MAT 170 Homework 2 Project Report

Wanzhu Zheng
Student ID: 918166670

July 25, 2024

1 Problem 2A

The least squares problem can be formulated as minimizing the residual sum of squares:

$$\min \ \|Ax - b\|_2^2$$

Then,

$$\begin{aligned}
f(x) &= \|Ax - b\|_2^2 \\
&= (Ax - b)^T(Ax - b) \\
&= ((Ax)^T - b^T)(Ax - b) \\
&= (x^T A^T - b^T)(Ax - b) \\
&= x^T A^T Ax - (x^T A^T b) - (b^T Ax) + b^T b \\
&= x^T (A^T A)x - 2b^T Ax + b^T b
\end{aligned}$$

After applying the gradient,

$$\begin{aligned}
\nabla f(x) &= 2(A^T Ax - A^T b) \\
\nabla f(x^*) &= 0 \\
2(A^T Ax^* - A^T b) &= 0 \\
A^T Ax^* - A^T b &= 0 \\
A^T Ax^* &= A^T b \\
(A^T A)^{-1}(A^T A)x^* &= (A^T A)^{-1} A^T b \\
x^* &= (A^T A)^{-1} A^T b
\end{aligned}$$

2 Problem 2B

2.1 Model

Let $\mathbf{X}$ be an $m \times n$ feature matrix where m is the number of instances and n is the number of features.
Let $\mathbf{w}$ be the vector of coefficients/weights.
Let $\mathbf{y}$ be an $m \times 1$ vector that represent the true value.
Let $\hat{y}$ be an $m \times 1$ vector that represent the predicted value.
Our objective is to minimize the residual sum of squares:

$$\begin{aligned} \min L &= \min \Sigma_{i=1}^{m}(y_i - \hat{y}_i)^2 \\ &= \min \|Xw - y\|_2^2 \end{aligned}$$

2.2 Code in CVXPY

In [9]:
```
import numpy as np
import cvxpy as cp
from sklearn import datasets
```

In [3]:
```
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```

In [4]:
```
diabetes_X.shape
```

Out[4]: (442, 10)

In [5]:
```
diabetes_y.shape
```

Out[5]: (442,)

In [6]:
```
A = np.append(diabetes_X, np.ones((diabetes_X.shape[0],1)),axis=1)
b = diabetes_y
```

In [7]:
```
A.shape
```

Out[7]: (442, 11)

In [8]:
```
b.shape
```

Out[8]: (442,)

In [10]:
```
w = cp.Variable(A.shape[1])  # define weights
obj = cp.Minimize(cp.sum_squares(A @ w - b))  # define objective

prob = cp.Problem(obj)

# Solve the problem
prob.solve()

# Retrieve the optimal coefficients
optimal_weights = w.value

print("Optimal Weights):", optimal_weights)

```

```
Optimal Weights): [ -10.01219782 -239.81908937  519.83978679  324.39042769 -792.18416163
  476.74583782  101.04457032  177.06417623  751.27932109   67.62538639
  152.13348416]
```

In [ ]:
```

```

2.3 Comparison of Results

Yes, the weights are the same.

