1. Mechanical Inverses

a) $A^{-1} = \left[\begin{array}{cc|cc} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

When A applied to $\vec{v}$, it switches the x and y axis.
For example, if $\vec{v}$ is $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $A\vec{v}$ makes it $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$.

b) $A^{-1} = \left[\begin{array}{cc|cc} -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right] = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

When A applied to $\vec{v}$, it reflects it about the y-axis.
For example, $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ becomes $\begin{bmatrix} -1 \\ 2 \end{bmatrix}$.

e) $A^{-1} = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & -2 & -2 & 1 \end{array}\right] = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & -\frac{1}{2} \end{array}\right] =$

$= \left[\begin{array}{cc|cc} 1 & 0 & 0 & \frac{1}{2} \\ 0 & 1 & 1 & -\frac{1}{2} \end{array}\right] = \begin{bmatrix} 0 & \frac{1}{2} \\ 1 & -\frac{1}{2} \end{bmatrix}$

f) $A^{-1} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 & 0 \\ 1 & 4 & 4 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 & 0 \\ 0 & 4 & 4 & -1 & 0 & 1 \end{array}\right] =$

$= \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & -2 & 1 \end{array}\right]$

A^{-1} doesn't exist. A row of 0s means the matrix is linearly dependent. Thus, A is not invertible and A^{-1} doesn't exist.

2. Finding Null spaces and Column spaces

a) $\underline{3}$ is the max possible number of linearly independent column vectors any 3×5 matrix can have. This is because you can have only 3 vectors to represent any vector in $\mathbb{R}^3$.

b) For $A = \begin{bmatrix} 1 & 1 & 0 & -2 & 3 \\ 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$, the set $\left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right\}$ will span the column space of A.

You need $\boxed{2}$ unique vectors to span the column space of A.

c) $A\vec{x} = 0$

$$\begin{array}{c} \left[\begin{array}{ccccc|c} 1 & 1 & 0 & -2 & 3 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \\ \begin{array}{cccccc} x & y & z & u & v & \end{array} \end{array} \Rightarrow \begin{array}{l} x = -y + 2u - 3v \\ z = u - v \end{array}$$

solution: $\begin{bmatrix} -y + 2u - 3v \\ y \\ u - v \\ u \\ v \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} y + \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} u + \begin{bmatrix} -3 \\ 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} v$

So, the set $\left\{ \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -3 \\ 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} \right\}$ will span the nullspace of A

The dimension of the null space of A is $\boxed{3.}$

d) The sum of the dimensions of the column space and null space is equal to the dimension of the row vectors of A and # of columns in A.

3. Properties of Pump Systems

a) Consider a system consisting of 2 reservoirs such that the entries of each column in the system's state transition matrix sum to one. If s is the total water in the system at timestep n, then total water at timestep $n+1$ is also s.

b) Given: $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ $\quad \vec{x}[n] = \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix}$

$a_{11} + a_{21} = 1$ $\quad x_1[n] + x_2[n] = s$

$a_{12} + a_{22} = 1$

c) $x_1[n+1] + x_2[n+1] = s$

d) $A\vec{x}[n] = \vec{x}[n+1]$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix} = \begin{bmatrix} x_1[n+1] \\ x_2[n+1] \end{bmatrix}$$

$$\begin{cases} a_{11} x_1[n] + a_{12} x_2[n] = x_1[n+1] \\ a_{21} x_1[n] + a_{22} x_2[n] = x_2[n+1] \end{cases}$$

$$\begin{aligned} x_1[n+1] + x_2[n+1] &= a_{11}x_1[n] + a_{12}x_2[n] + a_{21}x_1[n] + a_{22}x_n[n] \\ &= (a_{11} + a_{21})x_1[n] + (a_{12} + a_{22})x_2[n] \\ &= (1)x_1[n] + (1)x_2[n] \\ &= x_1[n] + x_2[n] \end{aligned}$$

$$\underline{x_1[n+1] + x_2[n+1] = s}$$

Thus, we showed that sum of reservoirs at timestep $n+1$ is s.

e) Let $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix}$

Given that the sum of columns add to 1.

