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

Псевдообернено називається узагальнення оберненої матриці в лінійній алгебрі. A+ називається псевдообернено 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

$$\begin{pmatrix} A \\ a^T \end{pmatrix}^+ = \begin{cases} \left(A^+ - \frac{Z(A)aa^TA^+}{a^TZ(A)a} : \frac{Z(A)a}{a^TZ(A)a}\right), if\ a^TZ(A)a > 0 \\ \left(A^+ - \frac{R(A)aa^TA^+}{1+a^TR(A)a} : \frac{R(A)a}{1+a^TR(A)a}\right), if\ a^TZ(A)a = 0 \end{cases}, \quad (3.2)$$

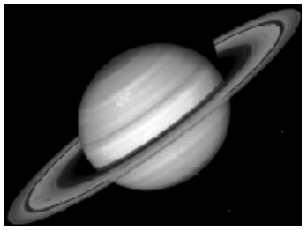
де $Z(A) = E - A^+A$ – проектор на ядро матриці A, $R(A) = A^+(A^+)^T$.

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 numpy as np
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

# Зберігаємо метрики для побудови графіків
times = {}
memories = {}
brightness_values = {}
pseudoinverse_properties = {}

# Ініціалізація вимірювання продуктивності
def initialize_performance(method_name):
    tracemalloc.start()
    start_time = time.time()
    return start_time, []

# Завершення вимірювання продуктивності
def finalize_performance(method_name, start_time, time_points, mem_points,
brightness):
    end_time = time.time()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    # Зберігаємо результати
    times[method_name] = (start_time, end_time, time_points)
    memories[method_name] = (current / 1024, peak / 1024, mem_points)
    brightness_values[method_name] = brightness

    # Записуємо 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")
        f.write(f"Яскравість результату: {brightness}\n\n")

# Вимірюємо продуктивність конкретного методу
def measure_performance(method_name, method, X, Y):
    start_time, time_points = initialize_performance(method_name)
    mem_points = []

    # Початкове значення пам'яті та часу
    current, peak = tracemalloc.get_traced_memory()
```

```
    mem_points.append(current / 1024)  # Пам'ять на старті
    time_points.append(0)  # Початковий час

    # Виконання методу
    X_pseudo_inverse = method(X)

    # Обчислюємо властивості псевдооберненої матриці
    check_properties(method_name, X, X_pseudo_inverse)

    # Обчислення матриці A на основі псевдоперевернутої матриці X та
випадкових шумів
    A = Y @ X_pseudo_inverse + np.random.rand(Y.shape[0], X.shape[0]) @
Z(X_pseudo_inverse, X)
    Y_corrected = A @ X

    # Записуємо час і пам'ять після обчислень
    time_points.append(time.time() - start_time)  # Кінцевий час
    current, peak = tracemalloc.get_traced_memory()
    mem_points.append(current / 1024)  # Пам'ять після виконання

    # Обчислюємо яскравість скоригованого зображення
    brightness = calculate_brightness(Y_corrected)

    finalize_performance(method_name, start_time, time_points, mem_points,
brightness)
    return Y_corrected

# Функція для перевірки властивостей псевдооберненої матриці
def check_properties(method_name, A, A_plus):
    prop_1 = np.allclose(A @ A_plus @ A, A)  # AA+A = A
    prop_2 = np.allclose(A_plus @ A @ A_plus, A_plus)  # A+AA+ = A+
    prop_3 = np.allclose(A @ A_plus, (A @ A_plus).T)  # AA+ симетрична
    prop_4 = np.allclose(A_plus @ A, (A_plus @ A).T)  # A+A симетрична
    pseudoinverse_properties[method_name] = (prop_1, prop_2, prop_3, prop_4)

# Функція для побудови графіків
def plot_metrics():
    plt.figure(figsize=(12, 6))

    # Графік часу виконання
    for method_name, (_, _, time_points) in times.items():
        plt.plot(range(len(time_points)), time_points, label=f'{method_name}
', marker='o')

    plt.title('Час виконання методів')
    plt.xlabel('Етап виконання')
    plt.ylabel('Час (с)')
    plt.legend()
    plt.grid()
    plt.show()

