EECS 16A Designing Information Devices and Systems I
Spring 2017 Babak Ayazifar, Vladimir Stojanovic Homework 3

This homework is due February 13, 2017, at 23:59.
Self-grades are due February 16, 2017, at 23:59.

Submission Format
Your homework submission should consist of **two** files.

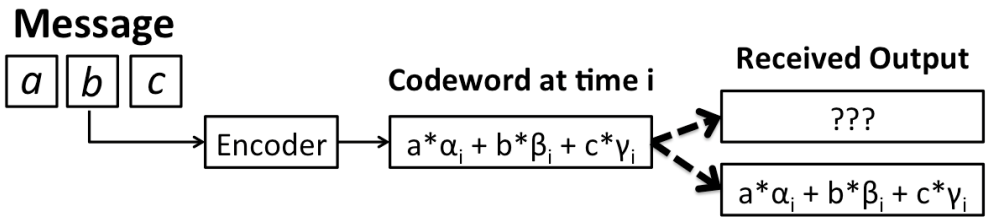
- `hw3.pdf`: A single pdf file that contains all your answers (any handwritten answers should be scanned) as well as your IPython notebook saved as a pdf. You can do this by printing the IPython notebook page in your browser and selecting the save to pdf option. Make sure any plots and results are showing. Also make sure you combine any separate pdfs into one file.
- `hw3.ipynb`: A single IPython notebook with all your code in it.

Submit each file to its respective assignment in Gradescope.

1. Fountain Codes

Consider a sender, Alice, and a receiver, Bob. Alice wants to send a message to Bob, but the message is too big to send all at once. Instead, she breaks her message up into little chunks which she sends across a wireless channel one at a time (think radio transmitter to antenna). She knows some of the packets will be corrupted or erased along the way (someone might turn a microwave on...), so she needs a way to protect her message from errors. This way, even if Bob gets a message missing parts of words he can still figure out what Alice is trying to say! One coding strategy is to use Fountain Codes. Fountain codes are a type of error-correcting codes based on principles of Linear Algebra. They were actually developed right here at Berkeley! The company that commercialized them, Digital Fountain, (started by a **Berkeley grad, Mike Luby**), was later acquired by Qualcomm. In this problem, we will explore some of the underlying principles that make Fountain codes work in a very simplified setting.

In this problem, we concentrate on the case with transmission erasures, i.e. where a bad transmission causes some parts the message to be erased. Let us say Alice wants to convey the set of her three favorite ideas covered in 16A lecture each day to Bob. For this, she maps each idea to a real number and wants to convey the 3-tuple $\begin{bmatrix} a & b & c \end{bmatrix}^T$ (Let us say there are an infinite number of ideas covered in 16A). At each time step, she can send one number, which we will call a "symbol" across. So one possible way for her to send the message is to first send a, then send b, and then send c. However, this method is particularly susceptible to losses. For instance, if the first symbol is lost, then Bob will receive $\begin{bmatrix} ? & b & c \end{bmatrix}^T$, and he will have no way of knowing what Alice's favorite idea is.



(a) The main idea in coding for erasures is to send redundant information so that we can recover from losses. So if we have three symbols of information, we might transmit six symbols for redundancy. One of the most naive codes is called the repetition code. Here, Alice would transmit $\begin{bmatrix} a & b & c & a & b & c \end{bmatrix}^T$. How much erasure can this transmission recover from? Are there specific patterns it cannot handle?

(b) A better strategy for transmission is to send linear combinations of symbols. Alice and Bob decide in advance on a collection of vectors $\vec{v}_i = \begin{bmatrix} \alpha_i & \beta_i & \gamma_i \end{bmatrix}$, $1 \leq i \leq 6$. These vectors define the code: at time i, Alice transmits the scalar

$$k_i = \vec{v}_i \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \alpha_i a + \beta_i b + \gamma_i c.$$

Formulate the setup / the six transmitted symbols using matrix/vector notation.

(c) What are the vectors $\vec{v}_i = \begin{bmatrix} \alpha_i & \beta_i & \gamma_i \end{bmatrix}$, $1 \leq i \leq 6$ that generate the repetition code strategy in part (a)?

(d) Suppose now we choose a collection of seven vectors

$$\vec{v}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}, \quad \vec{v}_4 = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix},$$
$$\vec{v}_5 = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}, \quad \vec{v}_6 = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}, \quad \vec{v}_7 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

Again, at time i, Alice transmits the scalar $k_i = \vec{v}_i \begin{bmatrix} a & b & c \end{bmatrix}^T$. Under what conditions, (i.e. what patterns of losses) can Bob still recover the message?

