EECS 16B Fall 2019 — Designing Information Devices and Systems II, UC Berkeley — HW 7

1. Discrete systems and their orbits

The concept of controllability exists to tell us whether or not a system can be eventually driven from any initial condition to any desired state given no disturbances, perfect knowledge of the system, and knowledge of the initial state.

For a discrete-time system with n-dimensional state $\vec{x}$ driven by a scalar input $u(t)$:

$$\vec{x}(t+1) = A\vec{x}(t) + \vec{b}u(t) \tag{1}$$

this can be checked by seeing whether the controllability matrix

$$C = \begin{bmatrix} \vec{b} & A\vec{b} & A^2\vec{b} & \cdots & A^{n-1}\vec{b} \end{bmatrix} \tag{2}$$

has a range (span of the columns) that encompasses all of $\mathbb{R}^n$. In other words, $\text{span}(C) = \mathbb{R}^n$. If it does span the whole space, then the system (1) is controllable. If it does not, then the system is not controllable.

Everything seems to hinge on the "orbit" that the vector $\vec{b}$ takes as it repeatedly encounters A. Does this orbit confine itself to a subspace, or does it explore the whole space? (Formally, this orbit is the infinite sequence $\vec{b}, A\vec{b}, A^2\vec{b}, \cdots$)

For almost the entire rest of the problem, let us assume that the square matrix A has n distinct and real eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ with corresponding nontrivial eigenvectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n$.

(a) Suppose that we choose $\vec{b} = \alpha\vec{v}_i$ for some eigenvector $\vec{v}_i$. **Show that the rank of C will be 1.** i.e. Show that the subspace spanned by the orbit of $\vec{b}$ through A will be one dimensional.

(b) If $\vec{b} = \vec{v}_i$ as above, **would the system be controllable if $n > 1$?**

(c) If $\vec{b} = \alpha_1\vec{v}_1 + \alpha_2\vec{v}_2$, then **show that $A^2\vec{b} = \beta_1 A\vec{b} + \beta_0\vec{b}$ for some choice of β_1 and β_0.**

(d) In the previous part, if all the $\alpha_i \neq 0$, **do the coefficients β_i depend on the exact value of the α_i?**

(e) Consequently, in the previous part, if $n > 2$, **would the system be controllable if the $\vec{b}$ was a linear combination of only two eigenvectors?**

(f) Now consider a general square matrix A (not necessarily with distinct eigenvectors, etc.) with a specific $\vec{b}$ such that the system defined by the pair $(A, \vec{b})$ is controllable. For this specific vector $\vec{b}$, there exist $\{\beta_i\}_{i=0}^{n-1}$ so that it satisfies:

$$A^n\vec{b} = \sum_{i=0}^{n-1} \beta_i A^i\vec{b} \tag{3}$$

Show that for all $j > 0$, the vector $\vec{w}_j = A^j\vec{b}$, also satisfies $A^n\vec{w}_j = \sum_{i=0}^{n-1} \beta_i A^i\vec{w}_j$.
(HINT: Multiply both sides of an equation by something.)

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.

(g) Suppose your general square matrix A above has a specific $\vec{b}$ that satisfies (3) above and such that the system defined by the pair $(A,\vec{b})$ is controllable. **Use the previous part to show that in fact, $A^n - \beta_{n-1}A^{n-1} - \beta_{n-2}A^{n-2} - \cdots - \beta_1 A - \beta_0 I$ is the matrix of all zeros.**

(HINT: The previous part might provide you with a very convenient basis to use. Then remember that the signature of the zero matrix is that no matter what it multiplies, it returns zero.)

It turns out that this specific polynomial ($\lambda^n - \beta_{n-1}\lambda^{n-1} - \cdots - \beta_1\lambda - \beta_0$) must in fact be the characteristic polynomial ($\det(\lambda I - A)$) of the matrix A. So, this argument above shows that these kinds of matrices must satisfy their own characteristic polynomials. It is also easy to see this for diagonalizable matrices A (i.e. those with an eigenbasis), but this example shows that it holds more generally for controllable matrices.

