Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Лабораторна робота №2

З «Моделювання складних систем»

Виконала студентка 3-го курсу
Групи ІПС-31
Величко Діана Сергіївна

Варіант 3

2024

Завдання

Матрицю X будемо інтерпретувати як двовимірне вхідне зображення, а матрицю Y – як вихідне зображення. Потрібно побудувати лінійний оператор перетворення вхідного сигналу X у вихідний сигнал Y на основі формули (3.9).

1. Вивчити означення псевдооберненої матриці і її основні властивості.

2. Створити програму, яка за заданими двома зображеннями знаходить лінійний оператор переходу між цими зображеннями. Основою для програми є формула (3.9), де V – довільна матриця (наприклад, нульова). Псевдообернену матрицю в (3.9) шукати двома методами: на основі формули Мура-Пенроуза (див. (3.3) або (3.4)) і на основі формули Гревіля. Правильність знаходження псавдооберненої матриці перевірити за допомогою теореми 3.1 про характеристичну властивість псевдооберненої матриці.

3. Вивести вихідне зображення і образ вхідного зображення при одержаному перетворенні. Зробити порівняння. Проаналізувати одержаний результат.

4. Оформити в друкованій формі звіт про виконання роботи, в якому викласти результати проведених обчислень.

Теорія

Псевдооберненою називається узагальнення оберненої матриці в лінійній алгебрі. A+ називається псевдооберненою до матриці A, якщо вона задовольняє такі умови:

1. AA+A = A (AA+чи A+A не обов'язково дорівнюватимуть одиничній матриці)
2. A+AA+= A+
3. AA+*=AA+ (це означає, що AA+-ермітова матриція)
4. A+A*=A+A (A+A – також ермітова матриця)

Де A*- ермітово-спряжена матриця до матриці A.

Властивості:

1. Псевдообернена матриця існує і вона єдина.
2. Псевдообернення нульової матриці дорівнює її транспонуванню
3. Псевдообернення є оборотним до самого себе A++=A

4. Псевдообернення комутує з транспонуванням, спряженням і ермітовим спряженням: AT+=A+T, A+= A+, A*+=A+*
5. Ранг матриці дорівнює рангу її псевдооберненої rank A+=rank A
6. Псевдообернення добутку матриці A на скаляр α дорівнює добутку матриці A+ на обернене число -1.
7. Якщо вже відома матриця A*A+ чи матриця AA*+, то їх можна використати для обчислення A+. A+=A*A+A*, A+=A*AA*+
8. Якщо матриця Ai утворена за матриці A за допомогою вставки ще одного нульового рядка/стовпця в i-ту позицію, то Ai+ буде утворюватись з A+ додаванням нульового стовпця/рядка в i-ту позицію.
9. Якщо рядок/стовпець в попередній процедурі не є нульовим i0 то існує формула Гревіля для вираження Ai+ через A,A+,i.

Формула Гревіля

Якщо для матриці A відома псевдообернена (обернена) матриця A^+, то для розширеної матриці $\begin{pmatrix} A \\ a^T \end{pmatrix}$ справедлива формула

$$\begin{pmatrix} A \\ a^T \end{pmatrix}^+ = \begin{cases} \left(A^+ - \frac{Z(A)aa^T A^+}{a^T Z(A)a} \vdots \frac{Z(A)a}{a^T Z(A)a} \right), \, if \; a^T Z(A)a > 0 \\ \left(A^+ - \frac{R(A)aa^T A^+}{1+a^T R(A)a} \vdots \frac{R(A)a}{1+a^T R(A)a} \right), \, if \; a^T Z(A)a = 0 \end{cases}, \qquad (3.2)$$

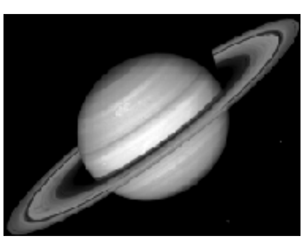
де $Z(A) = E - A^+A$ – проектор на ядро матриці A, $R(A) = A^+ (A^+)^T$.

