EE546 HW3

1. Randomized matrix multiplication:

Solve: (i) Compute the spectral norms of the matrix $B^{m\times N}$ and $C^{N\times n}$ and scale them so that their spectral norms are equal to one (The maximum singular value of a matrix)

$$\|B\| = \|C\| = 1$$

The stable rank: $\text{srank}(B) = \frac{\|B\|_F^2}{\|B\|^2}$ (The Frobeninus Norm of a matrix is defined as the square root of the sum of the squares of the elements of the matrix)

$$\text{srank}(C) = \frac{\|C\|_F^2}{\|C\|}$$

Let $\text{asr} = \frac{1}{2}(\text{srank}(B) + \text{srank}(C))$

$$\therefore \frac{E[\|\hat{A} - A\|]}{\|B\| \cdot \|C\|} \xrightarrow{\text{after scaling } B \text{ and } C} E[\|\hat{A} - A\|] \le \sqrt{\frac{4 \cdot \text{asr} \cdot \log(m+n)}{r}} + \frac{2 \cdot \text{asr} \cdot \log(m+n)}{3r}$$

(ii) $p_i = \frac{\|b_i\|^2 + \|C_i^T\|^2}{\|B\|_F^2 + \|C\|_F^2}$ for $i = 1, 2, 3, \cdots, N$, where $b_i$ are the columns of $B$ and $C_i^T$ are the columns of $C^T$, ($C_i$ are the rows of $C$)

(iii) Let $r \ge r_0^{-2} \cdot \text{asr} \cdot \log(m+n)$

since $\frac{E[\|\hat{A} - A\|]}{\|B\| \cdot \|C\|} \le \varepsilon$

$$\therefore 2r_0^2 + \frac{2}{3}r_0^2 = \varepsilon$$

$$2r_0^2 + 6r_0 - 3\varepsilon = 0$$

$$\therefore r_0 = \frac{-3 \pm \sqrt{9+6\varepsilon}}{2} \Rightarrow r \ge \frac{4 \cdot \text{asr} \cdot \log(m+n)}{(-3+\sqrt{9+6\varepsilon})^2}$$

(iv) The computational cost is $O\left(4(-3+\sqrt{9+6\varepsilon})^{-2} \cdot \text{asr} \cdot mn \cdot \log(m+n)\right)$

**Problem2: Randomized SVD**

First, draw $X \in R^{m\times m}$ and $Y \in R^{n\times m}$ with $m \geq n$ uniformly at random from the space of orthonormal matrices with $m = 1000$ and $n = 100000$. I use Python to implement this step as shown below:

```
m, n = 1000, 100000
# np.random.seed(seed=546)
X = special_ortho_group.rvs(m)
print("X.shape", X.shape)
Y = np.random.random(size=(n, m))  # Generate a random matrix (n, m)
Y, _ = np.linalg.qr(Y)  # Perform a QR decomposition.
print("Y.shape", Y.shape)
```

Then, set the diagonal matrix $D \in R^{m\times m}$ with diagonal entries

$$D_{ii} = \begin{cases} r - i + 1, & i \leq r \\ 4 \times 10^{-3} & i > r \end{cases}$$

with $r = 10$. The Python code is:

```
def get_D(m, r):
    """
       Get the required diagonal matrix D.
    Inputs:
        the size
        the value of r
    Outputs:
        The diagonal matrix
    """
    D = np.zeros((m, m), dtype="float")
    for i in range(m):
        if i + 1 <= r:
            D[i, i] = r - (i + 1) + 1
        else:
            D[i, i] = 4e-3

    return D
```

Next, use the randomized SVD of Drineas et al. to calculate the r = 10 top eigenvectors of $A$. The algorithm is

*Input: Data matrix* $A \in R^{m\times n}$

*Output: Approximate for left eigenspace* $H \in R^{m\times r}$

*For* $t \in \{1, 2, \dots, c\}$

*Pick one column among* $\{1, 2, \dots, n\}$ *at random with the probability distribution* $\{P_1, P_2, \dots, P_n\}$, *where* $P_i = \frac{\|A_i^2\|}{\|A\|_F^2}$.

*The column of B:* $B_t = \frac{A_i}{\sqrt{c\times P_i}}$.

*End for*

*Compute top r left singular vectors of B:* $h_1, h_2, \dots, h_r$

*Return* $H = [\, h_1, h_2, \dots, h_r]$

The Python code is:

```
def generalized_power(A, row_value, r):
    """
```

```
        Utilize the generalized power method to find the A_U_r and A_V_r.
    Inputs:
        the input data matrix
        the value of the row
        the number of top eigenvectors
    Outputs:
        the objective eigen-matirx
    """
    Q = np.random.random(size=(row_value, r))  # Generate a random matrix
(row_value, r)
    Q, _ = np.linalg.qr(Q)  # Perform a QR decomposition.
    for _ in range(10):
        U = np.dot(A, np.dot(A.T, Q))
        Q, _ = np.linalg.qr(U)

    return Q


def simple_power(A_r, B_r, row_value):
    """
        Utilize the power method to find the spectral norm.
    Inputs:
        the real top r eigen-matrix
        the estimated top r eigen-matrix
        the value of the row
    Outputs:
        the spectrum norm
    """
    x = np.ones((row_value, 1), dtype='float')

    for _ in range(10):
        x = dot(A_r, dot(A_r.T, dot(A_r, dot(A_r.T, x)))) \
            - dot(A_r, dot(A_r.T, dot(B_r, dot(B_r.T, x)))) \
            - dot(B_r, dot(B_r.T, dot(A_r, dot(A_r.T, x)))) \
            + dot(B_r, dot(B_r.T, dot(B_r, dot(B_r.T, x))))

    x = x / np.max(x)

    numer_value = dot(x.T, dot(A_r, dot(A_r.T, dot(A_r, dot(A_r.T, x))))) \
                  - dot(x.T, dot(A_r, dot(A_r.T, dot(B_r, dot(B_r.T, x))))) \
                  - dot(x.T, dot(B_r, dot(B_r.T, dot(A_r, dot(A_r.T, x))))) \
                  + dot(x.T, dot(B_r, dot(B_r.T, dot(B_r, dot(B_r.T, x)))))
    max_eig_value = fabs(numer_value) / dot(x.T, x)

    spec_norm = sqrt(np.squeeze(max_eig_value))

    return spec_norm


class hw3_randomizedSVD(object):
    def __init__(self, X, Y, m, n):
        self.X = X    # The matrix X
```

```
        self.Y = Y    # The matrix Y
        self.m = m    # The value of m
        self.n = n    # The value of n

    def check_c(self, r, prob):
        """
            Check how large c has to be to satisfy specifications.
        :param r: The number of top eigenvalues
        :param prob: The ith problem
        :return: The values of c
        """
        D = get_D(m=self.m, r=r)
        A = dot(dot(self.X, D), self.Y.T)  # Generate the matrix A: (m, n)
        A_i2_1, A_i2_2 = sum(square(A), axis=0), sum(square(A.T), axis=0)
        A_F2_1, A_F2_2 = sum(square(A)), sum(square(A.T))
        col_pdf1 = A_i2_1 / A_F2_1
        col_pdf2 = A_i2_2 / A_F2_2
        c = 1
        verbose1, verbose2 = True, True
        while True:
            select_cols1 = np.random.choice(a=self.n, size=c, p=col_pdf1)  # For
U
            select_cols2 = np.random.choice(a=self.m, size=c, p=col_pdf2)  # For
V
            B1, B2 = 0, 0
            for t in range(c):
                col_index1 = select_cols1[t]
                B_t1 = expand_dims(A[:, col_index1], axis=1) / sqrt(c *
col_pdf1[col_index1])
                col_index2 = select_cols2[t]
                B_t2 = expand_dims(A.T[:, col_index2], axis=1) / sqrt(c *
col_pdf2[col_index2])
                if t == 0:
                    B1 = B_t1
                    B2 = B_t2
                else:
                    B1 = np.hstack((B1, B_t1))
                    B2 = np.hstack((B2, B_t2))

            B_U_r = generalized_power(A=B1, row_value=self.m, r=r)
            B_V_r = generalized_power(A=B2, row_value=self.n, r=r)

            A_U_r = generalized_power(A=A, row_value=self.m, r=r)
            A_V_r = generalized_power(A=A.T, row_value=self.n, r=r)

            # Utilize the power method to find the spectral norm
            U_spec_norm = simple_power(A_r=A_U_r, B_r=B_U_r, row_value=self.m)
            # print("U_spec_norm", U_spec_norm)
            V_spec_norm = simple_power(A_r=A_V_r, B_r=B_V_r, row_value=self.n)
            # print("V_spec_norm", V_spec_norm)
            if prob == "prob_i":
                if U_spec_norm <= 0.1 and V_spec_norm <= 0.1:
```

```
                    if verbose1:
                        print("When epsilon is 0.1, the c is %d, " % (c))
                        verbose1 = False
                if U_spec_norm <= 0.05 and V_spec_norm <= 0.05:
                    if verbose2:
                        print("When epsilon is 0.05, the c is %d" % (c))
                        verbose2 = False
                if U_spec_norm <= 0.01 and V_spec_norm <= 0.01:
                    print("When epsilon is 0.01, the c is %d" % (c))
                    return True

            elif prob == "prob_ii":
                if U_spec_norm <= 0.05 and V_spec_norm <= 0.05:
                    print("When r is %d, the c is %d" % (r, c))
                    return True

            else:
                print("Please input the correct problem index!")
                return False

            # print("The c=%d is finished!" % (c))
            c += 1

    def check_time(self, c, r, epsilon):
        """
            Check the running time with specific c, r, and epsilon.
        :param c: The number of drawn columns
        :param r: The number of top eigenvalues
        :param epsilon: The value of epsilon
        :return: The running time
        """
        D = get_D(m=self.m, r=r)
        A = dot(dot(self.X, D), self.Y.T)  # Generate the matrix A: (m, n)

        t0 = time.time()
        A_i2_1, A_i2_2 = sum(square(A), axis=0), sum(square(A.T), axis=0)
        A_F2_1, A_F2_2 = sum(square(A)), sum(square(A.T))
        col_pdf1 = A_i2_1 / A_F2_1
        col_pdf2 = A_i2_2 / A_F2_2
        select_cols1 = np.random.choice(a=self.n, size=c, p=col_pdf1)  # For U
        select_cols2 = np.random.choice(a=self.m, size=c, p=col_pdf2)  # For V
        B1, B2 = 0, 0
        for t in range(c):
            col_index1 = select_cols1[t]
            B_t1 = expand_dims(A[:, col_index1], axis=1) / sqrt(c *
col_pdf1[col_index1])
            col_index2 = select_cols2[t]
            B_t2 = expand_dims(A.T[:, col_index2], axis=1) / sqrt(c *
col_pdf2[col_index2])
            if t == 0:
                B1 = B_t1
                B2 = B_t2
```

```
            else:
                B1 = np.hstack((B1, B_t1))
                B2 = np.hstack((B2, B_t2))

        B_U_r = generalized_power(A=B1, row_value=self.m, r=r)
        B_V_r = generalized_power(A=B2, row_value=self.n, r=r)

        A_U_r = generalized_power(A=A, row_value=self.m, r=r)
        A_V_r = generalized_power(A=A.T, row_value=self.n, r=r)

        # Utilize the power method to find the spectral norm
        _ = simple_power(A_r=A_U_r, B_r=B_U_r, row_value=self.m)
        _ = simple_power(A_r=A_V_r, B_r=B_V_r, row_value=self.n)

        t = time.time() - t0
        print("When c is %d, r is %d, and epsilon is %f, the running time
is %f" % (c, r, epsilon, t))
```