Also given $\sum_{i=1}^{k} x_i = s$

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

$$\begin{bmatrix} x_1[n+1] \\ x_2[n+1] \\ \vdots \\ x_k[n+1] \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix} \begin{bmatrix} x_1[n] \\ x_2[n] \\ \vdots \\ x_k[n] \end{bmatrix}$$

$$\begin{bmatrix} x_1[n+1] \\ x_2[n+1] \\ \\ x_k[n+1] \end{bmatrix} = \begin{bmatrix} a_{11}x_1[n] + a_{12}x_2[n] + \ldots + a_{1k}x_k[n] \\ a_{21}x_1[n] + a_{22}x_2[n] + \ldots + a_{2k}x_k[n] \\ \vdots \\ a_{k1}x_1[n] + a_{k2}x_2[n] + \ldots + a_{kk}x_k[n] \end{bmatrix}$$

$x_1[n+1] + x_2[n+1] + \ldots + x_k[n+1] =$

$= (a_{11} + a_{21} + \ldots + a_{k1})x_1[n] + \ldots + (a_{1k} + a_{2k} + \ldots + a_{kk})x_k[n]$

$= (1)\,x_1[n] + \ldots + (1)\,x_k[n+1]$ (sum of columns add to 1)

$\underline{\underline{= s}}$ (given that $\sum_{i=1}^{k} x_i = s$)

Thus, this proves the theorem in the general case.

4. Traffic Flows

a) $\begin{cases} t_1 + t_3 = 0 \\ t_2 - t_1 = 0 \\ -t_3 - t_2 = 0 \end{cases}$ $\begin{cases} 10 + t_3 = 0 \\ t_2 - 10 = 0 \\ -t_3 - t_2 = 0 \end{cases}$ $\begin{array}{|l} t_3 = -10 \\ t_2 = 10 \\ 10 - 10 = 0 \end{array}$

$\begin{cases} t_3 = -10 \\ t_2 = 10 \end{cases}$

b) $\begin{cases} t_1 + t_3 - t_4 = 0 \\ t_2 - t_1 = 0 \\ t_5 - t_3 - t_2 = 0 \\ t_4 - t_5 = 0 \end{cases}$

Berkeley student: we know t_3 and t_5

$\begin{cases} t_1 - t_4 = -t_3 \\ t_2 = t_1 \\ t_2 = t_5 - t_3 \\ t_4 = t_5 \end{cases}$

Let $t_3 = u$, $t_5 = v$, then $\begin{cases} t_1 = v - u \\ t_2 = v - u \\ t_4 = v \end{cases}$

We $\boxed{\text{can}}$ find $[t_1, t_2, t_3, t_4, t_5]^T$ with the Berkeley student's suggestion.

Stanford student: let $t_1 = u$, $t_2 = v$

$\begin{cases} t_3 - t_4 = -t_1 \\ t_2 = t_1 \\ t_5 - t_3 = t_2 \\ t_4 - t_5 = 0 \end{cases}$ $\left[\begin{array}{ccccc|c} 0 & 0 & 1 & -1 & 0 & -u \\ \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} & \cancel{0} & u \\ 0 & 0 & -1 & 0 & 1 & v \\ 0 & 0 & 0 & 1 & -1 & 0 \end{array}\right]$

$\left[\begin{array}{ccc|c} 1 & -1 & 0 & -u \\ -1 & 0 & 1 & v \\ 0 & 1 & -1 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & -1 & 0 & -u \\ 0 & -1 & 1 & v-u \\ 0 & 1 & -1 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 1 & -1 & 0 & -u \\ 0 & -1 & 1 & v-u \\ 0 & 0 & 0 & v-u \end{array}\right]$

You $\boxed{\text{cannot}}$ find using Stanford student's suggestion because the rows are linearly dependent.

$B\vec{t} = \vec{0}$

c) $$\begin{bmatrix} 1 & 0 & 1 & -1 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The sum of each column add up to 0, meaning that no cars accumulate.

d) $$\left[\begin{array}{ccccc|c} 1 & 0 & 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{ccccc|c} 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$\left[\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \end{array}\right] \qquad \begin{cases} t_1 = -t_3 + t_5 \\ t_2 = -t_3 + t_5 \\ t_4 = t_5 \end{cases}$$

$$N(A) = \left\{ \begin{bmatrix} -1 \\ -1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \right\}$$

dimension of null space of $A = \boxed{2}$

e) $$M_s\vec{t} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \vec{t} = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

f) $\boxed{\text{No}}$, measuring any k roads is not sufficient to recover all of the true flows.

Counterexample: The measurement the Stanford student proposed didn't give us any new information. Choosing 2 measurements from an intersection with only 2 roads doesn't give us any new information because the 2 measurements are equal. Thus, it causes the matrix to become linearly dependent and making it impossible to find the flow values.