Визначення Мура-Пенроуза

$$A^+ = \lim_{\delta \to 0} (A^* A + \delta I)^{-1} A^* = \lim_{\delta \to 0} A^* (AA^* + \delta I)^{-1}$$

Ці границі існують, навіть якщо $(AA^*)^{-1}$ і $(A^* A)^{-1}$ не комутують.

Вхідні дані:




Код розв’язку:

Лабораторну роботу було виконано у формі проекту на мові Python .

Main.py:

```
import time
import tracemalloc
import matplotlib.pyplot as plt
from src.image_processor import load_image, save_image
from src.pseudoinverse_methods import (
    greville_pseudoinversion,
    moore_penrose_pseudoinversion,
    svd_pseudoinversion
)
from src.utils import Z, compare_images, calculate_brightness
import numpy as np

# окремо для графіків
times = {}
memories = {}
brightness_diff = {}

def measure_performance(method_name, method, X, Y):
    # вимірюємо час та пам'ять
    tracemalloc.start()
    start_time = time.time()

    X_pseudo_inverse = method(X)
    A = Y @ X_pseudo_inverse + np.random.rand(Y.shape[0], X.shape[0]) @ 
Z(X_pseudo_inverse, X)
    Y_corrected = A @ X

    end_time = time.time()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    # зберігаємо результати часу і пам'яті
    times[method_name] = end_time - start_time
    memories[method_name] = current / 1024  # в КБ

    # зберігаємо результати у файл
    with open('/Users/macbookpro/Desktop/MCC/Lab2/output.txt', 'a') as f:
        f.write(f"Метод: {method_name}\n")
        f.write(f"Час виконання: {end_time - start_time} секунд\n")
        f.write(f"Використано пам'яті: {current / 1024} KB (пік: {peak / 
1024} KB)\n\n")

    return Y_corrected

def plot_metrics():
    # графік часу виконання
    plt.figure()
    plt.bar(times.keys(), times.values())
    plt.title('Час виконання методів')
    plt.ylabel('Час (с)')
    plt.show()
```

```
    # графік використаної пам'яті
    plt.figure()
    plt.bar(memories.keys(), memories.values())
    plt.title('Використання пам\'яті методами')
    plt.ylabel('Пам\'ять (KB)')
    plt.show()

    # графік порівняння яскравості
    plt.figure()
    plt.bar(brightness_diff.keys(), brightness_diff.values())
    plt.title('Різниця в яскравості між результатами')
    plt.ylabel('Яскравість')
    plt.show()

def main():
    # додавання зображень
    X_path = '/Users/macbookpro/Desktop/MCC/Lab2/x1.bmp'
    Y_path = '/Users/macbookpro/Desktop/MCC/Lab2/y3.bmp'
    result_path_greville =
'/Users/macbookpro/Desktop/MCC/Lab2/results/result_greville.bmp'
    result_path_mp =
'/Users/macbookpro/Desktop/MCC/Lab2/results/result_mp.bmp'

    X = load_image(X_path)
    X = np.vstack([X, np.ones(X.shape[1])])  # додаємо одиничний рядок

    Y = load_image(Y_path)

    # метод Гревіля
    Y_greville_corrected = measure_performance("Greville",
greville_pseudoinversion, X, Y)
    save_image(Y_greville_corrected, result_path_greville)

    # метод Мура-Пенроуза з двома алгоритмами
    Y_mp_corrected_method1 = measure_performance("Moore-Penrose 1", lambda X:
moore_penrose_pseudoinversion(X, 'method1'), X, Y)
    save_image(Y_mp_corrected_method1, result_path_mp.replace('.bmp',
'_method1.bmp'))

    Y_mp_corrected_method2 = measure_performance("Moore-Penrose 2", lambda X:
moore_penrose_pseudoinversion(X, 'method2'), X, Y)
    save_image(Y_mp_corrected_method2, result_path_mp.replace('.bmp',
'_method2.bmp'))

    # Метод SVD
    Y_svd_corrected = measure_performance("SVD", svd_pseudoinversion, X, Y)
    save_image(Y_svd_corrected, result_path_mp.replace('.bmp', '_svd.bmp'))

    # порівняння
    compare_images(Y, Y_greville_corrected, title='Результат методу Гревіля')
    compare_images(Y, Y_mp_corrected_method1, title='Результат методу Мура-
Пенроуза 1')
    compare_images(Y, Y_mp_corrected_method2, title='Результат методу Мура-
Пенроуза 2')
    compare_images(Y, Y_svd_corrected, title='Результат методу SVD')
```

```
    # підрахунок різниці яскравості
    brightness_diff["Greville"] = calculate_brightness(Y) - calculate_brightness(Y_greville_corrected)
    brightness_diff["Moore-Penrose 1"] = calculate_brightness(Y) - calculate_brightness(Y_mp_corrected_method1)
    brightness_diff["Moore-Penrose 2"] = calculate_brightness(Y) - calculate_brightness(Y_mp_corrected_method2)
    brightness_diff["SVD"] = calculate_brightness(Y) - calculate_brightness(Y_svd_corrected)

    # графіки
    plot_metrics()

if __name__ == "__main__":
    # видалення даних з файлу перед записом нових результатів
    with open('/Users/macbookpro/Desktop/MCC/Lab2/output.txt', 'w') as f:
        f.write("Результати вимірювань для методів Гревіля та Мура-Пенроуза:\n\n")

    main()
```