This argument can actually be used to extend to all square matrices, even those that don't have a full complement of linearly independent eigenvectors and aren't controllable with a single scalar input. When extended all the way, it is called the Cayley-Hamilton theorem. It shows that a square matrix satisfies its own characteristic polynomial — no matter what. This is a theorem that is proved in the upper-division linear-algebra course Math 110.

2. System Identification

You are given a discrete-time system as a black-box. You don't know the specifics of the system but you know that it takes one scalar input and has two states that you can observe. You assume that the system is linear and of the form

$$\vec{x}(t+1) = A\vec{x}(t) + Bu(t) + \vec{w}(t), \tag{4}$$

where $\vec{w}(t)$ is an external unseen disturbance that you hope is small, $u(t)$ is a scalar input, and

$$A = \begin{bmatrix} a_0 & a_1 \\ a_2 & a_3 \end{bmatrix}, \quad B = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}, \quad x(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \end{bmatrix}. \tag{5}$$

You want to identify the system parameters from measured data. You need to find the unknowns: a_0, a_1, a_2, a_3, b_0 and b_1, however, you can only interact with the system via a blackbox model.

(a) You observe that the system has state $\vec{x}(t) = [x_0(t), x_1(t)]^T$ at time t. You pass input $u(t)$ into the blackbox and observe the next state of the system: $\vec{x}(t+1) = [x_0(t+1), x_1(t+1)]^T$.

Write scalar equations for the new states, $x_0(t+1)$ and $x_1(t+1)$. Write these equations in terms of the a_i, b_i, the states $x_0(t)$, $x_1(t)$ and the input $u(t)$. Here, assume that $\vec{w}(t) = \vec{0}$ (i.e. the model is perfect).

(b) Now we want to identify the system parameters. We observe the system at the start state $\vec{x}(0) = \begin{bmatrix} x_0(0) \\ x_1(0) \end{bmatrix}$. We can then input $u(0)$ and observe the next state $\vec{x}(1) = \begin{bmatrix} x_0(1) \\ x_1(1) \end{bmatrix}$. We can continue this for an m long sequence of inputs.

Let us define an m long trace to be $[x_0(0), x_1(0), u(0), x_0(1), x_1(1), u(1), x_0(2), x_1(2), u(2), \ldots, x_0(m-1), x_1(m-1), u(m-1), x_0(m), x_1(m)]$. **What is the minimum value of m you need to identify the system parameters?**

(c) Say we feed in a total of 4 inputs $[u(0), u(1), u(2), u(3)]$ into our blackbox. This allows us to observe the following states $[x_0(0), x_0(1), x_0(2), x_0(3), x_0(4)]$ and $[x_1(0), x_1(1), x_1(2), x_1(3), x_1(4)]$, which we can use to identify the system.

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.

To identify the system we need to set up an approximate (because of potential disturbances) matrix equation

$$D\vec{p} \approx \vec{y}$$

using the observed values above and the unknown parameters we want to find. Suppose you are given the form of D in terms of some of the observed data:

$$D = \begin{bmatrix} x_0(0) & x_1(0) & u(0) & 0 & 0 & 0 \\ x_0(1) & x_1(1) & u(1) & 0 & 0 & 0 \\ x_0(2) & x_1(2) & u(2) & 0 & 0 & 0 \\ x_0(3) & x_1(3) & u(3) & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0(0) & x_1(0) & u(0) \\ 0 & 0 & 0 & x_0(1) & x_1(1) & u(1) \\ 0 & 0 & 0 & x_0(2) & x_1(2) & u(2) \\ 0 & 0 & 0 & x_0(3) & x_1(3) & u(3) \end{bmatrix}. \tag{6}$$

For this D, **what are $\vec{y}$ and the unknowns $\vec{p}$ so that $D\vec{p} \approx \vec{y}$ makes sense?** Tell us what the components of these vectors are, written in vector form.

(Feel free to assume the columns of D are linearly independent if that is causing concern.)

