Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук так кібернетики

Лабораторна робота № 2

З курсу «Моделювання складних систем»

Варіант № 9

Виконав

Студент групи ІПС-31

Павлюченко Василь

**Постановка задачі**

Необхідно побудувати математичну модель перетворення вхідних даних у вихідні у класі лінійних операторів на основі формули:

де:

* X – матриця вхідних сигналів,
* Y – матриця вихідних сигналів,
* X+ - псевдообернена матриця Х,
* Z = I – XX+,
* V – додаткова матриця (у нашому випадку нульова).

Псевдообернену матрицю необхідно знайти двома методами: за формулою

Мура-Пенроуза та за формулою Гревілля.

**Хід роботи**

У варіанті 9 маємо вхідне зображення «x1.bmp» та вихідне «y9.bmp». Інтерпритуємо наші зображення як двовимірні масиви пікселів, X є нашим вхідним зображенням, а Y – вихідним. Додаємо до X рядок одиниць.

input = io.read\_image("x1.bmp")

output = io.read\_image("y9.bmp")

# calculate

X = np.array(input)

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

Y = np.array(output)

**Мура-Пенроуза**

Спочатку розглянемо пошук псевдооберненої матриці за формулою Мура-Пенроуза. З означення Мура-Пенроуза випливає, що для наближеного визначення псевдооберненої матриці можна застосовувати одну з формул:

Вибір залежить від того, що є меншим: m чи n. – число, яке підбирається експериментально. Алгоритм працює наступним чином:

1. Задаємо початкове значення , у нашому випадку
2. Розраховуємо початкове наближення ;
3. На кроці k нове значення ;
4. Наближення ;
5. Якщо , то зупинитись з , інакше k := k + 1 і продовжуємо ітераційний процес з пункту 3

def moore\_penrose(A):

    delta = 10

    m = A.shape[0]

    n = A.shape[1]

    E = np.identity(n)

    if m < n:

        E = np.identity(m)

    previous = multiply(A.T, np.linalg.inv(multiply(A, A.T) + delta \* E))

    while True:

        delta /= 2

        inverse = multiply(A.T, np.linalg.inv(multiply(A, A.T) + delta \* E))

        if find\_norm(previous, inverse) < 1e-10:

            return inverse

        else:

            previous = inverse

**Гревіля**

Далі розглянемо пошук псевдооберненої матриці за допомогою функції Гревіля. Необхідно зазначити, що у нашому випадку матриця Х = матриці А. Представляємо матрицю А у наступному вигляді:

Для першого кроку алгоритму маємо , при ; якщо ж , то . На наступному кроці додаємо другий рядок, шукаємо псевдообернену матрицю згідно формули Гревіля.

Де – проектор на ядро матриці A, .

def greville(matrix):

    # first step

    inverse = np.array(matrix)

    A0 = matrix[0]

    denominator = multiply(A0.T, A0)

    if (denominator == 0):

        inverse = np.vstack(A0)

    else:

        inverse = np.vstack(A0 / denominator)

    A = np.array([matrix[0]])

    n = matrix.shape[0]

    for i in range(1, n):

        a = matrix[i].reshape(-1, 1)

        Z = find\_Z(A, inverse)

        A = np.vstack([A, matrix[i]]) # add row

        denominator = multiply(a.T, Z, a)[0, 0]

        if np.abs(denominator) == 0: #  or try eps < 0.001

            R = find\_R(inverse)

            denominator = 1 + multiply(a.T, R, a)

            temp = inverse - multiply(R, a, a.T, inverse) / denominator

            inverse = np.hstack((temp, np.vstack((multiply(R, a) / denominator))))

        else:

            temp = inverse - multiply(Z, a, a.T, inverse) / denominator

            inverse = np.hstack((temp, np.vstack((multiply(Z, a) / denominator))))

    return inverse

**Матриця А**

Після знаходження псевдооберненої матриці X+, незалежно від обраного методу, підставляємо усе необіхдне в нашу формулу:

Z = np.identity(X.shape[0]) - multiply(X, X\_inverse)

V = np.zeros((Y.shape[0], X.shape[0]))

A = multiply(Y, X\_inverse) + multiply(V, Z)

# output

result = multiply(A, X)

io.write\_image("result.bmp", result)

Та отримуємо вихідне зображення за нашою моделлю.

**Порівняння зображень**

Рисунок Вихідне зображення "y9.bmp"

Рисунок Вхідне зображення "x1.bmp"

****

Рисунок Результат роботи програми

Рисунок Вихідне зображення "y9.bmp"

**Повний код програми**

Модуль imageio.py

from PIL import Image

def read\_image(path):

    try:

        with Image.open(path) as img:

            img = img.convert("L") # convert to grayscale

            pixels = list(img.getdata())

            width, height = img.size

            pixels = [pixels[i \* width:(i + 1) \* width] for i in range(height)]

            return pixels

    except Exception as e:

        print(f"Error reading image: {e}")

        return None

def write\_image(path, pixels):

    try:

        width = len(pixels[0])

        height = len(pixels)

        img = Image.new("L", (width, height))

        # Flatten the 2D list into 1D

        flat\_pixels = [pixel for row in pixels for pixel in row]

        img.putdata(flat\_pixels)

        img.save(path)

    except Exception as e:

        print(f"Error writing image: {e}")

Модуль utils.py

import numpy as np

def multiply(\*args):

    if len(args) == 0:

        return 0

    if len(args) == 1:

        return args[0]

    result = np.dot(args[0], args[1])

    for i in range(2, len(args)):

        result = np.dot(result, args[i])

    return result

def find\_Z(A, inverse):

    E = np.identity(A.shape[1])

    return E - multiply(inverse, A)

def find\_R(inverse):

    return multiply(inverse, inverse.T)

def find\_norm(m1, m2):

    return np.sqrt(np.sum((m1 - m2) \*\* 2))

Модуль inverse.py

import numpy as np

from utils import \*

def greville(matrix):

    # first step

    inverse = np.array(matrix)

    A0 = matrix[0]

    denominator = multiply(A0.T, A0)

    if (denominator == 0):

        inverse = np.vstack(A0)

    else:

        inverse = np.vstack(A0 / denominator)

    A = np.array([matrix[0]])

    n = matrix.shape[0]

    for i in range(1, n):

        a = matrix[i].reshape(-1, 1)

        Z = find\_Z(A, inverse)

        A = np.vstack([A, matrix[i]]) # add row

        denominator = multiply(a.T, Z, a)[0, 0]

        if np.abs(denominator) == 0: #  or try eps < 0.001

            R = find\_R(inverse)

            denominator = 1 + multiply(a.T, R, a)

            temp = inverse - multiply(R, a, a.T, inverse) / denominator

            inverse = np.hstack((temp, np.vstack((multiply(R, a) / denominator))))

        else:

            temp = inverse - multiply(Z, a, a.T, inverse) / denominator

            inverse = np.hstack((temp, np.vstack((multiply(Z, a) / denominator))))

    return inverse

def moore\_penrose(A):

    delta = 10

    m = A.shape[0]

    n = A.shape[1]

    E = np.identity(n)

    if m < n:

        E = np.identity(m)

    previous = multiply(A.T, np.linalg.inv(multiply(A, A.T) + delta \* E))

    while True:

        delta /= 2

        inverse = multiply(A.T, np.linalg.inv(multiply(A, A.T) + delta \* E))

        if find\_norm(previous, inverse) < 1e-10:

            return inverse

        else:

            previous = inverse

Модуль main.py

import numpy as np

import imageio as io

import inverse

from utils import \*

input = io.read\_image('x1.bmp')

output = io.read\_image('y9.bmp')

# calculate

X = np.array(input)

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

Y = np.array(output)

X\_inverse = inverse.moore\_penrose(X)

#X\_inverse = reverse.greville(X)

Z = np.identity(X.shape[0]) - multiply(X, X\_inverse)

V = np.zeros((Y.shape[0], X.shape[0]))

A = multiply(Y, X\_inverse) + multiply(V, Z)

# output

result = multiply(A, X)

io.write\_image("result.bmp", result)

# test

print("Норма:", find\_norm(result, Y))