image_processor.py :

```
from PIL import Image
import numpy as np

def load_image(image_path):

   # завантаження зображення та конвертація його у матрицю.
    return np.array(Image.open(image_path).convert('L'))

def save_image(image, filepath):

    # зберігаємо матрицю як зображення.
    img = Image.fromarray(image.astype(np.uint8))
    img.save(filepath)
```

pseudoinverse_methods.py :

```
import numpy as np
from src.utils import Z

def greville_pseudoinversion(A):
    # перевіряємо, чи треба транспонувати матрицю A (якщо кількість рядків більше за кількість стовпців)
    is_swap = False
    if A.shape[0] > A.shape[1]:
        is_swap = True
        A = A.T

    # обираємо поточний вектор як першого рядка матриці A
    current_vector = A[0, :].reshape(-1, 1)
    vector_scalar = np.dot(current_vector.T, current_vector)

    # якщо вектор скаляр дорівнює 0, то псевдообернена матриця є самим
```

```
вектором
    if vector_scalar == 0:
        A_pseudo_inverse = current_vector
    else:
        # ділимо вектор на його скаляр
        A_pseudo_inverse = current_vector / vector_scalar

    A_i = current_vector.T
    # проходимо по кожному рядку матриці A
    for i in range(1, A.shape[0]):
        current_vector = A[i, :].reshape(-1, 1)
        Z_A = Z(A_i, A_pseudo_inverse)  # обчислюємо матрицю Z
        A_i = np.vstack([A_i, current_vector.T])  # додаємо поточний вектор
до A_i
        denom_Z = np.dot(current_vector.T, np.dot(Z_A, current_vector))

        if denom_Z > 0:
            # оновлюємо псевдообернену матрицю
            A_pseudo_inverse = np.hstack([
                A_pseudo_inverse - (np.dot(Z_A, np.dot(current_vector,
current_vector.T)) @ A_pseudo_inverse) / denom_Z,
                np.dot(Z_A, current_vector) / denom_Z
            ])
        else:
            # якщо знаменник Z <= 0, обчислюємо матрицю R_A
            R_A = np.dot(A_pseudo_inverse, A_pseudo_inverse.T)
            denom_R = 1 + np.dot(current_vector.T, np.dot(R_A,
current_vector))
            A_pseudo_inverse = np.hstack([
                A_pseudo_inverse - (np.dot(R_A, np.dot(current_vector,
current_vector.T)) @ A_pseudo_inverse) / denom_R,
                np.dot(R_A, current_vector) / denom_R
            ])

    # якщо матриця була транспонована, то знову транспонуємо результат
    if is_swap:
        A_pseudo_inverse = A_pseudo_inverse.T

    return A_pseudo_inverse

def moore_penrose_pseudoinversion(A, initial_approximation='method1'):
    # перевіряємо, чи потрібно транспонувати матрицю A
    is_swap = False
    if A.shape[0] > A.shape[1]:
        is_swap = True
        A = A.T

    # точность
    CONST_E = 1e-8
    delta = 10.0
    # початкова ітерація псевдооберненої матриці
