Large Scale Matrix Computation, Optimization and Learning (EE270) HW #1

Winter 2021 Due: **Friday, January 29, 23:59pm**

- After a genuine attempt to solve the homework problems by yourself, you are free to collaborate with your fellow students to find solutions to the homework problems. Regardless of whether you collaborate with other students, you are required to type up or write your own solutions. Copying homework solutions from another student or from existing solutions is a serious violation of the honor code. Please take advantage of the professor's and TA's office hours. We are here to help you learn, and it never hurts to ask!
- **The assignments should be submitted via Gradescope including your code attached as pdf**

1. **Linear Algebra (15 pts)**

   Are the following statements **true** or **false**? If true, prove it; if false, show a counterexample.

   (a) The inverse of a symmetric matrix is itself symmetric.

   (b) All 2×2 orthogonal matrices have the following form

   $$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \text{ or } \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix}.$$

   (c) Let $A = \begin{bmatrix} -8 & -1 & -6 \\ -3 & -5 & -7 \\ -4 & -9 & -2 \end{bmatrix}$. A can be written as $A = CC^T$ for some matrix C.

SOLUTION:

(a) True, $I = (A^{-1}A)^\top = A^\top(A^\top)^{-1}$, thus $A^\top(A^{-1})^\top = A^\top(A^\top)^{-1}$ and $(A^{-1})^\top = (A^\top)^{-1}$.

(b) True. We note that $\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}$ is an orthogonal matrix if $a_1^2 + a_2^2 = b_1^2 + b_2^2 = 1$ and $[a_1\ a_2][b_1\ b_2]^T = 0$. Now, let θ denote $\theta = \tan^{-1}(a_2/a_1)$ so that $a_1 = \cos\theta$ and $a_2 = \sin\theta$. From the orthogonality property, i.e., $[a_1\ a_2][b_1\ b_2]^T = 0$, we have

$$b_1\cos\theta + b_2\sin\theta = 0 \implies \frac{b_1}{b_2} = \frac{-\sin\theta}{\cos\theta}$$

Now, let's use the normalization property, i.e.,$b_1^2 + b_2^2 = 1$,

$$b_2^2 = \frac{1}{1 + \frac{\sin\theta^2}{\cos\theta^2}} = \cos\theta^2$$

Therefore, $b_2 = \pm\cos\theta$ and $b_1 \mp \sin\theta$.

(c) False, A is not symmetric.

2. **Divide and Conquer Matrix Multiplication (15 pts)**

If A is a matrix, then $A^2 = AA$ is the square of A.

(a) Show that five multiplications are sufficient to compute the square of a 2×2 matrix $A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$, where a_1, a_2, a_3, a_4 are scalars.

(b) Generalize the formula in part (a) to a 2×2 block matrix $A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$ where A_1, A_2, A_3, A_4 are arbitrary matrices.

(c) Instead of using the classical matrix multiplication (three-loop) algorithm for computing A^2, we may apply the block formula you derived in (b) to reduce $2n \times 2n$ problems to several $n \times n$ computations, which can be tackled with classical matrix multiplication. Compare the total number of arithmetic operations. Generate $2n \times 2n$ random A matrices and plot the wall-clock time of the classical matrix multiplication algorithm and the algorithm using the formula in (b) to compute A^2 for $n = 4, ..., 10000$ (or as large as your system memory allows). You can use standard packages for matrix multiplication, e.g., numpy.matmul.

(d) Show that if you have an algorithm for squaring an $n \times n$ matrix in $O(n^c)$ time, then you can use it to multiply any two arbitrary $n \times n$ matrices in $O(n^c)$ time. [Hint: Consider multiplying two matrices A and B. Can you define a matrix whose square contains AB?]

SOLUTION:

(a)

$$AA = \begin{bmatrix} a_{11}^2 + a_{12}a_{21} & a_{12}(a_{11} + a_{22}) \\ a_{21}(a_{11} + a_{22}) & a_{21}a_{12} + a_{22}^2 \end{bmatrix}$$

Therefore, we only need to make five multiplications: $a_{11}^2, a_{22}^2, a_{12}a_{21}, a_{12}(a_{11} + a_{22})$, and $a_{21}(a_{11} + a_{22})$.

(b) Since matrix multiplication is not commutative, the approach in the previous part does not apply to this case. In general, we can use the Straussen's algorithm to compute

$$A^2 = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix}$$

with 7 multiplications as follows

$$M_1 = (A_1 + A_4)(A_1 + A_4),\ M_2 = (A_3 + A_4)A_1,\ M_3 = A_1(A_2 - A_4),\ M_4 = A_4(A_3 - A_1),$$
$$M_5 = (A_1 + A_2)A_4,\ M_6 = (A_3 - A_1)(A_1 + A_2),\ M_7 = (A_2 - A_4)(A_3 + A_4),$$

then

$$B_1 = M_1 + M_4 - M_5 + M_7,\ B_2 = M_3 + M_5,\ B_3 = M_2 + M_4,\ B_4 = M_1 - M_2 + M_3 + M_6.$$

(c) We observe that the method in (b) is slower than the classical matrix multiplication. We also note that relative error between the classical method and (b) is negligible.