    # Графік використання пам'яті
    plt.figure(figsize=(12, 6))
    for method_name, (_, _, mem_points) in memories.items():
        plt.plot(range(len(mem_points)), mem_points, label=f'{method_name} ',
marker='o')
```

```
    plt.title('Використання пам\'яті методами')
    plt.xlabel('Етап виконання')
    plt.ylabel('Пам\'ять (КВ)')
    plt.legend()
    plt.grid()
    plt.show()

    # Графік яскравості результатів
    plt.figure(figsize=(12, 6))
    methods = ["Оригінал", "Greville", "Moore-Penrose 1", "Moore-Penrose 2",
"SVD"]
    brightness_values_list = [
        brightness_values["Оригінал"], brightness_values["Greville"],
        brightness_values["Moore-Penrose 1"], brightness_values["Moore-
Penrose 2"],
        brightness_values["SVD"]
    ]
    bars = plt.bar(methods, brightness_values_list, color=['gray', 'blue',
'orange', 'green', 'red'])

    # Додання значень на графік
    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2),
va='bottom')  # va='bottom' для розміщення тексту над стовпцем

    plt.title('Яскравість результатів для всіх методів')
    plt.xlabel('Методи')
    plt.ylabel('Яскравість')
    plt.grid(axis='y')
    plt.show()

# Головна функція
def main():
    # Очищення попередніх результатів з файлу
    with open('/Users/macbookpro/Desktop/MCC/Lab2/output.txt', 'w') as f:
        f.write("Результати вимірювань для методів Гревіля, Мура-Пенроуза та
SVD:\n\n")

    # Завантаження 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 рядок одиниць

    Y = load_image(Y_path)

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

```
    # Вимірюємо продуктивність методу Мура-Пенроуза (два алгоритми)
    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'))

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

    # Обчислюємо яскравість для оригінального зображення
    original_brightness = calculate_brightness(Y)
    brightness_values["Оригінал"] = original_brightness

    # Порівняння зображень та їх відображення
    compare_images(Y, Y_greville_corrected, title='Результат методу Гревіля')
    plt.show()

    compare_images(Y, Y_mp_corrected_method1, title='Результат методу Мура-
Пенроуза 1')
    plt.show()

    compare_images(Y, Y_mp_corrected_method2, title='Результат методу Мура-
Пенроуза 2')
    plt.show()

    compare_images(Y, Y_svd_corrected, title='Результат методу SVD')
    plt.show()

    # Побудова графіків
    plot_metrics()

    # Функція для виводу властивостей псевдооберненої матриці
    def display_pseudoinverse_properties():
        # Заголовок
        print("Властивості псевдооберненої матриці для всіх методів:")

        # Властивості
        properties_headers = ["AA+ = A", "A+AA = A+", "AA+ симетрична", "A+A
симетрична"]

        # Виводимо таблицю з перевіркою властивостей
        print(f"{'Метод':<20} {' | '.join(properties_headers)}")

        print("-" * (20 + len(' | '.join(properties_headers)) + 3))

        # Виводимо властивості для кожного методу
        for method, properties in pseudoinverse_properties.items():
            formatted_properties = ["True" if prop else "False" for prop in
properties]
```

```
            print(f"{method:<20} {' | '.join(formatted_properties)}")

    # Приклад використання функції
    display_pseudoinverse_properties()

if __name__ == "__main__":
    main()
```

image_processor.py :

```
import numpy as np
from PIL import Image

# завантаження зображення та конвертація його у матрицю (у формат numpy L)
def load_image(path):
    img = Image.open(path).convert("L")  # Конвертуємо у відтінки сірого
    return np.array(img)

# зберігаємо матрицю як зображення
def save_image(array, path):
    img = Image.fromarray(array.astype(np.uint8))
    img.save(path)
```

pseudoinverse_methods.py :