```
weights = np.linalg.solve(A.T @ A, A.T @ b)
print(weights)
```

```
[ -10.01219782 -239.81908937  519.83978679  324.39042769 -792.18416163
  476.74583782  101.04457032  177.06417623  751.27932109   67.62538639
  152.13348416]
```

3 Problem 2C

3.1 (a)

Since $f(x) = -log(x)$ is twice differentiable on the domain $x > 0$, we need to show that $f"(x) \geq 0$ for all $x > 0$ to prove convexity. Then,

$$f'(x) = -\frac{1}{x}$$
$$f"(x) = \frac{d}{dx}(-\frac{1}{x}) = \frac{1}{x^2}$$

Since our domain is $x > 0$, and we know that $\frac{1}{x^2}$ is positive for all $x > 0$, we conclude that $f"(x) \geq 0$ for all $x > 0$ so by the second order condition for convexity, $f(x) = -log(x)$ is convex for all $x > 0$.

3.2 (b)

We use the fact that $f(x) = -\log(x)$ is a convex function to prove the inequality $\frac{x+y}{2} \geq \sqrt{xy}, \quad \forall x, y > 0$. The epigraph of $f(x) = -log(x)$ is the set of points lying above the graph of the function. Suppose there exists two arbitrary points $(x_1, f(x_1))$ and $(x_2, f(x_2))$. Let there be a line segment connecting these two points above the graph. By convexity, we have:

$$f(\frac{x+y}{2}) \leq \frac{f(x) + f(y)}{2}$$
$$-log(\frac{x+y}{2}) \leq \frac{-log(x) + (-log(y))}{2}$$
$$-log(\frac{x+y}{2}) \leq \frac{-log(xy)}{2}$$
$$\frac{x+y}{2} \geq \sqrt{xy} \ \forall \ x, y \geq 0$$

3.3 (c)

Let $f_1(x) = 0$ and $f_2(x) = 3x$.
At $x = 0$, $f_1 = f_2$ so the subdifferential is $\partial g(0) = \{\lambda \ \epsilon \ \mathbb{R} : \lambda \leq 0\}$
At $x = 1$, $f_1 < f_2$ so the subdifferential is $\partial g(1) = \{3\}$.

3.4 (d)

Let $f_1(x) = 2x_1 + 3x_2$ and $f_2(x) = x_1 - x_2$.
Then, $\partial f_1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ and $\partial f_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$
At $x = (1, 1)$, $f_1 > f_2$ so the subdifferential is $\partial h(1, 1) = 2, 3$.
At $x = (0, 0)$, $f_1 = f_2$ so the subdifferential is $\partial h(0, 0) = conv(\partial f_1, \partial f_2) = \{\lambda(2, 3), (1 - \lambda)(1, -1) : 0 \leq \lambda \leq 1\}$.

4 Problem 2D

4.1 (a)

In [9]:

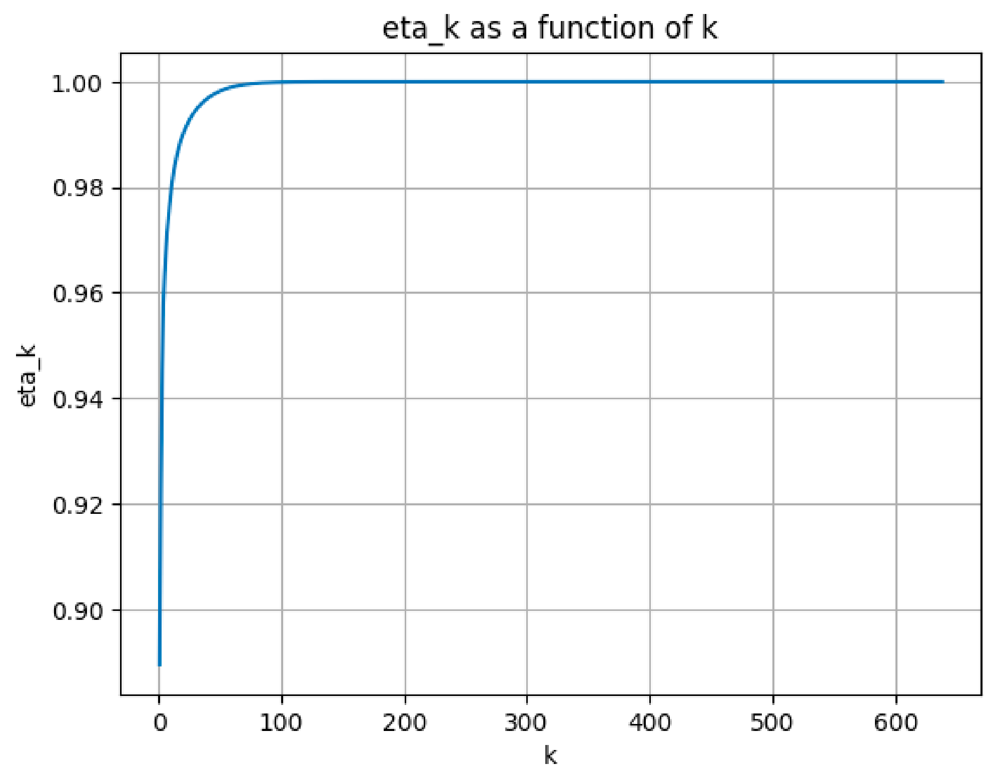
```python
import matplotlib.pyplot as plt