5. Seguay Tours

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

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

$\vec{x}[3] = A\vec{x}[2] + \vec{b}u[2] = A\left(AA\vec{x}[0] + A\vec{b}u[0] + \vec{b}u[1]\right) + \vec{b}u[2]$

$\vec{x}[4] = A\vec{x}[3] + \vec{b}u[3] =$

$= A\left(AAA\vec{x}[0] + AA\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2]\right) + \vec{b}u[3]$

c) $\vec{x}[N] = A^N\vec{x}[0] + \sum_{i=0}^{N-1} A^i \vec{b}u[N-1-i]$

Apply A N times $(A_1 \cdot A_2 \cdot \ldots \cdot A_N)$

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

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

Since $\vec{x}_f = \vec{0}$, $\vec{x}[2] = \vec{x}_f = \vec{0}$

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

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

$\left[\begin{array}{cc|c} A\vec{b} & \vec{b} & -AA\vec{x}[0] \end{array}\right]$

After plugging iPython, I get the matrix:

$$\left[\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{array}\right]$$

Since $0 \neq 1$, we don't have solution to this matrix.

Thus, you $\boxed{\text{can't}}$ reach $\vec{x}_f$ in 2 steps.

e) $\vec{x}[3] = AAA\vec{x}[0] + AA\vec{b}u[0] + A\vec{b}u[1] + \vec{b}u[2]$

$$\begin{bmatrix} AA\vec{b} & A\vec{b} & \vec{b} \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \end{bmatrix} = \vec{x}_f - AAA\vec{x}[0]$$

$$\left[\begin{array}{ccc|c} AA\vec{b} & A\vec{b} & \vec{b} & -AAA\vec{x}[0] \end{array}\right]$$

Plugging to iPython gives me the matrix:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]$$

Same as part (d), we get the contradiction that $0 = 1$

Thus, you $\boxed{\text{can't}}$ reach $\vec{x}_f$ in 3 steps.

f) $\vec{x}[4] = AAAA\vec{x}[0] + AAA\vec{b}u[0] + AA\vec{b}u[1] + A\vec{b}u[2] + \vec{b}u[3]$

$$\begin{bmatrix} AAA\vec{b} & AA\vec{b} & A\vec{b} & \vec{b} \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \end{bmatrix} = \vec{x}_f - AAAA\vec{x}[0]$$

$$\left[\begin{array}{cccc|c} AAA\vec{b} & AA\vec{b} & A\vec{b} & \vec{b} & -AAAA\vec{x}[0] \end{array}\right]$$

After plugging to iPython, I get the matrix:

$$\left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & -13.24875075 \\ 0 & 1 & 0 & 0 & 23.73325125 \\ 0 & 0 & 1 & 0 & -11.57181872 \\ 0 & 0 & 0 & 1 & 1.46515973 \end{array}\right]$$

We have a unique answer, so you $\boxed{\text{can}}$ reach $\vec{x}_f$ in 4 steps.

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

The combination of controls $u[0]$ and $u[1]$ allow you to move in any $\vec{v} \in \text{span}\{A\vec{b}, \vec{b}\}$

i) The positions that can be reached in N time steps can be expressed as $\text{span}\{A^{N-1}\vec{b}, A^{N-2}\vec{b}, \dots, A\vec{b}, \vec{b}\}$

7. Homework Process

I worked on this homework alone. I read until Note 8, watched the lecture, and worked on the homework.

EECS16A: Homework 4

Problem 5: Bieber's Segway

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

In [4]:
```
# %load gauss_elim.py
from gauss_elim import gauss_elim
```

In [5]:
```
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])
```

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

In [15]:
```
# You may use gauss_elim to help you find the row reduced echelon form.

# part D
left = np.concatenate((np.dot(A, b).reshape(4,1), b.reshape(4,1)), 1).re
m = np.c_[left, (-1) * np.dot(A, np.dot(A, state0))]
print(gauss_elim(m))


# part E
leftE = np.concatenate((np.dot(A, np.dot(A,b)).reshape(4,1), np.dot(A,b)
me = np.c_[leftE, (-1) * np.dot(A, np.dot(A, np.dot(A, state0)))]
print(gauss_elim(me))


# part F
aaab = np.dot(A, np.dot(A, np.dot(A,b))).reshape(4,1)
aab = np.dot(A, np.dot(A,b)).reshape(4,1)
ab = np.dot(A,b).reshape(4,1)
leftF = np.concatenate((aaab, aab, ab, b.reshape(4,1)), 1).reshape(4,4)
mf = np.c_[leftF, (-1) * np.dot(A, np.dot(A, np.dot(A, np.dot(A, state0)
print(gauss_elim(mf))
```

```
[[ 1.   0.   0.]
 [ 0.   1.   0.]
 [-0.  -0.   1.]
 [ 0.   0.   0.]]
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
[[  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.

In [16]:
```
# 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 anim
    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), aroun
    xmax = max(around(states[:, 0].max() + segway_height / 2.0, 1), arou

