Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук і кібернетики

Звіт
з лабораторної роботи №2
з моделювання складних систем
Варіант 4

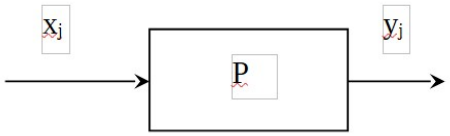
Виконав:
Студент групи ІПС-31
Міцкевич Костянтин Олександрович

Київ
2025

Мета лабораторної роботи

Побудова лінійної моделі з допомогою псевдообернених операторів.

Будемо вважати, що на вхід системи перетворення, математична модель якої невідома, поступають послідовно дані у вигляді m-1 вимірних векторів x_j. На виході системи спостерігається сигнал у вигляді вектора y_j розмірності p.



Постановка задачі

Для послідовності вхідних сигналів x_j, j =1, 2, …, n та вихідних сигналів y_j, j =1, 2, …, n знайти оператор p перетворення вхідного сигналу у вихідний.

Будемо шукати математичну модель оператора об'єкту в класі лінійних операторів $A\begin{pmatrix} x_j \\ 1 \end{pmatrix} = y_j,\ j=1, 2, \dots, n.$ (1)

Невідома матриця A математичної моделі об'єкту розмірності p × n. Систему (1) запишемо у матричній формі

$A\begin{pmatrix} x_1 & x_2 & \dots & x_n \\ 1 & 1 & \dots & 1 \end{pmatrix} = (y_1, y_2, \dots, y_n)$, або AX = Y, (2)

де $X = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ 1 & 1 & \dots & 1 \end{pmatrix}$ – матриця вхідних сигналів розмірності m × n,

Y = (y_1, y_2, …, y_n)– матриця вихідних сигналів розмірності p × n.

Матрицю X будемо інтерпретувати як двовимірне вхідне зображення, а матрицю Y вихідне зображення.

Тоді $A = YX^+ + VZ^T(X^T)$, де матриця

$$V = \begin{pmatrix} v_{(1)}^T \\ v_{(2)}^T \\ \vdots \\ v_{(p)}^T \end{pmatrix},$$

розмірності p × m, $Z(X^T) = I_m - XX^+$.

Формула Гревіля для псевдообернення матриці:

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

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

де $Z(A) = E - A^+A$, $R(A) = A^+(A^+)^T$.

Для першого кроку алгоритму $\left(a_1^T\right)^+ = \frac{a_1}{a_1^Ta_1}$, де $A = \begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{pmatrix}$.

Формула Мура - Пенроуза для знаходження оберненої (псевдооберненої) матриці:

$$A^+ = \lim_{\delta^2 \to 0} \left\{ \left(A^TA + \delta^2 E_n \right)^{-1} A^T \right\} = \lim_{\delta^2 \to 0} \left\{ A^T \left(AA^T + \delta^2 E_m \right)^{-1} \right\}.$$

матриця A розмірності m × n.

игнал – x1.bmp, вихідний сигнал – y4.bmp.

Основні терміни

1. Псевдообернена матриця (Moore-Penrose inverse)
Псевдообернена матриця A^+ — це узагальнення оберненої матриці для випадків, коли звичайна обернена матриця не існує (для прямокутних або вироджених матриць). Для матриці A розміру m×n псевдообернена матриця A^+ має розмір n×m і задовольняє чотирьом умовам Мура-Пенроуза:
1. $A \cdot A^+ \cdot A = A$
2. $A^+ \cdot A \cdot A^+ = A^+$
3. $(A \cdot A^+)^T = A \cdot A^+$ (симетричність)
4. $(A^+ \cdot A)^T = A^+ \cdot A$ (симетричність)

2. RMSE (Root Mean Square Error)
Середньоквадратична похибка — метрика для оцінки точності відновлення зображення:

$$\text{RMSE} = \sqrt{\frac{1}{n} * \sum (y_i - \bar{y}_i)^2}$$

де:
- Y — очікуване зображення
- $\hat{Y}$ — отримане зображення
- n — кількість пікселів

3. Оператор лінійного перетворення
Матриця A, що перетворює вхідне зображення X у вихідне зображення Y:
$Y = A \cdot X$
Для знаходження оператора A використовується псевдообернена матриця:
$A = Y \cdot X^+$

4. Нормалізація зображення
Приведення значень пікселів до діапазону [0, 1]