(i) In this part, I am about to report the value of c (number of drawn columns / rows) based on the average of 10 random draws from the columns of A as shown in the Table 1, which means drawing c columns from the matrix A 10 independent times when the epsilon is 0.01, 0.05, and 0.1 respectively.

$$\frac{\|\hat{U}_r\hat{U}_r^T - U_rU_r^T\|}{\|U_rU_r^T\|} \leq \epsilon \; and \; \frac{\|\hat{V}_r\hat{V}_r^T - V_rV_r^T\|}{\|V_rV_r^T\|} \leq \epsilon$$

Table 1: the value of c among 10 independent trials when r=10

| Trial | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Epsilon=0.1 | 13 | 13 | 12 | 12 | 13 | 12 | 13 | 15 | 12 | 13 | 13 |
| Epsilon=0.05 | 19 | 18 | 18 | 19 | 21 | 15 | 18 | 16 | 19 | 16 | 18 |
| Epsilon=0.01 | 192 | 189 | 191 | 201 | 201 | 185 | 203 | 187 | 205 | 190 | 195 |

Table 2: the running time (seconds) with a specific c (mean value) when r=10

| Trial | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Epsilon=0.1 | 4.57 | 4.39 | 4.44 | 4.34 | 4.41 | 4.48 | 4.56 | 4.71 | 4.57 | 4.74 | 4.52 |
| Epsilon=0.05 | 4.45 | 4.56 | 4.52 | 4.51 | 4.43 | 4.47 | 4.54 | 4.52 | 4.55 | 4.65 | 4.52 |
| Epsilon=0.01 | 10.98 | 11.05 | 10.93 | 10.97 | 10.98 | 11.02 | 10.88 | 11.05 | 11.15 | 11.21 | 11.02 |

Therefore, the value of c among 10 independent trials when r=10 is shown in the Table 1. Then I set the c as 13, 18, and 195 respectively to get the running time as shown in the Table 2. I will use the randomized SVD algorithm since it can avoid the memory error when matrices are too large and the final error is acceptable. Also, using the built-in function "*sp.linalg.svd(a=A, full_matrices=False, lapack_driver='gesvd')*" will takes about 10 seconds, which is a bit longer when we only want epsilon to be around 0.05. Finally, when I calculate the spectral norm of $\hat{U}_r\hat{U}_r^T - U_rU_r^T$, which is its largest singular value, I should use the power method to calculate the dominant eigenvalue of $(\hat{U}_r\hat{U}_r^T - U_rU_r^T)^2$.

(ii) In this part, I am about to repeat the experiment of the previous part with r = 2, 5, 15, and 20. In these experiments. I will use epsilon = 0.05 and calculate the minimal number of required columns c to get to a relative error over the 10 independent trials as shown in the Table 3. The value of c as a function of r is shown in the Fig. 1, which has a positive correlation relationship consistent with the theorem in class since c is $O(r/\epsilon^2)$.

Table 3: the value of c among 10 independent trials when epsilon=0.05

| Trial | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| r=2 | 9 | 9 | 9 | 8 | 10 | 10 | 8 | 8 | 15 | 8 | 10 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| r=5 | 12 | 12 | 14 | 16 | 13 | 11 | 14 | 11 | 16 | 14 | 14 |
| r=15 | 22 | 20 | 23 | 22 | 22 | 22 | 22 | 22 | 24 | 25 | 23 |
| r=20 | 26 | 28 | 30 | 29 | 26 | 26 | 28 | 24 | 26 | 28 | 28 |

Table 4: the running time (seconds) with a specific c among 10 independent trials when epsilon=0.05

| Trial | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Mean |
|---|---|---|---|---|---|---|---|---|---|---|---|
| r=2 | 3.44 | 3.44 | 3.45 | 3.51 | 3.54 | 3.51 | 3.47 | 3.55 | 3.49 | 3.45 | 3.48 |
| r=5 | 3.87 | 3.73 | 3.72 | 3.83 | 3.82 | 3.82 | 3.78 | 3.89 | 3.85 | 3.81 | 3.81 |
| r=15 | 5.34 | 5.33 | 5.39 | 5.35 | 5.51 | 5.64 | 5.71 | 5.55 | 5.56 | 5.39 | 5.47 |
| r=20 | 6.01 | 6.08 | 5.98 | 6.03 | 6.13 | 5.99 | 6.55 | 6.16 | 6.19 | 6.04 | 6.11 |

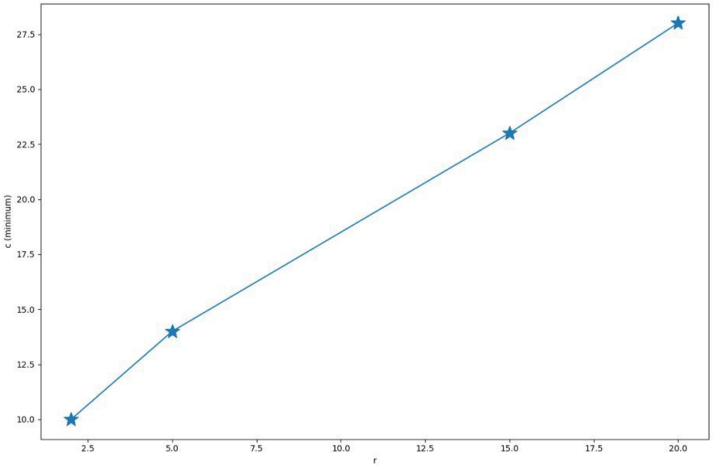


Figure 1: The value of c as a function of r