EECS 16A HW04

Bryan Ngo

2019-09-24

1 Mechanical Inverses

1.a

$$\left[\begin{array}{cc|cc} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right] \tag{1.1}$$

$$\overset{r_1 \leftrightarrows r_2}{\Longrightarrow} \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] \tag{1.2}$$

$$\mathbf{A}^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.3}$$

A reflects the vector across the line $y = x$.

1.b

$$\left[\begin{array}{cc|cc} -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right] \tag{1.4}$$

$$\overset{-r_1 \to r_1}{\Longrightarrow} \left[\begin{array}{cc|cc} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right] \tag{1.5}$$

$$\mathbf{A}^{-1} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \tag{1.6}$$

A reflects the vector across the y-axis.

1.e

$$\left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{array}\right] \tag{1.7}$$

$$\overset{r_2-2r_1\to r_2}{\Longrightarrow} \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & -2 & -2 & 1 \end{array}\right] \tag{1.8}$$

$$\overset{r_2/-2\to r_2}{\Longrightarrow} \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & -\frac{1}{2} \end{array}\right] \tag{1.9}$$

$$\overset{r_1-r_2\to r_1}{\Longrightarrow} \left[\begin{array}{cc|cc} 1 & 0 & 0 & \frac{1}{2} \\ 0 & 1 & 1 & -\frac{1}{2} \end{array}\right] \tag{1.10}$$

$$\mathbf{A}^{-1} = \begin{bmatrix} 0 & \frac{1}{2} \\ 1 & -\frac{1}{2} \end{bmatrix} \tag{1.11}$$

$\mathbf{A}$ flips $\hat{\jmath}$ to the x-axis and shears a given vector by 2.

1.f

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 & 0 \\ 1 & 4 & 4 & 0 & 0 & 1 \end{array}\right] \tag{1.12}$$

$$\overset{r_3-r_1\to r_3}{\Longrightarrow} \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 0 & 1 & 0 \\ 0 & 4 & 4 & -1 & 0 & 1 \end{array}\right] \tag{1.13}$$

$$\overset{2r_2\to r_2}{\Longrightarrow} \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 4 & 4 & 0 & 2 & 0 \\ 0 & 4 & 4 & -1 & 0 & 1 \end{array}\right] \tag{1.14}$$

$$\overset{r_3-r_2\to r_3}{\Longrightarrow} \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 4 & 4 & 0 & 2 & 0 \\ 0 & 0 & 0 & -1 & -2 & 1 \end{array}\right] \tag{1.15}$$

The row of zeroes indicates that $\mathbf{A}^{-1}$ does *not* exist.

$\mathbf{A}$ flattens the x-component to the xz-plane, then flattens the y-component and z-component to the yz-plane.

2 Finding Null Spaces and Column Spaces

2.a

The maximum number of linearly independent columns in a 3×5 matrix is 3, since there are 3 possible pivots in the matrix.

2.b

Given our matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & -2 & 3 \\ 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.1}$$

$\text{colsp}(\mathbf{A}) \in \mathbb{R}^3$ even if there are all zeroes in the last row. Thus, we need only pick a basis set of vectors to satisfy the columnspace. Since the last two vectors can be made out of a linear combination of the 2nd and 3rd, they are within the span of the 2nd and 3rd columns. The choice is obvious that

$$\text{colsp}(\mathbf{A}) = \text{span}\left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} = \mathbb{R}^2 \tag{2.2}$$

meaning the columnspace has dimension 2.

2.c

$$\text{null}(\mathbf{A}) = \left[\begin{array}{ccccc|c} 1 & 1 & 0 & -2 & 3 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{2.3}$$

The row of zeroes indicates that the nullspace is infinite. A system of linear equations of this matrix would look like

$$\begin{aligned} x_1 + x_2 \quad - 2x_4 + 3x_5 \; &= 0 \\ x_3 - \; x_4 + \; x_5 \; &= 0 \\ 0 &= 0 \end{aligned} \tag{2.4}$$

Thus, we must express our nullspace in terms of our free variables, which in this case is $x_{2,4,5}$ for the x_n column. This means we can express our nullspace as

$$\begin{aligned} x_1 &= -x_2 + 2x_4 - 3x_5 \\ x_2 &= x_2 \\ x_3 &= x_4 \qquad - \qquad x_5 \\ x_4 &= x_4 \\ x_5 &= x_5 \end{aligned} \tag{2.5}$$

Separating our free variables into vectors:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} -x_2 + 2x_4 - 3x_5 \\ x_2 \\ x_4 - x_5 \\ x_4 \\ x_5 \end{bmatrix} \tag{2.6}$$

$$= \begin{bmatrix} -x_2 \\ x_2 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 2x_4 \\ 0 \\ x_4 \\ x_4 \\ 0 \end{bmatrix} + \begin{bmatrix} -3x_5 \\ 0 \\ -x_5 \\ 0 \\ x_5 \end{bmatrix} \tag{2.7}$$

$$= x_2 \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + x_4 \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} + x_5 \begin{bmatrix} -3 \\ 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} \tag{2.8}$$

Since every vector in the nullspace can be written as a linear combination of the free variables, the three column vectors form a basis for the nullspace:

$$\text{null}(\mathbf{A}) = \text{span} \left\{ \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -3 \\ 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} \right\} \tag{2.9}$$

meaning the nullspace has dimension 3.

