1. Symmetric Matrices

We want to show that every real symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors. In other words, a symmetric matrix has full complement of eigenvectors that are all orthogonal to each other.

In discussion section, you have seen a recursive derivation of a related fact. Formally however, such recursive derivations are usually turned into proofs by using induction. This problem serves to both freshen your mind regarding induction as well as to give you a chance to prove for yourself this very important theorem. (This is the same essential proof as that of Schur upper-triangularization. So understanding this problem will help solidify your understanding of that proof as well.)

(a) You will start by proving a basic lemma about real symmetric matrices under an orthonormal change of basis. Prove that if $S$ is a symmetric matrix ($S = S^T$) and $U$ is a matrix whose columns are orthonormal, then $U^T S U$ (that is, $S$ represented in the basis $U$) is also symmetric.

(b) Another useful lemma is that real symmetric matrices have real eigenvalues. Prove that the eigenvalues $\lambda$ of real, symmetric matrix $S$ are real. Hint: Suppose that $S$ had a complex eigenvalue $\lambda$ with eigenvector $\vec{v}$. Because $S$ is a real matrix, what do you know about $S\overline{\vec{v}}$ — applying $S$ to the complex conjugate of $\vec{v}$? What happens when you take a potentially complex number and multiply it by its own complex conjugate? Consequently, what do you know happens if you multiply a complex vector $\vec{v}$ by the conjugate of its transpose: i.e. consider $\overline{\vec{v}^T} \vec{v}$? Since $S$ is symmetric, what do you know about $\vec{v}^T S$? Put these ingredients together as shown in lecture.

(c) A third useful lemma is one about finding orthonormal bases. Show that given a single nonzero vector $\vec{u}_0$ of dimension $n$, that it is possible to find an orthonormal set of $n$ vectors, $\vec{v}_0, \ldots, \vec{v}_{n-1}$ such that $\vec{v}_0 = \alpha \vec{u}_0$ for some scalar $\alpha$.

(Hint: Use the Gram-Schmidt process on the list of $n + 1$ vectors obtained by starting with the given vector and appending the standard basis — i.e. the columns of the identity matrix.)

(d) For the main proof that every real symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors, we will proceed by formal induction. Recall that for a proof by induction, we have to start with a base case - this is also the base case in a recursive derivation. Consider the trivial case of $S$ having dimensions $[1 \times 1]$ ($n = 1$). Does $S$ have an eigenvector? Can this eigenbasis be made orthonormal? Is the matrix diagonal in this basis? Are the entries real?

(e) After the base case, we do an inductive stage of the main proof. The first step in the inductive stage is to write down the induction hypothesis. Assume that the property that every real symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors holds for all symmetric matrices with size $[(n-1) \times (n-1)]$. Write down the statement of the fact you want to assume in your own words using mathematical notation. (Hint: In general for proofs by induction, you want to start with the strongest version of what you want to prove. This gives you the most powerful inductive hypothesis.)

(f) Now think about a symmetric matrix $S$ with size $[n \times n]$. Consider a real eigenvalue $\lambda_0$ of $S$ and the corresponding eigenvector $\vec{u}_0$ (a column vector with size $n$). Use an appropriate orthonormal
change of basis \( V \) to show that \( S = V XV^T \), where \( X \) is of the form

\[
X = \begin{bmatrix}
\lambda_0 & x_{1,2} & \cdots & x_{1,n} \\
0 & x_{2,2} & \cdots & x_{2,n} \\
\vdots & \vdots & \ddots & \vdots \\
0 & x_{n,2} & \cdots & x_{n,n}
\end{bmatrix}
\]

That is, the first column of \( X \) is

\[
\begin{bmatrix}
\lambda_0 \\
0 \\
\vdots \\
0
\end{bmatrix}.
\]

(g) Continue the previous part to show that in fact the matrix relation can be written as

\[
S = V \begin{bmatrix}
\lambda_0 & \vec{0}^T \\
\vec{0} & Q
\end{bmatrix} V^T
\]

where \( Q \) is an \([n - 1 \times n - 1]\) symmetric matrix. Hint: Recall that \( S \) is a symmetric matrix.