(d) Now that we have set up $D\vec{p} \approx \vec{y}$, **explain how you would use this approximate equation to estimate the unknown values a_0, a_1, a_2, a_3, b_0 and b_1.** In particular, give an expression for your estimate $\widehat{\vec{p}}$ for the unknowns in terms of the D and $\vec{y}$.

(HINT: Don't forget that D is not a square matrix. It is taller than it is wide.)

3. Identifying systems from their responses to known inputs

In many problems, we have an unknown system, and would like to characterize it. One of the ways of doing so is to observe the system response with different initial conditions (or inputs). This problem is also called system identification. It is a prototypical example of a problem that today is called machine learning — inferring an underlying pattern from data, and doing so well enough to be able to exploit that pattern in some practical setting.

Because you have a potential exam redo at the same time, we have decided to simply release this problem as a demonstration using NumPy instead of making you work through it. You will have a chance to do system identification by least squares in the lab later, but here, the only thing you need to do is play with the included Jupyter notebook and note down your observations.

(a) **Please share your observations on Example 2.**

(b) **Please share your observations on Example 3.**

(c) **Please share your observations on Example 4.**

(d) **Please share your observations on Example 5.**

4. Outlier Removal via OMP

The problem of "outliers" (bad data points) is ubiquitous in the real world of experimentally collected data. This problem is about how we can leverage known techniques (from 16A) to do something about them in a way that doesn't require a human to manually look at points and subjectively decide which ones are good or bad.

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.

Suppose we have a system where we believe that vector-inputs $\vec{x}$ lead to scalar outputs in a linear way $\vec{p}^T\vec{x}$. However, the parameters $\vec{p}$ are unknown and must be learned from data. Our data collection process is imperfect and instead of directly seeing $\vec{p}^T\vec{x}$, we get observations $y = \vec{p}^T\vec{x} + w$ where the w is some kind of disturbance or noise that nature introduces.

To allow us to learn the parameters, we have n experimentally obtained data points: input-output pairs $(\vec{x}_i, y_i)$ where the index $i = 1, \ldots, n$. However, because we believe that our observed outputs might be a bit noisy, we only have an approximate system of equations. In particular, if we group the data points into a matrix and vector as follows: $X = \begin{bmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vdots \\ \vec{x}_n^T \end{bmatrix}$ where clearly X is a matrix that has d columns and n rows, and $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$ is an n-vector. Then we can express the approximate system of equations that we want to solve as $X\vec{p} \approx \vec{y}$.

The classic least-squares problem views the goal as minimizing

$$\|\vec{y} - X\vec{p}\|^2 = \sum_{i=1}^{n} (y_i - \vec{x}_i^T \vec{p})^2 \tag{7}$$

over the choice of $\vec{p}$ and thereby making the residual $\vec{y} - X\vec{p}$ have as small a Euclidean norm as possible. This is a reasonable thing to do when we believe that the individual residuals $y_i - \vec{x}_i^T \vec{p}$ are all small on average, meaning that we believe that the true disturbance vector $\vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$ is small.

However, nature (or our observation process) isn't always this well behaved. When a small subset of observations don't line up well like the other data points, we call these *outliers*. For these, we don't always know what is wrong. It could be that there was a lot of disturbance or observation noise. It could also be that there was something wrong with how the experiment that we conducted and the $\vec{x}$ in reality was very different from what we think it was. Or it could simply be that the physical system was just behaving in a very unusual way that we cannot hope to predict or understand with our model.

An exaggerated example is when we have a set of observations that satisfy perfectly $y_i = \sum_{j=1}^{d} \vec{x}_i[j]$ with the $|y_i| < 1$ for all $i \neq c$, but there is one crazy $\vec{x}_c$ such that $y_c = \sum_{j=1}^{d} \vec{x}_c[j] + 10^{100}$. Then, as we can see, if we were to do a standard least-squares solution that attempts to minimize (7), this single crazy observation would be enough to shift our estimated $\hat{\vec{p}}$ from the "true" $\vec{p}^* = [1, \ldots, 1]^T$ by a large amount. Why? Because $\hat{\vec{p}} = (X^TX)^{-1}X^T\vec{y}$ is a linear function of $\vec{y}$ and hence the crazy huge deviation in the c-th component of $\vec{y}$ is going to cause a huge multiple of the c-th column of $(X^TX)^{-1}X^T$ to be added to the estimate for $\vec{p}$. The c-th column of $(X^TX)^{-1}X^T$ is just $(X^TX)^{-1}\vec{x}_c$ by our definition of the matrix X above. And so, from the perspective of being an outlier that corrupts our estimate, it really doesn't much matter whether the observation fault was in the scalar y_c or in the vector $\vec{x}_c$ — whichever side of the approximate equation representing the c-th data point is actually messed up, our estimate is going to be pretty badly messed up if we just blindly use least-squares.