2.d

The sum of the dimensions of the nullspace and the columnspace is the number of columns in $\mathbf{A}$.

3 Properties of Pump Systems

3.a

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

3.b

We are given two reservoirs, meaning that the system can be written as a matrix-vector multiplication

$$\underbrace{\begin{bmatrix} x_1[n+1] \\ x_2[n+1] \end{bmatrix}}_{\mathbf{x}[n+1]} = \underbrace{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1[n] \\ x_2[n] \end{bmatrix}}_{\mathbf{x}[n]} \tag{3.1}$$

where we are given the constraint that

$$a_{11} + a_{21} = a_{12} + a_{22} = 1 \tag{3.2}$$

3.c

What we are trying to prove is that

$$s_3 + s_4 = s_1 + s_2 \tag{3.3}$$

under the transformation

$$\begin{bmatrix} s_3 \\ s_4 \end{bmatrix} = \mathbf{A} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \tag{3.4}$$

for some state vector $\mathbf{s}$.

3.d

Proof. Given our matrix multiplication

$$\begin{bmatrix} s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \tag{3.5}$$

If we carry out the matrix multiplication,

$$\begin{bmatrix} s_3 \\ s_4 \end{bmatrix} = s_1 \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} + s_2 \begin{bmatrix} a_{12} \\ a_{22} \end{bmatrix} \tag{3.6}$$

$$= \begin{bmatrix} s_1 a_{11} + s_2 a_{12} \\ s_1 a_{21} + s_2 a_{22} \end{bmatrix} \tag{3.7}$$

Suppose we take the sum of $s_{3,4}$. If we rearrange the terms and factor out $s_{1,2}$,

$$s_3 + s_4 = s_1 a_{11} + s_2 a_{12} + s_1 a_{21} + s_2 a_{22} \tag{3.8}$$

$$= s_1 \cancelto{1}{(a_{11} + a_{21})} + s_2 \cancelto{1}{(a_{12} + a_{22})} \tag{3.9}$$

$$= s_1 + s_2 \tag{3.10}$$

□

3.e

Theorem 2. *Given the matrix-vector multiplication*

$$\underbrace{\begin{bmatrix} x_1[n+1] \\ x_2[n+1] \\ \vdots \\ x_k[n+1] \end{bmatrix}}_{\mathbf{x}[n+1]} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1[n] \\ x_2[n] \\ \vdots \\ x_k[n] \end{bmatrix}}_{\mathbf{x}[n]} \tag{3.11}$$

for some state vector $\mathbf{x}[n]$ *with* k *unknowns multiplied by a transition matrix* $\mathbf{A} \in \mathbb{R}^{m \times k}$*, where*

$$\sum_{i=1}^{m} a_{i1} = \sum_{i=1}^{m} a_{i2} = \cdots = \sum_{i=1}^{m} a_{ik} = 1 \tag{3.12}$$

then it follows that

$$\sum_{i=1}^{k} x_i[n+1] = \sum_{i=1}^{k} x_i[n] \tag{3.13}$$

Proof. Carrying out the matrix multiplication,

$$\begin{bmatrix} x_1[n+1] \\ x_2[n+1] \\ \vdots \\ x_k[n+1] \end{bmatrix} = x_1[n] \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + x_2[n] \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + x_k[n] \begin{bmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{mk} \end{bmatrix} \tag{3.14}$$

$$= \begin{bmatrix} x_1[n]a_{11} + x_2[n]a_{12} + \cdots + x_k[n]a_{1k} \\ x_1[n]a_{21} + x_2[n]a_{22} + \cdots + x_k[n]a_{2k} \\ \vdots \\ x_1[n]a_{m1} + x_2[n]a_{m2} + \cdots + x_k[n]a_{mk} \end{bmatrix} \tag{3.15}$$