(h) According to our induction hypothesis, we can write \( Q \) as \( U \Lambda U^T \) where \( U \) is an orthonormal \([n - 1 \times n - 1]\) square matrix and \( \Lambda \) is a diagonal matrix with real entries along the diagonal and 0s everywhere else. Use this fact to show that indeed there must exist an orthonormal \([n \times n]\) square matrix \( W \) such that

\[
S = W \begin{bmatrix}
\lambda_0 & \vec{0}^T \\
\vec{0} & \Lambda
\end{bmatrix} W^T
\]

(Hint: What is the product of orthonormal matrices?)

By induction, we are now done since we have proved that having the desired property for \( n - 1 \) implies that we have the property for \( n \) and we also have a valid base case at \( n = 1 \).

According to the base case and inductive steps we just proved, the statement, “every real symmetric matrix is diagonalized by a matrix of its real orthonormal eigenvectors” is proved by induction.

2. The Moore-Penrose pseudoinverse for “wide” matrices

Say we have a set of linear equations given by \( A\vec{x} = \vec{y} \). If \( A \) is invertible, we know that the solution for \( \vec{x} \) is \( \vec{x} = A^{-1}\vec{y} \). However, what if \( A \) is not a square matrix? In 16A, you saw how this problem could be approached for tall “standing up” matrices \( A \) where it really wasn’t possible to find a solution that exactly matches all the measurements, using linear least-squares. The linear least-squares solution gives us a reasonable answer that asks for the “best” match in terms of reducing the norm of the error vector.

This problem deals with the other case — when the matrix \( A \) is wide and “lying down” — with more columns than rows. In this case, there are generally going to be lots of possible solutions — so which should we choose? Why? We will walk you through the Moore-Penrose pseudoinverse that generalizes the idea of the matrix inverse and is derived from the singular value decomposition.

This approach to finding solutions complements the OMP approach that you learned in 16A and that we used earlier in 16B in the context of outlier removal during system identification.
(a) Say you have the matrix
\[
A = \begin{bmatrix}
1 & -1 & 1 \\
1 & 1 & -1
\end{bmatrix}.
\]
To find the Moore-Penrose pseudoinverse we start by calculating the SVD of \(A\). That is to say, calculate \(U, \Sigma, V\) such that
\[
A = U\Sigma V^T
\]
where \(U\) and \(V\) are orthonormal matrices.
Here we will give you that the decomposition of \(A\) is:
\[
A = \begin{bmatrix}
1 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
0 & \frac{\sqrt{2}}{\sqrt{2}} & 0
\end{bmatrix} \begin{bmatrix}
2 & 0 & 0 \\
0 & \frac{\sqrt{2}}{\sqrt{2}} & 0 \\
0 & 0 & \frac{\sqrt{2}}{\sqrt{2}}
\end{bmatrix} \begin{bmatrix}
0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
1 & 0 & 0 \\
0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}
\end{bmatrix}
\]
where:
\[
U = \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
0 & \frac{\sqrt{2}}{\sqrt{2}}
\end{bmatrix}
\]
\[
\Sigma = \begin{bmatrix}
2 & 0 & 0 \\
0 & \frac{\sqrt{2}}{\sqrt{2}} & 0
\end{bmatrix}
\]
\[
V^T = \begin{bmatrix}
0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
1 & 0 & 0 \\
0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}
\end{bmatrix}
\]
It is a good idea to be able to calculate the SVD yourself as you may be asked to solve similar questions on your own in the exam.
Let us now think about what the SVD does. Let us look at matrix \(A\) acting on some vector \(\vec{x}\) to give the result \(\vec{y}\). We have
\[
A\vec{x} = U\Sigma V^T \vec{x} = \vec{y}.
\]
Observe that \(V^T \vec{x}\) rotates the vector, \(\Sigma\) scales the result, and \(U\) rotates it again. We will try to "reverse" these operations one at a time and then put them together to construct the Moore-Penrose pseudoinverse.
If \(U\) "rotates" the vector \((\Sigma V^T) \vec{x}\), what operator can we derive that will undo the rotation?
(b) Derive a matrix that will "unscale", or undo the effect of \(\Sigma\) where it is possible to undo. Recall that \(\Sigma\) has the same dimensions as \(A\). Ignore any division by zeros (that is to say, let it stay zero).
(c) Derive an operator that would "unrotate" by \(V^T\).
(d) Try to use this idea of "unrotating" and "unscaling" to derive an "inverse", denoted as \(A^\dagger\). That is to say,
\[
\vec{x} = A^\dagger \vec{y}
\]
The reason why the word inverse is in quotes (or why this is called a pseudo-inverse) is because we’re ignoring the "divisions" by zero.
(e) Use \(A^\dagger\) to solve for a vector \(\vec{x}\) in the following system of equations.
\[
\begin{bmatrix}
1 & -1 & 1 \\
1 & 1 & -1
\end{bmatrix} \vec{x} = \begin{bmatrix}
2 \\
4
\end{bmatrix}
\]
(f) Now we will see why this matrix is a useful proxy for the matrix inverse in such circumstances. Show that the solution given by the Moore-Penrose pseudoinverse satisfies the minimality property that if \( \hat{x} \) is the pseudo-inverse solution to \( Ax = \bar{y} \), then \( \|\hat{x}\| \leq \|\bar{z}\| \) for all other vectors \( \bar{z} \) satisfying \( AZ = \bar{y} \). (Hint: look at the vectors involved in the V basis. Think about the relevant nullspace and how it is connected to all this.)