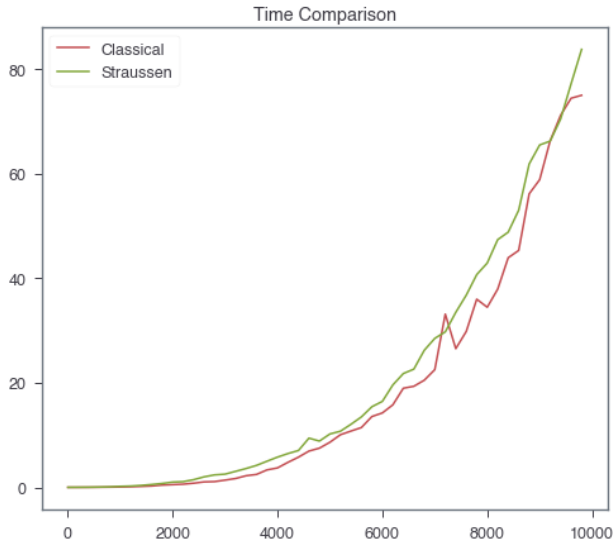

Figure 1: Time comparison

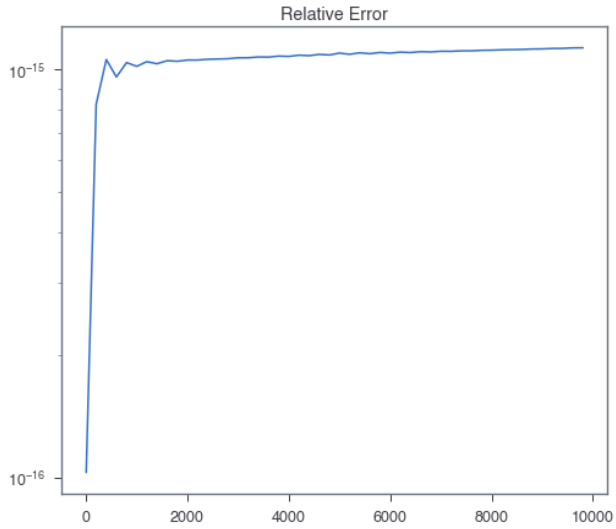

Figure 2: Relative error (log scale)

Code:

```
import numpy as np
import time
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size':18})


def mult_straussen(A):
    m,n=A.shape
    if not ((m==n) and m%2==0):
        raise Expcetion('A is not a 2n x 2n matrix')

    n_h=n//2
    A1=A[:n_h,:n_h]
    A2=A[:n_h,n_h:]
    A3=A[n_h:,:n_h]
    A4=A[n_h:,n_h:]

    M1=(A1+A4)@(A1+A4)
    M2=(A3+A4)@(A1)
    M3=(A1)@(A2-A4)
    M4=(A4)@(A3-A1)
    M5=(A1+A2)@(A4)
    M6=(A3-A1)@(A1+A2)
    M7=(A2-A4)@(A3+A4)

    B1=M1+M4-M5+M7
    B2=M3+M5
    B3=M2+M4
    B4=M1-M2+M3+M6

    return np.vstack((np.hstack((B1,B2)),np.hstack((B3,B4))))



ns=np.arange(50)*200
ns[0]=4
time_vec=np.zeros(len(ns))
time_vec_straussen=np.zeros(len(ns))

err=np.zeros(len(ns))

for (e,n) in enumerate(ns):
    print(n)
    A=np.random.randn(n,n)
    tic=time.process_time()
    A2_mul=A@A
    toc=time.process_time()
    time_vec[e]=toc -tic

```

```
        tic=time.process_time()
        A2_mul_straussen=mult_straussen(A)
        toc=time.process_time()

        time_vec_straussen[e]=toc -tic

        err[e]=np.linalg.norm(A2_mul-A2_mul_straussen)/(1+np.linalg.
            norm(A2_mul))


plt.figure()
plt.plot(ns,time_vec,'r',label='Classical')
plt.plot(ns,time_vec_straussen,'g',label='Straussen')
plt.legend()
plt.title('Time Comparison')


plt.figure()
plt.plot(ns,err,'b')
plt.yscale('log')
plt.title('Relative Error')
```

(d) Let us first define a new matrix as $D = \begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix} \in \mathbb{R}^{2n \times 2n}$. Then,

$$D^2 = \begin{bmatrix} AB & 0 \\ 0 & BA \end{bmatrix}.$$

Therefore, we can compute AB by squaring D in $O((2n)^c) = O(n^c)$ using the provided algorithm.

