Homework 2
Problem 1 - Working with Low-Rank Approximations

Christian Howard
howard28@illinois.edu

Contents

1 Algorithm Design

In the problem statement for this part, we are to come up with an algorithm to tackle least square problems $Ax \cong b$ given that $A \in \mathbb{R}^{m \times n}$ is low rank and approximated as $A \approx QB$, where $B = Q^T A$ with $m > n \geq k$ and $Q \in \mathbb{R}^{m \times k}$. Additionally, for cases where the least square solution is not unique, i.e. when $k < n$, we are asked to add that the solution x will minimize $|x|_2$.

If we first assume that $k = n$, we can find the least square solution by finding $x^* = \arg\min_x J(x)$ where $J(x)$ is defined below as:

$$\begin{aligned} J(x) &= (Ax - b)^T (Ax - b) \\ &= x^T A^T A x - 2x^T A^T b + b^T b \end{aligned}$$

If we differentiate with respect to x^T, we can find the solution by doing the following:

$$\begin{aligned} \frac{\partial J}{\partial x^T} &= 0 = 2A^T A x^* - 2A^T b \\ &= A^T A x^* - A^T b \end{aligned}$$

which implies a solution of:

$$x^* = \left(A^T A\right)^{-1} A^T b$$

If we assume A is in the form of a Singular Value Decomposition, i.e. $A = U \Sigma V^T$, then we can show the following solution for the Least Square solution when $k = n$:

$$\begin{aligned} x^* &= \left(V \Sigma U^T U \Sigma V^T\right)^{-1} V \Sigma U^T b \\ &= \left(V \Sigma^2 V^T\right)^{-1} V \Sigma U^T b \\ &= V \Sigma^{-2} V^T V \Sigma U^T b \\ &= V \Sigma^{-1} U^T b \\ &= V \Sigma^{+} U^T b \end{aligned} \tag{1}$$

where Σ^+ is the pseudoinverse of Σ where basically all non-zero diagonal elements are replaced by their reciprocals. Now let us assume that $k < n$ so that we need to enforce that the solution x^* must also minimize $|x|_2$. We can find this solution x^* by solving the following optimization problem based on Lagrange Multipliers:

$$\begin{aligned} (x^*, \lambda^*) &= \arg\min_{x, \lambda} J(x, \lambda) \\ \text{where } J(x, \lambda) &= x^T x + \lambda^T (Ax - b) \end{aligned}$$

If we differentiate $J(\cdot,\cdot)$ with respect to x^T and λ^T, we get:

$$\frac{\partial J}{\partial x^T} = 0 = 2x^* + A^T\lambda^*$$
$$\frac{\partial J}{\partial \lambda^T} = 0 = Ax^* - b$$

The first equation implies $x^* = -\frac{1}{2}A^T\lambda$, the second leads to $\lambda^* = -2(AA^T)^{-1}b$. With these relationships, we find the solution for x^* is the following:

$$x^* = A^T\left(AA^T\right)^{-1}b$$

If we again substitute the SVD form of A into the expression for x^*, we can obtain the following:

$$\begin{aligned} x^* &= V\Sigma U^T\left(U\Sigma V^T V\Sigma U^T\right)^{-1}b \\ &= V\Sigma U^T\left(U\Sigma^2 U^T\right)^{-1}b \\ &= V\Sigma U^T U\Sigma^{-2}U^T b \\ &= V\Sigma^{-1}U^T b \\ &= V\Sigma^{+}U^T b \end{aligned} \tag{2}$$

What becomes obvious is that if we compare (1) with (2), the two solutions are identical. This means that using the SVD form of A to solve the Least Square problem works when $k \leq n$ and enforces minimizing $|x|_2$ when $k < n$. With this information, the algorithms we come up with only need to use the information we have to construct efficient SVD decompositions and then the rest is trivial.

1.1 Algorithm 1 - Baseline

1. Find SVD of $A = QB$
   (a) Find SVD of $B \ni B = \hat{U}\Sigma V^T$
   (b) Define $U = Q\hat{U} \ni A = U\Sigma V^T$
2. Compute $x^* = V\Sigma^{+}U^T b$

1.2 Algorithm 2 - Interpolative Decomposition

1. Find SVD of $A = QB$
   (a) Find J and P using an Interpolative Decomposition $\ni Q^T = Q^T_{(:,J)}P^T$
   (b) Compute QR factorization $(A_{(J,:)})^T = \bar{Q}\bar{R}$
   (c) Upsample row coefficients, $Z = P\bar{R}^T$

