Vedaant Tiwari
9-15-17

EE16A HW3

1A $\begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$ $\det = (1\cdot 0) - (2\cdot 1)$
$= -2 \neq 0$ invertible

1B $\begin{bmatrix} 3 & 2 \\ 1 & -1 \end{bmatrix}$ $\det = (3\cdot -1) - (2\cdot 1)$
$-3 - 2 = -5 \neq 0$ invertible

1C $\begin{bmatrix} -\frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} \end{bmatrix}$ $\det = \left(\frac{1}{2}\right)^2 - \left(\frac{\sqrt{3}}{2}\right)^2$
$\frac{1}{4} - \frac{3}{4} = \frac{-1}{2} \neq 0$ invertible

1D $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$ $\det = 4 \neq 0$ invertible.

1E $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ $\det = (-1 + 0 + 0) - (0 + 1 + 0) = 1 - 1 = 0$
not invertible

1F $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 2 \\ 1 & 4 & 4 \end{bmatrix}$ $\det = (8 + 0 + 0) - (0 + 8 + 0)$ $8 - 8 = 0$
not invertible

1G $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$ $\det = 1 \det \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} + 0 + 0$

$\det \begin{vmatrix} -1 & 1 \\ 1 & -1 \end{vmatrix} = 1 - 1 = 0$ $1 \cdot 0 = 0 \neq 0$
not invertible.

1H $\begin{bmatrix} -1 & 1 & -\frac{1}{2} \\ 1 & 1 & -\frac{1}{2} \\ 0 & 1 & 1 \end{bmatrix}$ $\det = (-1 + 0 + \frac{-1}{2}) - (0 + \frac{1}{2} + 1) = -2 \neq 0$
invertible.

11 $\begin{matrix} + & - & + & - \\ 1 & -1 & 1 & -1 \end{matrix}$

$$\begin{vmatrix} 1 & 3 & 0 & -2 \\ -1 & 0 & 2 & 1 \\ 1 & 3 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{vmatrix} \quad \det = -1 \begin{vmatrix} 0 & -2 & 1 \\ 2 & 1 & 3 \\ 1 & 0 & 1 \end{vmatrix} + \begin{vmatrix} 3 & 0 & -2 \\ 0 & 2 & 1 \\ 3 & 1 & 0 \end{vmatrix}$$

$$(-1)\begin{vmatrix} 0 & -2 & 1 \\ 2 & 1 & 3 \\ 1 & 0 & 1 \end{vmatrix} = \begin{matrix} (0 - 6 + 0) \\ -(1 + 0 - 16) \end{matrix} \quad -(-6-(-15)) \Rightarrow -9$$

$$(1)\begin{vmatrix} 3 & 0 & -2 \\ 0 & 2 & 1 \\ 3 & 1 & 0 \end{vmatrix} = \begin{matrix} (0 + 0 + 0) \\ -(-12 + 3 + 0) \end{matrix} = 9$$

$\det = -9 + 9 = 0$ not invertible

2A $\begin{matrix} 1 \\ 2 \end{matrix}\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ = corresponding transition matrix

$$\vec{x}[n+1] = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \vec{x}[n]$$

2B $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \quad \vec{x}[1] = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Universes 1

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} \quad \vec{x}[1] = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Alt. Universe

2C Using the data from timestep 1, we are unable to determine the water levels at timestep 0 because there are many previous timestep or initial states that when multiplied together with the corresponding transition matrix.

For Example in case 1:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

alt

$$x = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

As these two answers with different starting points show, it is impossible to determine the previous timestep, as this information is not preserved.

2D $A \cdot \vec{x}_1 = \vec{x}_2$ In order to determine the previous timestep $\vec{x}_1$, A must be invertible.

Since $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ is not invertible, we are unable to recover the information in the previous matrix

In ~~A A~~ $\cdot x_1 = \vec{x}_2 \, A^{-1}$

$\vec{x}_1 = \vec{x}_2 \cdot A^{-1}$

Matrix A must be invertible for this to be true.

2E This statement implies that A is invertible, and by definition of invertibility, it is linearly independent, one to one, and onto.

