# EE16A: Homework 3

## Problem 4: Bieber's Segway

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

In [65]:
```
# %load gauss_elim.py
from gauss_elim import gauss_elim
```

In [66]:
```
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 [67]:
```
# 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])
```

## Part (d), (e), (f)

In [68]:
```
# Part D
DCol1 = np.dot(A, b)
DCol2 = b
DCol3 = stateFinal
DCol4 = stateFinal
```

```
DmatrixA = np.vstack([DCol1, DCol2])
DmatrixA = np.vstack([DmatrixA, DCol3])
DmatrixA = np.vstack([DmatrixA, DCol4])

DvectorB = np.dot(A, A)
DvectorB = np.dot(DvectorB, b)

Dfinal = gauss_elim(np.vstack([DvectorB, DmatrixA]))
print("E")
print(Dfinal)

# Part E
ECol1 = np.dot(A, A)
ECol1 = np.dot(ECol1, b)
ECol2 = np.dot(A, b)
ECol3 = b
ECol4 = stateFinal

EmatrixA = np.vstack([ECol1, ECol2])
EmatrixA = np.vstack([EmatrixA, ECol3])
EmatrixA = np.vstack([EmatrixA, ECol4])

EvectorB = np.dot(A, A)
EvectorB = np.dot(EvectorB, A)
EvectorB = np.dot(EvectorB, b)

Efinal = gauss_elim(np.vstack([EvectorB, EmatrixA]))
print("E")
print(Efinal)

# Part F
FCol1 = np.dot(A, A)
FCol1 = np.dot(FCol1, A)
FCol1 = np.dot(FCol1, b)
FCol2 = np.dot(A, A)
FCol2 = np.dot(FCol2, b)
FCol3 = np.dot(A, b)
FCol4 = b

FmatrixA = np.vstack([FCol1, FCol2])
FmatrixA = np.vstack([FmatrixA, FCol3])
FmatrixA = np.vstack([FmatrixA, FCol4])

FvectorB = np.dot(A, A)
FvectorB = np.dot(FvectorB, A)
FvectorB = np.dot(FvectorB, A)
FvectorB = np.dot(FvectorB, b)

Ffinal = gauss_elim(np.vstack([FvectorB, FmatrixA]))
print("F")
print(Ffinal)
```

```
Eoh wel
[[  1.           0.           0.          14.41214875]
 [  0.           1.           0.          -1.66942929]
 [  0.           0.           1.           7.78471123]
 [  0.           0.           0.           0.        ]
 [  0.           0.           0.           0.        ]]
E
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [-0. -0. -0.  1.]
 [ 0.  0.  0.  0.]]
F
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]
 [ 0.  0.  0.  0.]]
```

# Part (g)

## Preamble

This function will take care of animating the segway.

In [13]:
```
# 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 animate
    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), around(segw
ayStick_x.min(), 1))
    xmax = max(around(states[:, 0].max() + segway_height / 2.0, 1), around(seg
wayStick_y.max(), 1))

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

```
t='equal')

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

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

    # create rectangle for the segway
    rect = Rectangle([states[0, 0] - segway_width / 2.0, -segway_height / 2],
        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,
color='blue')

    # vector for the control (force)
    force_vec = ax.quiver([],[],[],[],angles='xy',scale_units='xy',scale=1)

    # 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 / 2))
        stick_line.set_data([states[i, 0], segwayStick_x[i]], [0, segwayStick_
y[i]])
        return time_text, rect, stick_line, control_text

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

## Plug in your controller here

In [14]:
```
controls = np.array([0,0,0,0]) # here
```

## Simulation

In [15]:
```
# This will add an extra couple of seconds to the simulation after the input c
ontrols with no control
# the effect of this is just to show how the system will continue after the co
ntroller "stops controlling"
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 visualization
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 visualizat
ion)
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_final[ste
pId-1] * bfine

# 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