Homework Set 2, CPSC 8420, Fall 2023

Parampreet Singh

Due 10/26/2023, Thursday, 11:59PM EST

Problem 1

For PCA, from the perspective of maximizing variance, please show that the solution of $\boldsymbol{\phi}$ to maximize $\|\mathbf{X}\boldsymbol{\phi}\|_2^2$, $s.t.$ $\|\boldsymbol{\phi}\|_2 = 1$ is exactly the first column of $\mathbf{U}$, where $[\mathbf{U}, \mathbf{S}] = svd(\mathbf{X}^T\mathbf{X})$. (Note: you need prove why it is optimal than any other reasonable combinations of $\mathbf{U}_i$, say $\hat{\boldsymbol{\phi}} = 0.8 * \mathbf{U}(:,1) + 0.6 * \mathbf{U}(:,2)$ which also satisfies $\|\hat{\boldsymbol{\phi}}\|_2 = 1$.)

Problem 2

Given matrix $\mathbf{X} \in \mathbb{R}^{n\times p}$ (assume each column is centered already), where n denotes sample size while p feature size. To conduct PCA, we need find eigenvectors to the largest eigenvalues of $\mathbf{X}^T\mathbf{X}$, where usually the complexity is $\mathcal{O}(p^3)$. Apparently when $n \ll p$, this is not economic when p is large. Please consider conducting PCA based on $\mathbf{X}\mathbf{X}^T$ and obtain the eigenvectors for $\mathbf{X}^T\mathbf{X}$ accordingly and use experiment to demonstrate the acceleration.

Problem 3

Let's revisit Least Squares Problem: $\underset{\boldsymbol{\beta}}{\text{minimize}}\ \frac{1}{2}\|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2$, where $\mathbf{A} \in \mathbb{R}^{n \times p}$.

1. Please show that if $p > n$, then vanilla solution $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$ is not applicable any more.
2. Let's assume $\mathbf{A} = [1, 2, 4; 1, 3, 5; 1, 7, 7; 1, 8, 9], \mathbf{y} = [1; 2; 3; 4]$. Please show via experiment results that Gradient Descent method will obtain the optimal solution with Linear Convergence rate if the learning rate is fixed to be $\frac{1}{\sigma_{max}(\mathbf{A}^T\mathbf{A})}$, and $\boldsymbol{\beta}_0 = [0; 0; 0]$.
3. Now let's consider ridge regression: $\underset{\boldsymbol{\beta}}{\text{minimize}}\ \frac{1}{2}\|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2$, where $\mathbf{A}, \mathbf{y}, \boldsymbol{\beta}_0$ remains the same as above while learning rate is fixed to be $\frac{1}{\lambda + \sigma_{max}(\mathbf{A}^T\mathbf{A})}$ where λ varies from $0.1, 1, 10, 100, 200$, please show that Gradient Descent method with larger λ converges faster.

Problem 4

We consider matrix completion problem. As we discussed in class, the main issue of *softImpute (Matrix Completion via Iterative Soft-Thresholded SVD)* is when the matrix size is large, conducting *SVD* is computational demanding. Let's recall the original problem where $\mathbf{X}, \mathbf{Z} \in \mathbb{R}^{n \times d}$:

$$\min_{\mathbf{Z}} \frac{1}{2}\|P_\Omega(\mathbf{X}) - P_\Omega(\mathbf{Z})\|_F^2 + \lambda\|\mathbf{Z}\|_* \tag{1}$$

People have found that instead of finding optimal $\mathbf{Z}$, it might be better to make use of *Burer-Monteiro* method to optimize two matrices $\mathbf{A} \in \mathbb{R}^{n \times r}, \mathbf{B} \in \mathbb{R}^{d \times r} (r \geq rank(\mathbf{Z}^*))$ such that $\mathbf{A}\mathbf{B}^T = \mathbf{Z}$. The new objective is:

$$\min_{\mathbf{A},\mathbf{B}} \frac{1}{2}\|P_\Omega(\mathbf{X} - \mathbf{A}\mathbf{B}^T)\|_F^2 + \frac{\lambda}{2}(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2). \tag{2}$$