3. **Probability (30 pts)**

(a) Random variables X and Y have a joint distribution $p(x, y)$. Prove the following results. You can assume continuous distributions for simplicity.

i. $\mathbb{E}[X] = \mathbb{E}_Y[\mathbb{E}_X[X|Y]]$

ii. $\mathbb{E}[I[X \in \mathcal{C}]] = P(X \in \mathcal{C})$, where $I[X \in \mathcal{C}]$ is the indicator function[1] of an arbitrary set $\mathcal{C}$.

iii. $\text{var}[\text{X}] = \mathbb{E}_Y[\text{var}_X[X|Y]] + \text{var}_Y[\mathbb{E}_X[X|Y]]$

iv. If X and Y are independent, then $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$.

v. If X and Y take values in $\{0, 1\}$ and $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$, then X and Y are independent.

(b) Show that the approximate randomized counting algorithm described in Lemma 1 of Lecture 2 slides (page 14) is unbiased:

$$\mathbb{E}\tilde{n} = n\,. \tag{1}$$

(c) Prove the variance formula in Lemma 2 of Lecture 2 slides (page 38) for Approximate Matrix Multiplication $AB \approx CR$

$$\textbf{Var}\left[(CR)_{ij}\right] = \frac{1}{m}\sum_{k=1}^{d}\frac{A_{ik}^2 B_{kj}^2}{p_k} - \frac{1}{m}(AB)_{ij}^2\,. \tag{2}$$

[1] $I[X \in \mathcal{C}] := 1$ if $X \in \mathcal{C}$ and 0 otherwise

where $\{p_k\}_{k=1}^d$ are sampling probabilities.

SOLUTION:

(a) i. $\mathbb{E}[X] = \int\int xp(x,y)\mathrm{d}x\mathrm{d}y = \int(\int xp(x|y)\mathrm{d}x)p(y)\mathrm{d}y = \mathbb{E}_Y[\mathbb{E}_X[X|Y]]$

ii. $\mathbb{E}[I[X \in \mathcal{C}]] = \int_{x\in C} p(x)\mathrm{d}x = P(X \in \mathcal{C})$

iii.

$$\begin{aligned}
\mathrm{var}[X] &= \mathbb{E}[(X - \mathbb{E}[X])^2] \\
&= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \\
&= \mathbb{E}_Y[\mathbb{E}_X[X^2|Y]] - \mathbb{E}[X]^2 \\
&= \mathbb{E}_Y[\mathbb{E}_X[X^2|Y]] - (\int xp(x|y)\mathrm{d}x)^2 + (\int xp(x|y)\mathrm{d}x)^2) - \mathbb{E}[X]^2 \\
&= \mathbb{E}_Y[\mathbb{E}_X[X^2|Y]] - \mathbb{E}_X[X|Y]]^2] + \mathbb{E}_Y[\mathbb{E}_X[X|Y]]^2] - (\mathbb{E}_Y[\mathbb{E}_X[X|Y]])^2 \\
&= \mathbb{E}_Y[\mathrm{var}_X[X|Y]] + \mathrm{var}_Y[\mathbb{E}_X[X|Y]]
\end{aligned}$$

iv. $\mathbb{E}[XY] = \int\int xyp(x,y)\mathrm{d}x\mathrm{d}y = \int xp(x)\mathrm{d}x \int yp(y)\mathrm{d}y = \mathbb{E}[X]\mathbb{E}[Y]$

v. $P(X = 1, Y = 1) = \mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] = P(X = 1)P(Y = 1)$. Similar steps also apply to the other cases.

(b)Let all of the possible values for the counter X be $1, ..., k$. Then, using the law of total expectation, we have

$$\mathbb{E}[2^{X_n}] = \sum_{j=0}^{k} \mathbb{E}[2^{X_n}|X_{n-1} = j]P(X_{n-1} = j),$$

where the conditional expectations can be computed as

$$\begin{aligned}
\mathbb{E}[2^{X_n}|X_{n-1} = j] &= P(Increment)2^{j+1} + (1 - P(Increment))2^j \\
&= 2^{-j}2^{j+1} + (1 - 2^{-j})2^j = 1 + 2^j.
\end{aligned}$$

Plugging this result into the sum gives

$$\begin{aligned}
\mathbb{E}[2^{X_n}] &= \sum_{j=0}^{k} \mathbb{E}[2^{X_n}|X_{n-1} = j]P(X_{n-1} = j) \\
&= \sum_{j=0}^{k} (1 + 2^j)P(X_{n-1} = j) \\
&= \mathbb{E}[2^{X_{n-1}}] + 1
\end{aligned}$$

Thus, we have $\mathbb{E}[2^{X_n}] = \mathbb{E}[2^{X_{n-1}}] + 1$. Since $\mathbb{E}[2^{X_0}] = 1$ due to $X_0 = 0$, we have $\mathbb{E}[2^{X_n} - 1] = n + 1 - 1 = n$.

(c) For $t = 1, ..., m$ define $X_t = \left(A^{i_t}B_{i_t}\right)_{ij} = \frac{A_{ii_t}B_{i_tj}}{mp_{i_t}}$. Therefore,

$$\mathbb{E}[X_t] = \frac{1}{m}(AB)_{ij} \text{ and } \mathbb{E}[X_t^2] = \sum_{k=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{m^2 p_k}$$

Since by construction $(CR)_{ij} = \sum_{t=1}^m X_t$, we have $\mathbb{E}[(CR)_{ij}] = \sum_{t=1}^m \mathbb{E}[X_t] = (AB)_{ij}$. Additionally, since $(CR)_{ij}$ is the sum of m independent random variable, $\mathbf{Var}[(CR)_{ij}] = \sum_{t=1}^m \mathbf{Var}[X_t]$. Then,

$$\mathbf{Var}[(CR)_{ij}] = m\mathbf{Var}[X_t] = \sum_{k=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{mp_k} - \frac{1}{m}(AB)_{ij}^2$$

4. **Positive (Semi-)Definite Matrices (15 pts)**

Let A be a real, symmetric $d \times d$ matrix. We say A is *positive semi-definite* (PSD) if, for all $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{x}^\top A\mathbf{x} \geq 0$. We say A is *positive definite* (PD) if, for all $\mathbf{x} \neq 0$, $\mathbf{x}^\top A\mathbf{x} > 0$. We write $A \succeq 0$ when A is PSD, and $A \succ 0$ when A is PD.

The *spectral theorem* says that every real symmetric matrix A can be expressed $A = U\Lambda U^\top$, where U is a $d \times d$ matrix such that $UU^\top = U^\top U = I$ (called an orthogonal matrix), and $\Lambda = diag(\lambda_1, ..., \lambda_d)$. Multiplying on the right by U we see that $AU = U\Lambda$. If we let u_i denote the i^{th} column of U, we have $Au_i = \lambda_i u_i$ for each i. This expression reveals that the λ_i are eigenvalues of A, and the corresponding columns u_i are eigenvectors associated to λ_i.

(a) A is PSD iff $\lambda_i \geq 0$ for each i.

(b) A is PD iff $\lambda_i > 0$ for each i.

Hint: Use the following representation

$$U\Lambda U^T = \sum_{i=1}^{d} \lambda_i u_i {u_i}^T .$$

SOLUTION:

(a) First suppose A is positive semi-definite, then we have each $i = 1, ..., d$,

$$\lambda_i = \lambda_i \mathbf{u}_i^\top \mathbf{u}_i = \mathbf{u}_i^\top (\lambda_i \mathbf{u}_i) = \mathbf{u}_i^\top (A\mathbf{u}_i) \geq 0$$

where the last inequality follows from the positive semi-definite assumption.
Now suppose $\lambda_i \geq 0$ for each i, then we have for all $\mathbf{x} \in \mathbb{R}^d$,

$$\mathbf{x}^\top A\mathbf{x} = \mathbf{x}^\top (U\Lambda U^\top)\mathbf{x} = \mathbf{x}^\top (\sum_{i=1}^{d} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top)\mathbf{x} = \sum_{i=1}^{d} \lambda_i (\mathbf{x}^\top \mathbf{u})(\mathbf{u}_i^\top \mathbf{x}) = \sum_{i=1}^{d} \lambda_i \|\mathbf{u}_i^\top \mathbf{x}\|_2^2 \geq 0$$

Therefore, by definition A is positive semi-definite.

(b) The proof for the positive definite case is nearly identical to part (a). Note that to prove the converse implication, we must additionally assume that $\mathbf{x} \neq \mathbf{0}$.

5. **Norms (20 pts)**

(a) For $p = 1, 2, \infty$, verify that the functions $\|\cdot\|_p$ are norms. Then, for a vector $x \in \mathbb{R}^n$, show that

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2 \leq n\|x\|_\infty$$

and for each inequlaity, provide an example demonstrating that the inequality can be tight.

(b) For vectors x,$y \in \mathbb{R}^n$, show that $|x^T y| \leq \|x\|_2 \|y\|_2$ with equality if and only if x and y are linearly dependent. More generally, show that $x^T y \leq \|x\|_1 \|y\|_\infty$. Note that this implies that $\|x\|_2^2 \leq \|x\|_1 \|x\|_\infty$; and that these are special cases of Hölder's inequality.

(c) For $A \in \mathbb{R}^{m \times n}$, show that $\text{Trace}(A^T A) = \sum_{ij} A_{ij}^2$, and show that $\sqrt{\sum_{ij} A_{ij}^2}$ is a norm on $m \times n$ matrices. This is the Frobenius norm, denoted $\|\cdot\|_F$. Show that, in addition to

satisfying the definining properties of a norm, the Frobenius norm is a submultiplicative norm, in that

$$\|AB\|_F \leq \|A\|_F \|B\|_F$$

whenever the dimensions are such that the product AB is defined.

(d) Recall the definiton of the spectral norm of an $m \times n$ matrix A : $\|A\|_2 = \sqrt{\lambda_{max}(A^T A)} = \sigma_{max}(A)$, where $\lambda_{max}(A^T A)$ is the largest eigenvalue pf $A^T A$ and σ_{max} is the largest singular value of A. Show that the Frobenius norm and the spectral norm are unitarily invariant: if U and V are unitary (orthogonal in the real case) matrices, then $\|U^T A V\|_\xi = \|A\|_\xi$, for $\xi = 2, F$.

SOLUTION:

(a) The inequalities can be proven as follows

$$\|x\|_\infty = \max_i |x_i| \leq \sqrt{\sum_{i=1}^n x_i^2} = \|x\|_2 \quad \text{Example: } [1\ 0 \ldots\ 0]^T$$

$$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2} \leq \sum_{i=1}^n \sqrt{x_i^2} = \|x\|_1 \quad \text{Example: } [1\ 0 \ldots\ 0]^T$$

$$\|x\|_1 = \sum_{i=1}^n |x_i| 1 \leq \sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n 1} = \sqrt{n}\|x\|_2 \quad \text{Example: } [1\ 1 \ldots\ 1]^T$$

$$\sqrt{n}\|x\|_2 = \sqrt{n} \sqrt{\sum_{i=1}^n x_i^2} \leq n \max_i |x_i| = n\|x\|_\infty \quad \text{Example: } [1\ 1 \ldots\ 1]^T$$

(b) Suppose x and y are linearly dependent, i.e., $x = cy$, where $c \in \mathbb{R}$. Then,

$$|x^T y| = c\|y\|_2^2 = \|cy\|_2 \|y\|_2 = \|x\|_2 \|y\|_2$$

Now suppose that $|x^T y| = \|x\|_2 \|y\|_2$ but $x = cy$ does not hold, i.e., x and y are not linearly dependent. Then,

$$|x^T y| = \|x\|_2 \|y\|_2 \cos\theta = \|x\|_2 \|y\|_2 \implies \cos\theta = 1$$

which is a contradiction since x and y are not linearly dependent. Additionally, we have

$$\begin{aligned} |x^T y| &= \left| \sum_{i=1}^d x_i y_i \right| \\ &\leq \sum_{i=1}^d |x_i||y_i| \\ &\leq \max_i |y_i| \sum_{i=1}^d |x_i| = \|y\|_\infty \|x\|_1. \end{aligned}$$

(c)

$$\text{Trace}(A^T A) = \sum_{i=1}^n A_i^T A_i = \sum_{i=1}^n \sum_{j=1}^m A_{ij}^2.$$

Let $\|A\|_F^2 = \text{Trace}(A^T A)$, then we have $\|A\|_F^2 \geq 0$ with equality iff $A = 0$. Now, if we consider the vectorized version of A, $\|A\|_F^2$ becomes ℓ_2 norm and it obviously satisfies all properties of a norm. Finally, we prove the following

$$\begin{aligned}
\|AB\|_F &= \sum_{i,j} (A_i B^j)^2 \\
&\leq \sum_{i,j} \|A_i\|_2^2 \|B^j\|_2^2 \\
&\leq \sum_i \|A_i\|_2^2 \sum_j \|B^j\|_2^2 \\
&= \|A\|_F \|B\|_F.
\end{aligned}$$

(d) For the Frobenius norm, we have

$$\begin{aligned}
\|UAV\|_F^2 &= \text{Trace}((UAV)^T UAV) \\
&= \text{Trace}(V^T A^T AV) \\
&= \text{Trace}((VV^T A^T A) \\
&= \text{Trace}(A^T A) \\
&= \|A\|_F^2.
\end{aligned}$$

Similarly, for the spectral norm, we have

$$\begin{aligned}
\lambda_{max}((UAV)^T UAV) &= \lambda_{max}(V^T A^T AV) \\
&= \lambda_{max}(V^T QDQ^T V) \\
&= \lambda_{max}(V^T QDQ^T V) \\
&= \max_i D_{ii} \\
&= \|A\|_2,
\end{aligned}$$

where the SVD of A is defined as $A = MDQ^T$.

6. **Approximate Matrix Multiplication (20 pts)**

Here, we will consider the empirical performance of random sampling and random projection algorithms for approximating the product of two matrices. You may use Matlab, or C, or R, or any other software package you prefer to do your implementations. Please be sure to describe what you used in sufficient detail that someone else could reproduce your results. Let A be an $n \times d$ matrix, with $n \gg d$, and consider approximating the product $A^T A$. First, generate the matrices A from one of three different classes of distributions introduced below.

- Generate a matrix A from multivariate normal $N(1_d, \Sigma)$, where the (i, j)th element of $\Sigma_{ij} = 2 \times 0.5^{|i-j|}$.(Refer to as GA data.)
- Generate a matrix A from multivariate t-distribution with 3 degree of freedom and covariance matrix Σ as before. (Refer to as T3 data.)
- Generate a matrix A from multivariate t-distribution with 1 degree of freedom and covariance matrix Σ as before. (Refer to as T1 data.)

To start, consider matrices of size $n \times d$ equal to 500×50. (So, you should have three matrices, one matrix A generated in each of the above ways.)

(a) For each matrix, approximate the product $(A^T A)$ with the random sampling algorithm we discussed in class, i.e., by sampling with respect to a probability distribution that depends on the norm squared of the rows of the input matrix. Plot the probability distribution. Does it look uniform or nonuniform? Plot the performance of the spectral and Frobenius norm error as a function of the number of samples.

(b) For each matrix, approximate the product $(A^T A)$ with the random sampling algorithm we discussed in class, except that the uniform distribution, rather than the norm-squared distribution, should be used to construct the random sample. Plot the performance of the spectral and Frobenius norm error as a function of the number of samples. For which matrices are the results similar and for which are they different than when the norm-squared distribution is used ?

(c) Now you will implement the matrix approximation technique on the MNIST dataset for handwritten digit classification. Details about MNIST dataset can be found at `http://yann.lecun.com/exdb/mnist/`. We provide the dataset in **.mat** file so that you can easily import it into Matlab by using **load('mnist_matrix.mat')**. To import the dataset in Python you can use:
import scipy.io
data = scipy.io.loadmat('mnist_matrix.mat')
In **.mat** file you will find one matrix, $A \in \mathbb{R}^{60000 \times 784}$. For this matrix, approximate the product $(A^T A)$ with the random sampling algorithm we discussed in class, i.e., by sampling with respect to a probability distribution that depends on the norm squared of the rows of the input matrix. Plot the probability distribution. Dos it look uniform or nonuniform? Plot the performance of the spectral and Frobenius norm error as a function of the number of samples.

SOLUTION:

(a) The probability distribution looks like uniform only for the Gaussian dataset (GA).

(b) For GA, since the row norm probabilities are almost uniform, both uniform and norm based sampling perform similarly. However, for T1 and T3, these sampling methods yielded significantly different results.

(c) Even though the norm probabilit distribution of MNIST does not look as uniform as GA, it is still close to the uniform distribution compared with T3 and T1.