Suppose we take the sum of the entries of $\mathbf{x}[n+1]$, We can rearrange the sum such that we can factor out the components of the state vector. Then using the given information in Equation 3.12, we can cancel those sums to 1:

$$\sum_{i=1}^{k} x_i[n+1] = x_1[n](\cancelto{1}{a_{11} + a_{21} + \cdots + a_{m1}}) + x_2[n](\cancelto{1}{a_{12} + a_{22} + \cdots + a_{m2}}) + \cdots \tag{3.16}$$

$$\sum_{i=1}^{k} x_i[n+1] = \sum_{i=1}^{k} x_k[n] \tag{3.17}$$

□

4 Traffic Flows

4.a

Given t_1, we can determine the values of $t_{2,3}$. This is because $t_{2,3}$ is defined in terms of t_1. In the example given, we have

$$\begin{aligned} t_3 + 10 &= 0 \\ t_2 - 10 &= 0 \end{aligned} \tag{4.1}$$

meaning that

$$\begin{aligned} t_1 &= 10 \\ t_2 &= 10 \\ t_3 &= -10 \end{aligned} \tag{4.2}$$

4.b

We can represent the given diagram as

$$\begin{aligned} t_1 + t_3 - t_4 &= 0 \\ -t_1 + t_2 &= 0 \\ -t_2 - t_3 + t_5 &= 0 \\ t_4 - t_5 &= 0 \end{aligned} \tag{4.3}$$

If we know $t_{3,5}$ according to the Berkeley student, then it is possible to deduce the rest of the traffic vector, by first determining $t_{2,4}$, then using it to determine t_1. On the other hand, knowing $t_{1,2}$ renders it impossible to determine the rest of the traffic vector because there is no way to recover any of the other vectors. This means the Berkeley student's sensor placement is superior.[1]

4.c

Given Equation 4.3, we can construct a transition matrix out of the road system

$$\underbrace{\begin{bmatrix} 1 & 0 & 1 & -1 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}}_{\mathbf{B}} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.4}$$

The sum of the columns of $\mathbf{B}$ is 0, representing the fact that the net flux of traffic is zero.

[1] Go Bears!

4.d

Finding the nullspace of $\mathbf{B}$,

$$\text{null}(\mathbf{B}) = \left[\begin{array}{ccccc|c} 1 & 0 & 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{array}\right] \tag{4.5}$$

$$\xRightarrow{r_2+r_1\to r_2} \left[\begin{array}{ccccc|c} 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{array}\right] \tag{4.6}$$

$$\xRightarrow{r_3+r_2\to b_3} \left[\begin{array}{ccccc|c} 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{array}\right] \tag{4.7}$$

$$\xRightarrow{-r_4\to r_4} \left[\begin{array}{ccccc|c} 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{array}\right] \tag{4.8}$$

$$\xRightarrow{r_4+r_3\to r_4} \left[\begin{array}{ccccc|c} 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{4.9}$$

$$\xRightarrow{r_3+r_4\to r_3} \left[\begin{array}{ccccc|c} 1 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{4.10}$$

$$\xRightarrow{r_2+r_4\to r_2} \left[\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \tag{4.11}$$

We end up with a row of zeroes, meaning that our nullspace can be represented as a linear combination of basis vectors. Our free variables are $t_{3,5}$. Solving for the vectors in terms of the free variables and reconverting to a system of linear

equations:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} -x_3 + x_5 \\ -x_3 + x_5 \\ x_3 \\ x_5 \\ x_5 \end{bmatrix} \tag{4.12}$$

$$= x_3 \begin{bmatrix} -1 \\ -1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + x_5 \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{4.13}$$

meaning our nullspace is

$$\text{null}(\mathbf{B}) = \text{span} \left\{ \begin{bmatrix} -1 \\ -1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \right\} \tag{4.14}$$

The dimension of the nullspace is 2.

4.e

The Stanford measurement is

$$\mathbf{M}_S \mathbf{t} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \tag{4.15}$$

4.f

The statement is not true, given the Stanford vector as a counterexample. Even though the dimension of the nullspace was 2, 2 measurements was not sufficient to resolve the system.

5 Segway Tours

5.a

$$\mathbf{x}[1] = \mathbf{A}\mathbf{x}[0] + \mathbf{b}u[0] \tag{5.1}$$

5.b

$$\mathbf{x}[2] = \mathbf{A}^2\mathbf{x}[0] + \mathbf{A}\mathbf{b}u[0] + \mathbf{b}u[1] \tag{5.2}$$

$$\mathbf{x}[3] = \mathbf{A}^3\mathbf{x}[0] + \mathbf{A}^2\mathbf{b}u[0] + \mathbf{A}\mathbf{b}u[1] + \mathbf{b}u[2] \tag{5.3}$$

$$\mathbf{x}[4] = \mathbf{A}^4\mathbf{x}[0] + \mathbf{A}^3\mathbf{b}u[0] + \mathbf{A}^2\mathbf{b}u[1] + \mathbf{A}\mathbf{b}u[2] + \mathbf{b}u[3] \tag{5.4}$$

