Sprawozdanie z laboratorium 7 - Rozwiązania układów równań linowych

Hubert Miklas

13-05-2025

1 Wstęp

Tematem laboratorium było rozwiązywanie układów równań liniowych, korzystając z metody LU, rozkładu QR i inwersji macierzy.

2 Treści zadań

- Napisz program, który:
  1. Jako parametr pobiera rozmiar układu równań n
  2. Generuje macierz układu $A(n \times n)$ i wektor wyrazów wolnych $b(n)$
  3. Rozwiązuje układ równań $Ax = b$ na trzy sposoby:
     (a) poprzez dekompozycję LU macierzy A: $A = LU$;
     (b) poprzez odwrócenie macierzy A: $x = A^{-1}b$, sprawdzić czy $AA^{-1} = I$ i $A^{-1}A = I$ (macierz jednostkowa);
     (c) poprzez dekompozycję QR macierzy A: $A = QR$.
  4. Sprawdzić poprawność rozwiązania (tj. czy $Ax = b$)
  5. Zmierzyć całkowity czas rozwiązania układu.
  6. Porównać czasy z trzech sposobów: poprzez dekompozycję LU, poprzez **odwrócenie macierzy** i poprzez **dekompozycję QR**
- **Zadanie domowe:** Narysuj wykres zależności całkowitego czasu rozwiązywania układu (LU, QR, odwrócenie macierzy) od rozmiaru układu równań. Wykonaj pomiary dla 5 wartości z przedziału od 10 do 100.
  Uwaga: można się posłużyć funkcjami z biblioteki numerycznej dla danego języka programowania.

3 Metodyka

W eksperymencie wykorzystano następującą procedurę pomiarową:

1. **Generowanie danych:** dla każdego rozmiaru układu $n \in \{10, 30, 50, 70, 100\}$ generowano losową macierz $A \in \mathbb{R}^{n\times n}$ oraz wektor prawej strony $b \in \mathbb{R}^n$ o wartościach całkowitych z przedziału $[0, 10)$.
2. **Metody rozwiązania:**

- **Dekompzycja LU [2]:** z wykorzystaniem funkcji `scipy.linalg.lu` oraz `solve_triangular`.
- **Odwrócenie macierzy:** obliczenie A^{-1} funkcją `numpy.linalg.inv` i przemnożenie przez b.
- **Dekompzycja QR [3]:** obliczenie $A = QR$ przez `numpy.linalg.qr` i rozwiązanie trójkątnego układu.

3. **Weryfikacja poprawności:** dla każdej metody sprawdzano, czy $\|Ax - b\|_\infty < 10^{-8}$.
4. **Pomiar czasu:** czas każdej operacji mierzono przy pomocy modułu `time`, bez uwzględniania czasu generowania danych.
5. **Powtarzalność:** dla każdej liczby danych uruchomiono **5 niezależnych pomiarów**, a w wynikach wykorzystano średnią arytmetyczną czasu.
6. **Wizualizacja wyników:** zebrane czasy przedstawiono na wykresie zależności czasu od n, z oddzielnymi krzywymi dla każdej metody.