- Assume $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] = svd(\mathbf{Z})$, show that if $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}, \mathbf{B} = \mathbf{V}\mathbf{\Sigma}^{\frac{1}{2}}$, then Eq. (2) is equivalent to Eq. (1).
- The *Burer-Monteiro* method suggests if we can find $\mathbf{A}^*, \mathbf{B}^*$, then the optimal $\mathbf{Z}$ to Eq. (1) can be recovered by $\mathbf{A}^*\mathbf{B}^{*T}$. It boils down to solve Eq. (2). Show that we can make use of least squares with ridge regression to update $\mathbf{A}, \mathbf{B}$ row by row in an alternating minimization manner as below. Assume $n = d = 2000, r = 200$, please write program to find $\mathbf{Z}^*$.

```
$T \leftarrow 100, i \leftarrow 1$ % you can also set T to be other number instead of 100
if $i \leq T$ then
    update A row by row while fixing B
    update B row by row while fixing A
    $i \leftarrow i + 1$
end if
```

Ans 1.

For matrix with self centered data, max. variance is given by

$$\max \|X\phi\|_2^2 \quad \text{——} \quad ①$$

such that $\|\phi\|_2 = 1$ and ϕ is a vector

To prove,

Complete closed solution of ① is exactly equal to the first column of U

where $(U, S) = \text{svd}(X^T X)$

Modify $\|X\phi\|_2^2$ as

$$[X\phi]^T [X\phi] = \phi^T X^T X \phi$$

As $X^T X$ is a spd matrix,
for SVD of $X^T X$, we have

$$V = U \quad \text{in } U \Sigma V^T$$

Also, singular values are absolute values of eigen values

Also, columns of $U = V$ are eigen vectors of A

Consider:

Case I - when $\phi = aU_1 + bV_2$
where $a, b \in R$
s.t $a^2 + b^2 = 1$

So,

$$||\phi||^2 = \phi^T . \phi$$
$$= (aU_1 + bU_2)^T (aU_1 + bU_2)$$
$$= a^2 U_1^T U + b^2 U_2^T U_2 + 2ab\, U_1^T V_2$$

$(\because U_1 \perp U_2,$
$U_1^T U_1 = I$
$U_2^T U_2 = I$

So,

$$||\phi||^2 = a^2 + b^2 = 1$$
s.t $a^2 + b^2 = 1$
satisfy the given constraint

If we substitute the value of ϕ in
$\phi^T X^T X \phi$

$$= \phi^T [U \Sigma U^T] \phi$$
$$= \phi^T [\sigma_1 U_1 U_1^T + \sigma_2 U_2 U_2^T + \sigma_3 U_3 U_3^T + \cdots] \phi$$

Scanned with CamScanner

$$= (av_1 + bv_2)^T [\sigma_1 v_1 v_1^T + \sigma_2 v_2 v_2^T + \sigma_3 v_3 v_3^T + \cdots] (av_1 + bv_2)$$

$$= (a v_1^T \sigma_1 v_1 v_1^T + b v_2^T \sigma_2 v_2 v_2^T + \cdots)(av_1 + bv_2)$$

$$= (a^2 \sigma_1 v_1^T v_1 v_1^T v_1 + b^2 \sigma_2 v_2^T v_2 v_2 v_2^T v_2 + \cdots)$$

As $v_1 \perp v_2$

$$v_1^T \cdot v_2 = v_2^T \cdot v_1 = 0$$

So, we are left with

$$a^2 \sigma_1 + b^2 \sigma_2$$

$$\sigma_1 \left(a^2 + b^2 \frac{\sigma_2}{\sigma_1} \right) \quad — \quad ①$$

As we know that the largest singular value is σ_1 & it is always greater than σ_2

and, $a^2 + b^2 = 1$

therefore, we have

$$a^2 + b^2\left(\frac{\sigma_2}{\sigma_1}\right) < 1$$

$$\sigma_1\left(a^2 + b^2\left(\frac{\sigma_2}{\sigma_1}\right)\right) < \sigma_1 \quad -\boxed{2}$$

