New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6763: Homework 5.
Due Monday, May 5th, 2025, 11:59pm.

Collaboration is allowed on this problem set, but solutions must be written-up individually. Please list collaborators for each problem separately, or write "No Collaborators" if you worked alone.

Problem 1: Approximating Eigenvalues Moments

Optional Extra Credit: Worth extra 20% of total problem set points.

Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a square symmetric matrix, which means it it guaranteed to have a symmetric eigendecomposition with real eigenvalues, $\lambda_1 \geq \ldots \geq \lambda_n$, and orthogonal eigenvectors. While computing these eigenvalues naively takes $O(n^3)$ time, we can compute their *sum* much more quickly: with n operations. This is because $\sum_{i=1}^n \lambda_i$ is exactly equal to the trace of $\mathbf{A}$, i.e. the sum of its diagonal entries $\text{tr}(\mathbf{A}) = \sum_{i=1}^n \mathbf{A}_{ii}$. We can also compute the sum of squared eigenvalues in $O(n^2)$ time by taking advantage of the fact that $\sum_{i=1}^n \lambda_i^2 = \|\mathbf{A}\|_F^2$ where $\|\mathbf{A}\|_F^2$ is the Frobenius norm. What about $\sum_{i=1}^n \lambda_i^3$? Or $\sum_{i=1}^n \lambda_i^4$? It turns out that no exact algorithms faster than a full eigendecomposition are known.

In this problem, however, we show how to *approximate* $\sum_{i=1}^n \lambda_i^k$ for any positive integer k in $O(n^2 k)$ time. Doing so turns out to have a lot of applications in machine learning and data science.

(a) Show that $\sum_{i=1}^n \lambda_i^k = \text{tr}(\mathbf{A}^k)$ where $\mathbf{A}^k$ denotes the chain of matrix products $\mathbf{A} \cdot \mathbf{A} \cdot \ldots \cdot \mathbf{A}$, repeated k times. For the remainder of the problem we use the notation $\mathbf{B} = \mathbf{A}^k$.

(b) Let $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)} \in \mathbb{R}^n$ be m independent random vectors, all with i.i.d. $\{+1, -1\}$ uniform random entries. Let $Z = \frac{1}{m} \sum_{i=1}^m (\mathbf{x}^{(i)})^T \mathbf{B} \mathbf{x}^{(i)}$. We will show that Z is a good estimator for $\text{tr}(\mathbf{B})$ and thus for $\sum_{i=1}^n \lambda_i^k$. Give a short argument that Z can be computed in $O(n^2 k m)$ time (recall that $\mathbf{B} = \mathbf{A}^k$).

(c) Prove that:

$$\mathbb{E}[Z] = \text{tr}(\mathbf{B}) \qquad \text{and} \qquad \text{Var}[Z] \leq \frac{2}{m} \|\mathbf{B}\|_F^2$$

Hint: Use linearity of variance but be careful about what things are independent!

(d) Show that if $m = O(\frac{1}{\epsilon^2})$ then, with probability 9/10,

$$|\,\text{tr}(\mathbf{B}) - Z| \leq \epsilon \|\mathbf{B}\|_F.$$

(e) Argue that, when $\mathbf{A}$ is positive semidefinite (equivalently, $\lambda_1, \ldots, \lambda_n$ are all positive), $\epsilon \|\mathbf{B}\|_F \leq \epsilon\, \text{tr}(\mathbf{B})$, so the above guarantee actually gives the relative error bound,

$$(1 - \epsilon)\, \text{tr}(\mathbf{B}) \leq Z \leq (1 + \epsilon)\, \text{tr}(\mathbf{B}),$$

all with just $O(n^2 k / \epsilon^2)$ computation time.

Problem 2: Gradient Descent as Power Method

(10 pts) Considering minimzing the function $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ for $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\mathbf{b} \in \mathbb{R}^n$ with gradient descent. $f(\mathbf{x})$ has gradient $\nabla f(\mathbf{x}) = 2\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{A}^T\mathbf{b}$. As mentioned in class, we can obtain a convergence bound for this function that scales with the *condition number* of $\mathbf{A}^T\mathbf{A}$, which is the ratio of the largest and smallest eigenvalues of $\mathbf{A}^T\mathbf{A}$, λ_1/λ_d. It turns out that this can be proven using an analysis very similar to power method!

1. Prove that, when gradient descent is run on $f(\mathbf{x})$, the difference between the i^{th} iterate and $\mathbf{x}^*$ can be written as:

$$\mathbf{x}^{(i)} - \mathbf{x}^* = (\mathbf{I} - 2\eta \mathbf{A}^T\mathbf{A})(\mathbf{x}^{(i-1)} - \mathbf{x}^*)$$

By induction, it follows that the error $\mathbf{x}^{(i)} - \mathbf{x}^*$ equals $(\mathbf{I} - 2\eta \mathbf{A}^T\mathbf{A})^i (\mathbf{x}^{(0)} - \mathbf{x}^*)$.