This minimality property is useful in many applications. You saw a control application in lecture. You’ll see a communications application in another problem. This is also used all the time in machine learning, where it is connected to the concept behind what is called ridge regression or weight shrinkage.

(g) Consider a generic wide matrix \( A \). We know that \( A \) can be written using \( A = U \Sigma V^T \) where \( U \) and \( V \) each are the appropriate size and have orthonormal columns, while \( \Sigma \) is the appropriate size and is a diagonal matrix — all off-diagonal entries are zero. Further assume that the rows of \( A \) are linearly independent. Prove that \( A^\dagger = A^T (AA^T)^{-1} \).

(HINT: Just substitute in \( U \Sigma V^T \) for \( A \) in the expression above and simplify using the properties you know about \( U, \Sigma, V \). Remember the transpose of a product of matrices is the product of their transposes in reverse order: \( (CD)^T = D^T C^T \).)

3. Using upper-triangularization to solve differential equations

You know that for any square matrix \( A \) with real eigenvalues, there exists a real matrix \( V \) with orthonormal columns and a real upper triangular matrix \( R \) so that \( A = VRV^T \). In particular, to set notation explicitly:

\[
V = \begin{bmatrix} \vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n \end{bmatrix}
\]

\[
R^T = \begin{bmatrix} \vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n \end{bmatrix}
\]

where the rows of the upper-triangular \( R \) look like

\[
\vec{r}_1^T = [\lambda_1, r_{1,2}, r_{1,3}, \ldots, r_{1,n}]
\]

\[
\vec{r}_2^T = [0, \lambda_2, r_{2,3}, r_{2,4}, \ldots, r_{2,n}]
\]

\[
\vec{r}_i^T = [0, \ldots, 0, \lambda_i, r_{i,i+1}, r_{i,i+2}, \ldots, r_{i,n}]
\]

\[
\vec{r}_n^T = [0, \ldots, 0, \lambda_n]
\]

where the \( \lambda_i \) are the eigenvalues of \( A \).

Here, we also use bracket notation to index into vectors so

\[
r_i[k] = \begin{cases} 
0 & \text{if } k < i \\
\lambda_i & \text{if } k = i \\
r_{i,k} & \text{if } k > i 
\end{cases}
\]

Note: we use 1-indexing so the first entry has index 1.

Suppose our goal is to solve the \( n \)-dimensional system of differential equations written out in vector/matrix form as:

\[
\frac{d}{dt} \vec{x}(t) = A\vec{x}(t) + \vec{u}(t),
\]

\[
\vec{x}(0) = \vec{x}_0,
\]
where \( \vec{x}_0 \) is a specified initial condition and \( \vec{u}(t) \) is a given vector of functions of time.

Assume that the \( V \) and \( R \) have already been computed and are accessible to you using the notation above.

Assume that you have access to a function \( \text{ScalarSolve}(\lambda, y_0, \vec{u}) \) that takes a real number \( \lambda \), a real number \( y_0 \), and a real-valued function of time \( \vec{u} \) as inputs and returns a real-valued function of time that is the solution to the scalar differential equation \( \frac{d}{dt} y(t) = \lambda y(t) + \vec{u}(t) \) with initial condition \( y(0) = y_0 \).

Also assume that you can do regular arithmetic using real-valued functions and it will do the right thing. So if \( u \) is a real-valued function of time, and \( g \) is also a real-valued function of time, then \( 5u + 6g \) will be a real valued function of time that evaluates to \( 5u(t) + 6g(t) \) at time \( t \).