$A \cdot \vec{x}_1 = \vec{x}_2$

in order to get $\vec{x}_1$ (previous timestep)

$A^{-1} \cdot A \cdot \vec{x}_1 = \vec{x}_2 A^{-1}$ A must be invertible

$I_n \vec{x}_1 = \vec{x}_2 A^{-1}$

2F If the entries of each column vector sums to one, the total water in the system is constant. No water is added or removed from this system

2G $\begin{bmatrix} 0 & 0 & 0 \\ 0.4 & 0.5 & 0.2 \\ 0 & 0.6 & 0.35 \end{bmatrix}$ The A matrix physically implies that there is an external source of water so the total amount of water in the system is changing and is NOT constant. We know this because the rows and columns do not add up to 1.

2H $\begin{bmatrix} a_{1,} & \dots & a_{1n} \\ \vdots & a_{ij} & \vdots \\ a_{n,} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x \\ \vdots \\ x \end{bmatrix} = \begin{bmatrix} x \\ \vdots \\ x \end{bmatrix}$ $[a_1 \dots a_n] = 1$

$[xa_1 \dots xa_{1n}] = x[a_{11} \dots a_{1n}] = x$

Because the sum of the rows is one, multiplying by a vector essentially becomes the uniform vector, regardless of how many columns / rows the two matrices have, still long as they are multiplicable

3A The state transition matrix A is like as follows:

$$\begin{bmatrix} a & f & d \\ b & e & d \\ c & e & f \end{bmatrix} = A \qquad \begin{aligned} a+f+d &= 1 \\ b+e+d &= 1 \\ c+e+f &= 1 \end{aligned}$$

$$0 < a, b, c, d, e, f < 1$$

$\vec{s}[n+1] = A\vec{s}[n]$ ← keeps track of the water distribution among the three reservoirs

3B $\begin{bmatrix} a & f & d \\ b & e & d \\ c & e & f \end{bmatrix}$ where $\begin{aligned} a+f+d &= 1 \\ b+e+d &= 1 \\ c+e+f &= 1 \end{aligned}$

Because we know that the reservoir system is conservative, we know that the state transition matrix above row sum to one which means that applying this system to a uniform vector will return the same uniform vector as proven in prop 2H.

Therefore $\vec{s}[n+1] = \vec{s}[n] = \vec{s}$ for any value of n.

The rows in this matrix add up to one, so we know that the total water stays the same and we can use the properties with uniform vectors.

3C

$$A = \begin{bmatrix} 1/5 & 2/5 & 2/5 \\ 2/5 & 1/5 & 2/5 \\ 2/5 & 2/5 & 1/5 \end{bmatrix} = S$$

In order to determine $\vec{s}[n]$ from the subsequent state $\vec{s}[n+1]$, A must be an invertible matrix

$$A \cdot \vec{s}[n] = \vec{s}[n+1]$$

$$A^{-1} A \cdot \vec{s}[n] = A^{-1} \vec{s}[n+1]$$

$$I \cdot \vec{s}[n] = A^{-1} \vec{s}[n+1]$$ so A must be invertible

In order to determine whether it is possible to determ $\vec{s}[n]$ from $\vec{s}[n+1]$

$$\det \begin{bmatrix} 1/5 & 2/5 & 2/5 \\ 2/5 & 1/5 & 2/5 \\ 2/5 & 2/5 & 1/5 \end{bmatrix} = \left(\frac{1}{5^3} + \frac{8}{5^3} + \frac{8}{5^3}\right) = \frac{17}{5^3}$$

$$- \left(\frac{4}{5^3} + \frac{4}{5^3} + \frac{4}{5^3}\right) = \frac{12}{5^3}$$

$$\frac{17}{5^3} - \frac{12}{5^3} = \frac{5}{5^3} = \boxed{\frac{1}{25}} \qquad \frac{1}{25} \neq 0$$ so A is invertible, so it is possible to determine $\vec{s}[n]$ from $\vec{s}[n+1]$