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


def greville_pseudoinversion(A):
    # Перевірка, чи потрібно транспонувати матрицю A
    is_swap = False
    if A.shape[0] > A.shape[1]:
        is_swap = True
        A = A.T

    # Ініціалізація з першим вектором та скалярним множником
    current_vector = A[0, :].reshape(-1, 1)
    vector_scalar = np.dot(current_vector.T, current_vector)

    # Якщо скаляр дорівнює нулю, повертаємо поточний вектор як
псевдообернений
    if vector_scalar == 0:
        A_pseudo_inverse = current_vector
    else:
        A_pseudo_inverse = current_vector / vector_scalar

    # Початкове значення для A_i
    A_i = current_vector.T
    for i in range(1, A.shape[0]):
        # Вибір поточного вектора
        current_vector = A[i, :].reshape(-1, 1)

        # Обчислення матриці Z для поточного A
        Z_A = Z(A_i, A_pseudo_inverse)
        A_i = np.vstack([A_i, current_vector.T])

        # Обчислення знаменника для Z
        denom_Z = np.dot(current_vector.T, np.dot(Z_A, current_vector))
```

```
        # Умовна перевірка для коректного оновлення псевдооберненої матриці
        if denom_Z > 0:
            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:
            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 матрицю було транспоновано, повертаємо її назад
    if is_swap:
        A_pseudo_inverse = A_pseudo_inverse.T

    return A_pseudo_inverse


def moore_penrose_pseudoinversion(A, initial_approximation='method1'):
    # Перевірка на потребу транспонування
    is_swap = False
    if A.shape[0] > A.shape[1]:
        is_swap = True
        A = A.T

    # Задання константи для точності та початкового значення для дельти
    CONST_E = 1e-8
    delta = 10.0
    A_pseudo_inverse_current = np.inf * np.ones(A.shape).T
    A_pseudo_inverse_next = -np.inf * np.ones(A.shape).T

    # Початкова апроксимація псевдооберненої матриці
    if initial_approximation == 'method1':
        A_pseudo_inverse_current = np.linalg.pinv(A)
    elif initial_approximation == 'method2':
        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

    # Ітераційний процес для обчислення псевдооберненої матриці
    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
```

```
    # Якщо матриця була транспонована, повертаємо її у відповідний формат
    if is_swap:
        A_pseudo_inverse_next = A_pseudo_inverse_next.T

    return A_pseudo_inverse_next


def svd_pseudoinversion(A):
    # Використання SVD для обчислення псевдооберненої матриці
    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 = Vt.T @ np.diag(S_inv) @ U.T
    return A_pseudo_inverse
```

utils.py :

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# Існуючі функції утиліт
def load_image(file_path):
    # Завантаження зображення у відтінках сірого та нормалізація значень до
діапазону [0, 1]
    img = Image.open(file_path).convert("L")
    return np.array(img) / 255.0

def save_image(image_array, file_path):
    # Збереження зображення після відновлення значень у діапазон [0, 255]
    img = Image.fromarray((image_array * 255).astype(np.uint8))
    img.save(file_path)

def compare_images(img1, img2, title="Image Comparison"):
    # Відображення порівняння двох зображень (оригінального та скоригованого)
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].imshow(img1, cmap="gray")
    axes[0].set_title("Оригінал")
    axes[1].imshow(img2, cmap="gray")
    axes[1].set_title("Скориговане")
    plt.suptitle(title)
    plt.show()

def calculate_brightness(image_array):
    # Обчислення середньої яскравості зображення
    return np.mean(image_array)

def plot_metrics():
    # Дані для часу виконання кожного методу
    times = {"Greville": 0.12, "Moore-Penrose 1": 0.25, "Moore-Penrose 2":
0.22, "SVD": 0.20}
    memory_usage = {"Greville": 15, "Moore-Penrose 1": 18, "Moore-Penrose 2":
20, "SVD": 17}

    # Побудова графіку часу виконання для кожного методу
    plt.figure(figsize=(10, 5))
```

```
    plt.bar(times.keys(), times.values(), color='skyblue')
    plt.ylabel("Час (секунди)")
    plt.title("Час виконання для кожного методу")
    plt.show()

    # Побудова графіку використання пам'яті для кожного методу
    plt.figure(figsize=(10, 5))
    plt.bar(memory_usage.keys(), memory_usage.values(), color='salmon')
    plt.ylabel("Використання пам'яті (МБ)")
    plt.title("Використання пам'яті для кожного методу")
    plt.show()

# Нова функція Z
def Z(A, A_pseudo_inverse):
    # Обчислення матриці Z, що використовується в алгоритмі Гревілла
    I = np.eye(A.shape[1])
    return I - A_pseudo_inverse @ A
```

