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
Вербицький Артем Віталійович

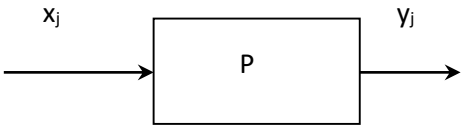
Київ
2024

1. 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 $\mathbf{x}_j$.

На виході системи спостерігається сигнал у вигляді вектора $\mathbf{y}_j$ розмірності p.



2. Постановка задачі

Для послідовності вхідних сигналів $\mathbf{x}_j$, $j=1,2,\dots,n$ та вихідних сигналів $\mathbf{y}_j$, $j=1,2,\dots,n$ знайти оператор P перетворення вхідного сигналу у вихідний.

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

$$\mathbf{A}\begin{pmatrix}\mathbf{x}_j\\1\end{pmatrix}=\mathbf{y}_j,\ j=1,2,\dots,n. \qquad (1)$$

Невідома матриця $\mathbf{A}$ математичної моделі об'єкту розмірності $p\times n$. Систему (1) запишемо у матричній формі

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

або

$$\mathbf{AX}=\mathbf{Y}, \qquad (2)$$

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

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

Матрицю $\mathbf{X}$ будемо інтерпретувати як двовимірне вхідне зображення, а матрицю $\mathbf{Y}$ вихідне зображення.

Тоді

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

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 $m \times n$.

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

3. Хід роботи

Програмна реалізація виконана мовою Python. Програма розділена на три файли: **MCS_Lab_2.py** – основний, **PseudoInverseMatrixMethods.py** – з методами знаходження псевдо-обернених матриць, **OperationsCounter.py** – з методами обрахунку кількості операцій.

Почнемо з ініціалізації всіх необхідних параметрів, а саме зчитаємо вхідне та вихідне зображення й побудуємо відповідні матриці, з якими надалі працюватимемо. Також одразу створимо оригінальне вихідне зображення у форматі png:

```
def main():
    X = readImage('x1.bmp')
    Y = readImage('y4.bmp')

    m = X.shape[1]
    X = np.vstack((X, np.ones((1, m))))

    saveImage(Y, 'original_Y.png')

    results = {}
```

```
def saveImage(X_, filename):
    results_dir = 'results'
    if not os.path.exists(results_dir):
        os.makedirs(results_dir)

    filepath = os.path.join(results_dir, filename)
    plt.figure(figsize=(10, 10))
    plt.imshow(X_, cmap='gray')
    plt.axis('off')
    plt.savefig(filepath, bbox_inches='tight', pad_inches=0)
    plt.close()
```

```python
def readImage(filename):
    X = imageio.imread(filename)
    return np.array(X)
```

Шукаємо обернені матриці оригінальним методом Мура-Пенроуза та з застосуванням градієнтного спуску, отримуємо вихідні зображення отримані цими методами та обраховуємо час виконання, використану пам’ять, кількість операцій та похибки відносно оригінального вихідного зображення:

```python
# Moore-Penrose method (original)
A_MP, time_MP, memory_MP, ops_MP = calculateOperator(X, Y, pimm.pseudoInverseMatrix_MoorePenrose, oc.count_operations_moo
Y_MP = applyOperator(X, A_MP)
error_norm_MP, mse_MP, rmse_MP = calculateError(Y, Y_MP)

# Moore-Penrose method (gradient descent)
A_MP_GD, time_MP_GD, memory_MP_GD, ops_MP_GD = calculateOperator(X, Y, pimm.pseudoInverseMatrix_MoorePenrose_GradientDesc
Y_MP_GD = applyOperator(X, A_MP_GD)
error_norm_MP_GD, mse_MP_GD, rmse_MP_GD = calculateError(Y, Y_MP_GD)
```

Порівнюємо похибки обох методів та обираємо точніший:

```python
# Compare and choose the better Moore-Penrose result
if rmse_MP < rmse_MP_GD:
    saveImage(Y_MP, 'moore_penrose_result.png')
    best_MP = {
        'method': 'Moore-Penrose',
        'time': time_MP,
        'memory': memory_MP,
        'operations': ops_MP,
        'error_norm': error_norm_MP,
        'mse': mse_MP,
        'rmse': rmse_MP
    }
else:
    saveImage(Y_MP_GD, 'moore_penrose_result.png')
    best_MP = {
        'method': 'Gradient Descent',
        'time': time_MP_GD,
        'memory': memory_MP_GD,
        'operations': ops_MP_GD,
        'error_norm': error_norm_MP_GD,
        'mse': mse_MP_GD,
        'rmse': rmse_MP_GD
    }
```