Consequently, it is evident that the least-squares-estimated $\hat{\vec{p}}$ is not what we really always want. Is there a way that allows us to reliably remove outliers when we know only a small proportion of the data points are outliers?

In this problem, we will demonstrate one of the simplest outlier removal methods that leverages the orthogonal matching pursuit (OMP) approach you learned in 16A.

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.

(a) As we saw in our example, we could improve our solution if we could remove outliers. One way to do this is by augmenting our matrices in the following way. Let $\vec{e}_i$ be the i-th standard basis vector. That is, $\vec{e}_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$ where the solitary 1 is in the i-row of the vector $\vec{e}_i$. Consider the augmented data matrix and parameter vectors:

$$X_{new} = \begin{bmatrix} X & \vec{e}_i \end{bmatrix}, \ \vec{y}_{new} = \vec{y}, \ \vec{p}_{new} = \begin{bmatrix} \vec{p} \\ f_i \end{bmatrix}. \tag{8}$$

Here the variable f_i is effectively a "fakeness" variable that we associate with the i-th data point. Apply the standard understanding of least squares to find the solution to

$$\min_{\vec{p}_{new}} \|\vec{y}_{new} - X_{new}\vec{p}_{new}\|^2. \tag{9}$$

and **write out the formula for the least-squares estimate** $\widehat{\vec{p}}_{new} = \begin{bmatrix} \widehat{\vec{p}} \\ \widehat{f}_i \end{bmatrix}$.

(b) In the previous part, the $(d+1) \times (d+1)$ matrix $X_{new}^T X_{new}$ played a role. Let's look at this matrix in "block" form:

$$X_{new}^T X_{new} = \begin{bmatrix} X & \vec{e}_i \end{bmatrix}^T \begin{bmatrix} X & \vec{e}_i \end{bmatrix} = \begin{bmatrix} X^T \\ \vec{e}_i^T \end{bmatrix} \begin{bmatrix} X & \vec{e}_i \end{bmatrix} = \begin{bmatrix} X^T X & X^T \vec{e}_i \\ \vec{e}_i^T X & \vec{e}_i^T \vec{e}_i \end{bmatrix}$$

What are $X^T \vec{e}_i, \quad \vec{e}_i^T X, \quad \vec{e}_i^T \vec{e}_i$**?**
Simplify these as much as possible in terms of the individual data points $\vec{x}_i$, etc.

(c) Based on what you know from the previous part, revisit the formula you got in the part before that and **prove that the value for y_i in the i-th data point only impacts the estimate $\widehat{f}_i$ and does not effect the least-squares estimate for $\widehat{\vec{p}}$ at all.**
(HINT: y_i sits in the i-th row of the vector $\vec{y}$. What does the nature of matrix multiplication imply for its influence on the least-squares estimate? Read the full problem text above for a reminder. What does the inverse of a matrix do to a particular column of that matrix?)

(d) Continuing the previous part, use what you know to **prove that** $y_i = \widehat{\vec{p}}^T \vec{x}_i + \widehat{f}_i$. That is, regardless of what y_i is, there is no residual left in the i-th position.