    # 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, '', 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
        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=

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

    # 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, segway
        return time_text, rect, stick_line, control_text

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

Plug in your controller here

In [20]:
```
controls = np.array([-13.24875075, 23.73325125, -11.57181872, 1.46515973
```

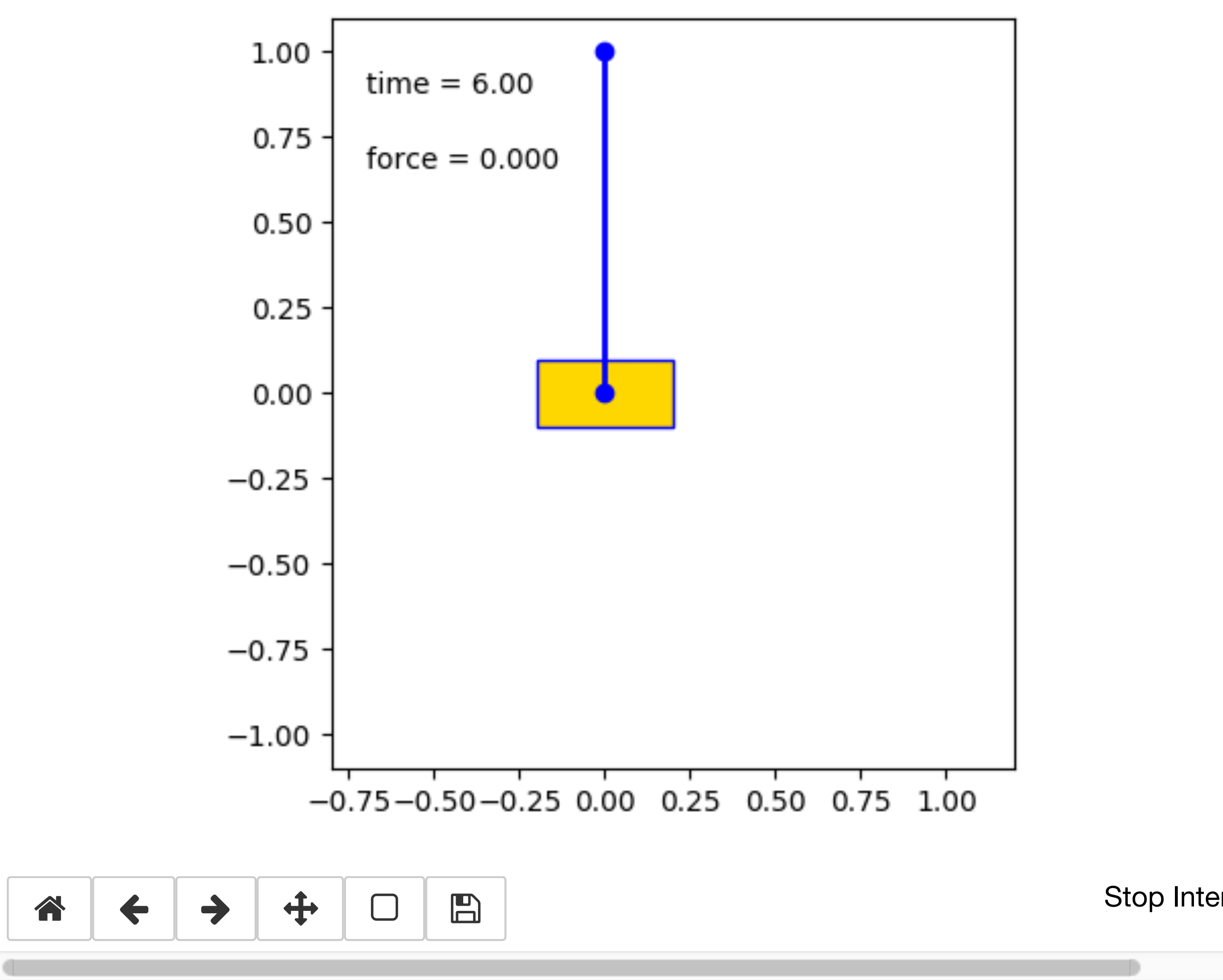
Simulation

In [21]:
```
# This will add an extra couple of seconds to the simulation after the i
# the effect of this is just to show how the system will continue after
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 visuali
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 visu
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_fin

# 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 [22]:
```
%matplotlib nbagg
# %matplotlib qt
anim = animate_segway(t, states, controls_final, stick_length)
anim
```

Figure 1



Out[22]: <matplotlib.animation.FuncAnimation at 0x127d35d30>

In [ ]: