

New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6763: Homework 4.

Due Thursday, December 14th, 2023, 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

(15 pts) Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a square symmetric matrix, which means it is guaranteed to have a symmetric eigendecomposition with real eigenvalues, $\lambda_1 \geq \dots \geq \lambda_n$, 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 $O(n^2k)$ time. By the way, this is not a contrived problem – it has a ton of applications in machine learning and data science that you can ask me about in office hours!

- Show that $\sum_{i=1}^n \lambda_i^k = \text{tr}(\mathbf{A}^k)$ where \mathbf{A}^k denotes the chain of matrix products $\mathbf{A} \cdot \mathbf{A} \cdot \dots \cdot \mathbf{A}$, repeated k times. For the remainder of the problem we use the notation $\mathbf{B} = \mathbf{A}^k$.
- Let $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)} \in \mathbb{R}^n$ 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 $O(n^2km)$ time (recall that $\mathbf{B} = \mathbf{A}^k$).
- Prove that:

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

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

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

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

- Argue that, when \mathbf{A} is positive semidefinite, $\epsilon \|\mathbf{B}\|_F \leq \epsilon \text{tr}(\mathbf{B})$, 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 $O(n^2k/\epsilon^2)$ computation time.

Problem 2: Accelerated Gradient Descent Through the Polynomial Lens

(10 pts) In Lecture 7, we saw how to analyze gradient descent for $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, which has gradient $\nabla f(\mathbf{x}) = 2\mathbf{A}^T \mathbf{A}\mathbf{x} - 2\mathbf{A}^T \mathbf{b}$. The dominant cost for each gradient descent iteration is multiplying \mathbf{x} by $\mathbf{A}^T \mathbf{A}$ to compute the gradient, which takes $O(nd)$ time when A is $n \times d$.

We obtained a convergence bound depending on the largest and smallest eigenvalues of $\mathbf{A}^T \mathbf{A}$, which we denote λ_1 and λ_d respectively. We did so by rearranging the gradient descent update rule:

$$\begin{aligned} \mathbf{x}^{(i)} &= \mathbf{x}^{(i-1)} - \eta \left(2\mathbf{A}^T \mathbf{A}\mathbf{x}^{(i-1)} - 2\mathbf{A}^T \mathbf{b} \right) \\ \mathbf{x}^{(i)} - \mathbf{x}^* &= \mathbf{x}^{(i-1)} - \eta \left(2\mathbf{A}^T \mathbf{A}\mathbf{x}^{(i-1)} - 2\mathbf{A}^T \mathbf{A}\mathbf{x}^* \right) - \mathbf{x}^* \quad \text{since } \nabla f(\mathbf{x}^*) = \mathbf{0}, \text{ so } \mathbf{A}^T \mathbf{A}\mathbf{x}^* = \mathbf{A}^T \mathbf{b} \\ \mathbf{x}^{(i)} - \mathbf{x}^* &= (\mathbf{I} - 2\eta \mathbf{A}^T \mathbf{A})(\mathbf{x}^{(i-1)} - \mathbf{x}^*). \end{aligned}$$

By induction, it follows that the error $\mathbf{x}^{(i)} - \mathbf{x}^*$ equals $\mathbf{x}^{(i)} - \mathbf{x}^* = (\mathbf{I} - 2\eta\mathbf{A}^T\mathbf{A})^i(\mathbf{x}^{(0)} - \mathbf{x}^*)$. This allowed us to obtain a convergence bound by arguing that, if we set $\eta = 1/2\lambda_1$ where λ_1 is the largest eigenvalue of $\mathbf{A}^T\mathbf{A}$, then $(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A})^i$ has top eigenvalue $< \epsilon$ after $i = O(\frac{\lambda_1}{\lambda_d} \log(1/\epsilon))$ iterations. In this problem you will prove an “accelerated” version of this bound with a significantly improve condition number dependence of $O(\sqrt{\frac{\lambda_1}{\lambda_d}} \log(1/\epsilon))$ iterations.

1. Let p be any degree q polynomial. I.e. $p = c_0 + c_1x + \dots + c_qx^q$. Show that, for any p with $p(1) = c_0 + c_1 + \dots + c_q = 1$ and any starting vector $\mathbf{x}^{(0)}$, we can compute in q iterations (i.e., using q gradient computations and up to $O(ndq)$ 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 $p(1) = 1$.

2. Prove that for $q = O(\sqrt{\frac{\lambda_1}{\lambda_d}} \log(1/\epsilon))$, there exists a polynomial p with coefficients $c_0 + c_1 + \dots + c_q = 1$ such that the top eigenvalue of $p\left(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A}\right) \leq \epsilon$. **Hint:** You might want to use Claim 4 in the supplemental notes on the Lanczos method posted for Lecture 11.

By Part 2, above, 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. run for $O(\sqrt{\frac{\lambda_1}{\lambda_d}} \log(1/\epsilon))$ iterations.

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 $i_1, \dots, i_k \in \{1, \dots, n\}$ be the indices of the k entries in v_1 with largest absolute value.
- Check if nodes i_1, \dots, 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 2!
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 $\mathbb{E}[A]$. 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.

Problem 4 (Bonus): Matrix Concentration from Scalar Concentration

(10 pts) This problem asks you to prove a simplified (and slightly weaker) version of the matrix concentration result used in Lecture 12. Construct a random *symmetric* matrix $R \in \mathbb{R}^{n \times n}$ by setting $R_{ij} = R_{ji}$ to +1 or -1, uniformly at random. Prove that, with high probability,

$$\|R\|_2 \leq c\sqrt{n \log n},$$

for some constant c . This is much better than the naive bound of $\|R\|_2 \leq \|R\|_F = n$ and it's nearly tight: we always have that $\|R\|_2^2 \geq \|R\|_F^2/n$ (do you see why?) so $\|R\|_2 \geq \sqrt{n}$ no matter what.

Here are a few hints that might help you along:

- Recall that for a matrix R , $\|R\|_2 = \max_{x \in \mathbb{R}^n} \frac{\|Rx\|_2}{\|x\|_2}$. When R is symmetric, it also holds that $\|R\|_2 = \max_{x \in \mathbb{R}^n} \frac{|x^T Rx|}{x^T x}$.
- Try to first bound $\frac{|x^T Rx|}{x^T x}$ for one particular x . You might want to use a Hoeffding bound.
- Then try to extend the result to hold for all x simultaneously, using an ϵ -net argument.