4 Zadanie: Porównanie metod wyznaczania układów równań liniowych

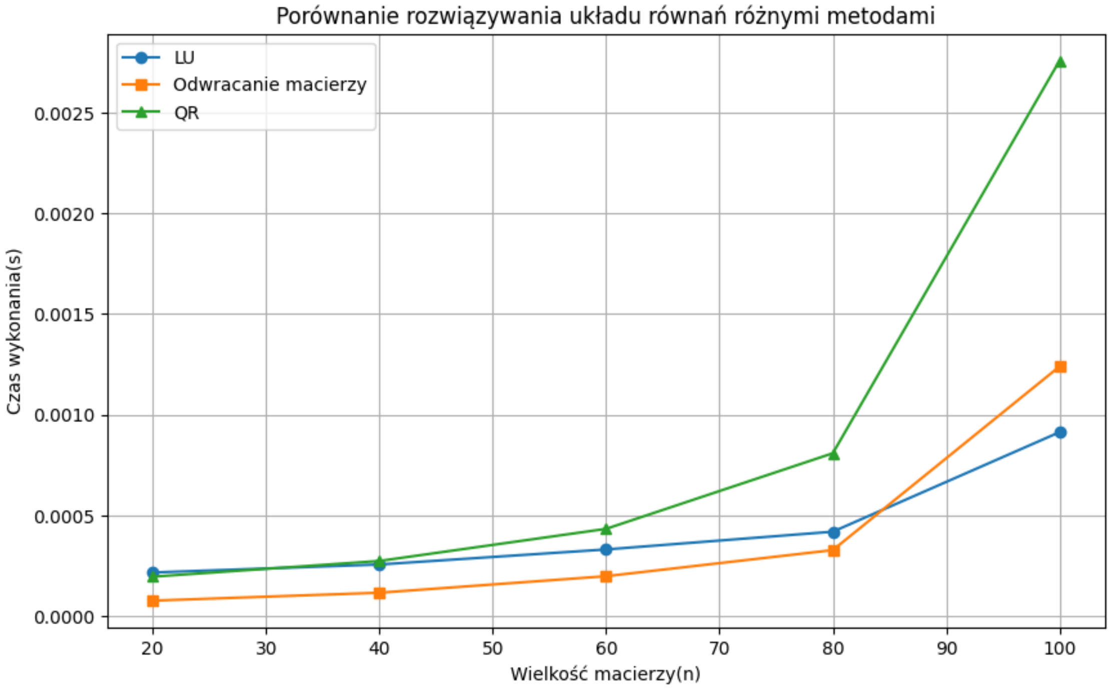
4.1 Opis rozwiązania

Program w Pythonie realizuje następujące kroki:

1. Generuje losową macierz A i wektor b.
2. Dla każdej z trzech metod:
   - mierzy czas wykonania,
   - oblicza rozwiązanie x,
   - weryfikuje, że $Ax \approx b$,
   - (dla odwrócenia) dodatkowo sprawdza własności macierzy odwrotnej $AA^{-1} = I$.
3. Wyświetla czasy, poprawność i porównuje wektory x między metodami.
4. Czynność powtarzana jest wielokrotnie dla ujednolicenia wyników i poprawnego porównania metod
5. Na podstawie przygotowanej listy rozmiarów generuje wykres porównawczy.

4.2 Wynikowy wykres

Program generuje wykres poniżej 1.



Rysunek 1: Zależność czasu wykonania od ilości danych dla metod LU, QR i inwersji

4.3 Kod w Python-ie rozwiązujący zadanie

```
import numpy as np
import time
import matplotlib.pyplot as plt
from random import randrange
import scipy.linalg

def gen_A(n, a=0, b=10):
    return [[randrange(a, b) for _ in range(n)] for _ in range(n)]

def gen_b(n, a=0, b=10):
    return [randrange(a, b) for _ in range(n)]

def solve_inverse_np(A, b):
    A_np = np.array(A)
    b_np = np.array(b)

    start = time.time()
    A_inv = np.linalg.inv(A_np)
    x = A_inv @ b_np
    end = time.time()

    I1 = A_np @ A_inv
    I2 = A_inv @ A_np

    identity = np.eye(len(A))
    is_I1 = np.allclose(I1, identity)
    is_I2 = np.allclose(I2, identity)

    print(f"A*A^(-1) = I: {is_I1}")
    print(f"A^(-1)*A = I: {is_I2}")
```

```python

    return x, end - start

def solve_LU(A, b):
    A_np = np.array(A)
    b_np = np.array(b)

    start = time.time()
    P, L, U = scipy.linalg.lu(A_np)

    y = scipy.linalg.solve_triangular(L, P @ b_np, lower=True)

    x = scipy.linalg.solve_triangular(U, y, lower=False)
    end = time.time()

    return x, end - start

def solve_QR(A, b):
    A_np = np.array(A)
    b_np = np.array(b)

    start = time.time()
    Q, R = np.linalg.qr(A_np)

    x = np.linalg.solve(R, Q.T @ b_np)
    end = time.time()

    return x, end - start

def verify_solution(A, b, x):
    A_np = np.array(A)
    b_np = np.array(b)
    x_np = np.array(x)

    result = A_np @ x_np
    is_correct = np.allclose(result, b_np)

    return is_correct

def plot_times(sizes, trials = 100):
    times_lu = []
    times_inv = []
    times_qr = []

    for n in sizes:
        sum_time_lu = 0
        sum_time_inv = 0
        sum_time_qr = 0
        for k in range(trials):
            print(f"Calculating for size n={n}")
            A = gen_A(n)
            b = gen_b(n)

            _, time_lu = solve_LU(A, b)
            sum_time_lu += time_lu

            _, time_inv = solve_inverse_np(A, b)
            sum_time_inv += time_inv

            _, time_qr = solve_QR(A, b)
```

```
            sum_time_qr += time_qr

        times_lu.append(sum_time_lu/trials)
        times_inv.append(sum_time_inv/trials)
        times_qr.append(sum_time_qr/trials)

    plt.figure(figsize=(10, 6))
    plt.plot(sizes, times_lu, 'o-', label='LU')
    plt.plot(sizes, times_inv, 's-', label='Odwracanie macierzy')
    plt.plot(sizes, times_qr, '^-', label='QR')
    plt.xlabel('Wielko  macierzy(n)')
    plt.ylabel('Czas wykonania(s)')
    plt.title('Por wnanie rozwi zywania uk adu r wna  r nymi metodami')
    plt.legend()
    plt.grid(True)
    plt.show()

def main():
    # n = int(input("Podaj liczb  n (rozmiar uk adu r wna ): "))
    n = 10
    A = gen_A(n)
    b = gen_b(n)

    print(f"Rozwi zywanie uk adu {n} r wna ...")

    x_lu, time_lu = solve_LU(A, b)
    is_correct_lu = verify_solution(A, b, x_lu)
    print(f"LU Decomposition: {time_lu:.6f} seconds, solution correct: {is_correct_lu}")

    x_inv, time_inv = solve_inverse_np(A, b)
    is_correct_inv = verify_solution(A, b, x_inv)
    print(f"Matrix Inversion: {time_inv:.6f} seconds, solution correct: {is_correct_inv}")

    x_qr, time_qr = solve_QR(A, b)
    is_correct_qr = verify_solution(A, b, x_qr)
    print(f"QR Decomposition: {time_qr:.6f} seconds, solution correct: {is_correct_qr}")

    print("\Por wnanie rozwi za :")
    print(f"LU i inwersja daj  ten sam wynik: {np.allclose(x_lu, x_inv)}")
    print(f"LU i QR daj  ten sam wynik: {np.allclose(x_lu, x_qr)}")
    print(f"Inwersja i QR daj  ten sam wynik: {np.allclose(x_inv, x_qr)}")

    print(f"Generowanie wykresu...")

    sizes = [20 + 20 * i for i in range(5)]
    plot_times(sizes)

if __name__ == "__main__":
    main()
```