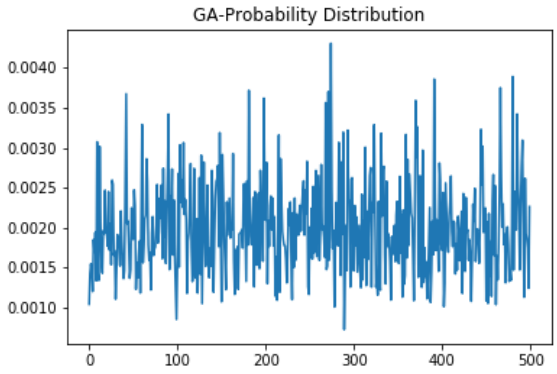

Figure 3: GA-Distribution

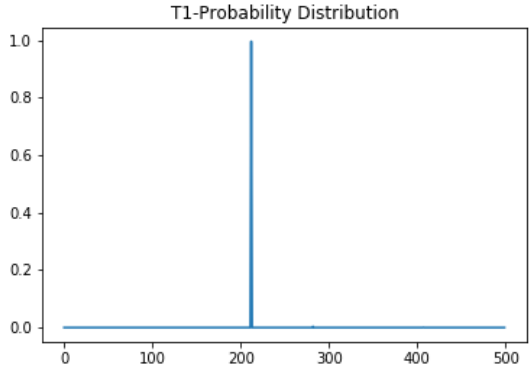

Figure 4: T1-Distribution

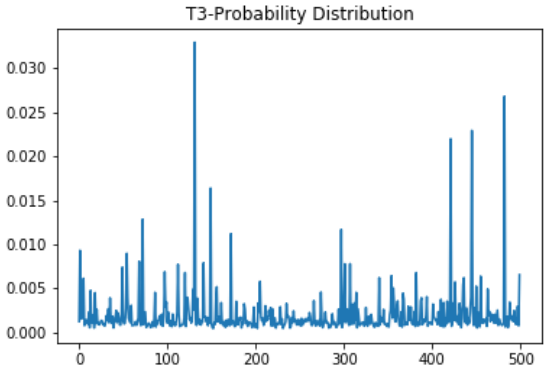

Figure 5: T3-Distribution

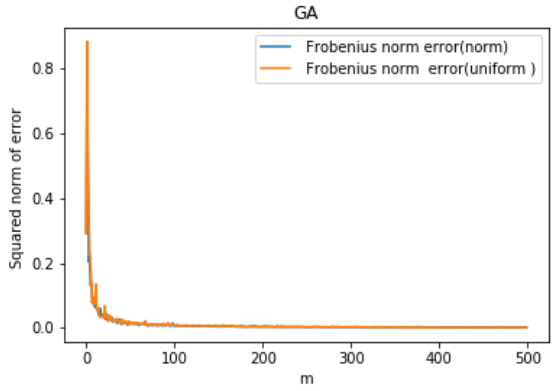

Figure 6: GA

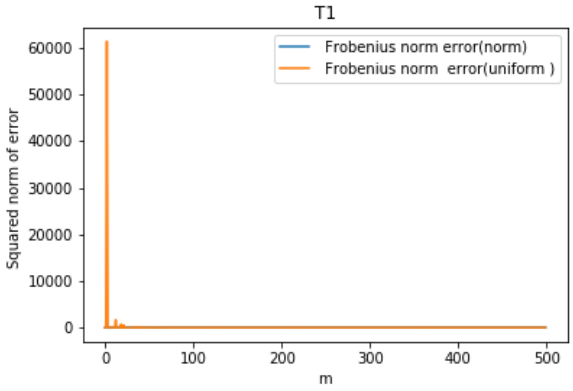

Figure 7: T1

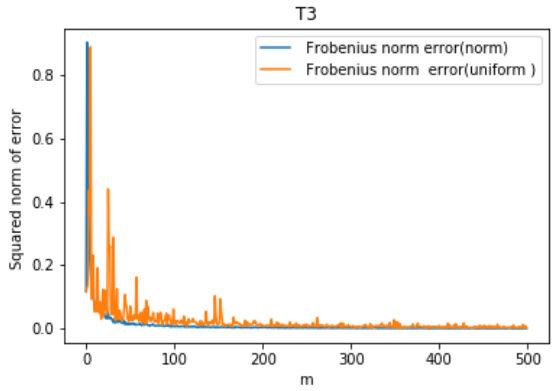

Figure 8: T3

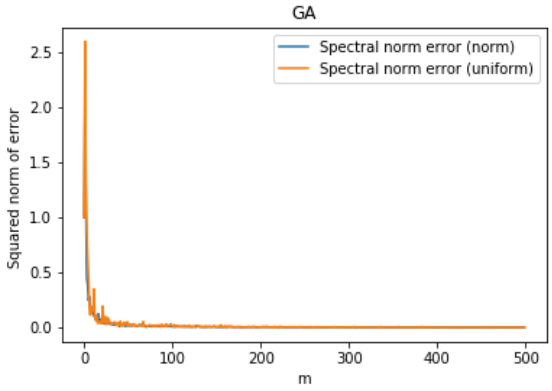

Figure 9: GA

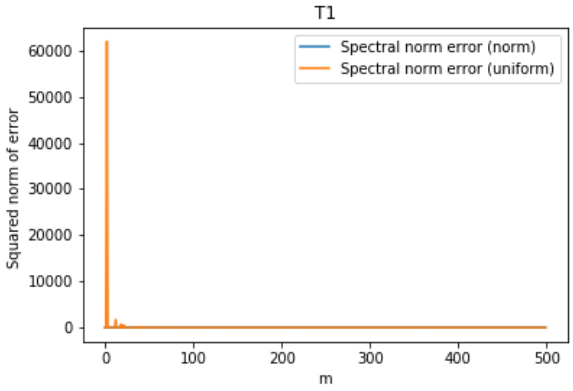

Figure 10: T1

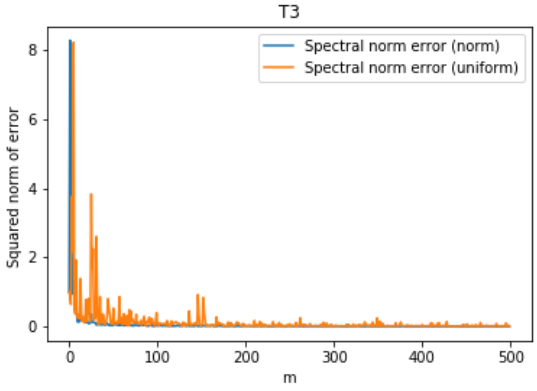

Figure 11: T3

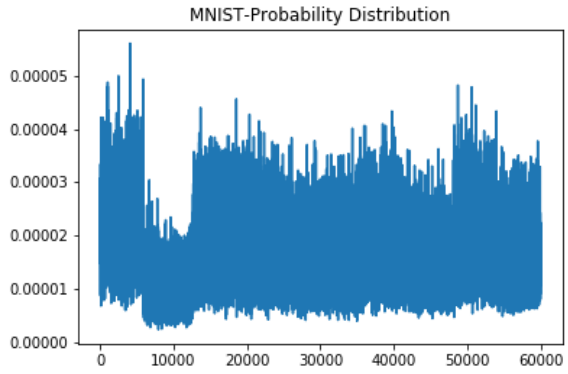

Figure 12: MNIST

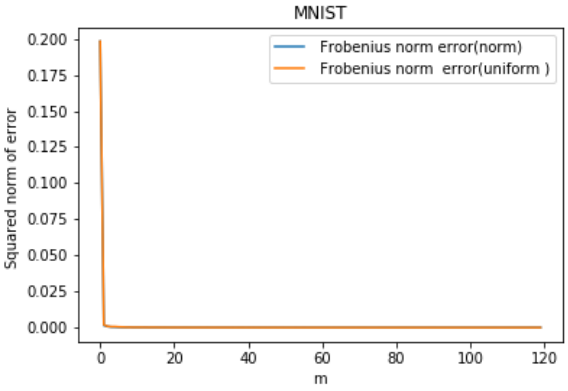

Figure 13: MNIST

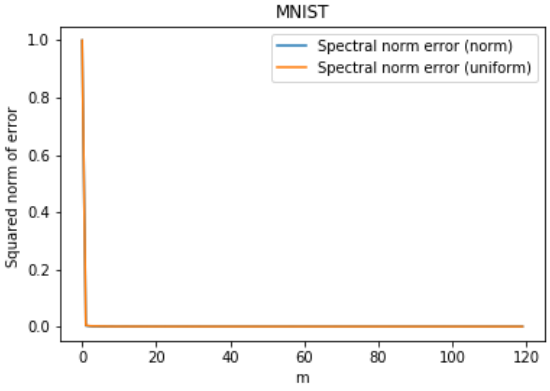

Figure 14: MNIST

Code:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
import scipy.io


def prob_dist(A) :
    norms = np.linalg.norm(A, axis =1)
    p =norms**2/np.sum(norms**2)
    return p

def unif_dist(A) :
    return np.ones(A.shape[0])/A.shape[0]


def t_dist(mu, sigma , df , n ):
    s = np.tile(st.chi2.rvs(df,size=n),(d,1)).T
    x = np.random . multivariate_normal(np.zeros(d ) ,sigma ,n)/np.
        sqrt(s/df)+np.tile(mu,(n,1))
    return x

def approx_mult(A,m,p,original ) :
    index=np.random.choice(np.arange(A.shape[0]),size=m,p=p)
    SA=A[index,:]/np.sqrt(np.tile(p[index],(A.shape[1],1)).T      *m)
    approximate=(SA.T)@SA
    s_error=np.linalg.norm(approximate-original,ord=2)**2/np.linalg
        .norm(A,ord=2)**4
    f_error= np.linalg.norm(approximate-original,ord='fro')**2 /np.
