Laboratorium 8

Układy równań – metody bezpośrednie

Bartłomiej Szubiak

30.04.2024

Zadanie:

Napisz program, który:

1. Jako parametr pobiera rozmiar układu równań n
2. Generuje macierz układu A(nxn) 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*

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.

Kod python z użyciem biblioteki numpy:

```
import numpy as np
import scipy.linalg
import time

def generate_matrix_and_vector(n):
    A = np.random.rand(n, n)
    b = np.random.rand(n)
    return A, b

def lu_decomposition(A, b):
    start = time.time()
    P, L, U = scipy.linalg.lu(A)
```

```
    x = scipy.linalg.solve_triangular(U, scipy.linalg.solve_triangular(L, P.T
@ b, lower=True))
    end = time.time()
    return x, end - start

def matrix_inversion(A, b):
    start = time.time()
    A_inv = np.linalg.inv(A)
    x = A_inv @ b
    end = time.time()
    identity_check = np.allclose(A @ A_inv, np.eye(A.shape[0])) and
np.allclose(A_inv @ A, np.eye(A.shape[0]))
    return x, identity_check, end - start

def qr_decomposition(A, b):
    start = time.time()
    Q, R = scipy.linalg.qr(A)
    x = scipy.linalg.solve_triangular(R, Q.T @ b)
    end = time.time()
    return x, end - start

def check_solution(A, b, x):
    return np.allclose(A @ x, b)

def main(n):
    A, b = generate_matrix_and_vector(n)

    x_lu, time_lu = lu_decomposition(A, b)
    print("LU Decomposition - correct?:", check_solution(A, b, x_lu), ", Time
taken:", time_lu)

    x_inv, identity_check, time_inv = matrix_inversion(A, b)
    print("Matrix Inversion - correct?:", check_solution(A, b, x_inv), ", Time
taken:", time_inv, ", identity check?:", identity_check)

    x_qr, time_qr = qr_decomposition(A, b)
    print("QR Decomposition - correct?:", check_solution(A, b, x_qr), ", Time
taken:", time_qr)

if __name__ == "__main__":
    n = int(input(">>"))
    main(n)
```

Przykład użycia:

>>100

LU Decomposition - correct?: True , Time taken: 0.014429569244384766

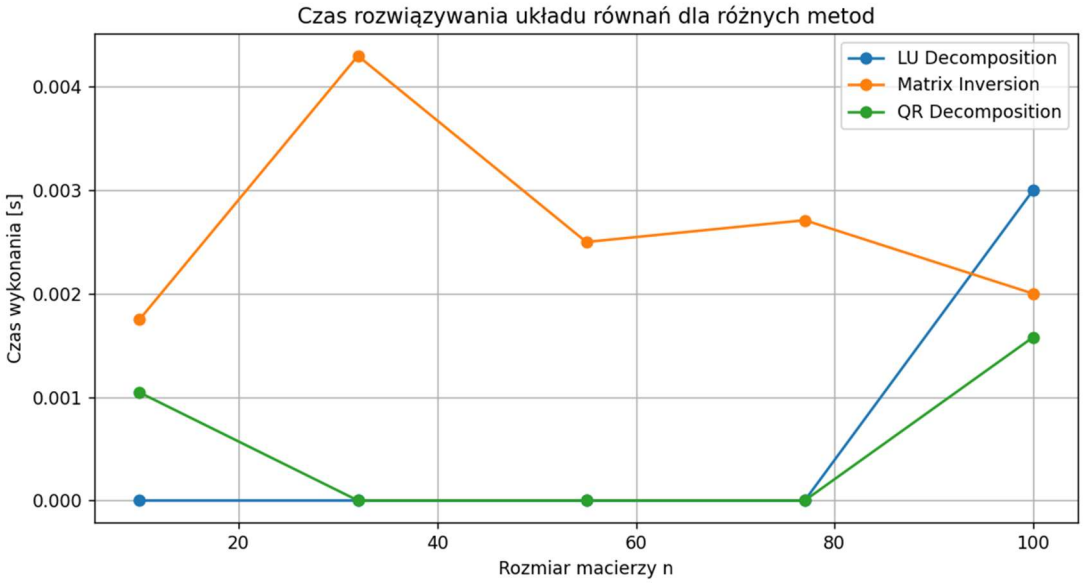
Matrix Inversion - correct?: True , Time taken: 0.003437519073486328 , identity check?: True