Wnioski

Na podstawie przeprowadzonych eksperymentów można sformułować następujące wnioski:

- **Skuteczność metod:** Wszystkie trzy metody (LU, odwrócenie macierzy, QR) poprawnie wyznaczają rozwiązanie układu równań liniowych, co potwierdza norma reszt ($\|Ax-b\|_\infty$) poniżej progu 10^{-8}.

- **Czas obliczeń:**
  1. **Dekompzycja LU** okazała się najbardziej wydajna czasowo dla większości rozmiarów macierzy. Jej czas wykonania wzrastał najwolniej, co jest zgodne z teoretyczną analizą złożoności $\mathcal{O}(n^3)$ z niskim stałym narzutem. Metoda ta uchodzi za optymalną [1] dla rozwiązywania dużych układów równań z dobrze uwarunkowanymi macierzami.
  2. **Odwracanie macierzy** wykazało zaskakująco dobrą wydajność dla mniejszych rozmiarów (do $n = 80$), jednak przy większych macierzach (np. $n = 100$) czas gwałtownie wzrasta, co wynika z kosztownego obliczania odwrotności oraz dodatkowego mnożenia przez wektor. Ta metoda nie powinna być stosowana do rozwiązywania układów równań, ze względu na jej niestabilność i koszt [1].
  3. **Dekompozycja QR** była najwolniejsza spośród trzech metod, zwłaszcza dla większych rozmiarów. Dodatkowe koszty wynikają z procesu ortogonalizacji (np. metoda Householdera), co czyni ją mniej efektywną czasowo. Jednakże, QR oferuje lepszą stabilność numeryczną [3], co czyni ją preferowaną metodą w zastosowaniach takich jak regresja liniowa.

- Do rozwiązywania dużych układów równań liniowych rekomenduje się dekompozycję LU ze względu na jej efektywność. QR może być wskazane w przypadkach, gdy najważniejsza jest stabilność numeryczna. Odwracanie macierzy należy ograniczyć do sytuacji, gdy rzeczywiście potrzebna jest macierz odwrotna.

- Uzyskane czasy wykonania były stabilne w kolejnych pomiarach, co świadczy o rzetelności pomiarów i niewielkim wpływie czynników losowych.

Literatura

[1] dr inż. Katarzyna Rycerz. Wykład z przedmiotu metody obliczeniowe w nauce i technice. Akademia Górniczo-Hutnicza im. Stanisława Staszica, 2025.

[2] Wikipedia. Metoda LU — Wikipedia, the free encyclopedia. `http://pl.wikipedia.org/w/index.php?title=Metoda\%20LU&oldid=70887012`, 2025. [Online; accessed 13-May-2025].

[3] Wikipedia. Rozkład QR — Wikipedia, the free encyclopedia. `http://pl.wikipedia.org/w/index.php?title=Rozk\%C5\%82ad\%20QR&oldid=64344398`, 2025. [Online; accessed 13-May-2025].