(e) Suppose, using the collection of vectors in part (d), Bob receives $\begin{bmatrix} 6 & ? & ? & 2 & 3 & ? & ? \end{bmatrix}^T$. What was the transmitted message? Express the problem as a system of linear equations using matrix/vector notation.

(f) Fountain codes build on these principles. The basic idea used by these codes is that Alice keeps sending linear combinations of symbols until Bob has received enough to decode the message. So at time 1, Alice sends the linear combination using $\vec{v}_1$, at time 2 she sends the linear combination using $\vec{v}_2$ and so on. After each new linear combination is sent, Bob will send back an acknowledgement *if* he can decode her message (i.e. figure out the original $\begin{bmatrix} a & b & c \end{bmatrix}^T$ that she intended to communicate). So clearly, the minimum number of transmissions for Alice is 3. If Bob receives the first three linear combinations that are sent, Alice is done in three steps! But because of erasures, she might hear nothing (he hasn't decoded yet). Suppose Alice used

$$\vec{v}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

as her first three vectors, but she has still not received an acknowledgement from Bob. Should she choose new vectors according to the strategy in part (a) or the strategy in part (d)? Why?

2. Cubic Polynomials

Consider the set of real-valued canonical polynomials given below:

$$\varphi_0(t) = 1 \quad \varphi_1(t) = t \quad \varphi_2(t) = t^2 \quad \varphi_3(t) = t^3,$$

where $t \in [a,b] \subset \mathbb{R}$.

(a) Show that the set of all cubic polynomials

$$p(t) = p_0 + p_1 t + p_2 t^2 + p_3 t^3,$$

where $t \in [a,b]$ and the coefficients p_k are real scalars, forms a vector space. What is the dimension of this vector space? Explain.

(b) Show that every real-valued cubic polynomial

$$p(t) = p_0 + p_1 t + p_2 t^2 + p_3 t^3$$

defined over the interval $[a,b]$ can be written as a linear combination of the canonical polynomials $\varphi_0(t)$, $\varphi_1(t)$, $\varphi_2(t)$, and $\varphi_3(t)$. In particular, show that

$$p(t) = \vec{c}^{\mathsf{T}} \vec{\varphi}(t),$$

where

$$\vec{c}^{\mathsf{T}} = [c_0 \; c_1 \; c_2 \; c_3]$$

is a vector of appropriately chosen coefficients, and

$$\varphi(t) = \begin{bmatrix} \varphi_0(t) \\ \varphi_1(t) \\ \varphi_2(t) \\ \varphi_3(t) \end{bmatrix} = \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}.$$

(c) Determine the coefficients c_k for $k = 0,1,2,3$. That is to say, for the setup above, determine what values would be in the c vector.

(d) True or False? The canonical polynomials $\varphi_k(t) = t^k$, for $k = 0,1,2,3$, constitute a basis for the vector space of real-valued cubic polynomials defined over the interval $[a,b]$. Briefly justify your answer

(e) A curve is a continuous mapping from the real line to $\mathbb{R}^N$. A cubic Bézier curve—used extensively in computer graphics—is a type of curve that uses as a basis the following special subset of what are more broadly known as *Bernstein polynomials*:

$$\beta_0(t) = (1-t)^3, \quad \beta_1(t) = 3t(1-t)^2, \quad \beta_2(t) = 3t^2(1-t), \quad \text{and } \beta_3(t) = t^3.$$

Prove that the Bernstein polynomials $\beta_k(t)$ defined above do indeed form a basis. To do this, show that any real-valued polynomial

$$p(t) = p_0 + p_1 t + p_2 t^2 + p_3 t^3$$

can be expressed as a linear combination of the polynomials $\beta_k(t)$, and determine the coefficients in that linear combination. In particular, determine the coefficients in the expansion

$$p(t) = \widehat{p}_0 \beta_0(t) + \widehat{p}_1 \beta_1(t) + \widehat{p}_2 \beta_2(t) + \widehat{p}_3 \beta_3(t)$$

.