    A_pseudo_inverse_current = np.inf * np.ones(A.shape).T
    A_pseudo_inverse_next = -np.inf * np.ones(A.shape).T

    # обираємо початкове наближення залежно від методу
    if initial_approximation == 'method1':
        #  1: початкове наближення за допомогою псевдооберненої матриці
```

```
        A_pseudo_inverse_current = np.linalg.pinv(A)

    elif initial_approximation == 'method2':
        # 2: початкове наближення за допомогою SVD
        U, S, Vt = np.linalg.svd(A, full_matrices=False)
        S_inv = np.zeros_like(S)
        for i in range(len(S)):
            if S[i] > 1e-10:  #
                S_inv[i] = 1 / S[i]
        A_pseudo_inverse_current = Vt.T @ np.diag(S_inv) @ U.T

    # ітеруємо, доки різниця між поточним і наступним наближенням більша за
точність
    while np.max(np.square(A_pseudo_inverse_current - A_pseudo_inverse_next))
> CONST_E:
        A_pseudo_inverse_current = A_pseudo_inverse_next
        A_pseudo_inverse_next = np.dot(A.T, np.linalg.inv(np.dot(A, A.T) +
delta * np.eye(A.shape[0])))
        delta /= 2.0  # Зменшуємо дельту

    # якщо матриця транспонована - повертаємо її
    if is_swap:
        A_pseudo_inverse_next = A_pseudo_inverse_next.T

    return A_pseudo_inverse_next

def svd_pseudoinversion(A):
    # псевдообернена матриця на основі сингулярного розкладу
    U, S, Vt = np.linalg.svd(A, full_matrices=False)
    S_inv = np.zeros_like(S)
    for i in range(len(S)):
        if S[i] > 1e-10:  # Уникаємо ділення на 0
            S_inv[i] = 1 / S[i]
    A_pseudo_inverse = Vt.T @ np.diag(S_inv) @ U.T
    return A_pseudo_inverse
```

utils.py :

```
import numpy as np
import matplotlib.pyplot as plt

# функція для обчислення залишкової різниці Z(A, A_pseudo_inverse)
def Z(A, A_pseudo_inverse):
    return np.eye(A_pseudo_inverse.shape[0]) - np.dot(A_pseudo_inverse, A)

# функція для порівняння зображень
def compare_images(original, transformed, title):
    plt.figure()
    plt.imshow(original.astype(np.uint8), cmap='gray')
    plt.title('Оригінальне зображення')

    plt.figure()
    plt.imshow(transformed.astype(np.uint8), cmap='gray')
    plt.title(title)
    plt.show()
```

```
# функція для підрахунку яскравості зображення
def calculate_brightness(image):
    return np.mean(image)
```

Алгоритм :

Додаємо зображення :

```
def main():
    # додавання зображень
    X_path = '/Users/macbookpro/Desktop/MCC/Lab2/x1.bmp'
    Y_path = '/Users/macbookpro/Desktop/MCC/Lab2/y3.bmp'
    result_path_greville =
'/Users/macbookpro/Desktop/MCC/Lab2/results/result_greville.bmp'
    result_path_mp =
'/Users/macbookpro/Desktop/MCC/Lab2/results/result_mp.bmp'

    X = load_image(X_path)
    X = np.vstack([X, np.ones(X.shape[1])])  # додаємо одиничний рядок

    Y = load_image(Y_path)
```

Ініціалізуємо методи :

```
# метод Гревіля
Y_greville_corrected = measure_performance("Greville",
greville_pseudoinversion, X, Y)
save_image(Y_greville_corrected, result_path_greville)

# метод Мура-Пенроуза з двома алгоритмами
Y_mp_corrected_method1 = measure_performance("Moore-Penrose 1", lambda X:
moore_penrose_pseudoinversion(X, 'method1'), X, Y)
save_image(Y_mp_corrected_method1, result_path_mp.replace('.bmp',
'_method1.bmp'))

Y_mp_corrected_method2 = measure_performance("Moore-Penrose 2", lambda X:
moore_penrose_pseudoinversion(X, 'method2'), X, Y)
save_image(Y_mp_corrected_method2, result_path_mp.replace('.bmp',
'_method2.bmp'))

# Метод SVD
Y_svd_corrected = measure_performance("SVD", svd_pseudoinversion, X, Y)
save_image(Y_svd_corrected, result_path_mp.replace('.bmp', '_svd.bmp'))
```

Перетворюємо зображення на матрицю і навпаки :

```
def load_image(image_path):

   # завантаження зображення та конвертація його у матрицю.
    return np.array(Image.open(image_path).convert('L'))

def save_image(image, filepath):

    # зберігаємо матрицю як зображення.
    img = Image.fromarray(image.astype(np.uint8))
    img.save(filepath)
```

Створюємо псевдообернену для методу Гревіля :

```
def greville_pseudoinversion(A):
    # перевіряємо, чи треба транспонувати матрицю A (якщо кількість рядків
більше за кількість стовпців)
    is_swap = False
    if A.shape[0] > A.shape[1]:
        is_swap = True
        A = A.T

    # обираємо поточний вектор як першого рядка матриці A
    current_vector = A[0, :].reshape(-1, 1)
    vector_scalar = np.dot(current_vector.T, current_vector)

    # якщо вектор скаляр дорівнює 0, то псевдообернена матриця є самим
вектором
    if vector_scalar == 0:
        A_pseudo_inverse = current_vector
    else:
        # ділимо вектор на його скаляр
        A_pseudo_inverse = current_vector / vector_scalar

    A_i = current_vector.T
    # проходимо по кожному рядку матриці A
    for i in range(1, A.shape[0]):
        current_vector = A[i, :].reshape(-1, 1)
        Z_A = Z(A_i, A_pseudo_inverse)  # обчислюємо матрицю Z
        A_i = np.vstack([A_i, current_vector.T])  # додаємо поточний вектор
до A_i
        denom_Z = np.dot(current_vector.T, np.dot(Z_A, current_vector))

        if denom_Z > 0:
            # оновлюємо псевдообернену матрицю
            A_pseudo_inverse = np.hstack([
                A_pseudo_inverse - (np.dot(Z_A, np.dot(current_vector,
current_vector.T)) @ A_pseudo_inverse) / denom_Z,
                np.dot(Z_A, current_vector) / denom_Z
            ])
        else:
            # якщо знаменник Z <= 0, обчислюємо матрицю R_A
            R_A = np.dot(A_pseudo_inverse, A_pseudo_inverse.T)
            denom_R = 1 + np.dot(current_vector.T, np.dot(R_A,
current_vector))
            A_pseudo_inverse = np.hstack([
                A_pseudo_inverse - (np.dot(R_A, np.dot(current_vector,
current_vector.T)) @ A_pseudo_inverse) / denom_R,
                np.dot(R_A, current_vector) / denom_R
            ])

    # якщо матриця була транспонована, то знову транспонуємо результат
    if is_swap:
        A_pseudo_inverse = A_pseudo_inverse.T

    return A_pseudo_inverse
```