4

$$\vec{x} = \begin{bmatrix} p \\ \dot{p} \\ \theta \\ \dot{\theta} \end{bmatrix} \begin{array}{l} \leftarrow \text{position} \\ \leftarrow \text{velocity} \\ \leftarrow \text{angle} \\ \leftarrow \text{angular velocity} \end{array}$$

$$\vec{x}[n+1] = A\vec{x}[n] + \vec{b}u[n]$$

$A \in \mathbb{R}^{4\times 4}$, $\vec{b} \in \mathbb{R}^{4\times 1}$

4A $\vec{x}[1]$ when $n = 0$

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

4B
$$\begin{aligned}\vec{x}[2] &= A\vec{x}[1] + \vec{b}u[1] \\ &= A[A\vec{x}[0] + \vec{b}u[0]] + \vec{b}u[1] \\ &= A^2\vec{x}[0] + A\vec{b}u[0] + \vec{b}u[1]\end{aligned}$$

$$\begin{aligned}\vec{x}[3] &= A\vec{x}[2] + \vec{b}u[2] \\ &= A[A^2\vec{x}[0] + A\vec{b}u[0] + \vec{b}u[1]] + \vec{b}u[2] \\ &= A^3\vec{x}[0] + A^2\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2]\end{aligned}$$

$$\begin{aligned}\vec{x}[4] &= A\vec{x}[3] + \vec{b}u[3] \\ &= A[A^3\vec{x}[0] + A^2\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2]] + \vec{b}u[3] \\ &= A^4\vec{x}[0] + A^3\vec{b}u[0] + A^2\vec{b}u[1] + A\vec{b}u[2] + \vec{b}u[3]\end{aligned}$$

4C
$$\vec{x}[n] = A^n\vec{x}[0] + \sum_{i=0}^{n-1} A^{n-1-i}\vec{b}u[i]$$

$$A = \begin{bmatrix} 1 & 0.05 & -0.01 & 0 \\ 0 & 0.22 & -0.17 & -0.01 \\ 0 & 0.10 & 1.14 & 0.10 \\ 0 & 1.6 & [illegible]5 & 1.14 \end{bmatrix} \qquad \vec{b} \begin{bmatrix} 0.01 \\ 0.21 \\ -0.03 \\ -0.44 \end{bmatrix}$$

4 (cont)

$$\vec{x}[0] = \begin{bmatrix} -0.3853493 \\ 6.1032227 \\ 0.8120005 \\ -14 \end{bmatrix}$$

$$\vec{x}[t] = A^t \cdot \vec{x}[0] + \vec{b} \sum_{i=0}^{t-1} A^{t-1-i} \cdot u[i]$$

4D Two time steps

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

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

$$Ax = b$$

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

Python, not possible in 3 time steps

4E Three time steps

$$\vec{x}[3] = A^3\vec{x}[0] + A^2bu[0] + Abu[1] + bu[2]$$

$Ax = b$, , not poss

$$\begin{bmatrix} | & | & | & 0 \\ A^2b & Ab & b & 0 \\ | & | & | & 0 \\ | & | & | & 0 \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \end{bmatrix} = \begin{bmatrix} | \\ A^3x[0] \\ | \end{bmatrix}$$

4F Born Time step

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

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

$Ax = b$

$$\begin{bmatrix} A^3b & A^2b & Ab & b \\ | & | & | & | \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ u[2] \\ u(3) \end{bmatrix} = A^4x[0]$$

yes 4 step is enough

4G Ipython

4H A must be invertible and be an $N \times N$ matrix

4G A must be invertible and be an $N \times N$ matrix

5 Check Invertibility

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ det = 4-6 = 2 ≠ 0, invertible

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ det = 1-0 ≠ 0 invertible

$\begin{bmatrix} 6 & 4 & 3 \\ 5 & 2 & 1 \\ 12 & 8 & 6 \end{bmatrix}$ det = (72 + 120 + 48)

−(72 + 48 + 120) = 0

not invertible

6 Worked on this HW alone. [illegible] [illegible] [illegible]
if I'm being honest. [illegible] [illegible] [illegible] [illegible],
24 [illegible] as [illegible] as [illegible] [illegible] [illegible]

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