2. Prove that if we set $\eta = 1/2\lambda_1$, then after $i = O(\frac{\lambda_1}{\lambda_d}\log(1/\epsilon))$ iterations, $\|(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A})^i\|_2 < \epsilon$. Recall that $\|\cdot\|_2$ denotes the spectral norm of a matrix, which is equivalent to it's largest singular value, or the largest absolute value of the matrix's eigenvalues .

3. Conclude that, after $O(\frac{\lambda_1}{\lambda_d}\log(1/\epsilon))$ iterations, gradient descent finds a vector $\mathbf{x}^{(i)}$ with $\|\mathbf{x}^{(i)} - \mathbf{x}^*\|_2 \leq \epsilon\|\mathbf{x}^{(0)} - \mathbf{x}^*\|_2$. **This can be a one line proof.**

4. We can also use this approach to obtain an "accelerated" version of gradient descent. To do so, prove that, for any degree q polynomial $p = c_0 + c_1 x + \ldots + c_q x^q$ with $p(1) = c_0 + c_1 + \ldots + c_q = 1$ and any starting vector $\mathbf{x}^{(0)}$, we can compute using q gradient computations and up to $O(dq)$ additional runtime a vector $\mathbf{x}^{(q)}$ such that:

$$\mathbf{x}^{(q)} - \mathbf{x}^* = p\left(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A}\right)(\mathbf{x}^{(0)} - \mathbf{x}^*).$$

   Note that this result strictly generalizes what we know from gradient descent, which computes $\mathbf{x}^{(q)}$ satisfying the equation for the polynomial $p(x) = x^q$, which satisfies our restriction that $p(1) = 1$.

5. Prove that for $q = O(\sqrt{\frac{\lambda_1}{\lambda_d}}\log(1/\epsilon))$, there exists a polynomial p with coefficients $c_0 + c_1 + \ldots + c_q = 1$ such that $\|p\left(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A}\right)\|_2 \leq \epsilon$. **Hint:** You might want to use Claim 4 in the supplemental notes on the Lanczos method posted on the course website.

Just as for regular gradient descent, it follows that $\|\mathbf{x}^{(q)} - \mathbf{x}^*\|_2 = \|p\left(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A}\right)(\mathbf{x}^{(0)} - \mathbf{x}^*)\|_2 \leq \epsilon\|\mathbf{x}^{(0)} - \mathbf{x}^*\|_2^2$ as long as we use degree $q = O(\sqrt{\frac{\lambda_1}{\lambda_d}}\log(1/\epsilon))$ – i.e. use $O(\sqrt{\frac{\lambda_1}{\lambda_d}}\log(1/\epsilon))$ gradient computations.

Problem 3: Spectral Methods for Cliques

(15 pts) A common tasks in data mining is to identify large *cliques* in a graph. For example, in social networks, large cliques can be indicators of fraudulent accounts or networks of accounts designed to promote certain content. In this problem, we consider a spectral heuristic for finding a large clique based on the top eigenvector of the graph adjacency matrix A:

- Compute the leading eigenvector v_1 of A.
- Let $i_1, \ldots, i_k \in \{1, \ldots, n\}$ be the indices of the k entries in v_1 with largest absolute value.
- Check if nodes $i_1, \ldots, i_k$ form a k-clique.

We will analyze this heuristic on a natural random graph model. Specifically, let G be an Erdos-Renyi random graph: we start with n nodes, and for every pair of nodes (i, j), we add an edge between the pair with probability $p < 1$. To simplify the math, also assume that we add a self-loop at every vertex i with probability p. Then, choose a fixed subset S of k nodes to form a clique. Connect all nodes in S with edges and add self-loops. We will argue that, for sufficiently large k, we can expect the heuristic above to identify the nodes in the clique.

1. Let A be the adjacency matrix of a random graph generated as above. What is $\mathbb{E}[A]$? Prove that the rank of $\mathbb{E}[A]$ is 2. In other words, the matrix only has two non-zero eigenvalues.

2. Derive expressions for the two non-zero eigenvalues of $\mathbb{E}[A]$, and their corresponding eigenvectors. **Hint:** First argue that, up to multiplying by a constant, any eigenvector v must have $v[i] = 1$ for all $i \notin S$ and $v[i] = \alpha$ for all $i \in S$, where α is a constant. Then use some high school algebra.

3. Using your results from (2) above, argue that, up to a positive scaling, the top eigenvector v_1 has $v[i] = 1$ for all $i \notin S$ and $v[i] = \alpha$ for all $i \in S$, where $\alpha > 1$. In other words, the largest entries of v_1 exactly correspond to the nodes in the clique!

4. To prove the algorithm works, it is possible to use a matrix concentration inequality to argue that the top eigenvector of A is close to that of $E[A]$. Instead of doing that, let's verify things experimentally. Generate a graph G according to the prescribed model with $n = 900$, $k = |S| = 30$, and $p = .1$. Compute the top eigenvector of A and look at its 30 largest entries in magnitude. What fraction of nodes in the clique S are among these 30 entries? Repeat the experiment and report the average fraction recovered.