Створюємо псевдообернену для методу Мура-Пенроуза :

```
def moore_penrose_pseudoinversion(A, initial_approximation='method1'):
    # перевіряємо, чи потрібно транспонувати матрицю A
```

```
    is_swap = False
    if A.shape[0] > A.shape[1]:
        is_swap = True
        A = A.T

    # точність
    CONST_E = 1e-8
    delta = 10.0
    # початкова ітерація псевдооберненої матриці
    A_pseudo_inverse_current = np.inf * np.ones(A.shape).T
    A_pseudo_inverse_next = -np.inf * np.ones(A.shape).T

    # обираємо початкове наближення залежно від методу
    if initial_approximation == 'method1':
        #  1: початкове наближення за допомогою псевдооберненої матриці
        A_pseudo_inverse_current = np.linalg.pinv(A)

    elif initial_approximation == 'method2':
        # 2: початкове наближення за допомогою SVD
        U, S, Vt = np.linalg.svd(A, full_matrices=False)
        S_inv = np.zeros_like(S)
        for i in range(len(S)):
            if S[i] > 1e-10:  #
                S_inv[i] = 1 / S[i]
        A_pseudo_inverse_current = Vt.T @ np.diag(S_inv) @ U.T

    # ітеруємо, доки різниця між поточним і наступним наближенням більша за
точність
    while np.max(np.square(A_pseudo_inverse_current - A_pseudo_inverse_next))
> CONST_E:
        A_pseudo_inverse_current = A_pseudo_inverse_next
        A_pseudo_inverse_next = np.dot(A.T, np.linalg.inv(np.dot(A, A.T) +
delta * np.eye(A.shape[0])))
        delta /= 2.0  # Зменшуємо дельту

    # якщо матриця транспонована - повертаємо її
    if is_swap:
        A_pseudo_inverse_next = A_pseudo_inverse_next.T

    return A_pseudo_inverse_next
```

Та для сингулярного розкладу :

```
def svd_pseudoinversion(A):
    # псевдообернена матриця на основі сингулярного розкладу
    U, S, Vt = np.linalg.svd(A, full_matrices=False)
    S_inv = np.zeros_like(S)
    for i in range(len(S)):
        if S[i] > 1e-10:  # Уникаємо ділення на 0
            S_inv[i] = 1 / S[i]
    A_pseudo_inverse = Vt.T @ np.diag(S_inv) @ U.T
    return A_pseudo_inverse
```

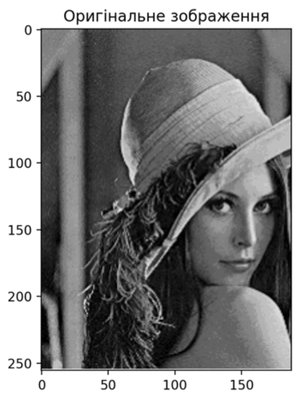
Обчислюємо залишкову різницю :

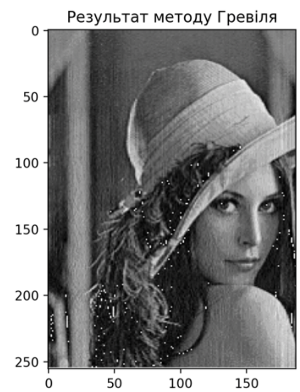
```
# функція для обчислення залишкової різниці Z(A, A_pseudo_inverse)
def Z(A, A_pseudo_inverse):
    return np.eye(A_pseudo_inverse.shape[0]) - np.dot(A_pseudo_inverse, A)
```

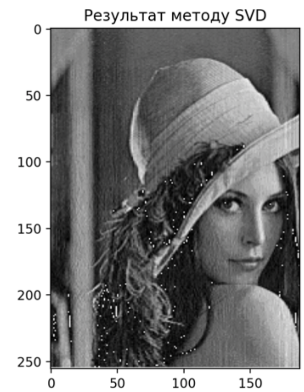
Виконуємо функції для порівняння :