QR Decomposition - correct?: True , Time taken: 0.0016851425170898438

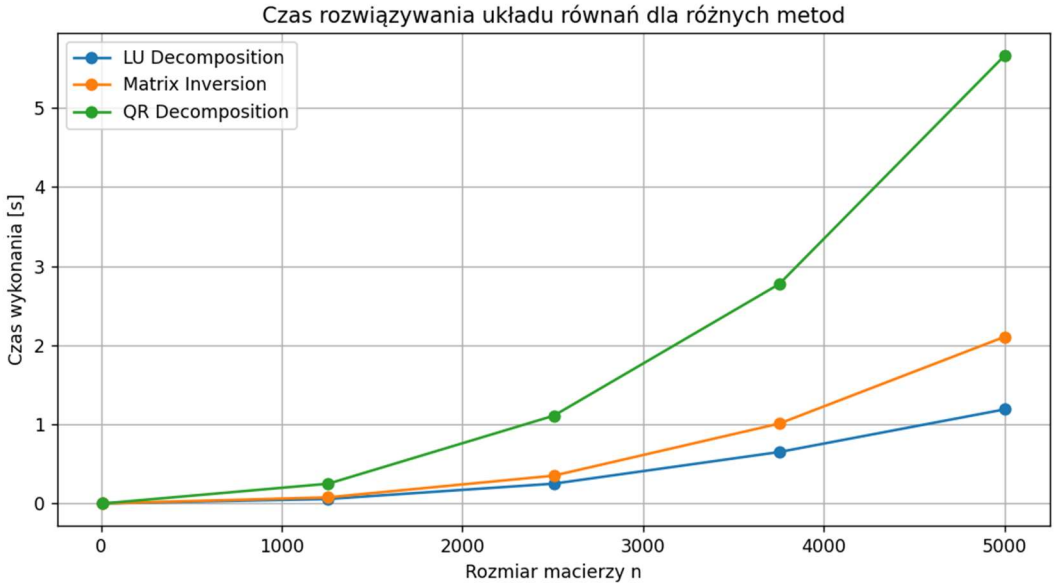
Kod python do rysowania wykresów:

```
import matplotlib.pyplot as plt

sizes = np.linspace(10, 100, 5, dtype=int)
lu_times = []
inv_times = []
qr_times = []

# Pomiar czasu dla każdej metody
for n in sizes:
    A, b = generate_matrix_and_vector(n)

    _, lu_time = lu_decomposition(A, b)
    lu_times.append(lu_time)

    _, _, inv_time = matrix_inversion(A, b)
    inv_times.append(inv_time)

    _, qr_time = qr_decomposition(A, b)
    qr_times.append(qr_time)

# Rysowanie wykresów
plt.figure(figsize=(10, 5))
plt.plot(sizes, lu_times, marker='o', label='LU Decomposition')
plt.plot(sizes, inv_times, marker='o', label='Matrix Inversion')
plt.plot(sizes, qr_times, marker='o', label='QR Decomposition')
plt.xlabel('Rozmiar macierzy n')
plt.ylabel('Czas wykonania [s]')
plt.title('Czas rozwiązywania układu równań dla różnych metod')
plt.legend()
plt.grid(True)
plt.show()
```

Poniżej przedstawiam wykresy ukazujące relacje czasu wykonania metody do rozmiaru macierzy n:

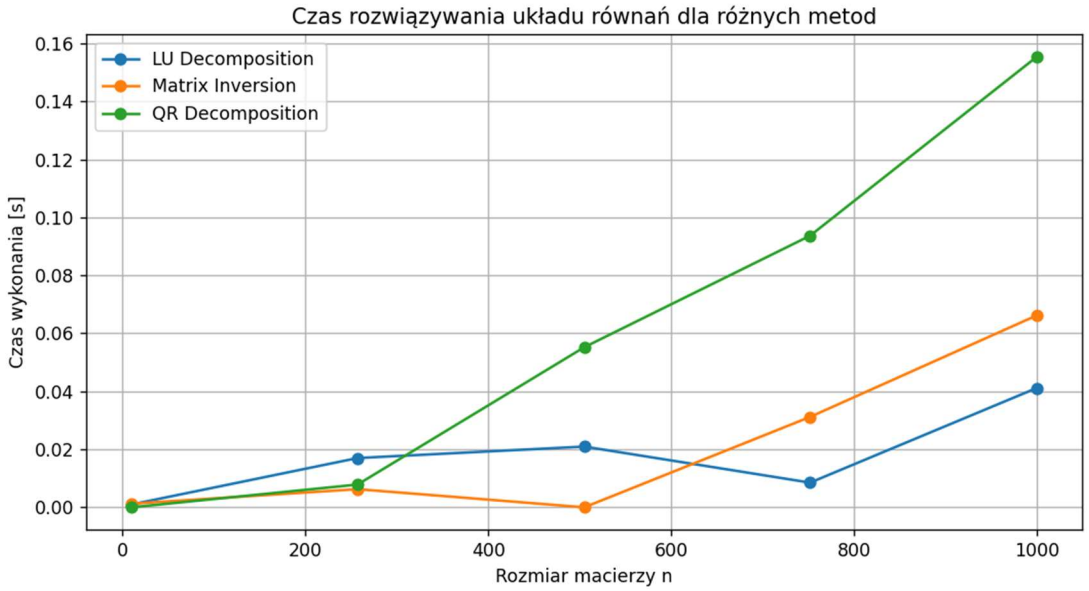


Rys 1: wykres zależności metody LU, QR dekompozycji i inwersji dla **n=100**

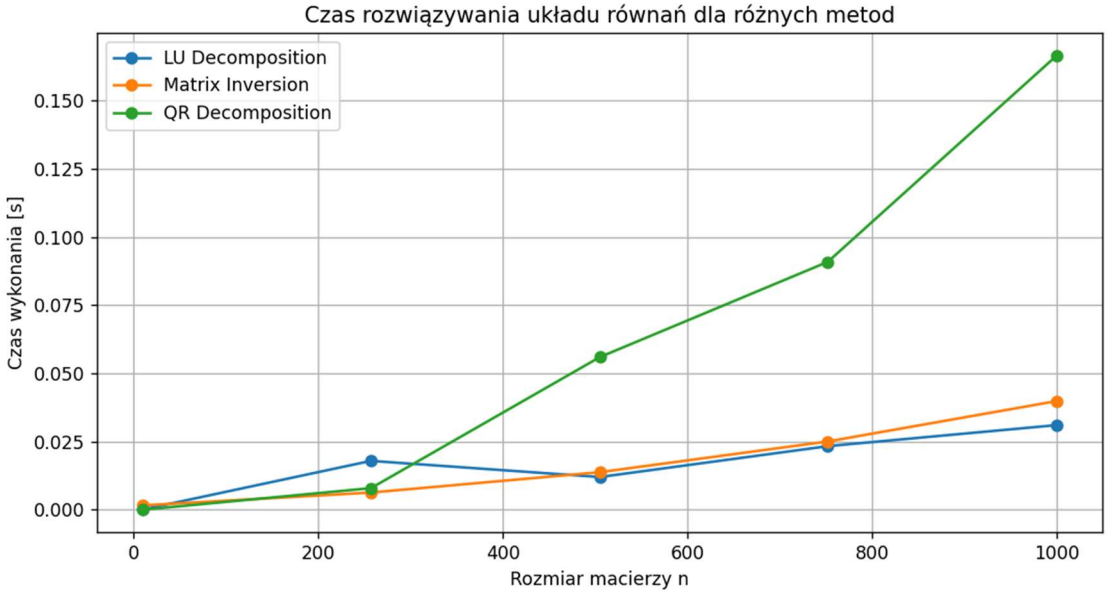


Rys 2: wykres zależności metody LU, QR dekompozycji i inwersji dla **n=5000**

Aby stworzyć bardziej poprawne wnioski będę porównywać metody przy **n=1000,** kilkukrotnie ponieważ macierze za każdym razem są inne (losowo generowane), wybieram n=1000 dlatego, że różnice już stają się widoczne a czas obliczeń akceptowalny:

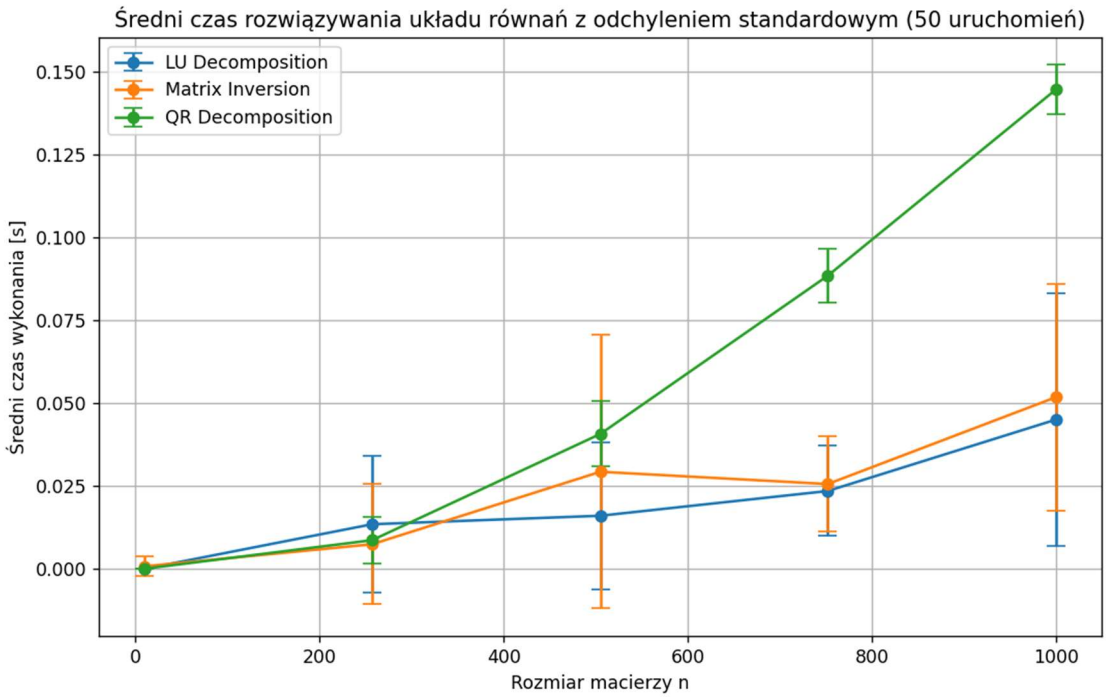


Rys 3: wykres zależności metody LU, QR dekompozycji i inwersji dla **n=1000, próba 1**



Rys 4: wykres zależności metody LU, QR dekompozycji i inwersji dla **n=1000, próba 2**

Dodatkowo tworzę zbiorczy wykres który pomoże w analizie, biorę pod uwagę średnią wartość dla każdego n oraz odchylenie standardowe od tej średniej:



Rys 5: wykres zależności metody LU, QR dekompozycji i inwersji dla **n=1000, próbkowanego 50 razy**

Biorąc pod uwagę uruchomienie programu kilkadziesiąt razy które eliminuje częściowo problem zależności od danych wejściowych (macierzy), możliwych opóźnień programu poprzez system operacyjny.

Z poprzednich uruchomień oraz z powyższego wykresu można stwierdzić, że:

Metoda QR dekompozycji jest najwolniejsza z nich wszystkich i ma najmniejsze odchylenia standardowe, to znaczy, że dla różnych danych wejściowych czas obliczeń będzie dość podobny.

Biorąc pod uwagę niepewności nie można jednoznacznie określić czy metoda inwersji macierzy jest lepsza od dekompozycji, obie są porównywalnie szybkie oraz silnie zależne od danych wejściowych (dla czasu wykonania)

Kod python:

```
# Parametry symulacji
sizes = np.linspace(10, 1000, 5, dtype=int)
num_runs = 50  # Liczba uruchomień dla każdego rozmiaru macierzy
lu_times = {size: [] for size in sizes}
inv_times = {size: [] for size in sizes}
qr_times = {size: [] for size in sizes}

# Pomiar czasu dla każdej metody
```

```
for _ in range(num_runs):
    for n in sizes:
        A, b = generate_matrix_and_vector(n)

        _, lu_time = lu_decomposition(A, b)
        lu_times[n].append(lu_time)

        _, _, inv_time = matrix_inversion(A, b)
        inv_times[n].append(inv_time)

        _, qr_time = qr_decomposition(A, b)
        qr_times[n].append(qr_time)

# Obliczenie średnich czasów i odchylenia standardowego
avg_lu_times = [np.mean(lu_times[size]) for size in sizes]
std_lu_times = [np.std(lu_times[size]) for size in sizes]

avg_inv_times = [np.mean(inv_times[size]) for size in sizes]
std_inv_times = [np.std(inv_times[size]) for size in sizes]

avg_qr_times = [np.mean(qr_times[size]) for size in sizes]
std_qr_times = [np.std(qr_times[size]) for size in sizes]

# Rysowanie wykresów z odchyleniem standardowym
plt.figure(figsize=(10, 6))
plt.errorbar(sizes, avg_lu_times, yerr=std_lu_times, marker='o', label='LU
Decomposition', capsize=5)
plt.errorbar(sizes, avg_inv_times, yerr=std_inv_times, marker='o',
label='Matrix Inversion', capsize=5)
plt.errorbar(sizes, avg_qr_times, yerr=std_qr_times, marker='o', label='QR
Decomposition', capsize=5)
plt.xlabel('Rozmiar macierzy n')
plt.ylabel('Średni czas wykonania [s]')
plt.title('Średni czas rozwiązywania układu równań z odchyleniem standardowym
(50 uruchomień)')
plt.legend()
plt.grid(True)
plt.show()
```

Bibliografia:

wykład dr inż. Katarzyna Rycerz

https://pl.wikipedia.org/wiki/Metoda_LU

https://pl.wikipedia.org/wiki/Rozk%C5%82ad_QR