        linalg.norm(A,ord='fro')**4
    return s_error,f_error




n = 500
d = 50

I, J=np.meshgrid(np.arange(d),np.arange(d),indexing='ij')
sigma =0.5**(np.abs(I-J))

T3 = t_dist(np.ones(d),sigma,3,n)
T1 = t_dist(np.ones(d),sigma,1,n)
GA=np.random.multivariate_normal(np.ones(d),sigma,n)
A_mnist =scipy.io.loadmat('mnist_matrix.mat')['A']
dataset="MNIST"
A=A_mnist#
original =(A.T)@A

dist=prob_dist(A)
```

```
47 plt.figure()
48 plt.title ( dataset+"-Probability Distribution" )
49 plt.plot(dist)
50 plt.savefig(dataset+'_dist.png')
51 plt.show()
52
53 s_normsq =[]
54 f_normsq =[]
55 s_uniform=[]
56 f_uniform=[]
57
58 for m in range(0,A.shape[0],500):
59     se_n,fe_n=approx_mult(A,m,prob_dist(A),original)
60     se_u, fe_u=approx_mult(A,m,unif_dist(A),original)
61
62     s_normsq.append(se_n)
63     f_normsq.append(fe_n)
64     s_uniform.append(se_u)
65     f_uniform.append(fe_u)
66
67 plt.figure( )
68 plt.title( dataset)
69 plt.xlabel("m")
70 plt.ylabel( "Squared norm of error " )
71 plt.plot(s_normsq,label="Spectral norm error (norm) ")
72 plt.plot(s_uniform,label="Spectral norm error (uniform)")
73 plt.legend( )
74 plt.savefig( dataset+ 's_error.png')
75 plt.show( )
76
77 plt.figure( )
78 plt.title(dataset)
79 plt.xlabel ( "m" )
80 plt.ylabel( "Squared norm of error " )
81 plt.plot(f_normsq,label=" Frobenius norm error(norm)" )
82 plt.plot(f_uniform,label=" Frobenius norm  error(uniform )")
83 plt.legend( )
84 plt.savefig(dataset+ 'f_error.png' )
85 plt.show()
```