U, S, V = np.linalg.svd(A)
fnorm_A = np.linalg.norm(A, ord='fro')

eta_vals = []

for k in range(1, min(A.shape)+1):
    Ak = np.dot(U[:, :k], np.dot(np.diag(S[:k]), V[:k, :]))
    fnorm_Ak = np.linalg.norm(Ak, ord='fro')
    eta_k = (fnorm_Ak**2) / (fnorm_A**2)
    eta_vals.append(eta_k)

# Plot the ratio eta_k from 1 to 266
plt.plot(range(1, min(A.shape)+1), eta_vals)
plt.xlabel('k')
plt.ylabel('eta_k')
plt.title('eta_k as a function of k')
plt.grid(True)
plt.show()
```



We compute the Singular Value Decomposition (SVD) of matrix A into three matrices: U, S, and V, such that $A = USV^T$. After obtaining the SVD, we compute the total sum of squared variances of the singular values (contained in matrix S). This sum represents the total variability of the image. For each value of k from 1 to the minimum dimension of A, we compute η_k. η_k is the ratio of the sum of squared variances of the first k singular values to the total sum of squared variances. It represents what proportion of the total variance is explained by rank k approximation of k. Finally, we plot the ratio η_k against k. This plot helps us understand what percentage of image variance is retained as we increase the number of singular values included in the reconstruction of the image. From the graph, we see that $k = 70$ already captures a majority of the image variance. Generally, when k is small, η_k is close to 0, indicating that only a small portion of the image variance is captured. As k increases, η_k approaches 1, indicating that more image variance is captured as more singular values are included in the reconstruction.

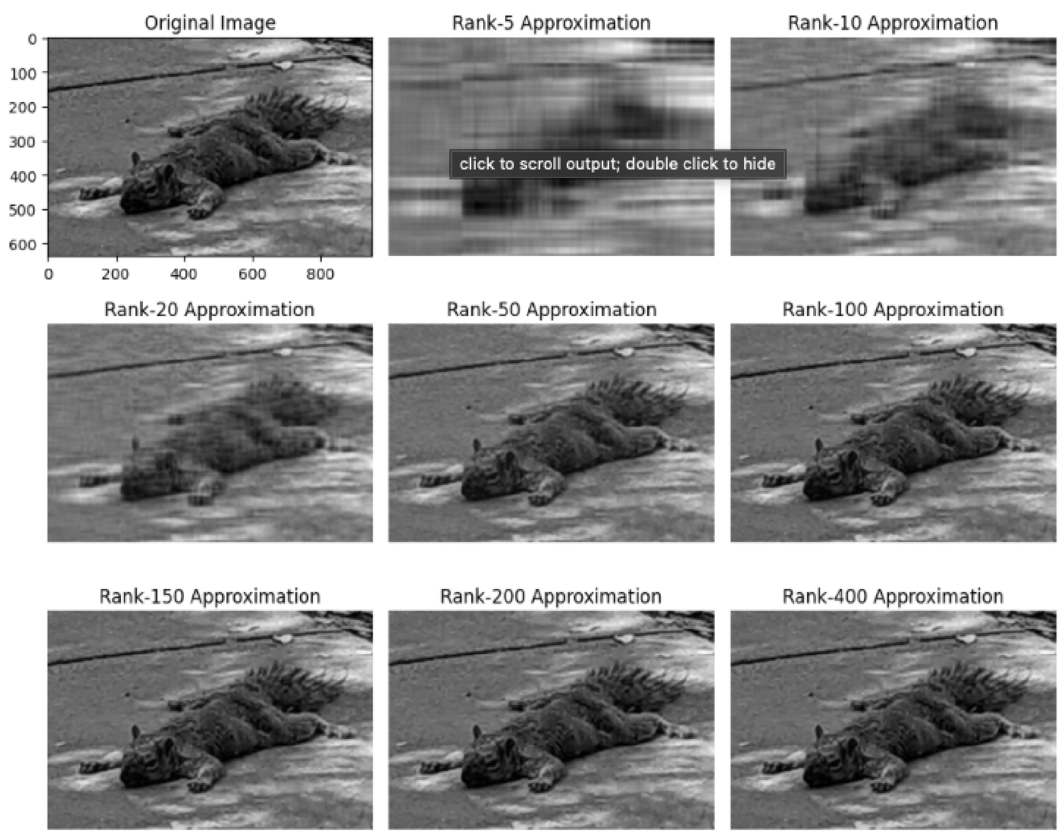
4.2 (b)

```python
ranks = [5, 10, 20, 50, 100, 150, 200, 400]

# Plot original image
plt.figure(figsize=(10, 8))
plt.subplot(3, 3, 1)
plt.imshow(A, cmap='gray')
plt.title('Original Image')

# Plot rank-k approximations
for i, k in enumerate(ranks):
    # Reconstruct the image using k singular values
    Ak = np.dot(U[:, :k], np.dot(np.diag(S[:k]), V[:k, :]))
    plt.subplot(3, 3, i+2)
    plt.imshow(Ak, cmap='gray')
    plt.title(f'Rank-{k} Approximation')
    plt.axis('off')

plt.tight_layout()
plt.show()
```



The output of the code presents a visual comparison of the original grayscale image and several rank-k approximations generated using Singular Value Decomposition (SVD). The initial subplot displays the unaltered original image. Subsequent subplots depict the rank-k approximations, where each subplot corresponds to a specific rank value. As the rank increases, more singular values are incorporated into the reconstruction, resulting in greater preservation of image details. Consequently, lower ranks yield smoother images with reduced detail, while higher ranks closely resemble the original image with finer nuances preserved.

5 Problem 2E

5.1 (a)

A $= \begin{bmatrix} -2 & 11 \\ -10 & 5 \end{bmatrix}$

$A^T A = \begin{bmatrix} 125 & 75 \\ 75 & 125 \end{bmatrix}$

For the eigenvalue of 200, the eigenvector is $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

For the eigenvalue of 50, the eigenvector is $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$

Then, $\Sigma = \begin{bmatrix} 10\sqrt{2} & 0 \\ 0 & 5\sqrt{2} \end{bmatrix} = \begin{bmatrix} 14.1421 & 0 \\ 0 & 7.0711 \end{bmatrix}$

and $U = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}$

$$\begin{aligned} v_1 &= \frac{1}{\sigma_1} \begin{bmatrix} -2 & 11 \\ -10 & 5 \end{bmatrix}^T \cdot u_1 \\ &= \frac{1}{10\sqrt{2}} \begin{bmatrix} -2 & -10 \\ 11 & 5 \end{bmatrix} \cdot \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \\ &= \begin{bmatrix} -\frac{3}{5} \\ \frac{4}{5} \end{bmatrix} \end{aligned}$$

$$\begin{aligned} v_2 &= \frac{1}{\sigma_2} \begin{bmatrix} -2 & 11 \\ -10 & 5 \end{bmatrix}^T \cdot u_1 \\ &= \frac{1}{5\sqrt{2}} \begin{bmatrix} -2 & -10 \\ 11 & 5 \end{bmatrix} \cdot \begin{bmatrix} -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \\ &= \begin{bmatrix} -\frac{4}{5} \\ -\frac{3}{5} \end{bmatrix} \end{aligned}$$

Then, $V = \begin{bmatrix} -\frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & -\frac{3}{5} \end{bmatrix} = \begin{bmatrix} -0.6 & -0.8 \\ 0.8 & -0.6 \end{bmatrix}$

5.2 (b)

```python
import numpy as np
A = np.array([[-2, 11],
              [-10, 5]])

# Compute the SVD of matrix A
U, S, VT = np.linalg.svd(A)
V = VT.T

# Print the results
print("Left Singular Vectors (U):")
print(U)
print("\nSingular Values (S):")
print(S)
print("\nRight Singular Vectors (V):")
print(V)
```

```
Left Singular Vectors (U):
[[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]

Singular Values (S):
[14.14213562  7.07106781]

Right Singular Vectors (V):
[[ 0.6 -0.8]
 [-0.8 -0.6]]
```

The computation by python and the manual computation yield the same results.