(d) Compute SVD of $Z \ni Z = U\Sigma\hat{V}^T$

(e) Define $V^T = \hat{V}^T\bar{Q}^T \ni A = U\Sigma V^T$

2. Compute $x^* = V\Sigma^+U^Tb$

2 Computational Complexity

Before going into the complexity analysis, I want to first note that [1] states that a Full Householder QR factorization of some matrix $A \in \mathbb{R}^{m\times n}$ is $O(m^2 n)$. Additionally, I will use the n-Truncated R-SVD such that for some matrix $A \in \mathbb{R}^{m\times n}$ where $m \geq n$, the complexity is $O(mn^2)$. For use in analyzing the Interpolative Decomposition (ID), most of the heavy lifting of ID is based on the RRQR Algorithm with complexity $O(mnk)$ for a matrix $A \in \mathbb{R}^{m\times n}$ that is rank k. These complexities will be used to describe the algorithms below.

2.1 Algorithm 1 - Baseline

1. Find SVD of $A = QB$ for $A \in \mathbb{R}^{m\times n}$, $Q \in \mathbb{R}^{m\times k}$, and $B \in \mathbb{R}^{k\times n}$
   (a) Find SVD of $B \ni B = \hat{U}\Sigma V^T$. Note that $\hat{U} \in \mathbb{R}^{k\times k}$, $\Sigma \in \mathbb{R}^{k\times k}$, and $V \in \mathbb{R}^{n\times k}$
      - Complexity: $O(nk^2)$

   (b) Define $U = Q\hat{U} \ni A = U\Sigma V^T$
      - Complexity: $O(mk^2)$
2. Compute $x^* = V\Sigma^+ U^T b$
   (a) Compute Σ^+ from Σ
      - Complexity: $O(k)$

   (b) Define $q_1 = U^T b$
      - Complexity: $O(km)$

   (c) Define $q_2 = \Sigma^+ q_1$
      - Complexity: $O(k^2)$

   (d) Define $x^* = Vq_2$
      - Complexity: $O(kn)$

The overall complexity of Algorithm 1 as $O(nk^2 + mk^2 + km) \in O(mk^2)$.

2.2 Algorithm 2 - Interpolative Decomposition

1. Find SVD of $A = QB$
   (a) Find J and P using an Interpolative Decomposition $\ni Q^T = Q^T_{(:,J)} P^T$
      - Complexity: $O(mk^2)$

   (b) Compute QR factorization $(A_{(J,:)})^T = \bar{Q}\bar{R}$
      - Compute $C = (A_{(J,:)})(A_{(J,:)})^T \to O(mk^2)$
      - Compute $\bar{R} = \text{cholesky}(C) \to O(k^3)$
      - Compute $\bar{R}^{-1} \to O(k^3)$

- Compute $\bar{Q} = (A_{(J,:)})^T \bar{R}^{-1} \to O(mk^2)$

(c) Upsample row coefficients, $Z = P\bar{R}^T$

- Complexity: $O(mk^2)$

(d) Compute SVD of $Z \in \mathbb{R}^{m\times k} \ni Z = U\Sigma\hat{V}^T$

- Complexity: $O(mk^2)$

(e) Define $V^T = \hat{V}^T\bar{Q}^T \ni A = U\Sigma V^T$

- Complexity: $O(mk^2)$

2. Compute $x^* = V\Sigma^+U^Tb \to O(km)$

The overall complexity of Algorithm 2 is $O(mk^2)$.

2.3 Discussion of Algorithms

So what is interesting to note between the algorithms is they are both of complexity $O(mn^2)$ when $k = n$ since both could benefit from the Truncated R-SVD algorithm. This appears to be a different result than what is talked about in lecture since it is assumed the SVD of $B \in \mathbb{R}^{k\times n}$ in Algorithm 1 should be $O(n^2k)$, but if we can construct the SVD of $Z \in \mathbb{R}^{m\times k}$ in $O(mk^2)$, then we should be able to also construct the SVD of B in $O(nk^2)$. This obviously goes against what is mentioned in the lecture notes, so not sure where the disparity lies.

3 Complexity of Finding Low-rank Projection Matrix

In this problem, we are tasked to come up with algorithms and complexity estimates for two Range Finding algorithms that wish to find a k-rank projection matrix Q such that an input matrix $A \approx QQ^T A$. The two algorithms we need to investigate mainly differ based on their tools to get measurement vectors from A's column space. The first algorithm, which I call the Nominal Non-Adaptive Range Finder, uses random vectors $\{\omega_i\}_{i=1}^k$ to sample A's column space where each vector element is sampled from $\mathcal{N}(0,1)$, i.e. a zero mean and unit variance Normal distribution. The latter algorithm uses a Subsampled Random Fourier Transform method to obtain measurements more efficiently than in the nominal case. With that said, let us proceed with the algorithm descriptions.