Hint: Determine a matrix R such that

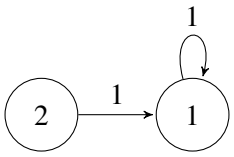
$$\vec{\beta}(t) = R \vec{\varphi}(t),$$

where

$$\vec{\beta}(t) = \begin{bmatrix} \beta_0(t) \\ \beta_1(t) \\ \beta_2(t) \\ \beta_3(t) \end{bmatrix} = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}.$$

Without solving for the inverse, show that R is invertible. Determine its inverse R^{-1}, from which you can determine the coefficients $\widehat{p}_k$. You may use IPython to find R^{-1}.

3. Pumps Properties Proofs



Figure

(a) Suppose we have a pump setup as in Figure , with associated matrix A. Write out the state transition matrix A.

$$\begin{bmatrix} \quad \\ A \\ \quad \end{bmatrix} \begin{bmatrix} \quad \\ x[n] \\ \quad \end{bmatrix} = \begin{bmatrix} \quad \\ x[n+1] \\ \quad \end{bmatrix}$$

(b) Suppose the reservoirs are initialized to have the following water levels: $x_1[0] = 0.5$, $x_2[0] = 0.5$. In a completely alternate universe the pools are initialized to have the following water levels: $x_1[0] = 0.3$, $x_2[0] = 0.7$. For both initial states, what are the water levels at timestep 1 ($x_1[1]$, $x_2[1]$)?

(c) If you observe the pools at timestep 1, can you figure out what the initial ($x_1[0]$, $x_2[0]$) water levels were? Why or why not?

(d) Now generalize: if there exists a state transition matrix where two different initial state vectors lead to the same water levels/state vectors at a timestep in the future, can you recover the initial water levels? Prove your answer. (Hint: What does this say about the matrix A?)

(e) Suppose there is a pump state transition matrix (and corresponding experiment matrix, similar to the last subpart) where every initial state is guaranteed to have a unique state vector in the next timestep. Consider what this statement implies about the system of linear equations represented by the state transition matrix A. Can you recover the initial state? Prove your answer, and explain what this implies about the experiment.

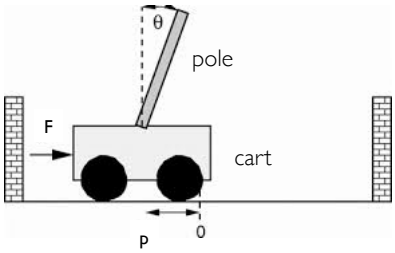
4. Bieber's Segway

After one too many infractions with the law J-Biebs decides to find a new mode of transportation, and you suggest he get a segway.

He becomes intrigued by your idea and asks you how it works.

You let him know that a force (through the spinning wheel) is applied to the base of the segway, and this in turn controls both the position of the segway and angle of the stand. As you push on the stand the segway tries to bring itself back to the upright position, and it can only do this by moving the base.

J-Biebs is impressed, to say the least, but he is a little concerned that only one input (force) is used to control two outputs (position and angle). He finally asks if it's possible for the segway to be brought upright and to a stop from any initial configuration. J-Biebs calls up a friend who's majoring in mechanical engineering, who tells him that a segway can be modeled as a cart-pole system:



A cart-pole system can be fully described by its position p, velocity $\dot{p}$, angle θ, and angular velocity $\dot{\theta}$. We write this as a "state vector":

$$\vec{x} = \begin{bmatrix} p \\ \dot{p} \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{1}$$

The input to this system u will just be the force applied to the cart (or base of the segway).[1]

At time step n, we can apply scalar input $u[n]$. The cart-pole system can be represented by a linear model:

$$\vec{x}[n+1] = A\vec{x}[n] + \vec{b}u[n], \tag{2}$$

where $A \in \mathbb{R}^{4\times 4}$ and $\vec{b} \in \mathbb{R}^{4\times 1}$. The model tells us how the the state vector will evolve over (discrete) time as a function of the current state vector and control inputs.

To answer J-Biebs' question, you look at this general linear system, and try to answer the following question: Starting from some initial state $\vec{x_0}$, can we reach a final desired state, $\vec{x_f}$ in N steps?

The challenge seems to be that the state is 4-dimensional and keeps evolving and we can only apply a one dimensional control at each time. Is it possible to control something 4-dimensional with only one degree of freedom that we can command?

You solve this problem by walking through several steps.

(a) Express $\vec{x}[1]$ in terms of $\vec{x}[0]$ and the input $u[0]$.

(b) Express $\vec{x}[2]$ in terms of *only* $\vec{x}[0]$ and the inputs, $u[0]$ and $u[1]$.

