COMS 3251 SU21 HW4

Numerical/Theory Problems Due: Mon Aug 09, 2021 at 11:59pm
Applications/Programming Problems Due: Mon Aug 16, 2021 at 11:59pm

This homework is to be done **alone**. No late homeworks are allowed. To receive credit, a typeset-ted copy of the homework pdf must be uploaded to Gradescope by the due date. You must show your work to receive full credit. Discussing possible solutions for homework questions is encouraged on course discussion board and with your peers, but you must write your own individual solutions and **not** share your written work/code. You must cite all resources (including online material, books, articles, help taken from specific individuals, etc.) you used to complete your work.

1 Numerical and Theory Problems (Due: Mon Aug 09)

1. Let $Q = \begin{bmatrix} | & & | \\ q_1 & \dots & q_n \\ | & & | \end{bmatrix}$ be an $m \times n$ matrix with columns q_i all unit length and mutually orthogonal to each other.

   (a) Show that the transformation associated with such a matrix preserves the lengths and angles of all vectors. That is, for all $x, y \in \mathbb{R}^n$, we have
   - $\|x\| = \|Qx\|$, and
   - $\angle(x, y) = \angle(Qx, Qy)$.

   (b) Show that $m \geq n$.

2. It is often useful to consider an *orthogonal basis*[1] for a subspace. One way to *orthogonalize* a vector v from a set of vectors $v_1, \dots, v_k$ is to *subtract off* the component that is parallel to the vectors $v_1, \dots, v_k$ from v.

   Consider two vectors $v_1, v_2 \in \mathbb{R}^n$ ($n \geq 2$) that form a basis for a 2-dimensional plane $P \subseteq \mathbb{R}^n$. Using the idea discussed above, we can come up with an orthogonal basis for P as follows:

   - $b_1 = v_1$
   - $b_2 = v_2 - \alpha$, where $\alpha + \beta = v_2$, $\alpha \in \text{span}\{v_1\}$, $\beta \in \text{span}\{v_1\}^\perp$.
     (That is, α is the component of v_2 that is parallel to the direction v_1, and β is the component of v_2 that is perpendicular to the direction v_1)

   (a) Write down an expression for b_2 that only uses the vectors v_1 and v_2.

   (b) Show that the following facts:

[1] An orthogonal basis $b_1, \dots, b_k$ is such that the basis vectors are mutually orthogonal. That is, $b_i \cdot b_j = 0$ for all $i \neq j$.

- $b_1 \cdot b_2 = 0$,
- span$\{b_1, b_2\} = P$.

(these two facts collectively show that $\{b_1, b_2\}$ form an orthogonal basis for the given subspace P).

Let v_1, v_2, v_3 form a basis for a given subspace W.

(c) Inspired from the discussion above, provide a procedure to compute an orthogonal basis b_1, b_2, b_3 for W.

(d) Verify that your basis set provided in part (c) is in fact orthogonal and spans the given subspace W.

3. **[Graph Theory Revisited]** In HW3 we wanted to establish a connection between graph connectivity and some of the linear algebra primitives, but we hit a snag. So we will try again.

   Given a graph $G = (V, E)$ with $v := |V|$ vertices and $e := |E|$ edges. We can derive two important matrices

   - The $v \times v$ *adjacency* matrix A, where A_{ij} entry denotes if there is an edge between the ith and the jth node. That is, $A_{ij} = \begin{cases} 1 & \text{exists edge } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$.
   - The $v \times v$ diagonal *degree* matrix D, where the D_{ii} entry denotes the degree of the ith node. That is $D_{ii} = \#$ of edges connected to v_i

   (a) Provide examples of three unweighted undirected (simple) graphs, each with six vertices. Make sure that each graph is disconnected and each graph has a different number of connected components from the other graphs. (You can use the same examples as you used in HW3)

   For each case, compute the dimension of the null space of the difference matrix $(D-A)$.

   (b) What connection/observation can you make between the dimension of the null space of the matrix $(D - A)$ and the number of connected components of the graph? (You don't need to prove anything.)