<u>Case II</u> consider if $\phi = v_1$

$\|\phi\|_2^2 = 1$ ($\because$ orthogonal vector)

Put $\phi = v_1$ in

$$\phi^T [v \Sigma v^T] \phi$$

$$\phi^T [\sigma_1 v_1 v_1^T + \sigma_2 v_2 v_2^T + \sigma_3 v_3 v_3^T + \cdots] \phi$$

$$= \sigma_1 v_1^T v_1 v_1^T v_1 + \sigma_2 v_2^T v_2 v_2^T v_2 + \cdots$$

As all columns of v are orthogonal,

we have,

$$~~\sigma_1 \alpha_1^T~~ \ \sigma_1 v_1^T v_1 v_1^T v_1$$

Also $v_1^T v_1 = I$

$\therefore \max \|X\phi\|_2^2 = \sigma_1$

Comparing $w = v$ as an argument to a general combination of vectors from v,

eq. 3 gives :

$$\underset{\|\phi\|_2^2 = 1}{\arg\max} \ \|X\phi\|_2^2 \Rightarrow w = av_1 + bv_2$$
$$(s.t \ a, b \in R \ \& \ a^2 + b^2 = 1)$$

gives $\sigma_1 \left(a^2 + b^2 \frac{\sigma_2}{\sigma_1} \right)$

which is less than σ_1 when $w = v_1$

So, max variance is achieved when ϕ = first column of v, where

$v = [v, s] = SVD(X^T X)$

Scanned with CamScanner

Ans 2:

Let $A = X^T X$
$B = X X^T$

Let U be eigen vectors of B

$$X X^T U = \lambda U$$

multiply both sides by X^T

$$X^T X X^T U = \lambda X^T U$$

from ①, $X^T X = A$

$$A(X^T U) = \lambda (X^T U)$$

So, we can conclude that $X^T U$ are eigen vectors of A.

So, we can find eigen vectors of B, which have the computational complexity of $O(N^3)$ instead of $O(p^3)$, when $X \in R^{N \times P}$

These eigen vectors can be used to find eigen vectors of $A(X^T X)$

To convert eigen vectors of XX^T to those of X^TX

→ Multiply eigen vectors of $B(XX^T)$ by X^T and then normalize

i.e eigen vectors of $A: X^TU$
But X^TU is not normalized. So, the matrix which gives us eigen vectors of A is X^TU with all of its column vectors normalized.

Scanned with CamScanner

Problem 2

```
% Experimental Setup
samples = 100;
features = 10000;

% Generate random data for matrix X (centered)
MatrixX = randn(samples, features);

% PCA using X^T X
start_XTX = tic;
CovMatrix1 = MatrixX' * MatrixX;
[EigVecs1, EigVals1] = eig(CovMatrix1);
[~, order1] = sort(diag(EigVals1), 'descend');
EigVecs1 = EigVecs1(:, order1);
elapsed_XTX = toc(start_XTX);

% PCA using XX^T
start_XXT = tic;
CovMatrix2 = MatrixX * MatrixX';
[EigVecs2, EigVals2] = eig(CovMatrix2);
EigVecs2 = MatrixX' * EigVecs2;
for k = 1:samples
    EigVecs2(:,k) = EigVecs2(:,k) / norm(EigVecs2(:,k));
end
[~, order2] = sort(diag(EigVals2), 'descend');
EigVecs2 = EigVecs2(:, order2);
elapsed_XXT = toc(start_XXT);

% Display the results
fprintf('Time for PCA using X^T X: %.4f seconds\n', elapsed_XTX);
```

```
Time for PCA using X^T X: 63.1824 seconds
```

```
fprintf('Time for PCA using XX^T: %.4f seconds\n', elapsed_XXT);
```

```
Time for PCA using XX^T: 0.0580 seconds
```