3.1 Nominal Non-Adaptive Range Finder

With this algorithm, the steps are pretty straight forward to get a k-rank matrix Q that approximates A's column space.

1. Construct random matrix $\Omega \in \mathbb{R}^{n \times k} \rightarrow O(nk)$
2. Get measurements from A by doing $Y = A\Omega \rightarrow O(mnk)$
3. Perform QR of $Y \ni Y = QR \rightarrow O(mk^2)$
4. Return Q

Using the above steps we can produce an estimate for Q such that $A \approx QQ^T A$ in $O(mnk)$. We can see the main weak point is the matrix multiplication against A in step 2. We can also see that this nominal Non-Adaptive Range Finder is slower than the approximate SVD computation algorithms described above by a factor of $\frac{n}{k}$.

3.2 Subsampled Random Fourier Transform Non-Adaptive Range Finder

This SRFT Range Finder is based on algorithms described in [2] that allow one to speed up the nominal FFT and inverse FFT computations when you only care about either a subset of inputs points or a subset of output points. Using this paper, we can compute $M\Omega^{'}$ for $M \in \mathbb{C}^{m \times n}$ and $\Omega^{'} \in \mathbb{C}^{n \times k}$ in $O(mn\log(k))$ where $\Omega^{'}$ is defined as:

$$\Omega^{'} = \sqrt{\frac{n}{k}} DFR$$

where

- D is an $n \times n$ diagonal matrix whose entries are independent random variables uniformly distributed on the complex unit circle
- F is the unitary $n \times n$ DFT matrix based on the relationship

$$F_{pq} = \frac{1}{\sqrt{n}} e^{-2\pi i(p-1)(q-1)/n}$$

- R is an $n \times k$ matrix that samples k coordinates from n uniformly at random, i.e., its k columns are drawn randomly without replacement from the columns of the $n \times n$ identity matrix

This sped up matrix multiplication against M can be done in the following steps:

1. Compute $\hat{M} = \sqrt{\frac{n}{k}} MD \to O(mn)$
2. Compute $Y = \hat{M}FR$ using the Transform Decomposition such that we only generate k terms, corresponding to the nonzero k columns R would produce, based on m rows from $\hat{M} \to O(mn\log(k))$

With this, we construct the following steps:

1. Construct matrices D and R minimally $\to O(n+k)$
2. Get measurements from A by performing Transform Decomposition based algorithm using $\Omega^{'}$ such that $Y = A\Omega^{'} \to O(mn\log k)$
3. Perform QR of $Y \ni Y = QR \to O(mk^2)$
4. Return Q

Using the above steps, we can produce an estimate for Q such that $A \approx QQ^T A$ in $O(mn\log(k))$. This SRFT algorithm obviously improves upon the Nominal Range Finder algorithm, though still slower than construction of the approximate SVD of A once we have Q.

4 Prove One Statement is True

We are given that C is a square, low-rank matrix. Given this information, show that exactly one of the below statements are true:

(A) The linear system $(I - C)x = b$ has a solution x

(B) The linear system $(I - C)^T y = 0$ has a solution y such that $y^T b \neq 0$

For only one of the above statements to be true, we must show $A \iff \neg B$. To do this, we must show that $A \rightarrow \neg B$ and $\neg B \rightarrow A$. To prove $A \rightarrow \neg B$, let us assume A is true and that $(I - C)^T y = 0$ for some y. Then we have:

$$\begin{aligned} (I - C)x &= b \\ y^T (I - C)x &= y^T b \\ 0 &= y^T b \end{aligned}$$

Since $y^T b = 0$, B cannot be true when A is true, implying that $A \rightarrow \neg B$. Now let us assume $\neg A$ and that $(I - C)^T y = 0$ for some y. Then we have:

$$\begin{aligned} (I - C)x &\neq b \\ y^T (I - C)x &\neq y^T b \\ 0 &\neq y^T b \end{aligned}$$

The above result implies $\neg A \rightarrow B$ since both $(I - C)^T y = 0$ and $y^T b \neq 0$ were satisfied. Since $\neg A \rightarrow B$ is the contrapositive of $\neg B \rightarrow A$, we have now shown $A \iff \neg B$. This proves that only A or B can be true for some low-rank matrix C.

References

[1] Gene H. Golub, Charles F. Van Loan. *Matrix Computations 4^{th} Edition.* The John Hopkins University Press, Baltimore, Maryland, 2013.

[2] Henrik V. Sorensen, C. Sidney Burrus. *Efficient Computation of the DFT with Only a Subset of Input or Output Points.* IEEE Transactions on Signal Processing, Volume: 41, Issue: 3, pgs. 1184 - 1200, Mar 1993.