Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Моделювання складних систем

Лабораторна робота №2

“Побудова лінійної моделі з допомогою псевдообернених операторів”

Варіант №18

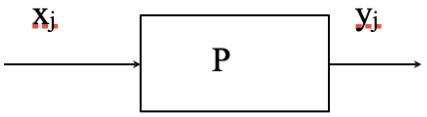
Виконала студентка групи ІПС-31

Сенечко Дана Володимирівна

Київ - 2025

Постановка задачі та основна теорія

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.



Для послідовності вхідних сигналів x_j, $j = 1, 2, \ldots, n$ та вихідних сигналів y_j, $j = 1, 2, \ldots, n$ знайти оператор P перетворення вхідного сигналу у вихідний.

Будемо шукати математичну модель оператора об'єкту в класі лінійних операторів

$$\mathbf{A}\begin{pmatrix}\mathbf{x}_j\\ 1\end{pmatrix} = \mathbf{y}_j,\ j = 1,2,\ldots,n.$$

Невідома матриця **A** математичної моделі об'єкту розмірності $p \times n$. Систему запишемо у матричній формі:

$$\mathbf{A}\begin{pmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_n\\ 1 & 1 & \ldots & 1\end{pmatrix} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n),$$

або

$$\mathbf{AX = Y},$$

де $\mathbf{X} = \begin{pmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_n\\ 1 & 1 & \ldots & 1\end{pmatrix}$ – матриця вхідних сигналів розмірності $m \times n$,

$\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$ – матриця вихідних сигналів розмірності $p \times n$.

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

Тоді

$$\mathbf{A = YX}^{+} + \mathbf{VZ}^{T}(\mathbf{X}^{T}),$$

де матриця

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

розмірності $p \times m$, $\mathbf{Z}(\mathbf{X}^T) = \mathbf{I}_m - \mathbf{X}\mathbf{X}^+$.

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

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

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

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

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

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

матриця A розмірності $m \times n$.

За варіантом роботи: вхідний сигнал – x3.bmp, вихідний сигнал – y10.bmp.

Основні терміни

Псевдообернена матриця (Moore-Penrose inverse)

Псевдообернена матриця A^+ – це узагальнення оберненої матриці для випадків, коли звичайна обернена матриця не існує, тобто для прямокутних або вироджених матриць. Для матриці A розміру $m \times n$ псевдообернена матриця A^+ має розмір $n \times m$ і задовільняє чотири умови:

1. $AA^{+}A = A$
2. $A^{+}AA^{+} = A^{+}$
3. $(AA^{+})^{*} = AA^{+}$
4. $(A^{+}A)^{*} = A^{+}A$

RMSE (Root Mean Square Error)

Середньоквадратична похибка – метрика, яка вимірює середнє значення абсолютної помилки між прогнозованими та фактичними значеннями (в даному випадку, для оцінки точності відновлення зображення):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y_i})^2},$$

де y – очікуване зображення, $\overline{y}$ – отримане зображення, а n – кількість пікселів.

Оператор лінійного перетворення

Матриця A, що перетворює вхідне зображення X у вихідне зображення Y: $Y = AX$. Для знаходження оператора A використовується псевдообернена матриця: $A = YX^{+}$.

Нормалізація зображення – приведення значень пікселів до діапазону $[0, 1]$.

Хід роботи

Програма реалізована мовою Python з використанням бібліотек NumPy (робота з матрицями і великими обрахунками), Matplotlib (побудова діаграм для візуалізації даних), time (вимірювання часу роботи методів), cv2 (зчитування файлів), psutil (моніторинг пам’яті), os (керування файлами, збереження результатів в зображення). Умовно поділимо її на кілька етапів:

1) Завантаження вхідного та очікуваного зображень, конвертація зображень у grayscale, нормалізація значень пікселів до діапазону [0, 1] та виведення розмірів матриць з діапазонами значень:

```python
def readImg():
    x_img = cv2.imread("x3.bmp", cv2.IMREAD_GRAYSCALE)
    y_img = cv2.imread("y10.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
```

```
dunnaya@MacBook-Air-Dana lab2 % python3 main.py
x size: (120, 188), y size: (140, 188)
x range: [0.000, 1.000]
y range: [0.000, 1.000]
Matrix sizes: X = (120, 188), Y = (140, 188)
```

2) Реалізація методі Мура-Пенроуза. Ініціалізація параметрів ($\varepsilon = 10^{-6}$, $\delta_0 = 1000$), обчислення початкового наближення: $A_0 = A^s(AA^s + \delta^2 I)^{-1}$, ітеративне зменшення δ вдвічі на кожному кроці, повернення збіжності $||A_0 - A^+|| < \varepsilon$ та повернення результату:

```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'Initial form 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'Converged after {iterations} iterations\n')
            return A_plus, iterations

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

        # protection against infinite loop
        if iterations > 1000:
            log(f'Warning: Maximum iterations reached (1000)\n')
            return A_plus, iterations
```

```
        Moore-Penrose Method
Sizes: X = (120, 188), Y = (140, 188)
[MoorePenrose] Initial form A0: (188, 120)
[MoorePenrose] Converged after 35 iterations

 X_pinv shape: (188, 120)
 A shape: (140, 120)
 Y_pred shape: (140, 188)
Time: 0.0475s, Memory: 2.00MB, Operations: 259224000, RMSE: 0.029164
```

3) Реалізація методу Гревіля. Обробка першого рядка ($A_1^+ = a_1/(a_1^T \cdot a_1)$); для кожного наступного рядка a_i – обчислення проектора $Z = I - A^+A$, перевірка лінійної незалежності $a_i^T Z a_j >$ ε, та застосування відповідної формули для незалежного/залежного рядка; розширення A^+ новим стовпцем. 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: first row processed.")
    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)
        quad_form = np.dot(a.T, np.dot(Z, a))[0, 0] # a^TZa

        if quad_form > eps:
            num1 = np.dot(Z, np.dot(a, np.dot(a.T, A_plus)))
            left = A_plus - num1 / quad_form
            right = np.dot(Z, a) / quad_form
            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"Finished. Final shape: {A_plus.shape}\n")
    return A_plus, m - 1
```

```
        Greville Method
Sizes: X = (120, 188), Y = (140, 188)
[Greville] Step 0: first row processed.
[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] Step 4: A_plus shape: (188, 5)
```

...

```
[Greville] Step 119: A_plus shape: (188, 120)
[Greville] Finished. Final shape: (188, 120)

 X_pinv shape: (188, 120)
 A shape: (140, 120)
 Y_pred shape: (140, 188)
Time: 0.0145s, Memory: 5.39MB, Operations: 378691339, RMSE: 0.029164
```

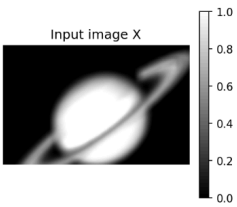
4) Обчислення оператора, оцінка результатів та побудова графіків:

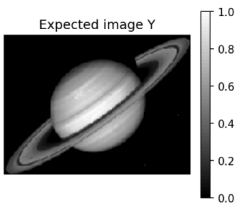
```
def calcOperator(X, Y, inversion_func, op_counter, eps = 1e-6, delta = 1000):
    start_time = time.time()
    start_mem = psutil.Process(os.getpid()).memory_info().rss / 1024 / 1024

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

    # find pseudoinverse of X
    X_pinv, iters = inversion_func(X, eps = eps, delta = delta)
    print(f" X_pinv shape: {X_pinv.shape}")

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

    # calculate result
    Y_pred = A @ X
    print(f" Y_pred shape: {Y_pred.shape}")
```

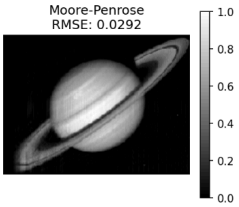
```
    # clip values to the range [0, 1]
    Y_pred = np.clip(Y_pred, 0, 1)

    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 = calcError(Y, Y_pred)

    return Y_pred, time_used, memory_used, ops, error, MSE, RMSE
```

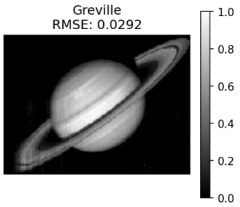
```
def count_operations_greville(m, n, iterations = None):
    operations = n + n  # init dot product and division

    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

    # init 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
```

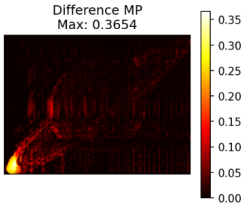
```
        Results comparison
Time difference: 0.0242s (Greville is faster)
Memory difference: 3.88MB
RMSE difference: 0.000000
```

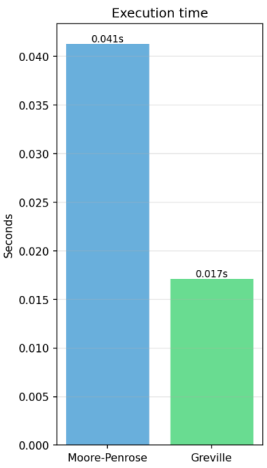

Input image X
1.0
0.8
0.6
0.4
0.2
0.0

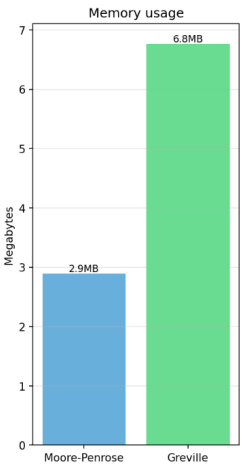

Expected image Y
1.0
0.8
0.6
0.4
0.2
0.0

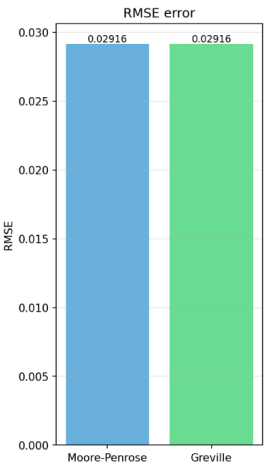

Moore-Penrose
RMSE: 0.0292
1.0
0.8
0.6
0.4
0.2
0.0

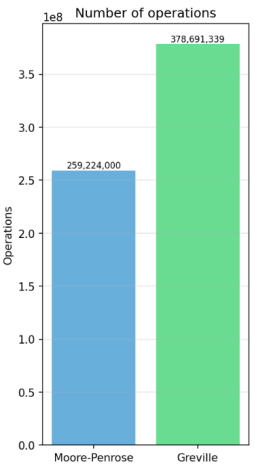

Greville
RMSE: 0.0292
1.0
0.8
0.6
0.4
0.2
0.0

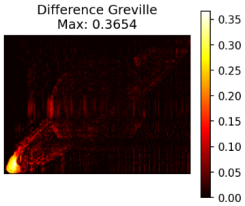

Difference MP
Max: 0.3654
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00


Execution time
Seconds
0.041s
0.017s
Moore-Penrose
Greville


Memory usage
Megabytes
2.9MB
6.8MB
Moore-Penrose
Greville


RMSE error
RMSE
0.02916
0.02916
Moore-Penrose
Greville


Number of operations
1e8
Operations
259,224,000
378,691,339
Moore-Penrose
Greville


Difference Greville
Max: 0.3654
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00