EECS 16A Designing Information Devices and Systems I
Summer 2017 D. Aranki, F. Maksimovic, V. Swamy Homework 2

This homework is due on Sunday, July 2, 2017, at 23:59.
Self-grades are due on Monday, July 3, 2017, at 23:59.

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

- `hw2.pdf`: A single PDF file that contains all of your answers (any handwritten answers should be scanned) as well as your IPython notebook saved as a PDF.

  If you do not attach a PDF of your IPython notebook, you will not receive credit for problems that involve coding. Make sure that your results and your plots are visible.

- `hw2.ipynb`: A single IPython notebook with all of your code in it.

  In order to receive credit for your IPython notebook, you must submit both a "printout" and the code itself.

Submit each file to its respective assignment on Gradescope.

1. 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 IPython for this problem.**

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

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

(f) **PRACTICE:** $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$

(g) **PRACTICE:** $\begin{bmatrix} -1 & 1 & -0.5 \\ 1 & 1 & -0.5 \\ 0 & 1 & 1 \end{bmatrix}$

(h) $\begin{bmatrix} 3 & 0 & -2 & 1 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 0 & 4 \\ 1 & 0 & 0 & 1 \end{bmatrix}$

2. Mechanical Eigenvalues

Find the eigenvalues and the corresponding eigenspaces of the following matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix}$$

Do NOT use IPython for this problem.

3. Four Fundamental Subspaces

Consider the matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & -3 & 4 \\ 3 & -3 & -5 & 8 \\ 1 & -1 & -1 & 2 \end{bmatrix}$$

Do NOT use IPython for this problem.

(a) Find the column space of $\mathbf{A}$. What is the dimension of this space?

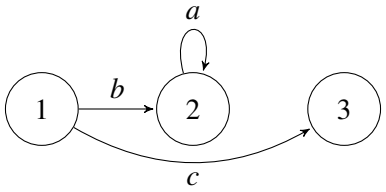
(b) Find the null space of $\mathbf{A}$. What is the dimension of this space?

(c) Find the row space of $\mathbf{A}$. What is the dimension of this space?

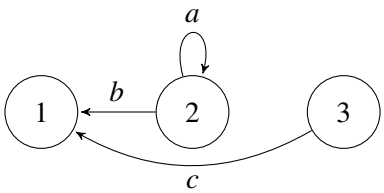
(d) Find the left null space of $\mathbf{A}$. What is the dimension of this space?

4. Transition Matrix Proofs

(a) Suppose there exists some network of websites, such as the "Original" example below. Assume the state vector at some time n is known. Would reversing the arrow directions, as shown in the "Reversed" example below, allow you to find the state vector at time $n-1$? If yes, argue why. If no, provide a counterexample.



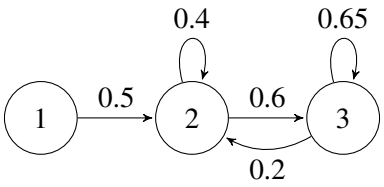
Original



Reverse

(b) Suppose there is a state transition matrix such that the entries of each column vector sum to one. What is the physical interpretation about the total number of people in the system?

(c) Set up the state transition matrix $\mathbf{A}$ for the network shown below. Explain what this $\mathbf{A}$ matrix physically implies about the total number of people in this system. (Note: If there is no "self-arrow/self-loop," then the people do not return to the original website.)



(d) There is a state transition matrix where the entries of its rows sum to one. Prove that applying this system to a uniform vector will return the same uniform vector. A uniform vector is a vector whose elements are all the same.

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & a_{ij} & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x \\ \vdots \\ x \end{bmatrix} = \begin{bmatrix} x \\ \vdots \\ x \end{bmatrix}$$

5. Pumps Properties Proofs

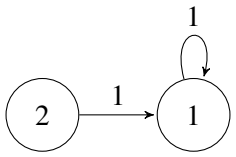


Figure 1

(a) Suppose we have a pump setup as in Figure 1 with the associated transition matrix **A**. Write out the state transition matrix **A**.

