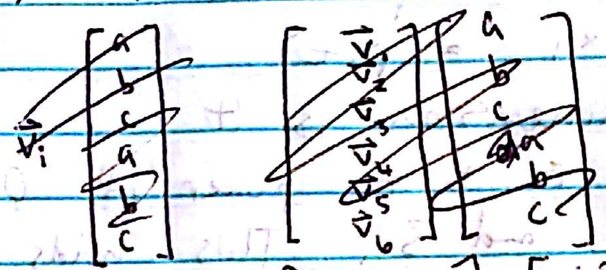
1. a) It can recover from only one erasure. It can't handle patterns where both a's are lost or both b's, etc.

b)



$$\begin{bmatrix} \alpha_1 & \beta_1 & \gamma_1 & \alpha_1 & \beta_1 & \gamma_1 \\ \alpha_2 & \beta_2 & \gamma_2 & \alpha_2 & \beta_2 & \gamma_2 \\ \alpha_3 & \beta_3 & \gamma_3 & \alpha_3 & \beta_3 & \gamma_3 \\ \alpha_4 & \beta_4 & \gamma_4 & \alpha_4 & \beta_4 & \gamma_4 \\ \alpha_5 & \beta_5 & \gamma_5 & \alpha_5 & \beta_5 & \gamma_5 \\ \alpha_6 & \beta_6 & \gamma_6 & \alpha_6 & \beta_6 & \gamma_6 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ a \\ b \\ c \end{bmatrix}$$

c) [1, 0, 0], [0 1 0], [0 0 1], [1 0 0], [0 1 0], [0 0 1]

d) When the letters in the place of the ones at that time i is present. For example, you wouldn't be able to figure out "a" if "a" was missing at i = 1 and "a" and "b" was missing at i = 4 and etc.

e)
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 6 \\ ? \\ ? \\ 2 \\ 3 \\ ? \\ ? \end{bmatrix}$$

$a = 6$
$b = -4$
$c = -3$

f) She should choose the strategy in part a because at this point he knows neither a, b, nor c, and it's hard to solve for combinations of summations of a, b, c if ~~he~~ ~~to~~ he doesn't know any of them.

Scanned by CamScanner

2. a) p(t) is closed under both addition and multiplication

$p(t) + 0 = p(t)$

$p(t) \times 1 = p(t)$

the dimension is 4

one is to the power of 0, and 1, and 2, and 3. This adds up to 4 different dimensions

b) p(t) is literally a linear combination of the canonical polynomials.

$$\begin{bmatrix} c_0 & c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

$$= c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

Does this look familiar?

It's the exact same as p(t) but with c's instead of p's

c) the c vector consists of the constants that we are multiplying the elements of $\varphi(t)$ by

d) True; $\varphi(t)$ is definitely linearly independent, as any combination of t^2 can't equal t and etc., and linear combinations of the elements of $\varphi(t)$ can form any ~~poly~~ cubic polynomial with real values

2e) $(1-t)^3 = (t^2-2t+1)(1-t) = t^2-2t+1-t^3+2t^2-t$
$= -t^3+3t^2-3t+1$

$3t(1-t)^2 = 3t^3-6t^2+3t$
$3t^2(1-t) = 3t^2-3t^3$

$$B(t) = \begin{bmatrix} -t^3+3t^2-3t+1 \\ 3t^3-6t^2+3t \\ -3t^3+3t^2 \\ t^3 \end{bmatrix}$$

calculator

$$R = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

because there is a pivot in each row, R is linearly independent and invertible

$$R^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{3} & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \qquad R^{-1}\vec{B}(t) = \vec{Q}(t)$$

$B_0(t) + \frac{4}{3}B_1(t) + 2B_2(t) + 4B_3(t)$
$= t + t + t^2 + t^3$

3. a) $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$

b) $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

c) no; $x_2[1]$ always equals zero so that gives us no information, and $x_1[1]$ is just $x_1[0] + x_2[0]$ and here we're trying to solve two variables with ~~the~~ one equation, which doesn't work.

d) It shows that ~~A~~ the form $A\vec{x} = \vec{b}$ doesn't have a unique solution and matrix A therefore isn't invertible. As a result, we can't do $A^{-1}\vec{b} = \vec{x}$ to find out the initial water levels

e) matrix A is therefore ~~linearly inde~~ invertible and we can recover the initial state, as when we put it in the form $A\vec{x} = \vec{b}$,

$$A^{-1}A\vec{x} = A^{-1}\vec{b}$$

$$\vec{x} = A^{-1}\vec{b}$$

and we can find $\vec{x}$, the initial state.

This means that the experiment is reproducible

4. a) $\vec{x}[1] = A\vec{x}[0] + \vec{b}u[0]$

b) $\vec{x}[2] = A(A\vec{x}[0] + \vec{b}u[0]) + \vec{b}u[1]$

$= A^2\vec{x}[0] + A\vec{b}u[0] + \vec{b}u[1]$

c) $\vec{x}[3] = A^3\vec{x}[0] + A^2\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2]$

d) $\vec{x}[4] = A^4\vec{x}[0] + A^3\vec{b}u[0] + A^2\vec{b}u[1] + A\vec{b}u[2] + \vec{b}u[3]$

e) $\vec{x}[N] = A^N\vec{x}[0] + \vec{b}(A^{N-1}u[0] + A^{N-2}u[1] + \dots + A^0u[N-1])$