(c) Express $\vec{x}[3]$ in terms of *only* $\vec{x}[0]$ and inputs $u[0]$, $u[1]$ and $u[2]$.

(d) Express $\vec{x}[4]$ in terms of *only* $\vec{x}[0]$ and inputs $u[0]$, $u[1]$, $u[2]$ and $u[3]$.

(e) Now, derive an expression for $\vec{x}[N]$ in terms of $\vec{x}[0]$ and the inputs from $u[0]...u[N-1]$. (*Hint: You can use a sum from* 0 *to* $N-1$.)

For the next four parts of the problem, you are given the matrix A and the vector b:

$$A = \begin{bmatrix} 1 & 0.05 & -0.01 & 0 \\ 0 & 0.22 & -0.17 & -0.01 \\ 0 & 0.10 & 1.14 & 0.10 \\ 0 & 1.66 & 2.85 & 1.14 \end{bmatrix} \tag{3}$$

$$\vec{b} = \begin{bmatrix} 0.01 \\ 0.21 \\ -0.03 \\ -0.44 \end{bmatrix} \tag{4}$$

[1] Some of you might be wondering why it is that applying a force in this model immediately causes a change in position. You might have been taught in high school physics that force creates acceleration, which changes velocity (both simple and angular), which in turn causes changes to position and angle. Indeed, when viewed in continuous time this is true instantaneously. However here in this engineering system, we have discretized time, i.e. we think about applying this force constantly for a given finite duration and we see how the system responds after that finite duration. In this finite time, indeed the application of a force will cause changes to all four components of the state. But notice that the biggest influence is indeed on the two velocities, as should comport with your intuition from high school physics.

The state vector $\vec{0}$ corresponds to the cart-pole (or segway) being upright and stopped at the origin. Assume the cart-pole starts in an initial state $\vec{x}[0] = \begin{bmatrix} -0.3853493 \\ 6.1032227 \\ 0.8120005 \\ -14 \end{bmatrix}$, and you want to reach the desired state $\vec{x_f} = \vec{0}$ using the control inputs $u[0], u[1], \ldots$. (Reaching $\vec{x_f} = \vec{0}$ in N steps means that, given we start at $\vec{x}[0]$, the state vector in the Nth time step, we can find control inputs such that we get $\vec{x}[N]$ equal to $\vec{x_f}$.)

Note: You can use IPython to solve the next four parts of the problem. We have provided a function `gauss_elim(matrix)` to help you find the row reduced echelon form of matrices.

(f) Can you reach $\vec{x_f}$ in *two* time steps? (*Hint: Express* $\vec{x}[2] - A^2\vec{x}[0]$ *in terms of the inputs* $u[0]$ *and* $u[1]$.)

(g) Can you reach $\vec{x_f}$ in *three* time steps?

(h) Can you reach $\vec{x_f}$ in *four* time steps?

(i) If the answer to the previous part is yes, find the required correct control inputs using IPython, and verify the answer by entering these control inputs into the code in the IPython notebook. The code has been written to simulate this system, and you should see the system come to a halt in four time steps! **Suggestion: See what happens if you enter all four control inputs equal to 0. This gives you an idea of how the system naturally evolves!**

(j) Let's return to a general matrix A and a general vector b. What condition do we need on

$$\text{span}\{\vec{b}, A\vec{b}, A^2\vec{b}, \ldots, A^{N-1}\vec{b}\}$$

for $\vec{x_f} = \vec{0}$ to be "reachable" from $\vec{x_0}$ in N steps?

(k) What condition would we need on $span\{\vec{b}, A\vec{b}, A^2\vec{b}, \ldots, A^{N-1}\vec{b}\}$ for *any* valid state vector to be reachable from $\vec{x_0}$ in N steps?

Wouldn't this be cool?

5. Homework process and study group

Who else did you work with on this homework? List names and student ID's. (In case of hw party, you can also just describe the group.) How did you work on this homework?

Working in groups of 3-5 will earn credit for your participation grade.

6. (PRACTICE) Mechanical Inverses

For each of the following matrices, state whether the inverse exists. If so, find the inverse. If not, prove that no inverse exists. **Solve these by hand! Do NOT use NumPy!**

(a) $\begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$

(b) $\begin{bmatrix} 3 & 2 \\ 1 & -1 \end{bmatrix}$

(c) $\begin{bmatrix} -\frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} \end{bmatrix}$

(d) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

(e) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$