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

(b) Suppose the reservoirs are initialized to the following water levels: $x_1[0] = 0.5, x_2[0] = 0.5$. In a completely alternate universe, the reservoirs are initialized to 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 ($\vec{x}[1]$)?

(c) If you observe the reservoirs at timestep 1, can you figure out what the initial ($\vec{x}[0]$) 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* $\boldsymbol{A}$*?*)

(e) Suppose there is a state transition matrix, 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 system.

6. Image Stitching

Often, when people take pictures of a large object, they are constrained by the field of vision of the camera. This means that they have two options how they can capture the entire object:

- Stand as far as away as they need to to include the entire object in the camera's field of view (clearly, we do not want to do this as it reduces the amount of detail in the image)
- (This is more exciting) Take several pictures of different parts of the object, and stitch them together, like a jigsaw puzzle.

We are going to explore the second option in this problem. Daniel, who is a professional photographer, wants to construct an image by using "image stitching". Unfortunately, Daniel took some of the pictures from different angles as well as from different positions and distances from the object. While processing these pictures, Daniel lost information about the positions and orientations from which the pictures were taken. Luckily, you and your friend Marcela, with your wealth of newly acquired knowledge about vectors and rotation matrices, can help him!

You and Marcela are designing an iPhone app that stitches photographs together into one larger image. Marcela has already written an algorithm that finds common points in overlapping images and it's your job to figure out how to stitch the images together. You recently learned about vectors and rotation matrices in EE16A, and you have an idea about how to do this.

Your idea is that you should be able to find a single rotation matrix, $\mathbf{R}$, which is a function of some angle, θ, and a translation vector, $\vec{T}$, that transforms every common point in one image to that same point in the other image. Once you find the the angle, θ, and the translation vector, $\vec{T}$, you will be able to transform one image so that it lines up with the other image.

Suppose $\vec{p}$ is a point in one image and $\vec{q}$ is the corresponding point (i.e., they represent the same thing in the scene) in the other image. You write down the following relationship between $\vec{p}$ and $\vec{q}$.

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}}_{\mathbf{R}(\theta)} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

This looks good, but then you realize that one of the pictures might be farther away than the other. You realize that you need to add a scaling factor, $\lambda > 0$.

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \lambda \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix} \tag{1}$$

(For example, if $\lambda > 1$, then the image containing q is closer (appears larger) than the image containing p. If $0 < \lambda < 1$, then the image containing q appears smaller.)

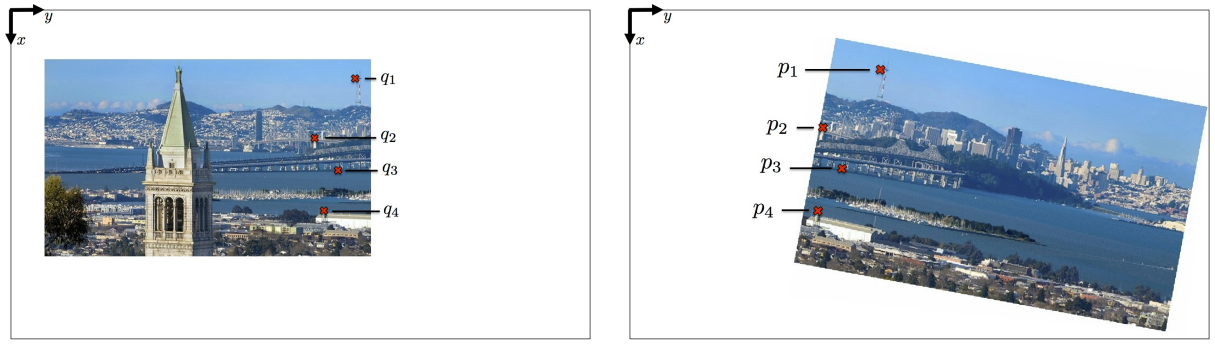


Figure 2: Two images to be stitched together with pairs of matching points labeled.