Хід роботи:

Мова реалізації: Python
Бібліотеки: time (вимірювання часу для методів), numpy (робота з матрицями та великими обрахунками), matplotlib.pyplot (побудови діаграм і візуалізації даних), cv2 (для зчитування файлів), psutil (для моніторингу пам’яті), os (для керування файлами, використовувався для збереження результатів в фото)

Крок 1. Підготовка даних

1. Завантаження вхідного зображення X (x1.bmp) та очікуваного зображення Y (y4.bmp)
2. Конвертація зображень у відтінки сірого (grayscale)
3. Нормалізація значень пікселів до діапазону [0, 1]
4. Виведення розмірів матриць та діапазонів значень

```python
def readImage():
    x_img = cv2.imread("x1.bmp", cv2.IMREAD_GRAYSCALE)
    y_img = cv2.imread("y4.bmp", cv2.IMREAD_GRAYSCALE)

    x = x_img.astype(float) / 255.0
    y = y_img.astype(float) / 255.0

    print(f"x size: {x.shape}, y size: {y.shape}")
    print(f"x range: [{x.min():.3f}, {x.max():.3f}]")
    print(f"y range: [{y.min():.3f}, {y.max():.3f}]")

    return x, y
```

```
x size: (140, 188), y size: (256, 188)
x range: [0.000, 1.000]
y range: [0.016, 1.000]
Розміри матриць: X=(140, 188), Y=(256, 188)
```

Крок 2. Реалізація методу Мура-Пенроуза

1. Ініціалізація параметрів ($\varepsilon = 10^{-6}$, $\delta_0 = 1000$)

2. Обчислення початкового наближення: $A_0 = A^S \cdot (A \cdot A^S + \delta^2 \cdot I)^{-1}$

3. Ітеративне зменшення δ вдвічі на кожному кроці

4. Перевірка збіжності: $\|A_0 - A^+\| < \varepsilon$

5. Повернення результату та кількості ітерацій

```python
def pseudoInverseMatrix_MoorePenrose(A, eps=1e-6, delta=1000):
    log = setupLogging('MoorePenrose')
    m, n = A.shape

    A0 = A.T @ np.linalg.inv(A @ A.T + delta**2 * np.identity(m))
    log(f'Початкова форма A0: {A0.shape}')

    delta = delta / 2

    iterations = 0
    while True:
        A_plus = A.T @ np.linalg.inv(A @ A.T + delta**2 * np.identity(m))

        if np.linalg.norm(A0 - A_plus, ord=2) < eps:
            log(f'Збіглося за {iterations} ітерацій\n')
            return A_plus, iterations

        delta = delta / 2
        A0 = A_plus
        iterations += 1

        # Захист від нескінченного циклу
        if iterations > 1000:
            log(f'ПОПЕРЕДЖЕННЯ: Досягнуто максимум ітерацій (1000)\n')
            return A_plus, iterations
```

```
=== Обчислення методом Мура–Пенроуза ===
  Розміри: X=(140, 188), Y=(256, 188)
[MoorePenrose] Початкова форма A0: (188, 140)
[MoorePenrose] Збіглося за 33 ітерацій

  X_pinv shape: (188, 140)
  A shape: (256, 140)
  Y_pred shape: (256, 188)
Час: 0.4640s, Пам'ять: 3.50MB, Операцій: 346265920, RMSE: 0.061069
```

Крок 3: Реалізація методу Гревілля

1. Обробка першого рядка: $A^{+}_1 = a_1 / (a_1^T \cdot a_1)$

2. Для кожного наступного рядка a_i:

- Обчислення проектора $Z = I - A^{+} \cdot A$

- Перевірка лінійної незалежності: $a_i^T \cdot Z \cdot a_i > \varepsilon$

- Застосування відповідної формули (для незалежного або залежного рядка)

- Розширення A^{+} новим стовпцем

3. Повернення результату та кількості ітерацій (m-1)

```python
def pseudoInverseMatrix_Greville(A, eps=1e-12, delta=None):
    log = setupLogging('Greville')
    A = np.array(A, dtype=float)
    m, n = A.shape

    a1 = A[0, :].reshape(-1, 1)
    denom = np.dot(a1.T, a1)
    if np.abs(denom) < eps:
        A_plus = np.zeros((n, 1))
    else:
        A_plus = np.dot(a1, 1 / denom)  # (a1) / (a1^T a1)
    current_A = np.array([A[0]])

    log("Step 0: Перший рядок оброблено")
    log(f"A_plus shape: {A_plus.shape}\n")

    for i in range(1, m):
        a = A[i, :].reshape(-1, 1)
        Z = np.identity(current_A.shape[1]) - np.dot(A_plus, current_A)
        aTZa = np.dot(a.T, np.dot(Z, a))[0, 0]

        if aTZa > eps:
            num1 = np.dot(Z, np.dot(a, np.dot(a.T, A_plus)))
            left = A_plus - num1 / aTZa
            right = np.dot(Z, a) / aTZa
            A_plus = np.hstack((left, right))
        else:
            R = np.dot(A_plus, A_plus.T)
            denom = 1 + np.dot(a.T, np.dot(R, a))[0, 0]
            num2 = np.dot(R, np.dot(a, np.dot(a.T, A_plus)))
            left = A_plus - num2 / denom
            right = np.dot(R, a) / denom
            A_plus = np.hstack((left, right))

        current_A = np.vstack([current_A, A[i]])
        log(f"Step {i}: A_plus shape: {A_plus.shape}")

    log(f"Завершено. Фінальна форма: {A_plus.shape}\n")
    return A_plus, m - 1
```

```
=== Обчислення методом Гревілля ===
  Розміри: X=(140, 188), Y=(256, 188)
[Greville] Step 0: Перший рядок оброблено
[Greville] A_plus shape: (188, 1)

[Greville] Step 1: A_plus shape: (188, 2)
[Greville] Step 2: A_plus shape: (188, 3)
[Greville] Step 3: A_plus shape: (188, 4)
```

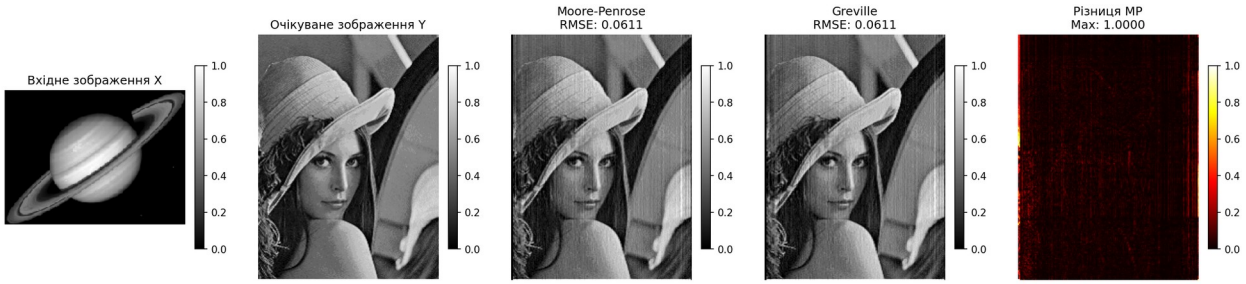
```
[Greville] Завершено. Фінальна форма: (188, 140)


  X_pinv shape: (188, 140)
  A shape: (256, 140)
  A shape: (256, 140)
  Y_pred shape: (256, 188)
Час: 0.0842s, Пам'ять: 0.58MB, Операцій: 537979999, RMSE: 0.061069
```

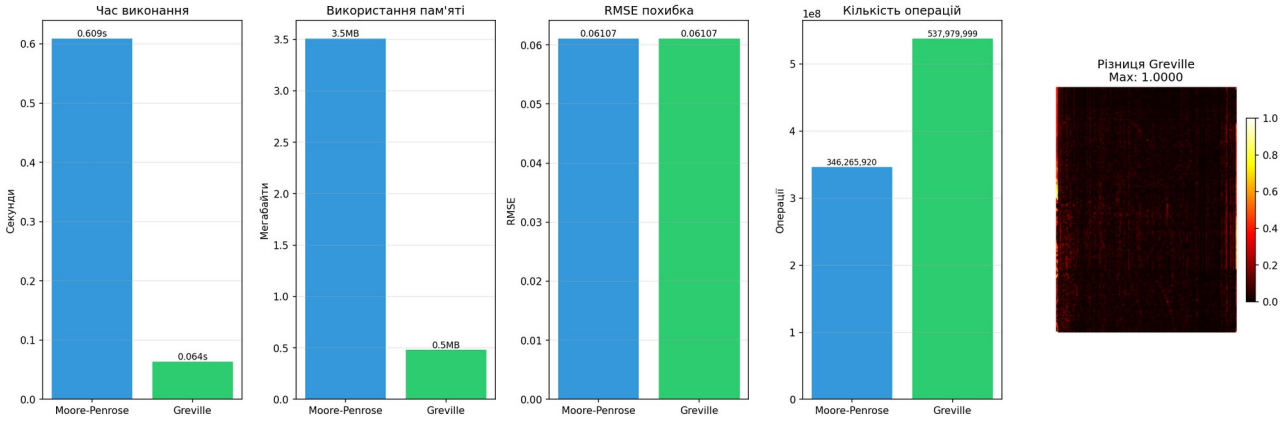
Крок 4: Обчислення, порівняння результатів та побудова графіка:

```python
def calculateOperator(X, Y, inversion_func, op_counter, eps=1e-6, delta=1000):
    start_time = time.time()
    start_mem = psutil.Process(os.getpid()).memory_info().rss / 1024 / 1024

    print(f"  Розміри: X={X.shape}, Y={Y.shape}")

    # Знаходимо псевдообернену матрицю для X
    X_pinv, iters = inversion_func(X, eps=eps, delta=delta)
    print(f"  X_pinv shape: {X_pinv.shape}")

    # Обчислюємо оператор A
    A = Y @ X_pinv
    print(f"  A shape: {A.shape}")

    # Обчислюємо результат
    Y_pred = A @ X
    print(f"  Y_pred shape: {Y_pred.shape}")

    # Обрізаємо значення до допустимого діапазону [0, 1]
    Y_pred = np.clip(Y_pred, 0, 1)

    end_time = time.time()
    end_mem = psutil.Process(os.getpid()).memory_info().rss / 1024 / 1024

    time_used = end_time - start_time
    memory_used = end_mem - start_mem
    ops = op_counter(X.shape[0], X.shape[1], iters if iters is not None else 1)
    error, mse, rmse = calculateError(Y, Y_pred)

    return Y_pred, time_used, memory_used, ops, error, mse, rmse
```

```python
def count_operations_greville(m, n, iterations=None):
    operations = n + n  # Initial dot product and division

    for i in range(1, m):
        operations += n * n * i  # A_plus @ current_matrix
        operations += n * n  # subtraction from identity
        operations += n * n  # z @ a
        operations += n  # a.T @ (z @ a)
        operations += n * i * i  # A_plus @ A_plus.T
        operations += n * i  # r @ a
        operations += n  # a.T @ (r @ a)
        operations += 1  # addition
        operations += n * i  # a.T @ A_plus
        operations += n * n  # z @ (a @ (a.T @ A_plus))
        operations += n * i  # subtraction and division
        operations += n * i  # r @ a and division
        operations += n * (i + 1)  # hstack

    return operations


def count_operations_moore_penrose(m, n, iterations):
    operations = 0

    # Initial calculation
    operations += m * n * m  # A @ A.T
    operations += m * m  # Adding delta**2 * np.identity(m)
    operations += m * m * m  # Matrix inversion
    operations += n * m * m  # A.T @ inv(...)

    # Per iteration
    operations += iterations * (
        m * n * m +  # A @ A.T
        m * m +  # Adding delta**2 * np.identity(m)
        m * m * m +  # Matrix inversion
        n * m * m +  # A.T @ inv(...)
        n * m +  # Matrix subtraction
        n * m  # Frobenius norm calculation
    )

    return operations
```

```
=== Порівняння результатів ===
Різниця у часі: 0.5451s (Greville швидше)
Різниця у пам'яті: 3.03MB
Різниця у пам'яті: 3.03MB
Різниця у RMSE: 0.000000
Різниця у RMSE: 0.000000
```


Вхідне зображення X
Очікуване зображення Y
Moore-Penrose
RMSE: 0.0611
Greville
RMSE: 0.0611
Різниця MP
Max: 1.0000


Час виконання
Секунди
0.609s
0.064s
Використання пам'яті
Мегабайти
3.5MB
0.5MB
RMSE похибка
RMSE
0.06107
0.06107
Кількість операцій
1e8
Операції
346,265,920
537,979,999
Moore-Penrose
Greville

Різниця Greville
Max: 1.0000