Use \( V, R \) to construct a procedure for solving this differential equation

\[
\frac{d}{dt} \vec{x}(t) = A\vec{x}(t) + \vec{u}(t),
\]

\[
\vec{x}(0) = \vec{x}_0,
\]

for \( \vec{x}(t) \) by filling in the following template in the spots marked ♣, ◊, ▼, ♠.

(HINT: Think back to the RLC circuit on homework 3. You also might want to write out for yourself what the differential equation looks like in \( V \)-coordinates.)

1: \( \vec{x}_0 = V^T \vec{x}_0 \)                      \> Change the initial condition to be in \( V \)-coordinates
2: \( \vec{u} = V^T \vec{u} \)                      \> Change the external input functions to be in \( V \)-coordinates
3: for \( i = n \) down to 1 do
   4: \( \vec{u}_i = \clubsuit + \sum_{j=i+1}^n \spadesuit \) \> Iterate up from the bottom
   5: \( \vec{x}_i = \text{ScalarSolve}(\diamond, \vec{x}_0[i], \vec{u}_i) \) \> Make the effective input for this level
5: end for
6: \( \vec{x}(t) = \heartsuit \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_N \end{bmatrix} (t) \) \> Solve this level’s scalar differential equation
7: \( \vec{x}(t) = \spadesuit \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_N \end{bmatrix} (t) \) \> Change back into original coordinates

(a) Give the expression for \( \heartsuit \) on line 7 of the algorithm above. (i.e. How do you get from \( \vec{x}(t) \) to \( \vec{x}(t) \)?)

(b) Give the expression for \( \diamond \) on line 5 of the algorithm above. (i.e. What are the \( \lambda \) arguments to \( \text{ScalarSolve} \) for the \( i \)-th iteration of the for-loop?)

(c) Give the expression for \( \clubsuit \) on line 4 of the algorithm above.

(d) Give the expression for \( \spadesuit \) on line 4 of the algorithm above.

(e) Let us complete the algorithm by investigating how \( \text{ScalarSolve}(\lambda, y_0, \vec{u}) \) works.

Consider an input that is a weighted sum of polynomials times exponentials.

\[
\vec{u}(t) = \sum_{k=1}^N \alpha[k] \beta[k] e^{\gamma[k]} t
\]

Here, the \( \alpha[k] \) are real constants, the \( \beta[k] \) are non-negative integer powers, and the \( \gamma[k] \) are real exponents. Assume that \( \alpha, \beta, \gamma \) are all lists of the same size \( N \).

What function should \( \text{ScalarSolve}(\lambda, y_0, \vec{u}) \) return for the above \( \vec{u} \)? Express the answer in terms of new lists \( \alpha'[k], \beta'[k], \gamma'[k] \) and a procedure to construct them.

(Hint: Recall the integral solution from HW 2 and consider integration by parts. Now, walk down the original lists and build your new lists as you go. You are going to have to deal with the case that \( \lambda \) equals the relevant \( \gamma \) entry differently from how you deal with the case where \( \lambda \) doesn’t equal that \( \gamma \). Finally, don’t forget about the initial condition.)
4. Weighted minimum norm

You saw in lecture in the context of open-loop control, how we consider problems in which we have a wide matrix $A$ and solve $Ax = y$ such that $x$ is a minimum norm solution:

$$\|x\| \leq \|z\|$$

for all $z$ such that $Az = y$. You then saw this idea again earlier in this HW where you saw how to compute the appropriate “pseudo-inverse” for such wide matrices.

But what if you weren’t interested in just the norm of $x$? What if you instead cared about minimizing the norm of a linear transformation $Cx$? For example, suppose that controls were more or less costly at different times.

The problem can be written out mathematically as:

Given a wide matrix $A$ and a matrix $C$ find $x$ such that $Ax = y$ and $\|Cx\| \leq \|Cz\|$ for all $z$ such that $Az = y$.

(a) Let’s start with the case of $C$ being invertible. **Solve this problem (i.e. find the optimal $x$ with the minimum $\|Cx\|$) for the specific matrices and $\vec{y}$ given below. Show your work.**

*HINT: You might want to change variables to solve this problem. Don’t forget to change back!*

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

For convenience, $C^{-1} = \begin{bmatrix} 0 & 0 & 0.5 \\ 0 & 1 & 0 \\ 0.5 & 0 & 0 \end{bmatrix}$ and you are also given some SVDs on the following page.