You are now confident that if you can find θ, $\vec{T}$, and λ, you will be able to reorient and scale one of the images, so that it lines up with the other image.

Before you get too excited, however, you realize that you have a problem. Equation 1 is not a linear equation with respect to θ, $\vec{T}$, and λ. You're worried that you don't have a good technique for solving nonlinear systems of equations. You decide to talk to Marcela and the two of you come up with a brilliant solution.

You decide to "relax" the problem, so that you're solving for a general matrix $\mathbf{R}$ rather than a perfect scaled rotation matrix. The new equation you come up with is:

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} R_{xx} & R_{xy} \\ R_{yx} & R_{yy} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix} \tag{2}$$

This equation is linear, so you can solve for $R_{xx}, R_{xy}, R_{yx}, R_{yy}, T_x$, and T_y. Also you realize that if $\vec{p}$ and $\vec{q}$ actually do differ by a rotation of θ degrees and a scaling of λ, you can expect that the general matrix $\mathbf{R}$ that you find will turn out to be a scaled rotation matrix with $R_{xx} = \lambda\cos(\theta)$, $R_{xy} = -\lambda\sin(\theta)$, $R_{yx} = \lambda\sin(\theta)$, and $R_{yy} = \lambda\cos(\theta)$.

(a) Multiply Equation 2 out into two scalar linear equations. What are the known values and what are the unknowns in each equation? How many unknowns are there? How many equations do you need to solve for all the unknowns? How many pairs of common points $\vec{p}$ and $\vec{q}$ will you need in order to write down a system of equations that you can use to solve for the unknowns?

(b) Write out a system of linear equations that you can use to solve for the values of $\mathbf{R}$ and $\vec{T}$.

(c) In the IPython notebook `prob2.ipynb`, you will have a chance to test out your solution. Plug in the values that you are given for p_x, p_y, q_x, and q_y for each pair of points into your system of equations to solve for the parameters $\mathbf{R}$ and $\vec{T}$. You will be prompted to enter your results, and the notebook will then apply your transformation to the second image and show you if your stitching algorithm works.

(d) We will now explore when this algorithm fails. For example, the three pairs of points must all be distinct points. Show that if $\vec{p}_1, \vec{p}_2, \vec{p}_3$ are *collinear*, the system of equations (2) is underdetermined. Does this make sense geometrically?
(Think about the kinds of transformations possible by a general affine transformation. An affine transformation is one that preserves points. For example, in the rotation of a line, the angle of the line might change, but the length will not. All linear transformations are affine. **Definition of Affine.**)
Use the following fact: $\vec{p}_1, \vec{p}_2, \vec{p}_3$ are collinear iff $(\vec{p}_2 - \vec{p}_1) = k(\vec{p}_3 - \vec{p}_1)$ for some $k \in \mathbb{R}$.

(e) **PRACTICE:** Show that if the three points are not collinear, the system is fully determined.

(f) **PRACTICE:** Marcela comments that perhaps the system (with three collinear points) is only underdetermined because we "relaxed" our model too much by allowing for general affine transformations, instead of just isotropic-scale/rotation/translation. Can you come up with a different representation of Equation 1, that will allow for recovering the transform from only *two* pairs of distinct points?
(*Hint: Let $a = \lambda\cos(\theta)$ and $b = \lambda\sin(\theta)$. In other words, enforce $R_{xx} = R_{yy}$ and $R_{xy} = -R_{yx}$*).

