COMS 3251 SU21 HW3

Due: Aug 2nd, 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 Problems

1. Which of the following sets are linear spaces? Justify your answer for each case.
   - $\{X = (x_1, x_2, x_3)^\mathsf{T} \in \mathbb{R}^3$ with the property $x_1 - 2x_3 = 0\}$
   - The set of solutions x of $Ax = 0$, where A is an $m \times n$ matrix.
   - The set of 2×2 matrices $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ with such that $a_{11}a_{22} - a_{12}a_{21} = 0$.
   - The set of polynomials $p(x)$ with $\int_{-1}^{1} p(x)dx = 0$.
   - The set of solutions $y = y(t)$ of $y'' + 4y' + y = 0$.
   - The set of solutions $y = y(t)$ of $y'' + 4y' + y = 7e^{2t}$.
   - Let S_f be the set of solutions $u(t)$ of the differential equation $u'' - xu = f(x)$. For which continuous functions f is S_f a linear space? Why? [Note: You are not being asked to actually solve this differential equation.]
2. For which real numbers x do the vectors: $(x, 1, 1, 1)^\mathsf{T}, (1, x, 1, 1)^\mathsf{T}, (1, 1, x, 1)^\mathsf{T}, (1, 1, 1, x)^\mathsf{T}$ not form a basis of $\mathbb{R}^4$? For each of the values of x that you find, what is the dimension of the subspace of $\mathbb{R}^4$ that they span?

2 Applications

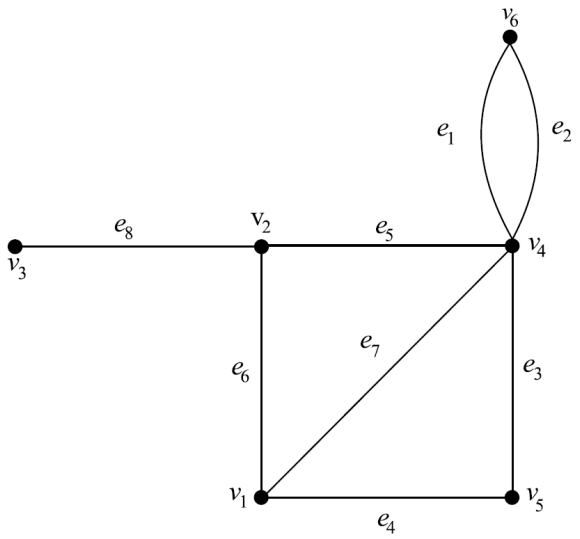
1. **[Graph Theory]** It turns out that one can can infer several graph properties by representing graphs as matrices and performing various matrix operations. We'll explore a few such connections between graphs and matrices here.

   Given a graph $G = (V, E)$ with $v := |V|$ vertices and $e := |E|$ edges. We can represent G as a $v \times e$ *incidence* matrix A, where the ij entry of A denotes if the jth edge is incident on the

ith vertex. That is,

$$a_{ij} := \begin{cases} 1 & \text{if the } j\text{th edge is incident on the } i\text{th vertex} \\ 0 & \text{otherwise} \end{cases}.$$

As an example, consider the undirected graph G_1 with 6 vertices and 8 edges below[1]:



The corresponding incidence matrix would be:

$$A(G_1) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Note the following:

- Each column has exactly two non-zero entries, because each edge is incident on exactly two vertices.[2]
- Total number of non-zero entries per row corresponds to the degree of the corresponding vertex.
- Permuting any row (or column) simply corresponds to relabelling the verties (or edges) of the corresponding graph.

[1] Usually we study *simple* graphs – those with at most one edge between a pair of vertices. This example example graph is not *simple*. There are two distinct edges, namely e_1 and e_2, between vertices v_4 and v_6.

[2] Here we are only considering graphs with no *hyper*-edges (that is, edges between three or more vertices simultaneously).

1. Compute the rank of the matrix $A(G_1)$. Show your steps for full credit.

2. Provide examples of three (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. For each case, compute the rank of the corresponding incidence matrix.

3. From solving the previous part you will see a pattern emerging. Specifically, the following is true.

   Let G be an undirected graph with n vertices and k connected components. Then the rank of the corresponding incidence matrix is necessarily $n - k$.

   We shall now prove this statement. In what follows, assume that G is a simple graph with n vertices. Let A be the corresponding incidence matrix.

   (a) Prove that $\text{rank}(A) < n$.