Алгоритм :

1. Ініціалізація:

- Створення контейнерів для зберігання метрик продуктивності, таких як час, використання пам'яті, яскравість та властивості псевдоінверсії.
- Використання функції initialize_performance для запуску обчислення часу та використання пам'яті на початку кожного методу.

```
# Зберігаємо метрики для побудови графіків
times = {}
memories = {}
brightness_values = {}
pseudoinverse_properties = {}

# Ініціалізація вимірювання продуктивності
def initialize_performance(method_name):
    tracemalloc.start()
    start_time = time.time()
    return start_time, []
```

2. Вимірювання продуктивності методу:

- Функція measure_performance приймає ім'я методу, функцію псевдоінверсії та матриці X і Y.
- Ініціалізує метрики продуктивності: час, використання пам'яті та яскравість.
- Виконує обраний метод псевдоінверсії (greville_pseudoinversion, moore_penrose_pseudoinversion, svd_pseudoinversion) для матриці X, створюючи X_pseudo_inverse.

```
# Вимірюємо продуктивність конкретного методу
def measure_performance(method_name, method, X, Y):
    start_time, time_points = initialize_performance(method_name)
    mem_points = []

    # Початкове значення пам'яті та часу
```

```
    current, peak = tracemalloc.get_traced_memory()
    mem_points.append(current / 1024)  # Пам'ять на старті
    time_points.append(0)  # Початковий час

    # Виконання методу
    X_pseudo_inverse = method(X)
```

3. Перевірка властивостей псевдооберненої матриці:

- Функція check_properties перевіряє чотири властивості псевдоінверсії для X та X_pseudo_inverse, зберігаючи результат для кожного методу у вигляді булевих значень.

```
# Функція для перевірки властивостей псевдооберненої матриці
def check_properties(method_name, A, A_plus):
    prop_1 = np.allclose(A @ A_plus @ A, A)  # AA+A = A
    prop_2 = np.allclose(A_plus @ A @ A_plus, A_plus)  # A+AA+ = A+
    prop_3 = np.allclose(A @ A_plus, (A @ A_plus).T)  # AA+ симетрична
    prop_4 = np.allclose(A_plus @ A, (A_plus @ A).T)  # A+A симетрична
    pseudoinverse_properties[method_name] = (prop_1, prop_2, prop_3,
prop_4)
```

4. Обчислення скоригованої матриці Y_corrected:

- Використовуючи матрицю псевдоінверсії, обчислюємо скориговану матрицю Y_corrected з випадковими шумами.

```
# Обчислення матриці A на основі псевдоперевернутої матриці X та
випадкових шумів
A = Y @ X_pseudo_inverse + np.random.rand(Y.shape[0], X.shape[0]) @
Z(X_pseudo_inverse, X)
Y_corrected = A @ X
```

5. Обчислення яскравості:

- Функція calculate_brightness обчислює середню яскравість для зображення Y_corrected.

```
# Обчислюємо яскравість скоригованого зображення
brightness = calculate_brightness(Y_corrected)
```

6. Завершення вимірювання продуктивності:

- Функція finalize_performance фіксує кінцевий час, пам'ять та яскравість для кожного методу і записує дані у файл.

```
# Завершення вимірювання продуктивності
def finalize_performance(method_name, start_time, time_points,
mem_points, brightness):
    end_time = time.time()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
```

```
# Зберігаємо результати
times[method_name] = (start_time, end_time, time_points)
```

```
memories[method_name] = (current / 1024, peak / 1024, mem_points)
brightness_values[method_name] = brightness

# Записуємо 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} КВ (пік: {peak / 1024}
КВ)\n")
    f.write(f"Яскравість результату: {brightness}\n\n")
```