5.c

$$\mathbf{x}[N] = \sum_{i=1}^{N} \mathbf{A}^i \mathbf{x}[0] + \sum_{i=0}^{N-1} \mathbf{A}^{N-1-i} \mathbf{b} u[i] \tag{5.5}$$

5.d

Isolating the scalars and converting to a matrix-vector multiplication,

$$\cancelto{0}{\mathbf{x}[2]} - \mathbf{A}^2 \mathbf{x}[0] = \mathbf{A}\mathbf{b}u[0] + \mathbf{b}u[1] \tag{5.6}$$

$$-\mathbf{A}^2 \mathbf{x}[0] = \begin{bmatrix} \mathbf{A}\mathbf{b} & \mathbf{b} \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \end{bmatrix} \tag{5.7}$$

When Gaussian elimination is performed on $\begin{bmatrix} \mathbf{A}\mathbf{b} & \mathbf{b} \end{bmatrix}$, the `iPython` notebook returns the matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.8}$$

the row of zeroes indicates that it is impossible to find a solution with only two timesteps.

5.e

Plugging in the following matrix

$$-\mathbf{A}^3 x[0] = \begin{bmatrix} \mathbf{A}^2\mathbf{b} & \mathbf{A}\mathbf{b} & \mathbf{b} \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \end{bmatrix} \tag{5.9}$$

We get our reduced matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.10}$$

Once more indicating it is impossible to find the unique control solution with three timesteps.

5.f

Plugging in the following matrix

$$-\mathbf{A}^4 x[0] = \begin{bmatrix} \mathbf{A}^3\mathbf{b} & \mathbf{A}^2\mathbf{b} & \mathbf{A}\mathbf{b} & \mathbf{b} \end{bmatrix} \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \end{bmatrix} \tag{5.11}$$

We get our reduced matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.12}$$

meaning that a unique solution is possible with four timesteps.

5.g

See `iPython` notebook.

5.h

The vectors we are allowed to move in in two timesteps is $\mathbf{Ab}, \mathbf{b}$. It follows that the range of possible controls within two timesteps is $\text{span}\{\mathbf{Ab}, \mathbf{b}\}$.

5.i

We can generalize this to say that the possible controls for any N timesteps is

$$\text{colsp}(\begin{bmatrix} A^{N-1}\mathbf{b} & \mathbf{A}^{N-2}\mathbf{b} & \cdots & \mathbf{b} \end{bmatrix}) = \text{span}\{A^{N-1}\mathbf{b}, \mathbf{A}^{N-2}\mathbf{b}, \ldots, \mathbf{b}\} \tag{5.13}$$

7 Homework Process and Study Group

I worked on this homework by myself.

prob4

September 27, 2019

1 EECS16A: Homework 4

1.1 Problem 5: Bieber's Segway

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

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

[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
```

1.2 Dynamics

[3]:
```
# 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])
```

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

[4]:
```
# You may use gauss_elim to help you find the row reduced echelon form.
two_column = np.transpose(np.array([np.dot(A, b), b]))
three_column = np.transpose(np.array([A.dot(A).dot(b), np.dot(A, b), b]))
four_column = np.array([A.dot(A).dot(A).dot(b), A.dot(A).dot(b), np.dot(A, b),␣
 ↪b])

controls = np.transpose(np.vstack([four_column, -A.dot(A).dot(A).dot(A).
 ↪dot(state0)]))
print(gauss_elim(controls))
print(gauss_elim(two_column))
print(gauss_elim(three_column))
print(gauss_elim(four_column))
```

```
[[  1.           0.           0.           0.         -13.24875075]
 [  0.           1.           0.           0.          23.73325125]
 [  0.           0.           1.           0.         -11.57181872]
 [  0.           0.           0.           1.           1.46515973]]
[[1. 0.]
 [0. 1.]
 [0. 0.]
 [0. 0.]]
[[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.]]
```

1.4 Part (g)

1.4.1 Preamble

This function will take care of animating the segway.

[5]:
```
# 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(segwayStick_x.min(), 1))
    xmax = max(around(states[:, 0].max() + segway_height / 2.0, 1),␣
↪around(segwayStick_y.max(), 1))

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

1.4.2 Plug in your controller here

[6]:
```
controls = np.array([-13.24875075,23.73325125,-11.57181872,1.46515973]) # here
```

1.4.3 Simulation

[7]:
```
# This will add an extra couple of seconds to the simulation after the input␣
↪controls with no control
# the effect of this is just to show how the system will continue after the␣
↪controller "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␣
↪visualization)
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[stepId-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)
```

1.4.4 Visualization

[8]:
```
%matplotlib nbagg
# %matplotlib qt
anim = animate_segway(t, states, controls_final, stick_length)
anim
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[8]:
```
<matplotlib.animation.FuncAnimation at 0x7f6c473b8898>
```