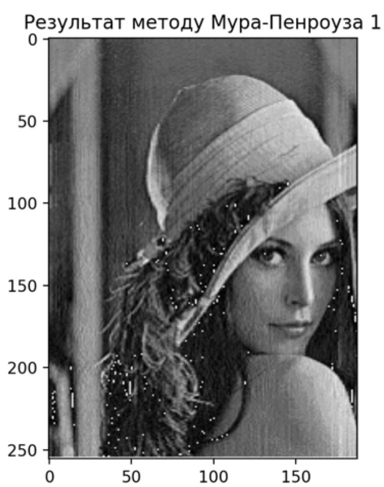
```python
# функція для порівняння зображень
def compare_images(original, transformed, title):
    plt.figure()
    plt.imshow(original.astype(np.uint8), cmap='gray')
    plt.title('Оригінальне зображення')

    plt.figure()
    plt.imshow(transformed.astype(np.uint8), cmap='gray')
    plt.title(title)
    plt.show()

# функція для підрахунку яскравості зображення
def calculate_brightness(image):
    return np.mean(image)
```

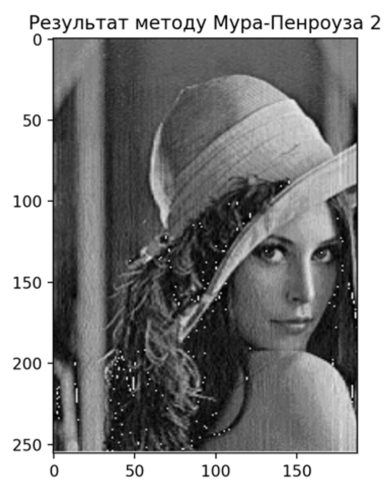
Отримані зображення :











Аналіз :

Розглянемо результати отримані з розрахунків проведених під час роботи проекту , а саме з файлу output.txt :

```
Результати вимірювань для методів Гревіля та Мура-Пенроуза:

Метод: Greville
Час виконання: 0.11781668663024902 секунд
Використано пам'яті: 1460.7265625 KB (пік: 1832.96875 KB)

Метод: Moore-Penrose 1
Час виконання: 0.019303321838378906 секунд
Використано пам'яті: 866.0234375 KB (пік: 1238.265625 KB)

Метод: Moore-Penrose 2
Час виконання: 0.013232707977294922 секунд
Використано пам'яті: 865.375 KB (пік: 1237.6171875 KB)

Метод: SVD
Час виконання: 0.0043430328369140625 секунд
Використано пам'яті: 865.375 KB (пік: 1237.6171875 KB)
```

Переглянувши ці дані , можна зробити такі висновки :

Час виконання:

- **Greville**: 0.1178 секунд — цей метод найповільніший з усіх, що може бути пов'язано зі складністю рекурсивного підходу до знаходження псевдооберненої матриці. Він має найбільший час виконання серед інших методів.
- **Moore-Penrose 1**: 0.0193 секунд — перший алгоритм для пошуку псевдооберненої матриці Мура-Пенроуза працює значно швидше, приблизно в 6 разів швидше за метод Гревіля.
- **Moore-Penrose 2**: 0.0132 секунд — другий алгоритм пошуку початкового наближення для методу Мура-Пенроуза є трохи швидшим, ніж перший, що вказує на його кращу ефективність при обчисленнях.
- **SVD**: 0.0043 секунд — метод на основі сингулярного розкладу (SVD) є найшвидшим серед усіх, що свідчить про його ефективність у таких обчисленнях.