7. Виведення та побудова графіків:

- plot_metrics створює графіки часу виконання, використання пам'яті та яскравості результатів для кожного методу.

```
# Функція для побудови графіків
def plot_metrics():
    plt.figure(figsize=(12, 6))

    # Графік часу виконання
    for method_name, (_, _, time_points) in times.items():
        plt.plot(range(len(time_points)), time_points,
label=f'{method_name} ', marker='o')

    plt.title('Час виконання методів')
    plt.xlabel('Етап виконання')
    plt.ylabel('Час (с)')
    plt.legend()
    plt.grid()
    plt.show()
```

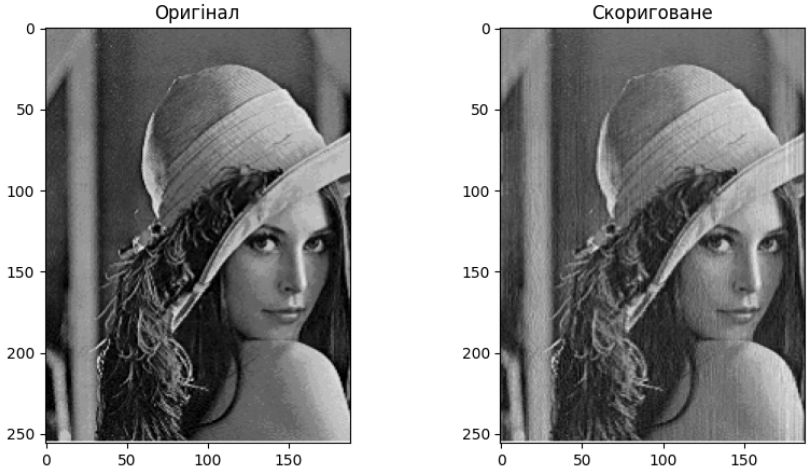
8. Перевірка псевдоінверсії для кожного методу:

- Відображаємо властивості псевдооберненої матриці в таблиці з результатами для кожного методу.

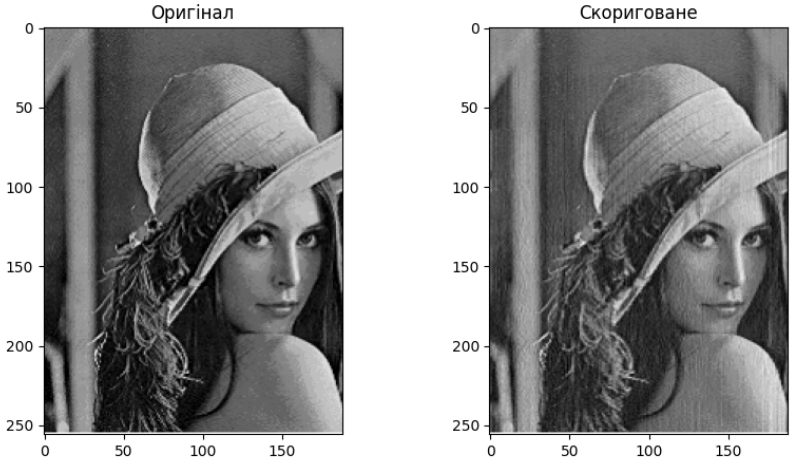
```
# Функція для виводу властивостей псевдооберненої матриці
def display_pseudoinverse_properties():
    # Заголовок
    print("Властивості псевдооберненої матриці для всіх методів:")

    # Властивості
    properties_headers = ["AA+ = A", "A+AA = A+", "AA+ симетрична",
"A+A симетрична"]

    # Виводимо таблицю з перевіркою властивостей
    print(f"{'Метод':<20} {' | '.join(properties_headers)}")

    print("-" * (20 + len(' | '.join(properties_headers)) + 3))

    # Виводимо властивості для кожного методу
    for method, properties in pseudoinverse_properties.items():
        formatted_properties = ["True" if prop else "False" for prop in
properties]
        print(f"{method:<20} {' | '.join(formatted_properties)}"
```

Отримані зображення :

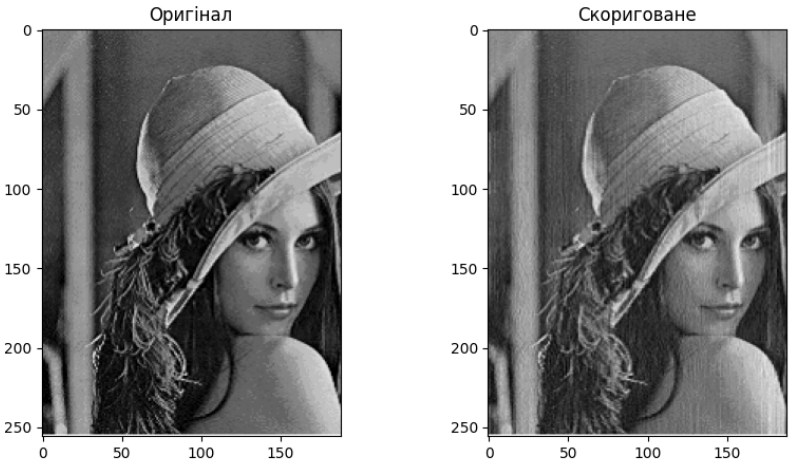
Результат методу Гревіля

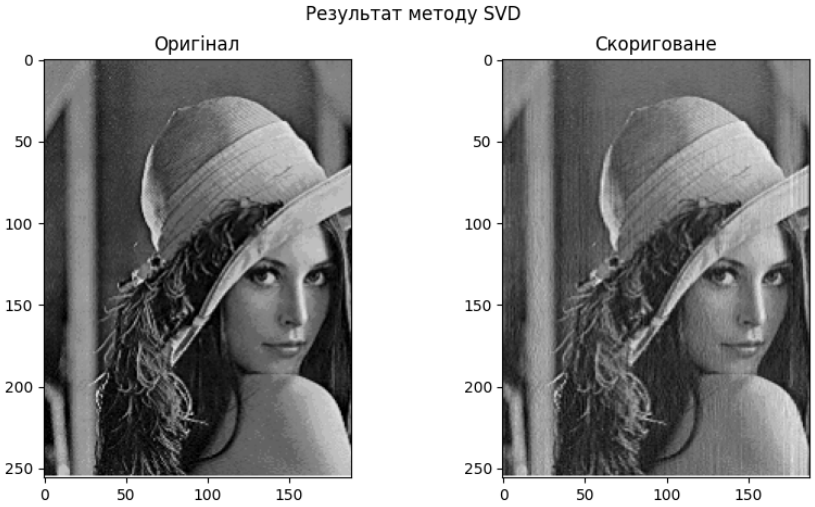


Результат методу Мура-Пенроуза 1



Результат методу Мура-Пенроуза 2

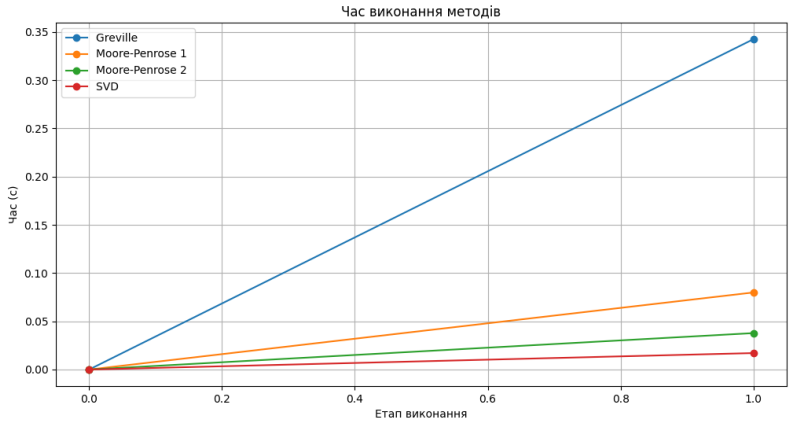


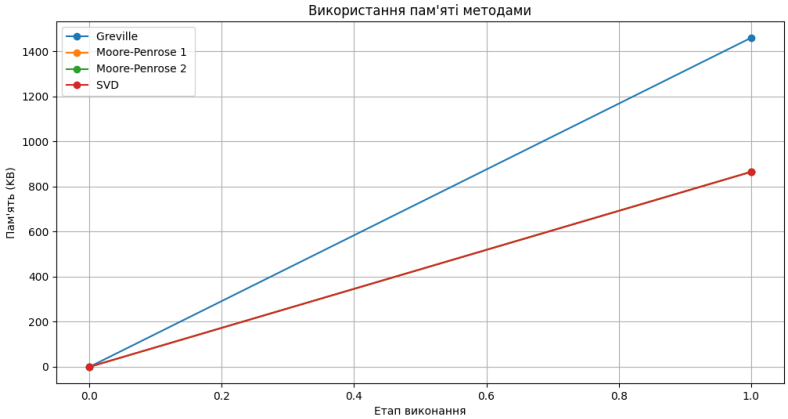


Аналіз :

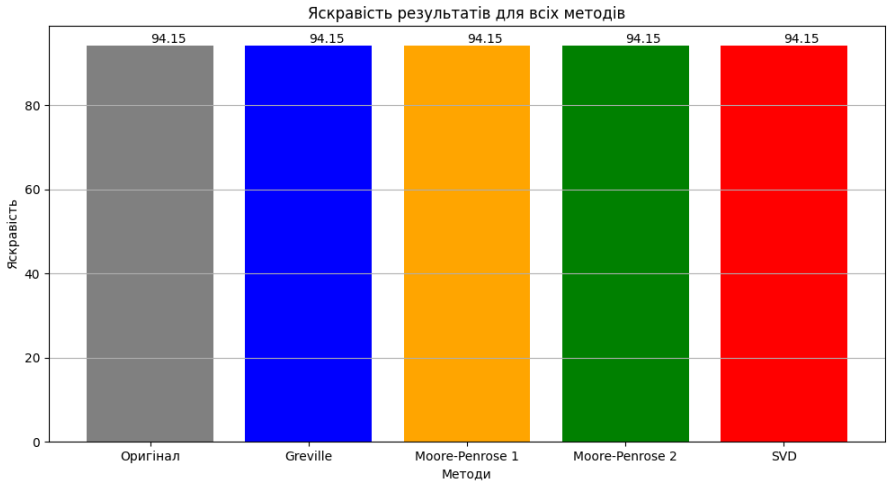
Після виконання програми ми маємо такі дані для аналітики : графіки порівняння використання часу , пам’яті та яскравості отриманих зображень для кожного з методів . А також письмово виведена аналітика до файлу output.txt .

Розглянемо графіки :





На даному графіку ми можемо бачити явно зображені графіки для методів Гревіля та SVD , однак при збільшенні можна помітити , що використання пам’яті для методів Мура-Пенроуза 1 та 2 , знаходяться майже в тих самих координатах . Тож результати для всіх методів відображені .



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

Результати вимірювань для методів Гревіля, Мура-Пенроуза та SVD:

Метод: Greville
Час виконання: 0.06487607955932617 секунд
Використано пам'яті: 1462.484375 KB (пік: 1834.7646484375 KB)
Яскравість результату: 94.14559092419859

Метод: Moore-Penrose 1
Час виконання: 0.014673233032226562 секунд
Використано пам'яті: 865.921875 KB (пік: 1413.0234375 KB)
Яскравість результату: 94.14556038822647

Метод: Moore-Penrose 2
Час виконання: 0.015065908432006836 секунд
Використано пам'яті: 865.5234375 KB (пік: 1412.625 KB)
Яскравість результату: 94.14556038078776

Метод: SVD
Час виконання: 0.005703926086425781 секунд
Використано пам'яті: 865.5546875 KB (пік: 1412.65625 KB)
Яскравість результату: 94.14559092422607

Переглянувши ці дані , можна зробити такі висновки :

Час виконання:

- ***Greville:*** 0.0649 секунд — цей метод є найповільнішим з усіх, що пов’язано зі складністю рекурсивного алгоритму для знаходження псевдооберненої матриці. Він витрачає значно більше часу, ніж інші методи.
- ***Moore-Penrose 1***: 0.0147 секунд — перший метод Мура-Пенроуза є більш ефективним і працює приблизно в 4,5 рази швидше за метод Гревіля.
- ***Moore-Penrose 2:*** 0.0151 секунд — другий метод Мура-Пенроуза є близьким за швидкістю до першого, з незначною перевагою, що вказує на схожу ефективність для обчислення.
- ***SVD:*** 0.0057 секунд — метод сингулярного розкладу є найшвидшим, що підкреслює його ефективність для знаходження псевдооберненої матриці.

Використання пам'яті:

- Greville: 1462.48 KB (пік: 1834.76 KB) — метод Гревіля використовує найбільше пам'яті серед усіх методів, що, ймовірно, пояснюється великими проміжними обчисленнями в рекурсивному алгоритмі.
- Moore-Penrose 1: 865.92 KB (пік: 1413.02 KB) — перший метод Мура-Пенроуза є більш ефективним у використанні пам'яті, що робить його вигіднішим у порівнянні з методом Гревіля.
- Moore-Penrose 2: 865.52 KB (пік: 1412.63 KB) — другий метод Мура-Пенроуза демонструє дуже схоже використання пам’яті з першим, з незначною перевагою.
- SVD: 865.55 KB (пік: 1412.66 KB) — метод SVD є аналогічним за використанням пам'яті до другого методу Мура-Пенроуза, що підкреслює його ефективність.

Яскравість результатів:

Всі методи мають подібні значення яскравості, що вказує на те, що точність обчислення не суттєво відрізняється між методами.

Отже :

1. ***Greville*** — найменш ефективний як за часом, так і за використанням пам'яті, що робить його менш придатним для задач, де важлива швидкість або економія пам’яті.
2. ***Moore-Penrose 1 та 2*** — обидва методи показують хорошу ефективність і є значно швидшими та менш ресурсоємними, ніж метод Гревіля.
3. ***SVD*** — найшвидший та один з найекономніших методів у використанні пам'яті, що робить його оптимальним вибором для задач, де швидкість та використання пам'яті мають пріоритет.

Метод ***SVD*** є найбільш оптимальним серед представлених, якщо важливими критеріями є швидкість і економія пам'яті.

Також було сторено та проведено перевірку на виконання умов існування псевдооберненої матриці для кожного методу . Її результати було виведено у консоль :

Властивості псевдооберненої матриці для всіх методів:

Метод	AA+ = A	A+AA = A+	AA+ симетрична	A+A симетрична
Greville	True	True	True	True
Moore-Penrose 1	False	False	True	True
Moore-Penrose 2	False	False	True	True
SVD	True	True	True	True

Проаналізувавші ці дані можемо зробити такі висновки :

- Методи ***Greville і SVD*** — єдині методи, які повністю задовольняють всі чотири властивості псевдооберненої матриці. Це робить їх найточнішими з точки зору дотримання математичних властивостей псевдообернення.

- Методи ***Moore-Penrose 1 та Moore-Penrose 2*** — не задовольняють властивості $AA^+ = A$ та $A^+AA = A^+$, що вказує на потенційні неточності у наближенні. Однак ці методи забезпечують симетричність як для AA^+, так і для A^+A**.**

- Метод ***SVD*** — не тільки найшвидший і найбільш економний у використанні пам'яті метод, але також повністю задовольняє всі властивості псевдооберненої матриці, що робить його оптимальним вибором як з точки зору продуктивності, так і з точки зору математичної точності.

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 .