Problem 3.1

Case: n < p

```
n = 10; % number of samples
p = 10000; % number of features

% Generate random data for matrix X (centered)
X = randn(n, p);

Y = X'* X;
disp("If n <p:")
```

```
If n <p:
```

```
disp("Size of the square matrix:");
```

```
Size of the square matrix:
```

```
disp(size(Y));
```

```
       10000       10000
```

```
disp("Rank of the square matrix");
```

```
Rank of the square matrix
```

```
disp(rank(Y));
```

```
    10
```

Case n > p:

```
disp("If n > p:");
```

```
If n > p:
```

```
n = 1000; % number of samples
p = 100; % number of features

% Generate random data for matrix X (centered)
X = randn(n, p);

Y = X'*X;
disp("Size of the square matrix:");
```

```
Size of the square matrix:
```

```
disp(size(Y));
```

```
   100   100
```

```
disp("Rank of the square matrix");
```

```
Rank of the square matrix
```

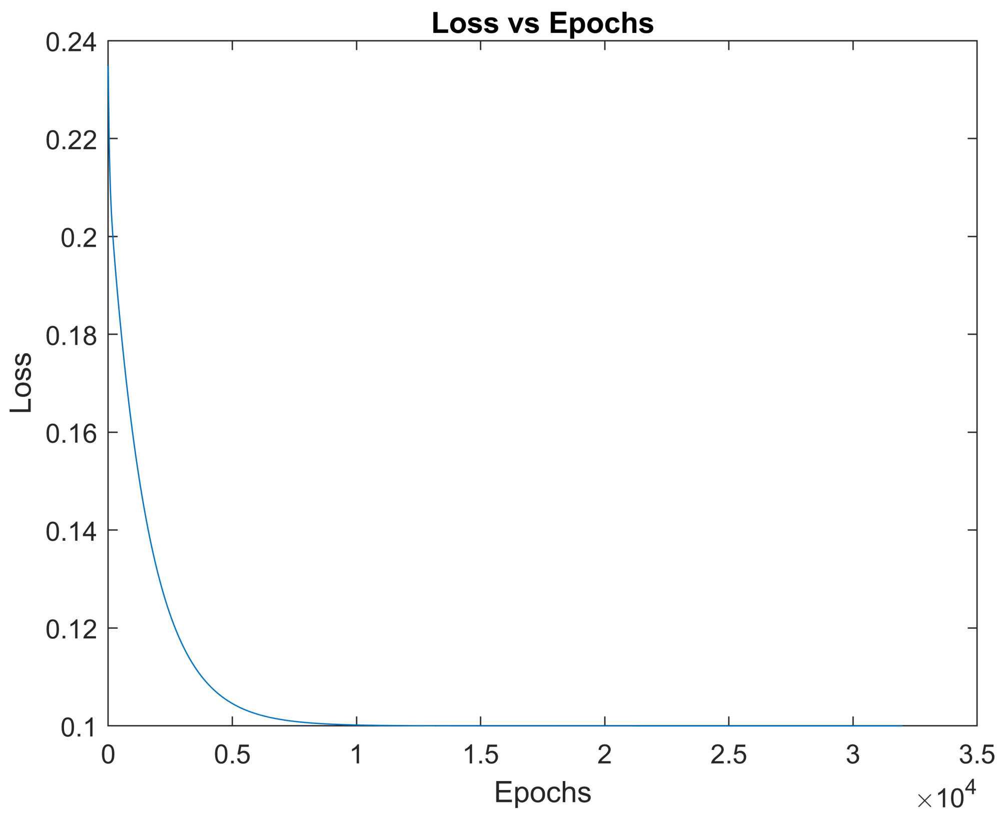
```
disp(rank(Y));
```

```
100
```

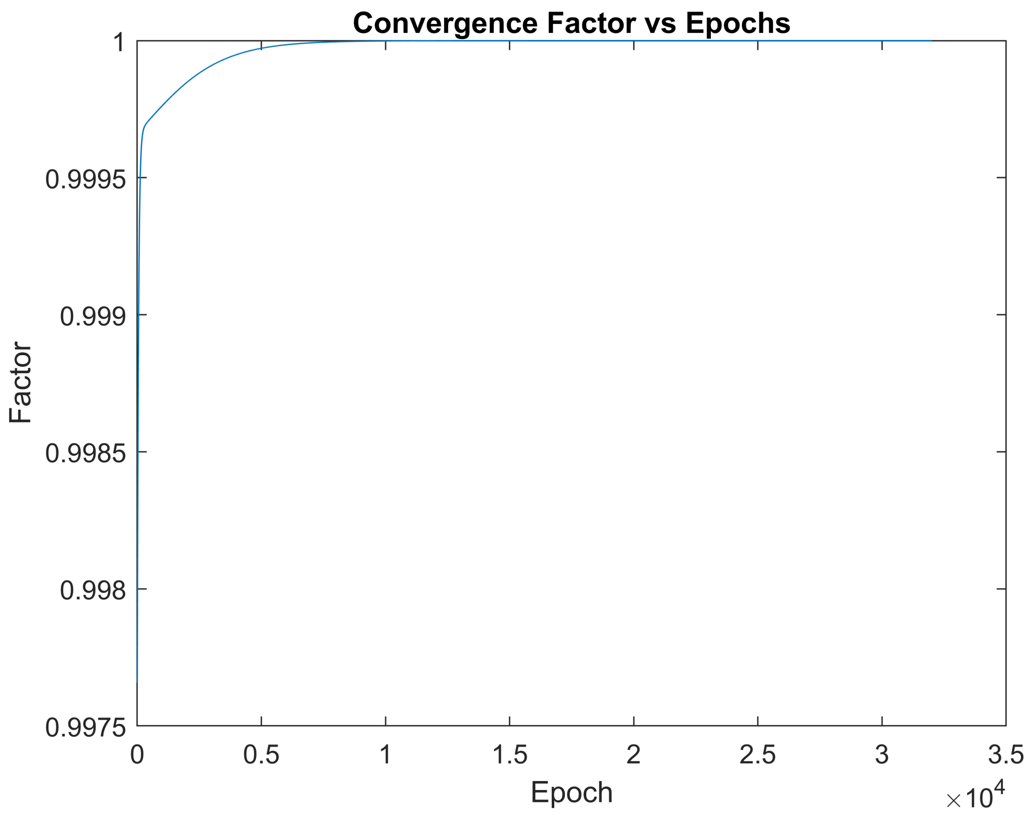
Given that the rank of the matrix is less than its size, and the number of features *p* exceeds the number of samples *n*, the resulting matrix is singular and therefore non-invertible. Consequently, this scenario precludes the viability of a closed-form solution.