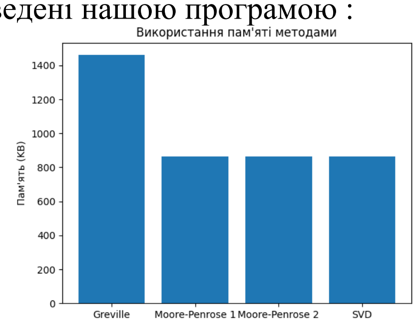
Використання пам'яті:

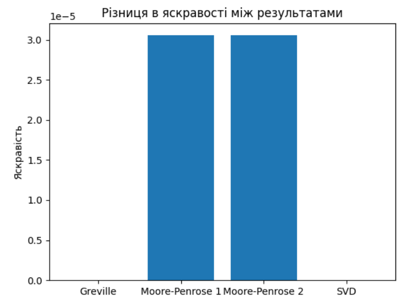
- **Greville**: 1460.73 KB (пік: 1832.97 KB) — цей метод також використовує найбільше пам'яті, що може бути результатом великих проміжних обчислень під час рекурсії.
- **Moore-Penrose 1**: 866.02 KB (пік: 1238.27 KB) — перший алгоритм Мура-Пенроуза використовує значно менше пам'яті, ніж метод Гревіля, майже вдвічі.
- **Moore-Penrose 2**: 865.38 KB (пік: 1237.62 KB) — другий алгоритм Мура-Пенроуза за використанням пам'яті дуже схожий на перший, з невеликими відмінностями. Він є трохи ефективнішим за перший варіант.
- **SVD**: 865.38 KB (пік: 1237.62 KB) — SVD має таке ж використання пам'яті, як і другий алгоритм Мура-Пенроуза, що робить його дуже ефективним у пам'яті.

1. **Метод Гревіля** є найменш ефективним як за часом, так і за використанням пам'яті. Це може бути пов'язано з особливостями обчислювального процесу.
2. **Метод Мура-Пенроуза** з обома алгоритмами показує добру ефективність як за часом, так і за пам'яттю. Другий алгоритм (на основі SVD) має перевагу в швидкості.
3. **Метод SVD** є найефективнішим з усіх. Він забезпечує швидкість виконання та ефективне використання пам'яті, що робить його найкращим варіантом для цієї задачі.

Отже метод на основі **SVD** виглядає найбільш оптимальним , якщо швидкість та використання пам'яті є основними факторами .

Також розглянемо графіки виведені нашою програмою :





Ці графіки тільки підтверджують наведену вище аналітику .

Ще нам необхідно порівняти між собою 2 алгоритми початкового наближення для методу Мура-Пенроуза :

Час:

Метод Мура-Пенроуза 1: 0.0193 секунд
Метод Мура-Пенроуза 2: 0.0132 секунд

Використання пам'яті:

Метод Мура-Пенроуза 1: 866.02 KB (пік: 1238.27 KB)
Метод Мура-Пенроуза 2: 865.38 KB (пік: 1237.62 KB)

Другий метод Мура-Пенроуза є швидшим на **0.0061 секунд** . Це свідчить про більш ефективний алгоритм для обчислення псевдооберненої матриці.Також 2 метод використовує на **0.64 KB** менше пам'яті, що є незначною різницею, але вказує на його кращу ефективність. Однак обидва методи мають дуже схожі показники використання пам'яті.

Висновок:

Під час виконання лабораторної роботи ми:

1. вивчили методи псевдообернення матриць, зокрема методи Гревіля та Мура-Пенроуза.
2. створили програму, яка реалізує два алгоритми знаходження псевдообернених матриць та дозволяє будувати математичну модель перетворення вхідного сигналу у вихідний на основі цих методів.

3. реалізували аналіз різниці між отриманими зображеннями після застосування кожного методу, з використанням гістограми яскравості зображень для порівняння результатів.
4. провели заміри часу виконання та використання оперативної пам’яті для кожного методу, результати яких були збережені в окремий файл.
5. зобразили та проаналізували результати для вихідних зображень після застосування кожного з методів, вивели графічні порівняння для подальшого аналізу.
6. створили цей звіт із результатами своєї роботи.