4. Let A be an $n \times n$ matrix whose rank is 1. Let $v := (v_1, \ldots, v_n)^\mathsf{T} \neq 0$ be a basis for col(A).

   (a) Show that $A_{ij} = v_i w_j$ for some vector $w := (w_1, \ldots, w_n)^\mathsf{T} \neq 0$.

   (b) If the vector $z = (z_1, \ldots, z_n)^\mathsf{T} \neq 0$ satisfies $z \cdot w = 0$, show that z is an eigenvector (of A) with eigenvalue 0.

   (c) The *trace* of a matrix is the sum of its diagonal entries i.e. tr$(A) = \sum_i^n A_{ii}$. With this definition show the following:

   If tr$(A) \neq 0$, then tr(A) is an eigenvalue of A. What is the corresponding eigenvector?

   (d) Two matrices X and Y are *similar* if there exists an invertible matrix Z such that $X = ZYZ^{-1}$. With this definition show the following:

   If tr$(A) \neq 0$, prove that A is similar to the following $n \times n$ matrix

$$\begin{bmatrix} c & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix},$$

where $c = \text{tr}(A)$.

(e) If $\text{tr}(A) = 1$, show that A is a projection, that is, $A^2 = A$.

2 Applications & Programming (Due: Mon Aug 16)

For this question, answer all relevant writing parts in your homework report.

Consider the case where you are given access to some data, but you discover that the data is too bulky for you to carry around. You will eventually want to use this data for some downstream task: if your data is weather statistics, you may want to use it to build a prediction model, or if it's an image, you may want to include it in a presentation. Unfortunately, you have no information about what this downstream task is- all you know is that you want to **compress** your data so that it fits in the memory that you have available, and that you have less memory available than what the data occupies right now. In this case, it is reasonable to assume you want to attempt the following:

- To the best of your ability, you want to go for **lossless compression**. This means that you want to eliminate all the possible redundancies in your data. For example, say that your data is a matrix of n columns in $\mathbb{R}^d$ but only $m < n$ of these columns are linearly independent. Then storing the remaining of the $n - m$ columns in their entirety is a redundancy, and there is room for lossless compression.
- If lossless compression is not possible, you want to work toward **lossy compression** that incurs the least amount of data loss. This means that in the pursuit of compression, you will loose some information, but you still want to quantify **how much** information you lost and minimize some notion of this quantity.

We will see how Singular Value Decomposition can help us achieve either of these outcomes. Let us consider the application of compression to image data. We will say that matrix $X \in \mathbb{R}^{m \times n}$ represents a grayscale image, with each element of these matrix representing the grayscale value at that pixel location. Assume through this question that it takes **1 unit of memory** to store one element of a matrix

1. How many units of memory does it take to store the matrix M? Your answer should be in terms of m and n.
2. Assume now that you have access to some algorithm A that provides you with a *low-rank approximation* of M. Essentially, given M, the algorithm A returns a matrix L and another matrix R such that $M \approx LR^T$ where L and R are taller than they are wide. Let us label the number of columns in L and R as k. How many rows must L and R have respectively?
3. This makes L and R a new representation of our matrix M, because (an approximation of) M can be recovered from them (since $M \approx LR^T$). Given your answer in (2), what can you say about how many units of memory it takes to store this new representation of M? Give an inequality relating k, m and n that describes the condition that must be met for L and R to be a more memory-efficient representation of our matrix M.
4. In parts (2) and (3), you assumed that an algorithm could magically give you the matrices L and R. However, you have already learnt one such algorithm: the Singular Value decomposition. If the full-rank SVD of $M = USV^T$, then you can simply take $L = US$ and $R = V$ to get the left and right matrices. In the full-rank approximation, we have that our estimate LR^T

exactly equals the original M. How many units of memory are required to store this matrix? Is it better or worse than part (1)?

To turn exact equality into an approximation that gives us a lower rank approximation of M, we should take only the first k columns of our decomposition matrices. Then, we can model $L = U_k S_k$ and $R = V_k$ where $U_k \in \mathbb{R}^{m \times k}$, $S_k \in \mathbb{R}^{k \times k}$ and $V_k \in \mathbb{R}^{n \times k}$ and recover $\tilde{M}_k = LR^T$ as an approximation of M.