PROBLEM 3.2

```
% Given matrix A, target vector y, learning rate, and initial weights
A = [1, 2, 4; 1, 3, 5; 1, 7, 7; 1, 8, 9];
y = [1; 2; 3; 4];

% Compute the maximum singular value of A
max_singular_value = max(svd(A' * A));

% Set learning rate as the inverse of the max singular value
learningRate = 1 / max_singular_value;
weights = zeros(size(A, 2), 1);

% Initialize a vector to track loss values
lossHistory = zeros(1, 32000);

% Vanilla Gradient Descent (32k iterations)
for epoch = 1:32000
    gradient = A' * (A * weights - y);
    weights = weights - learningRate * gradient;
    lossHistory(epoch) = norm(A * weights - y)^2;
end

% Calculate the convergence factor
convergenceFactor = lossHistory(2:end) ./ lossHistory(1:end-1);

% Plot Loss vs Epochs
plot(lossHistory);
title('Loss vs Epochs');
xlabel('Epochs');
ylabel('Loss');
```



```
% Plot Convergence Factor vs Epochs
figure;
plot(convergenceFactor);
title('Convergence Factor vs Epochs');
xlabel('Epoch');
ylabel('Factor');
```



```
% Closed form solution for verification
weightsClosedForm = pinv(A' * A) * A' * y;
display(weightsClosedForm);
```

```
weightsClosedForm = 3×1
   -1.0000
    0.0333
    0.5333
```

```
display(weights);
```