(e) Given that $\widehat{\vec{p}}_{new} = \begin{bmatrix} \widehat{\vec{p}} \\ \widehat{f}_i \end{bmatrix}$ minimizes (9), **show that $\widehat{\vec{p}}$ minimizes $\sum_{j \neq i} (y_j - \vec{x}_j^T \vec{p})^2$.**
Note that here, $\sum_{j \neq i}$ is just a compact way to write sum up over all indices j from 1 to n but skip the $j = i$ term in the sum.
(HINT: There are many ways to proceed. Let $\vec{p}'$ be the vector that minimizes $r = \sum_{j \neq i} (y_j - \vec{x}_j^T \vec{p})^2$. Use this to construct an f_i' so that $\vec{p}'_{new} = \begin{bmatrix} \vec{p}' \\ f_i' \end{bmatrix}$ achieves the same r in (9)*. Could the cost in* (9) *possibly go any lower? Recall that a square of a real number is always ≥ 0.)*

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.

(f) **Argue why the above implies that the resulting solution is the same parameter estimate that we would have gotten had we simply removed the i-th data-point entirely from our data set.**

(g) From what you have seen above together with what you see from running the attached jupyter notebook, **argue in words why augmenting the data matrix X with an identity and then running OMP is a reasonable approach to systematically eliminating outliers.**

The secret to actually running OMP successfully for this purpose of learning from corrupted data is finding the right stopping condition. A very reasonable condition (whose reason for working is a bit out of scope of this course) is to keep a random subset of data points aside (don't include them in our estimation process for which OMP is running, etc.), and then track the residuals on predicting the y_i in this side set of points (called a "held out set") using the parameters $\hat{\vec{p}}$ being estimated using the main data set. In particular, look at the median value for $|y_i - \vec{x}_i^T \hat{\vec{p}}|$ across this held-out set. If there are actual outliers present, this median residual will tend to drop as outliers get eliminated, because the pattern found is closer to the actual underlying pattern. Once all the true outliers are gone, this median will tend to bottom-out and then slowly start getting worse as more and more good data points are falsely accused of being outliers. We should stop OMP anywhere in this (typically wide) valley. Intuitively, the reason to take the median here is that some of these held-out points might also be outliers and the median is likely to be something that is not an outlier and is instead more typical. These kinds of approaches to stopping OMP are suitable when data is relatively plentiful, isn't horribly corrupted, and we value being able to deploy fully automated approaches to learning.

5. Gram-Schmidt Basic

(a) **Use Gram-Schmidt to find a matrix U whose columns form an orthonormal basis for the column space of V.**

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Show that you get the same resulting vector when you project $\vec{w} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ -1 \\ 0 \end{bmatrix}$ onto the columns of V as you do when you project onto the columns of U, i.e. **show that**

$$\mathbf{V}(\mathbf{V}^T\mathbf{V})^{-1}\mathbf{V}^T\vec{w} = \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\vec{w}.$$

Feel free to use numpy. No need to grind this out by hand.

6. Speeding Up OMP

Recall the imaging lab from EE16A where we projected masks on an object to scan it to our computer using a single pixel measurement device, that is, a photoresistor. In that lab, we were scanning a 30×40 image

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.

having 1200 pixels. In order to recover the image, we took exactly 1200 measurements because we wanted our 'measurement matrix' to be invertible.

However, we saw in 16A lecture near the end of the semester that an iterative algorithm that does "matching and peeling" can enable reconstruction of a sparse vector (i.e. one that has mostly zeros in it) while reducing the number of samples that need to be taken from it. In the case of imaging, the idea of sparsity corresponds to most parts of the image being black with only a small number of light pixels. In these cases, we can reduce the overall number of samples necessary. This would reduce the time required for scanning the image. (This is a real-world concern for things like MRI where people have to stay still while being imaged. It is also a key principle that underlays a lot of modern machine learning.)

In this problem, we have a 2D image I of size 91×120 pixels for a total of 10920 pixels. The image is made up of mostly black pixels except for 476 of them that are white.

Although the imaging illumination masks we used in the lab consisted of zeros and ones, in this question, we are going to have masks with real values — i.e. the light intensity is going to vary in a more finely grained way. Say that we have an imaging mask M_0 of size 91×120. The measurements using the photoresistor using this imaging mask can be represented as follows.