$$A = (U_A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix})(\Sigma_A = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix})(V_A^T = \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} \\ 1 & 0 \\ 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}) \quad (1)$$

$$C = (U_C = \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix})(\Sigma_C = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix})(V_C^T = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}) \quad (2)$$

$$AC = (U_{AC} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix})(\Sigma_{AC} = \begin{bmatrix} \sqrt{5} & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix})(V_{AC}^T = \begin{bmatrix} 0 & \frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 \end{bmatrix}) \quad (3)$$

$$AC^{-1} = (U_{AC^{-1}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix})(\Sigma_{AC^{-1}} = \begin{bmatrix} \frac{\sqrt{5}}{2} & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix})(V_{AC^{-1}}^T = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \end{bmatrix}) \quad (4)$$

(b) What if $C$ were a tall matrix with linearly independent columns? **Explicitly describe how you would solve this problem in that case, step by step.**
For convenience, we have copied the problem statement again here: Given a wide matrix $A$ and a matrix $C$ find $\tilde{x}$ such that $A\tilde{x} = \tilde{y}$ and $\|C\tilde{x}\| \leq \|C\tilde{z}\|$ for all $\tilde{z}$ such that $A\tilde{z} = \tilde{y}$.

Here, you can assume that the wide matrix $A$ has linearly-independent rows but is otherwise generic. Similarly, $\tilde{y}$ is a generic vector.

(HINT: Does $C$ have a nullspace? Does $C^T C$ have a nullspace? Does the SVD of $C$ suggest any (invertible) change of coordinates from $\tilde{x}$ to $\tilde{z}$ such that $\|\tilde{z}\| = \|C\tilde{x}\|$?)

5. Classification of Sinusoids

This HW problem can be viewed as a warm-up for the next topic in the course: which is going to be motivated by figuring out how to process signals recorded from the brain to decipher what a person wants to do in terms of a specific command to their robot arm. These kinds of problems are called “classification” problems. In this exercise, you will be using jupyter to classify sinusoids.

The iPython notebook Sinusoidal_Projection.fa19_prob.ipynb will guide you through the process of performing sinusoidal projections.

Suppose you already know the true potential frequencies $f_i$ and potential phases $\phi_i$ of a set of sinusoidal signals

$$S := \{\sin(2\pi f_i + \phi_i), i = 1, 2, \ldots, n\}, \quad (5)$$

and you have some noisy samples of these true sinusoidal signals. You want to determine the true sinusoidal signal for each of these noisy samples—How would you approach the problem?

We will show in this problem that we can project noisy sinusoidal signals onto noiseless sinusoids to achieve good classification.

In the realistic world, one often doesn’t have the complete waveform of a continuous function, instead oftentimes one works with samples of the continuous function.

In our case, we generate noisy samples of the true sinusoidal signals in the following way. For each of the $num\_sinusoids$ true frequencies, each noisy sample $y_i$ consists of $S$ sample points sampled with a sampling rate of $F_s$ sample rate, and corrupted by noise scaled by $\sigma$.

$$y_i(k) = \sin(2\pi f_i \cdot k/F_s) + \sigma \cdot \text{Noise} \quad k = 1, 2, \ldots, S.$$ 

In our example, we will work with $num\_sinusoids = 3$.

A higher $\sigma$ corresponds to more noise in our measurements.

Please complete the notebook by following the instructions given.

(a) Run the first part of the jupyter notebook to generate our noisy data points. Use $\sigma = 0.1, 1.0, 10.0, 100.0$ and comment on what you observe in the plots.

(b) Complete part (b) of the notebook to project noisy sinusoids onto potential true sinusoids. Sketch by hand the resulting 3D plot of projections qualitatively. Comment on what happens when you try the noise scalings $\sigma = 0.1, 1.0, 10.0, 100.0$.

(c) Complete part (c) of the notebook to classify the data points and calculate the number of misclassified points.

Report the number of misclassifications for $\sigma = 0.1, 1.0, 10.0, 100.0$. Explain what happens when there is a high level of noise. Recall that our noisy process is random so that there can be cases where there are misclassifications even in low noise.
(d) **For what qualitative regions of the noise level is it very beneficial for us to use projections?** For very low values of noise, do you have to do projections to successfully classify? What else could you have done? This question is asking you to reflect on what you have observed.

6. **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.

7. **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.)

(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:**

- Siddharth Iyer.
- Yu-Yun Dai.
- Sanjit Batra.
- Anant Sahai.
- Sidney Buchbinder.
- Justin Yim.
- Nikhil Shinde.
- Kuan-Yun Lee.