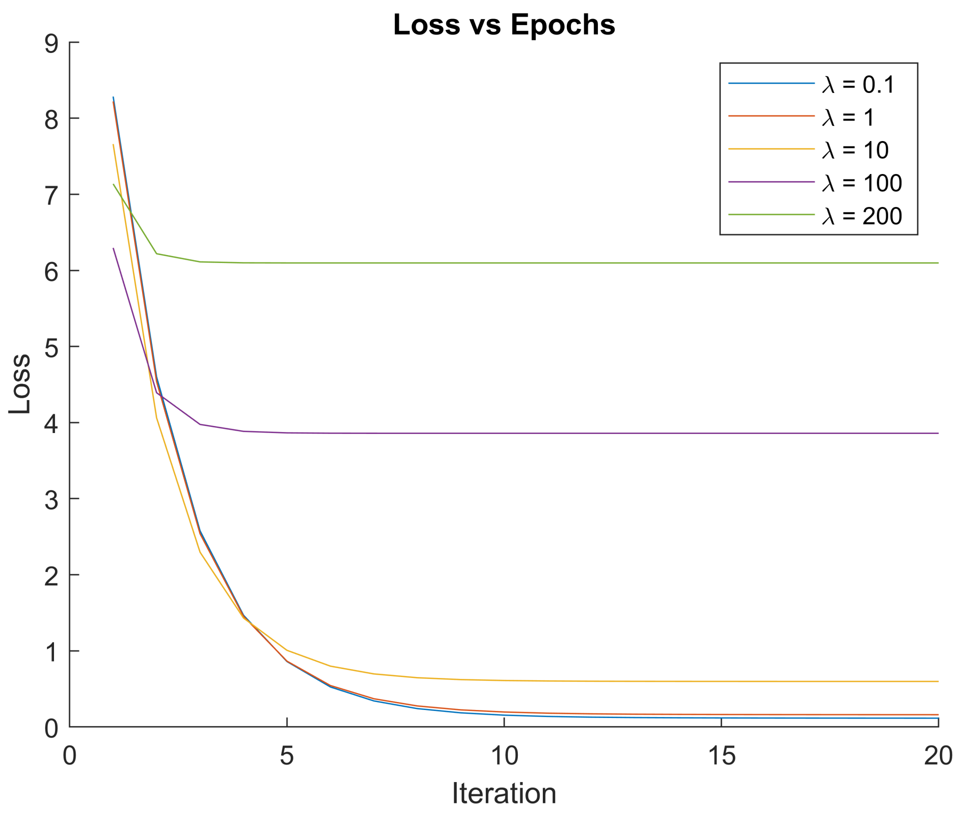
```
weights = 3×1
   -1.0000
    0.0333
    0.5333
```

From analyzing the weight matrices, it's evident that the Gradient Descent approach achieves the optimal solution, exhibiting nearly linear convergence with the specified learning rate.

PROBLEM 3.3

```
% Define the matrix A and vector y
A = [1, 2, 4; 1, 3, 5; 1, 7, 7; 1, 8, 9];
y = [1; 2; 3; 4];

% Define a range of λ values
lambda_values = [0.1, 1, 10, 100, 200];