First, let us vectorize our image to $\vec{i}$ which is a column vector of length 10920. Likewise, let us vectorize the mask M_0 to $\vec{m}_0$ which is a column vector of length 10920. Then the measurement using M_0 can be represented as

$$b_0 = \vec{m}_0^T \vec{i}.$$

Say we have a total of K measurements, each taken with a different illumination mask. Then, these measurements can collectively be represented as

$$\vec{b} = \mathbf{A}\vec{i},$$

where $\mathbf{A}$ is an $K \times 10920$ size matrix whose rows contain the vectorized forms of the illumination masks, that is

$$\mathbf{A} = \begin{bmatrix} \vec{m}_1^T \\ \vec{m}_2^T \\ \vdots \\ \vec{m}_K^T \end{bmatrix}.$$

To show that we can reduce the number of samples necessary to recover the sparse image I, we are going to only generate 6500 masks. The columns of $\mathbf{A}$ are going to be approximately uncorrelated with each other. The following question refers to the part of Jupyter notebook file accompanying this homework related to this question.

(a) In the jupyter notebook, we have completed a function `OMP` to run the naive OMP algorithm you learned in EE16A. Read through the code and understand the function `OMP`.

We have also supplied code that reads a PNG file containing a sparse image, takes measurements, and performs OMP to recover it. An example input image file is also supplied together with the code. Using `smiley.png`, generate an image of size 91×120 pixels of sparsity less than 400 and recover it using OMP with 6500 measurements.

Run the code `rec = OMP((height, width), sparsity, measurements, A)` and see the image being correctly reconstructed from a number of samples smaller than the number of pixels of your figure. What is the image? Report how many seconds this took to run.

Remark: Note that this remark is not important for solving this problem; it is about how such measurements could be implemented in our lab setting. When you look at the vector `measurements`

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.

you will see that it has zero average value. Likewise, the columns of the matrix containing the masks $\mathbf{A}$ also have zero average value. To satisfy these conditions, some entries need to have negative values. However, in an imaging system, we cannot project negative light. One way to get around this problem is to find the smallest value of the matrix $\mathbf{A}$ and subtract it from all entries of $\mathbf{A}$ to get the actual illumination masks. This will yield masks with positive values, hence we can project them using our real-world projector. After obtaining the readings using these masks, we can remove their average value from the readings to get measurements as if we had multiplied the image using the matrix $\mathbf{A}$. This is being done silently for you in the code.

(b) Now let us try using our naive implementation of OMP to recover a slightly less sparse image: An example input image file is supplied together with the code. Using `pika.png`, generate an image of size 100×100 pixels of sparsity less than 800 and recover it using OMP with 9000 measurements.

Run the corresponding code blocks in the accompanying jupyter notebook and report back the number of seconds it took to to reconstruct the image. (Take a well deserved break, this may take upwards of ten minutes to run!)

(c) As you saw, reconstructing the image with the staff's naive implementation took quite a while. Modify the code to run faster by using a Gram-Schmidt orthonormalization to speed it up. **Edit the code given to you in the jupyter notebook. Report back the number of seconds it took to run the reconstruct the image `pika.png`.** Note this should be less than previous part.

This is the only place in the problem where you should have to actually edit code.

(d) (Optional, not in scope) **Do any other modifications you want to further speed up the code.**

Hint: When possible, how would you safely extract multiple peaks corresponding to multiple pixels in one go and add them to the recovered list? Would this speed things up?

7. Write Your Own Question And Provide a Thorough Solution.

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top level. We rarely ask you any homework questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself (e.g. making flashcards). But we don't want the same to be true about the highest level. As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams. Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't ever happen.

8. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student! We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

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

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.

(c) **How did you work on this homework?** (For example, *I first worked by myself for 2 hours, but got stuck on problem 3, so I went to office hours. Then I went to homework party for a few hours, where I finished the homework.*)

(d) **Roughly how many total hours did you work on this homework?**

Contributors:

- Anant Sahai.
- Nikhil Shinde.
- Alex Devonport.
- Kuan-Yun Lee.
- Orhan Ocal.
- Vasuki Narasimha Swamy.

© UCB EECS 16B, Fall 2019. All Rights Reserved. This may not be publicly shared without explicit permission.