Зберігаємо отримані дані у масив results, аби в подальшому створити JSON:

```
results['Moore-Penrose'] = {
    'time': time_MP,
    'memory': memory_MP,
    'operations': ops_MP,
    'error_norm': error_norm_MP,
    'mse': mse_MP,
    'rmse': rmse_MP
}

results['Gradient Descent'] = {
    'time': time_MP_GD,
    'memory': memory_MP_GD,
    'operations': ops_MP_GD,
    'error_norm': error_norm_MP_GD,
    'mse': mse_MP_GD,
    'rmse': rmse_MP_GD
}

results['Best Moore-Penrose'] = best_MP
```

Аналогічні дії (окрім порівняння) робимо для методів Гревілля та Сингулярного розкладу матриці:

```
# Greville method
A_G, time_G, memory_G, ops_G = calculateOperator(X, Y, pimm.pseudoInverseMatrix_Greville, oc.count_operations_greville, e
Y_G = applyOperator(X, A_G)
saveImage(Y_G, 'greville_result.png')
error_norm_G, mse_G, rmse_G = calculateError(Y, Y_G)
results['Greville'] = {
    'time': time_G,
    'memory': memory_G,
    'operations': ops_G,
    'error_norm': error_norm_G,
    'mse': mse_G,
    'rmse': rmse_G
}

# SVD method
A_SVD, time_SVD, memory_SVD, ops_SVD = calculateOperator(X, Y, pimm.pseudoInverseMatrix_SVD, oc.count_operations_svd, eps
Y_SVD = applyOperator(X, A_SVD)
saveImage(Y_SVD, 'svd_result.png')
error_norm_SVD, mse_SVD, rmse_SVD = calculateError(Y, Y_SVD)
results['SVD'] = {
    'time': time_SVD,
    'memory': memory_SVD,
    'operations': ops_SVD,
    'error_norm': error_norm_SVD,
    'mse': mse_SVD,
    'rmse': rmse_SVD
}
```

Конвертуємо отримані дані та створюємо JSON файл, який використаємо для звіту в html форматі:

```python
# Convert NumPy types to Python native types
results = {k: {kk: numpy_to_python(vv) for kk, vv in v.items()} for k, v in results.items()}

# Save results to a JSON file
with open('results.json', 'w') as f:
    json.dump(results, f, indent=2)
```

4. Аналіз результатів

Порівнявши оригінальний *метод Мура-Пенроуза* та *з використанням Градієнтного спуску* **(Рис. 1)** спостерігаємо, що другий є швидшим та менш затратним у плані пам’яті. При цьому кількість операцій суттєво більша, ніж в оригінального методу. Точність зображення ж ідентична, тож обираємо метод з використанням Градієнтного спуску для подальшого аналізу.

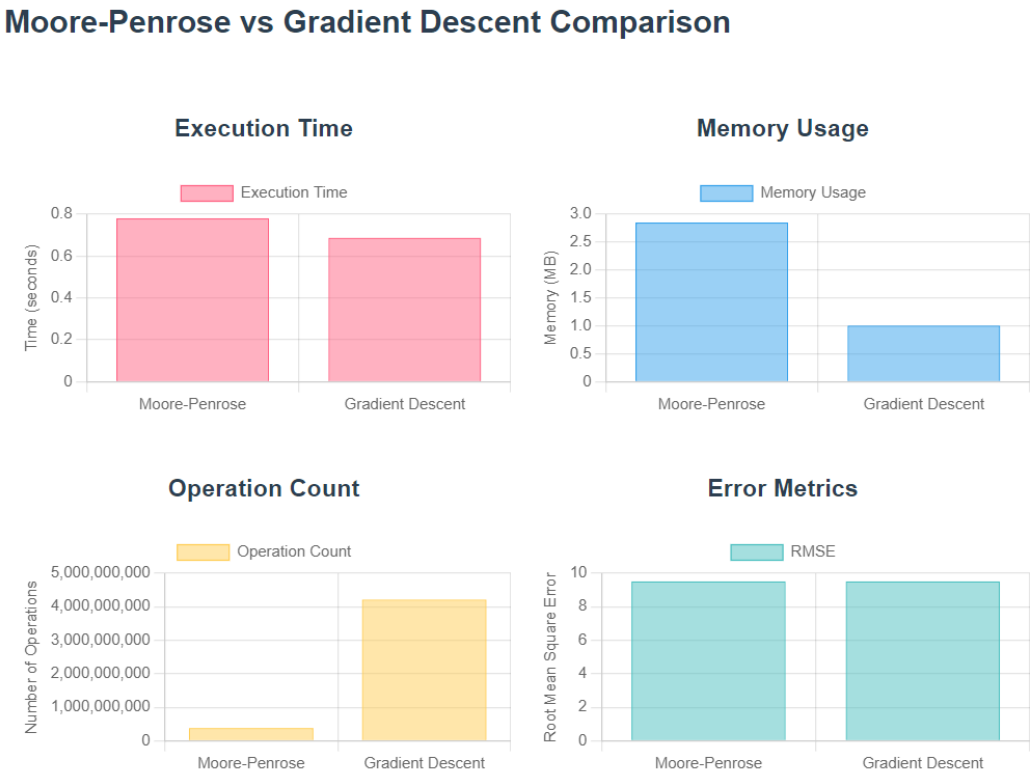


Рис. 1

Отож, на виході **(Рис. 2)** маємо три зображення: оригінальне, знайдене методом Градієнтного спуску та Гревілля:

Рис. 2

З отриманих графіків **(Рис. 3)** можемо побачити, що метод Сингулярного розкладу матриці значно поступається по швидкості, проте має значну перевагу у кількості використаної пам’яті. Значно виділяється метод Градієнтного спуску по кількості операцій, а також метод Гревілля по використанню пам’яті. Щодо точності знайдених матриць, то результати виявились ідентичними.

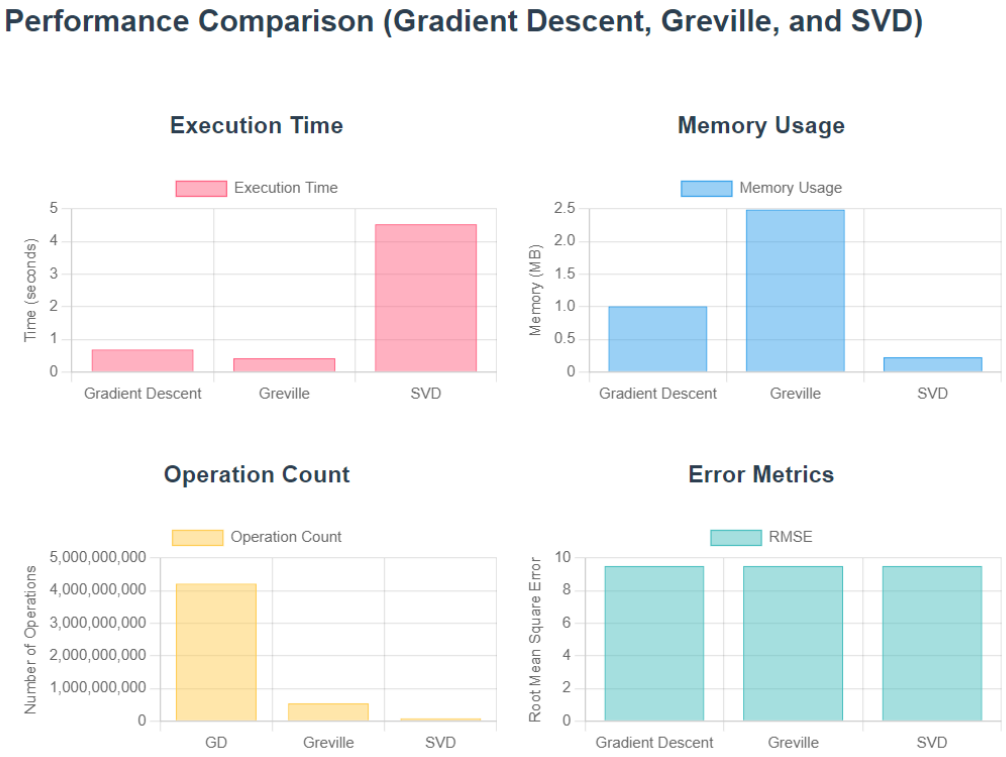


Рис. 3

5. Висновки

Отож, ми реалізували різні алгоритми знаходження псевдообернених операторів для побудови лінійної моделі на прикладі графічних зображень.

У ході роботи ми вияснили, що метод Мура-Пенроуза має широкий потенціал для оптимізації й застосування Градієнтного спуску – один з варіантів, який для цього можна застосувати.

Також ми з’ясували, що кожен метод має певні переваги та недоліки, хоч точність перетворень у нашому випадку виявилась ідентичною. Тож вибір конкретного методу залежить від наявних ресурсів та їхнього розподілу, який ми можемо забезпечити.