   (b) Prove that if G is connected, then $\text{rank}(A) = n - 1$.
   $\Big[$*Hint: if G is connected, then A cannot be written in a block diagonal form* $\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}$.$\Big]$

   (c) Prove that if G has k connected components, then $\text{rank}(A) = n - k$.
   $\Big[$*Hint: if G has k connected components, then (possibly after reordering the rows and/or columns) A can be be written in a block diagonal form (with k blocks).*$\Big]$

3 Theory

1. Let V be a vector space and $f : V \to \mathbb{R}$ be a linear map. If $z \in V$ is not in the nullspace of f, show that every $x \in V$ can be decomposed uniquely as $x = v + cz$, where v is in the nullspace of f and c is a scalar.

2. Let A be an $n \times n$ matrix. If $AB = BA$ for all invertible matrices B, show that $A = cI$ for some scalar c.

4 Programming

For this problem, follow the function definitions given in **benchmarking.ipnyb**. The only libraries permitted for this assignment are `NumPy` and `timeit`. Do not modify any of the function signatures.

1. Time complexity is an essential concern in many applications. In many data science applications, datasets often don't fit in random access memory. In such domains, whether each operation is linear versus quadratic versus cubic can mean the difference between minutes versus hours versus days of compute time. For this problem, you will benchmark the runtime of several matrix operations and implementations using the python library `timeit`. In particular, for matrices of varying size n, you will compare the time complexity of:

   - Your own matrix multiplication function using the schoolbook matrix multiplication algorithm, `my_matmul`

- Your own matrix addition function, `my_add`
- Your own matrix inverse function, `my_inv`
- NumPy's matrix multiplication function, `numpy.linalg.matmul`
- NumPy's matrix addition function, `numpy.linalg.add`
- NumPy's matrix inverse function, `numpy.linalg.inv`
- NumPy's system of equations solver, `numpy.linalg.solve`

Choose a range of n, for example $[0, 5, 10, 15, 20, ..., m]$, on which to sample the benchmark values for these operations. For each function, plot the resulting benchmark values on your chosen range and include the plots in your pdf.

2. We'll now use our understanding of the theoretical time complexities of these operations to fit a curve to each of our generated datasets using the method of **least squares**.
Let's take the example of matrix addition. We know theoretically that both `my_add` and `numpy.linalg.add` will run in time $O(n^2)$. Therefore, we expect to be able to fit a function of the form $f(n) = x_1 n^2 + x_2, x_1 \in \mathbb{R}, x_2 \in \mathbb{R}$ to the benchmark samples of those functions. But that is not a linear function, and this is linear algebra, so how can we find the parameters x_1 and x_2, in the function f? By our sampling, $f(n)$ is equal to the benchmark result at n, and is therefore known for all n in the sample, as is the value n^2. Therefore, if you have m samples, you will have m equations:

$f(n_1) = x_1 \cdot n_1^2 + x_2 \cdot 1$
$f(n_2) = x_1 \cdot n_2^2 + x_2 \cdot 1$
...
$f(n_m) = x_1 \cdot n_m^2 + x_2 \cdot 1$

This is a linear system of equations. We can express it in the more familiar matrix form:

$$Ax = \begin{bmatrix} n_1^2 & 1 \\ n_2^2 & 1 \\ ... & \\ n_m^2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} f(n_1) \\ f(n_2) \\ ... \\ f(n_m) \end{bmatrix} = b$$

However, this system of equations is **overdetermined**, meaning that there are more equations than unknowns. In this situation, it is highly unlikely that the system is consistent. In other words (equations), we expect that $\nexists x$ s.t $Ax = b$. But perhaps we can approximate a solution. That is, perhaps we can find some $\hat{x}$ such that $A\hat{x}$ is "close" to b. Think about it this way, $Ax = b$ is a perfect solution because the distance from Ax to b is 0, $||Ax - b|| = 0$. Hence our approximation should aim to minimize $||A\hat{x} - b||$.
This is the fundamental idea of least squares. We aim to minimize the squared distance from $A\hat{x}$ to b, i.e we aim to find $\min_{\hat{x}} ||A\hat{x} - b||^2$, which is equivalent to $\min_{\hat{x}} ||A\hat{x} - b||$.
A least squares solution is therefore the orthogonal projection of x onto the column space of A. Recall from lecture the closed form solution, $A^T Ab = A^T x$.

For each of the sample datasets you created, use least squares to fit a curve to the plot. You are welcome to use the `NumPy` library as well as `matplotlib` . For each of the function implementations listed above in 4.1, be sure to include a figure that contains both the data and the

curve you fitted to the data. Remember that the library implementations may have asymptotic complexity different from your own implementations and therefore you may need to adjust the exponent in your approximation of the time complexity, p in $f(n) = x_1 n^p + x_2$. Discuss how the time complexity of the library implementations compare to your implementations for matrix multiplication and matrix addition. Remember to include your analysis and plots in your pdf to receive credit.

To receive credit, submit a zip file containing **benchmark.py**. Your zip file should also contain a README.txt if you consult any third-party sources (the `NumPy` and `timeit` references need not be included).