f) $\vec{x}[2] = A^2\vec{x}[0] + A\vec{b}u[0] + \vec{b}u[1]$

$\vec{0} - A^2\vec{x}[0] = A\vec{b}u[0] + \vec{b}u[1]$

$$\begin{bmatrix} A\vec{b} \\ \vec{b} \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \end{bmatrix} = -A^2\vec{x}[0]$$

no; the columns of the coefficient matrix aren't linearly independent

g) $\vec{x}[3] - A^3\vec{x}[0] = A^2\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2]$

$$\begin{bmatrix} A^2\vec{b} \\ A\vec{b} \\ \vec{b} \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \end{bmatrix} = -A^3\vec{x}[0]$$

no; there isn't a pivot in every column so the coefficient matrix can't be solved with a unique solution

h) $$\begin{bmatrix} A^3\vec{b} \\ A^2\vec{b} \\ A\vec{b} \\ \vec{b} \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[3] \\ u[4] \end{bmatrix} = -A^4\vec{x}[0]$$

yes; ~~it row red~~ the coefficient matrix row reduces to the identity matrix, so the columns are linearly independent and there is a unique ~~sol~~ solution

j) $\vec{0} - A^N x[0]$ must be in the span of
$\{\vec{b}, A\vec{b}, A^2\vec{b}, \ldots, A^{N-1}\vec{b}\}$

k) $\vec{x}[N] - A^N x[0]$ needs to be in
span $\{\vec{b}, A\vec{b}, A^2\vec{b}, \ldots, A^{N-1}\vec{b}\}$

yes this would be cool!

5. By myself

Problem 2e

In [2]:
```
import numpy as np
from numpy.linalg import inv

R = np.matrix([[-1, 3, -3, 1], [3, -6, 3, 0], [-3, 3, 0, 0], [1, 0, 0, 0]])
print(inv(R))
```

```
[[  0.00000000e+00   3.70074342e-17   7.40148683e-17   1.00000000e+00]
 [ -0.00000000e+00   3.70074342e-17   3.33333333e-01   1.00000000e+00]
 [ -0.00000000e+00   3.33333333e-01   6.66666667e-01   1.00000000e+00]
 [  1.00000000e+00   1.00000000e+00   1.00000000e+00   1.00000000e+00]]
```

Problem 4 Bieber's Segway

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

In [3]:
```
# %load gauss_elim.py
from gauss_elim import gauss_elim
```

In [4]:
```
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 [5]:
```
# 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 (f), (g), (h)

In [6]:
```
# You may use gauss_elim to help you find the row reduced echelon form.
# part (f)
print(gauss_elim(np.vstack((np.dot(A, b), b))))

# part (g)
print(gauss_elim(np.vstack((np.dot(np.power(A, 2), b), np.dot(A, b), b))))

# part (h)
print(gauss_elim(np.vstack((np.dot(np.power(A, 3), b), np.dot(np.power(A, 2)
```

```
[[ 1.          0.         -2.71346103 -6.71136185]
 [ 0.          1.         -0.01364471 -1.77564944]]
[[  1.           0.           0.          25.51610971]
 [  0.           1.           0.          -1.61359272]
 [ -0.          -0.           1.          11.87688754]]
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [-0. -0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

Part (i)

Preamble

This function will take care of animating the segway.

In [7]:
```
# 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(se
    xmax = max(around(states[:, 0].max() + segway_height / 2.0, 1), around(s

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

    # 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='blu

    # 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, segwayStic
        return time_text, rect, stick_line, control_text

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

Plug in your controller here

In [8]:
```
coeffMatrix = np.vstack((np.dot(np.power(A, 3), b), np.dot(np.power(A, 2), b

controls = np.linalg.solve(coeffMatrix, stateFinal - np.dot(np.dot(A, A), st
```

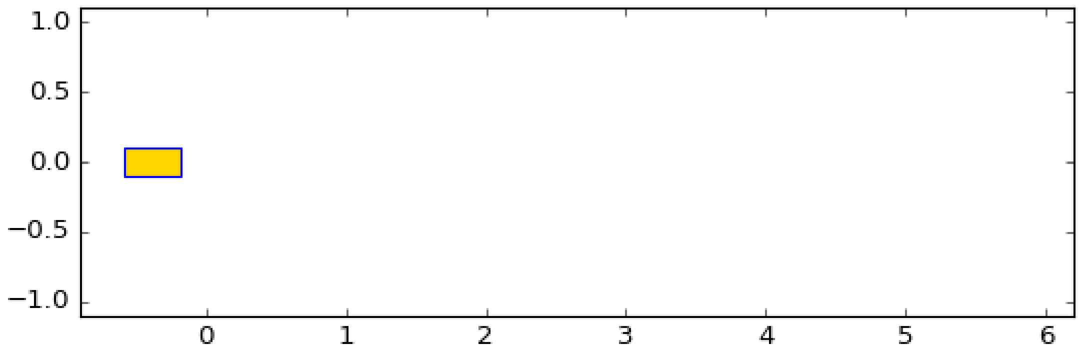
Simulation

In [9]:
```
# This will add an extra couple of seconds to the simulation after the input
# the effect of this is just to show how the system will continue after the
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 visualizati
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 visuali
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[s

# 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

In [10]:
```
%matplotlib nbagg
#%matplotlib qt
animate_segway(t, states, controls_final, stick_length)
```

Figure 1



In [ ]: