Dokumentacja implementacji metody Gaussa-Jordana

I. Część pierwsza - Opis teoretyczny

Należy zaimplementować metodę Gaussa-Jordana do rozwiązywania układów równań liniowych. Program powinien obsługiwać układy równań o wymiarze 6x6 i uwzględniać różne przypadki rozwiązań (jedno rozwiązanie, brak rozwiązań, nieskończenie wiele rozwiązań).

Opis teoretyczny metody

Metoda Gaussa-Jordana jest rozszerzeniem metody eliminacji Gaussa i służy do rozwiązywania układów równań liniowych. Polega na przekształceniu macierzy rozszerzonej układu do postaci zredukowanej schodkowej (macierz jednostkowa w części współczynnikowej).

Główne kroki metody:

1. Utworzenie macierzy rozszerzonej **[A|b]**

2. Dla każdej kolumny k:

   - Wybór elementu głównego (pivot) - największy co do modułu element w kolumnie k

   - Zamiana wierszy (jeśli potrzebna)

   - Dzielenie wiersza przez element główny

   - Zerowanie pozostałych elementów w kolumnie k

Przykład ilustrujący metodę

Rozważmy prosty układ równań:

$3x + 2y - z = 12$

$-2x + y + 3z = 5$

$x - y + 2z = 4$

Macierz rozszerzona **[A|b]**:

[3 2 -1 | 12]

[-2 1 3 | 5]

[1 -1 2 | 4]

Kolejne kroki:

1. Normalizacja pierwszego wiersza:

[1 2/3 -1/3 | 4]

[-2 1   3 | 5]

[1 -1 2 | 4]

2. Eliminacja x z pozostałych wierszy:

[1 2/3 -1/3 | 4]

[0 7/3 7/3 | 13]

[0 -5/3 7/3 | 0]

(i tak dalej...)

II. Opis implementacji

Struktura programu

Program został zaimplementowany w języku Python z wykorzystaniem biblioteki NumPy. Główna funkcja `gauss_jordan` przyjmuje dwa argumenty:

- `A`: macierz współczynników (typ: np.array)

- `b`: wektor wyrazów wolnych (typ: np.array)

Kluczowe elementy implementacji:

1. Tworzenie macierzy rozszerzonej:

augmented_matrix = np.hstack((A, b.reshape(-1, 1)))

2. Wybór elementu głównego (pivot):

max_row = i + np.argmax(np.abs(augmented_matrix[i:, i]))

3. Zamiana wierszy:

augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]

4. Normalizacja wiersza:

augmented_matrix[i] /= augmented_matrix[i, i]

5. Eliminacja:

augmented_matrix[j] -= augmented_matrix[j, i] * augmented_matrix[i]

Obsługa przypadków szczególnych:

Program sprawdza i odpowiednio reaguje na następujące przypadki:

1. Brak rozwiązań - gdy występuje wiersz zerowy z niezerowym wyrazem wolnym

2. Nieskończenie wiele rozwiązań - gdy występuje wiersz zerowy z zerowym wyrazem wolnym

3. Jedno rozwiązanie - w pozostałych przypadkach

III. Prezentacja działania programu

```
import numpy as np

def gauss_jordan(A, b):
    augmented_matrix = np.hstack((A, b.reshape(-1, 1)))
    rows, cols = augmented_matrix.shape

    for i in range(rows):
        max_row = i + np.argmax(np.abs(augmented_matrix[i:, i]))
        if augmented_matrix[max_row, i] == 0:
            continue

        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]

        augmented_matrix[i] /= augmented_matrix[i, i]

        for j in range(rows):
            if j != i:
                augmented_matrix[j] -= augmented_matrix[j, i] *
augmented_matrix[i]

    solution = augmented_matrix[:, -1]

    for i in range(rows):
        if np.allclose(augmented_matrix[i, :-1], 0) and not
np.isclose(augmented_matrix[i, -1], 0):
            return "Brak rozwiązań"

    if any(np.allclose(augmented_matrix[i, :-1], 0) for i in range(rows)):
        return "Nieskończenie wiele rozwiązań"

    return np.round(solution, decimals=3)

# Wprowadzanie danych
#print("Podaj liczbę równań:")
#n = int(input())
n=6
print(f"Podaj współczynniki macierzy A ({n}x{n}):")
```

```python
A = []
for i in range(n):
    row = list(map(float, input(f"Wiersz {i+1}: ").split()))
    A.append(row)
A = np.array(A, dtype=float)

print("Podaj wyrazy wolne wektora b:")
b = list(map(float, input().split()))
b = np.array(b, dtype=float)

# Rozwiązywanie układu równań
result = gauss_jordan(A, b)

if isinstance(result, str):
    print(result)
else:
    for i, val in enumerate(result):
        print(f"x{i+1} = {val}")
```

Przykład 1: Układ z jednym rozwiązaniem

Dane wejściowe:

A = [

[2, 1, -1, 0, 0, 0],

[-3, -1, 2, 0, 0, 0],

[1, 2, 3, -1, 0, 0],

[0, 0, -1, 2, 1, 0],

[0, 0, 0, 1, -1, 2],

[0, 0, 0, 0, 1, -2]

]

b = [8, -11, -3, 4, 6, -5]

Wynik:

```
PS C:\Users\Fenus\Documents\analiza_danych> & "C:/Program Files/Python312/python.exe" c:/Users/Fenus/Documents/analiza_danych/zad2.py
x1 = -1.5
x2 = 6.5
x3 = -4.5
x4 = 1.0
x5 = -2.5
x6 = 1.25
PS C:\Users\Fenus\Documents\analiza_danych>
```

Przykład 2: Układ z nieskończenie wieloma rozwiązaniami

Dane wejściowe:

A = [

[2, -1, 1, 1, 1, 1],

[4, -2, 2, 2, 2, 2],

[1, 1, 1, 0, 0, 0],

[0, 0, 0, 2, -1, 1],

[0, 0, 0, 4, -2, 2],

[0, 0, 0, 1, 1, 1]

]

b = [4, 8, 3, 2, 4, 2]

Wynik:

```
PS C:\Users\Fenus\Documents\analiza_danych> & "C:/Program Files/Python312/python.exe" c:/Users/Fenus/Documents/analiza_danych/zad2.py
Nieskończenie wiele rozwiązań
PS C:\Users\Fenus\Documents\analiza_danych>
```

Przykład 3: Układ bez rozwiązań

Dane wejściowe:

A = [

[1, 1, 1, 1, 1, 1],

[1, 1, 1, 1, 1, 1],

[2, 1, 1, 1, 1, 1],

[1, 2, 1, 1, 1, 1],

[1, 1, 2, 1, 1, 1],

[1, 1, 1, 2, 1, 1]

]

b = [1, 2, 2, 3, 4, 5]

Wynik:

```
PS C:\Users\Fenus\Documents\analiza_danych> & "C:/Program Files/Python312/python.exe" c:/Users/Fenus/Documents/analiza_danych/zad2.py
Brak rozwiązań
PS C:\Users\Fenus\Documents\analiza_danych>
```

Wnioski

Zaimplementowana metoda Gaussa-Jordana skutecznie rozwiązuje układy równań liniowych o wymiarze 6x6. Program poprawnie identyfikuje i obsługuje wszystkie możliwe przypadki rozwiązań. Dzięki wykorzystaniu biblioteki NumPy, implementacja jest zwięzła i efektywna obliczeniowo. Dokładność obliczeń jest kontrolowana przez zaokrąglanie wyników do trzech miejsc po przecinku, co jest wystarczające dla większości zastosowań praktycznych.