```

```
% Define the number of iterations
num_iterations = 20;

% Initialize variables to store losses
losses = zeros(length(lambda_values), num_iterations);

% Perform gradient descent for each λ
for k = 1:length(lambda_values)
    lambda = lambda_values(k);
    alpha = 1 / (max(max(A' * A)) + lambda_values(k));

    % Initialize θ0 to [0; 0; 0]
    theta = [0; 0; 0];
    for iteration = 1:num_iterations
        % Compute the gradient for ridge regression
        gradient = -A' * (y - A * theta) + lambda * theta;

        % Update θ using the gradient and learning rate
        theta = theta - alpha * gradient;

        % Calculate the cost (ridge regression loss)
        loss = (1 / (2 )) * norm(A * theta - y)^2 + (theta' ...
            * theta) * (lambda / 2);
        % Store the loss
        losses(k, iteration) = loss;
    end
end

% Plot the loss curves for different λ values
figure;
hold on;
for k = 1:length(lambda_values)
    semilogy(1:num_iterations, losses(k, :), 'DisplayName', ...
        ['\lambda = ', num2str(lambda_values(k))]);
end

hold off;
xlabel('Iteration');
ylabel('Loss');
title("Loss vs Epochs");
legend;
```



```
%Verifying the results obtained from closed form solution
weights2 = pinv(A'*A + lambda_values(5)*eye(size(A, 2)))*A'*y;
```

To verify, let us compare the closed form weights and weights

```
display(weights2);
```

```
weights2 = 3×1
    0.0195
    0.1231
    0.1423
```

```
display(weights);
```

```
weights = 3×1
   -1.0000
    0.0333
    0.5333
```

Ans 4.

$$\min_{A,B} \frac{1}{2}\|P_\Omega(X-AB^T)\|_F^2 + \frac{\lambda}{2}(\|A\|_F^2 + \|B\|_F^2) \quad \text{— } ①$$

Given : $A = U\Sigma^{1/2}$ $B = V\Sigma^{1/2}$

$[U, \Sigma, V] = \text{svd}(Z)$

$P_\Omega(X-AB^T) = P_\Omega(X) - P_\Omega(AB^T)$
$= P_\Omega(X) - P_\Omega(Z)$ — ②

$\|A\|_F^2 = \text{tr}(A^T.A) = \text{tr}[(U\Sigma^{1/2})^T(U\Sigma^{1/2})]$
$= \text{tr}[(\Sigma^{1/2})^T U^T. U\Sigma^{1/2}]$

As $U^T.U = I$

$\|A\|_F^2 = \text{Tr}[(\Sigma^{1/2})^T(\Sigma^{1/2})] = \text{tr}(\Sigma),$ — ③

Similarly, $B^T B = (V\Sigma^{1/2})^T(V\Sigma^{1/2})$

$\|B\|_F^2 = \text{tr}(B^T B) = \text{tr}(\Sigma),$ — ④

Putting ②,③,④ in ①

$$\min_{A,B} \frac{1}{2}\|P_\Omega(X) - P_\Omega(Z)\|_F^2 + \frac{\lambda}{2}(tr(\Sigma) + tr(\Sigma))$$

$$= \min_{A,B} \frac{1}{2}\|P_\Omega(X) - P_\Omega(Z)\|_F^2 + \lambda\, tr(\Sigma)$$

$tr(\Sigma)$ is sum of ~~the~~ the singular values

$\therefore$ it is equal to nuclear norm of Z

Therefore, we have

$$\min_{A,B} \frac{1}{2}\|P_\Omega(X) - P_\Omega(Z)\|_F^2 + \lambda\|Z\|_*$$

PROBLEM 4.2

```
% Parameters
n = 2000; % Number of samples
d = 2000; % Number of features
r = 200;  % Rank for A and B

% Generate a random matrix X for testing
X = randn(n, d);

% Regularization parameter
lambda = 0.1;

% Initialize matrices A and B
A = randn(n, r);
B = randn(d, r);

% Maximum number of iterations
num_iterations = 100;

% Learning rate
learning_rate = 1 / (max(svd(A))^2 + lambda);

% Iterative updates for A and B
for iter = 1:num_iterations
    AB_diff = X - A * B';
    A = A + learning_rate * (AB_diff * B - lambda * A);
    B = B + learning_rate * (AB_diff' * A - lambda * B);
end
```