7. 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

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

where

$$\vec{c}^T = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \end{bmatrix}$$

is a vector of appropriately chosen coefficients and

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

(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 $\vec{c}$ 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) **PRACTICE:** 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*:

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

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 $\mathbf{R}$, such that

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

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 $\mathbf{R}$ is invertible. Determine its inverse $\mathbf{R}^{-1}$, from which you can determine the coefficients $\widehat{p}_k$. You may use IPython to find $\mathbf{R}^{-1}$.

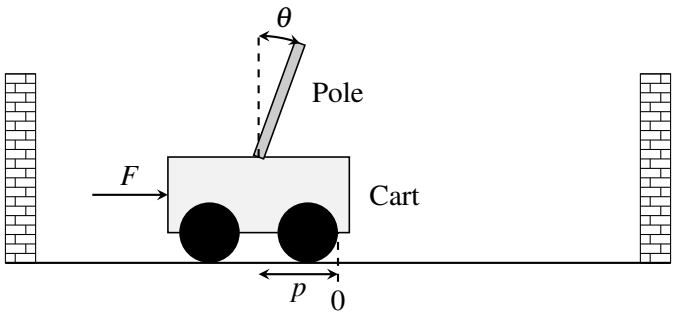
8. Bieber's Segway

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 the 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":

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

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:

$$\vec{x}[n+1] = \mathbf{A}\vec{x}[n] + \vec{b}u[n], \tag{3}$$

where $\mathbf{A} \in \mathbb{R}^{4\times 4}$ 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?

[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 your intuition from high school physics would predict.

The challenge seems to be that the state is 4-dimensional and keeps evolving and that 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 control?

You will 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 $u[1]$. Then express $\vec{x}[3]$ and $\vec{x}[4]$ in terms of *only* $\vec{x}[0]$ and the inputs, $u[0]$ and $u[1]$.

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

For the next four parts of the problem, you are given the matrix $\mathbf{A}$ and the vector $\vec{b}$:

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

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

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 that we start at $\vec{x}[0]$, we can find control inputs, such that we get $\vec{x}[N]$, the state vector at the Nth time step, 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 reduced row echelon form of matrices.

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

(e) Can you reach $\vec{x}_f$ in *three* time steps?

(f) Can you reach $\vec{x}_f$ in *four* time steps?

(g) 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!*

(h) Let's return to a general matrix $\mathbf{A}$ and a general vector $\vec{b}$. What condition do we need on

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

for $\vec{x}_f = \vec{0}$ to be "reachable" from $\vec{x}_0$ in N steps?

(i) What condition would we need on span $\left\{\vec{b}, \mathbf{A}\vec{b}, \mathbf{A}^2\vec{b}, \ldots, \mathbf{A}^{N-1}\vec{b}\right\}$ for *any* valid state vector to be reachable from $\vec{x}_0$ in N steps?
Wouldn't this be cool?

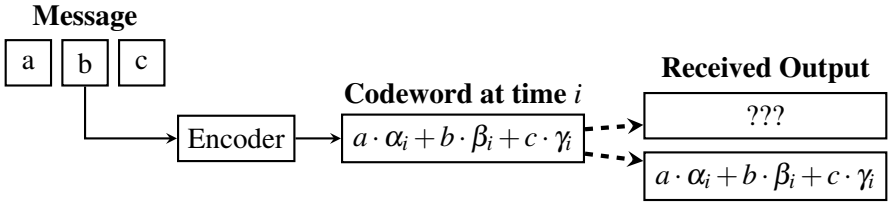
9. Homework Process and Study Group

Who else did you work with on this homework? List names and student ID's. (In case of homework party, you can also just describe the group.) How did you work on this homework?
Working in groups of 3-5 will earn you credit for your participation grade.

10. (PRACTICE) 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 that 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 EE16A 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 EE16A). 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. Thus, 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 they 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 $k_i = \vec{v}_i \begin{bmatrix}a\\b\\c\end{bmatrix}$. 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 $\begin{bmatrix}6 & ? & ? & 2 & 5 & ? & ?\end{bmatrix}^T$. 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?