Why does this make for a good approximation of our original M? And under what notion of approximation is this $\tilde{M}_k$ close to M.? A common metric of measuring the distance between matrices is the **Frobenius norm**. The Frobenius norm is the matrix generalization of the typical vector norm that we are familiar with. Then, we can measure the distance between two matrices of the same size using the Frobenius norm of their difference:

$$||M - \tilde{M}_k||_F = \sqrt{\sum_{ij} (M - \tilde{M}_k)_{ij}^2}$$

Consider $M = USV^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ where $v_1, v_2, ..., v_r$ are the right singular vectors of M and $u_1, u_2, ..., u_r$ are the left singular vector of M, all corresponding to M's top singular values (arranged in descending order of size) $\sigma_1, \sigma_2, ..., \sigma_r$. Then $\tilde{M}_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ where $k \leq r$.

5. This notation makes it explicit that $\tilde{M}_k$ has rank k . Why?

   It can be shown that $\tilde{M}_k$ is the best rank-k approximation of M under the Frobenius norm. Intuitively, this can be argued as follows: the rows of $\tilde{M}_k$ are the projections of the rows of M onto the subspace spanned by the first k singular vectors of M. Hence the rows of $\tilde{M}_k$ represent some notion of the "best-fit" k-dimensional subspace of M. Since we are working with the Frobenius norm as the metric, we can carry over our intuition for "best-fit" from vector norms. We will test this notion empericially.

   We will now move onto the programming parts.

6. Download the image provided in the HW release titled **camera.png**, which is a 226 $\times$ 226 full-rank matrix. You should open it using the **Image.open** function from the **Pillow** library. This should give you a [226, 226, 3] matrix of which you should only take the first channel, yielding a [226, 226] matrix. You can then check (approximately) that this matrix is full rank using the **matrix rank** function in the **numpy.linalg** library.

   (a) Study the documentation for **numpy.linalg.svd** function to calculate SVD carefully, and make sure how it returns the singular values and the left and right singular vectors. Using this function, calculate the SVD of M and recover S, U and V. Reorder S, U and V in decreasing order of the size of the singular values (this should be easy in NumPy). Make a plot where the horizontal axis is the rank of your singular value and the vertical axis is the n^{th} largest singular value in S. Precisely, your horizontal axis should range from 1 to 226, and the corresponding points on the vertical axis should be the largest and the smallest singular values. Include this plot in your report titled **Singular**, and retain the matrices for S, U and V in your code for further parts.

(b) Use the method outlined above to get S_k, U_k and V_k for four different values of $k \leq 226$. Compute L and R using these matrices and visualize your corresponding $\tilde{M}_k$. Include all your images in the report, labelled with corresponding values of k, and reflect on what you observe. Remember that if you use $k = 226$, computing LR^T should recover the original image exactly (barring numerical issues).

(c) For a particular value of k define I_k as the weight of the first k singular values relative to the total sum of all singular values. Therefore

$$I_k = \frac{\sum_{i=1}^{k} \sigma_i}{\sum_{i=1}^{128} \sigma_i}$$

This is a representation of the percent "information" about M carried by the first k singular vectors. Make another plot with the same horizontal axis as (a) ranging from $k = 1$ to $k = 226$ and vertical axis as I_k. Include this plot in your report, titled **Information**.

(d) Finally, make a third plot with the same horizontal axis as (a) and (c) ranging from $k = 1$ to $k = 226$ and the vertical axis referring to the memory usage in **units of memory** using your formula in part (3). Include this plot in your report, titled **Memory**.

(e) Using your graphs in part (a), (c) and (d), visual intuition from part (b), and your inequality from (3), pick a value of k that is suitable for storing the given image with memory constraints. Justify why this would make a good value for k. Include the resulting $\tilde{M}_k$ in your report titled **Image of rank k**.

.

Please make sure you submit code for all parts of the programming to GradeScope as a zip file, with a function for each part clearly labelled. **No points will be given unless both the code and the report section for this problem is answered.**