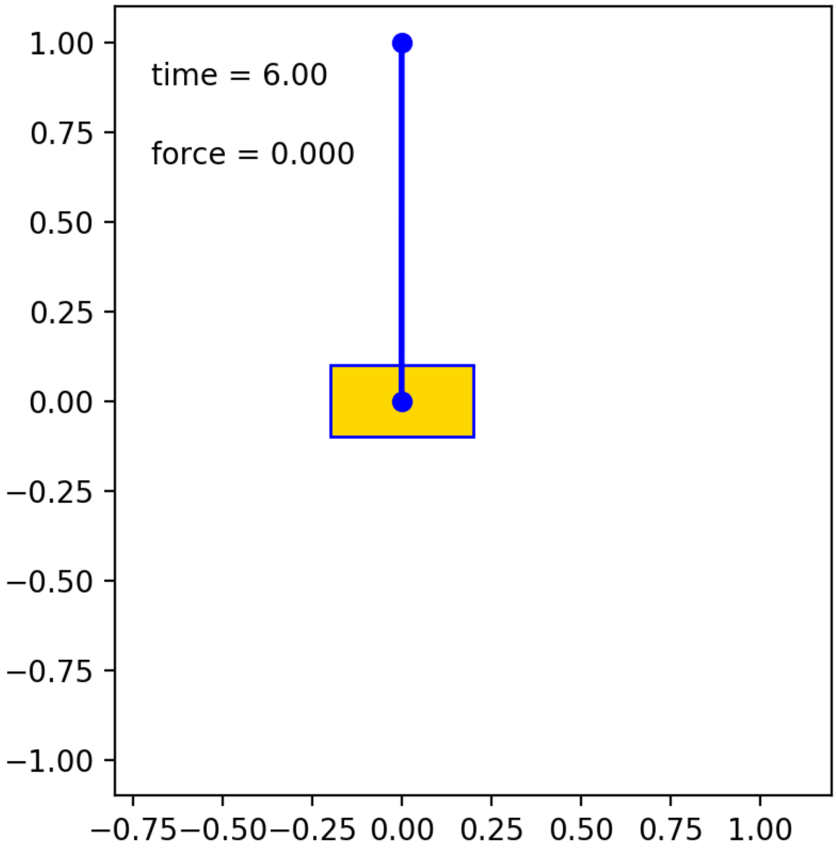
In [99]:
```
# This will add an extra couple of seconds to the simulation after the
# the effect of this is just to show how the system will continue afte
controls = np.append(controls,[0, 0])

# number of steps in the simulation
nr_steps = controls.shape[0]

# We now compute finer dynamics and control vectors for smoother visua
Afine = sp.linalg.fractional_matrix_power(A,(1/fps))
Asum = np.eye(nr_states)
for i in range(1, fps):
    Asum = Asum + np.linalg.matrix_power(Afine,i)

bfine = np.linalg.inv(Asum).dot(b)

# We also expand the controls in the "intermediate steps" (only for vi
controls_final = np.outer(controls, np.ones(fps)).flatten()
controls_final = np.append(controls_final, [0])

# We compute all the states starting from x0 and using the controls
states = np.empty([fps*(nr_steps)+1, nr_states])
states[0,:] = state0;
for stepId in range(1,fps*(nr_steps)+1):
    states[stepId, :] = np.dot(Afine,states[stepId-1, :]) + controls_f

# Now create the time vector for simulation
t = np.linspace(1/fps,nr_steps,fps*(nr_steps),endpoint=True)
t = np.append([0], t)
```

Visualization

DO NOT EDIT!

In [101]:
```
%matplotlib nbagg
# %matplotlib qt
anim = animate_segway(t, states, controls_final, stick_length)
anim
```

Figure 1



Out[101]: <matplotlib.animation.FuncAnimation at 0xd248ac438>

In [ ]: