Algorytmy macierzowe

Laboratorium 2
Sprawozdanie

Algorytmy rekurencyjnego odwracania macierzy, rekurencyjnej LU faktoryzacji macierzy i rekurencyjnego obliczania wyznacznika macierzy

Adam Naumiec

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki
Październik MMXXIII

Algorytmy macierzowe
Laboratorium 2

Adam Naumiec

Październik 2023

Spis treści

0 Abstrakt

Niniejszy dokument jest sprawozdaniem z wykonania Laboratorium 2 z przedmiotu Algorytmy macierzowe prowadzonego przez Pana prof. dr hab. Macieja Paszyńskiego w roku akademickim 2023/2024 na piątym semestrze studiów pierwszego stopnia na kierunku Informatyka prowadzonego na Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie na Wydziale Informatyki.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

Rysunek 1: Macierz kwadratowa - dla takiej macierzy można zdefiniować wyznacznik macierzy (2)

1 Streszczenie wykładu

Tematyką drugiego wykładu i laboratorium było omówienie rekurencyjnych algorytmów:

- odwracania macierzy
- LU faktoryzacji
- obliczania wyzancznika gęstych (elimicaji Gaussa)

oraz ich złożoności obliczeniowej.

Zaprezentowane zostało także wykorzystanie i znaczenie tych algorytmów w praktyce oraz historia wykorzystania, postrzegania i zastosowania macierzy na przestrzeni wieków, a także matematyczne, inofmratyczne i algorytmiczne podstawy operacji na macierzach.

2 Zadania zrealizowane w ramach laboratorium

W ramach laboratorium zrealizowano następujące zadanie polegające na implementacji, wykonaniu testów wydajnościowych i przygotowaniu sprawozdania:

Proszę wybrać ulubiony język programownaia, wygenerować macierze losowe o wartościach z przedziału otwartego (0.00000001, 1.0) i zaimpelemntować wybrane algorytmy.

1. Rekurencyjne odwracanie macierzy (10 punktów)
2. Rekurencyjna LU faktoryzacja (10 punktów)
3. Rekurencyjne obliczanie wyznacznika (10 punktów)

Proszę zliczać liczbę operacji zmienno-przecinkowych (+-*/) podczas mnożenia macierzy.

3 Pseudokody algorytmów i fragmenty kodu

PRZYGOTOWANO pseudokody testowancyh algorytmów oraz przedstawiono wybrane najważniejsze fragmenty kodu zaimplementowanych algorytmów w języku programowania wysokiego poziomu.

3.1 Algorytm rekurencyjnego odwracania macierzy

Implementacja lalgorytmu w Pythonie:

```
def inverse_matrix_recursive(A):
    n = A.shape[0]

    if n == 1:
        if  A[0, 0] != 0:
            return np.array([[1 / A[0, 0]]])
        else:
            return np.array([[1 / (A[0, 0] + 1e-10)]])

    A11 = A[:n // 2, :n // 2]
    A12 = A[:n // 2, n // 2:]
```

```
A21 = A[n // 2:, :n // 2]
A22 = A[n // 2:, n // 2:]

A11_inv = inverse_matrix_recursive(A11)
S22 = A22 - np.dot(np.dot(A21, A11_inv), A12)
S22_inv = inverse_matrix_recursive(S22)

B11 = np.dot(np.dot(A11_inv, (np.eye(n // 2) + np.dot(np.dot(A12, S22_inv),
B12 = -np.dot(np.dot(A11_inv, A12), S22_inv)
B21 = -np.dot(np.dot(S22_inv, A21), A11_inv)
B22 = S22_inv

B = np.vstack((np.hstack((B11, B12)), np.hstack((B21, B22))))

return B
```

- Zdefiniujmy funkcję odwracającą macierz A:

$$\text{inverse}(A)$$

- Zawołanie rekurencyjne dla A_{11}:

$$A_{11}^{-1} = \text{inverse}(A_{11})$$

- Obliczenie $S_{22} = A_{22} - A_{21} \cdot A_{11}^{-1} \cdot A_{12}$ (rekurencyjne mnożenie macierzy):

$$S_{22} = A_{22} - A_{21} \cdot A_{11}^{-1} \cdot A_{12}$$

- Zawołanie rekurencyjne dla S_{22}:

$$S_{22}^{-1} = \text{inverse}(S_{22})$$

- Obliczenie $B_{11} = A_{11}^{-1} \cdot (I + A_{12} \cdot S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1})$ (rekurencyjne mnożenie i odejmowanie macierzy):

$$B_{11} = A_{11}^{-1} \cdot (I + A_{12} \cdot S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1})$$

- Obliczenie $B_{12} = -A_{11}^{-1} \cdot A_{12} \cdot S_{22}^{-1}$ (rekurencyjne mnożenie i odejmowanie macierzy):

$$B_{12} = -A_{11}^{-1} \cdot A_{12} \cdot S_{22}^{-1}$$

- Obliczenie $B_{21} = -S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1}$ (rekurencyjne mnożenie i odejmowanie macierzy):

$$B_{21} = -S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1}$$

- Obliczenie $B_{22} = S_{22}^{-1}$:

$$B_{22} = S_{22}^{-1}$$

1. **Zaczynamy od macierzy A podzielonej na bloki:**

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$
$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = A^{-1}$$
$$\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1}$$

2. **Rekurencyjne odwrócenie A_{11}:**

$$A_{11}^{-1} = \text{inverse}(A_{11})$$

3. **Obliczenie S_{22}:**

$$S_{22} = A_{22} - A_{21} \cdot A_{11}^{-1} \cdot A_{12}$$

4. **Rekurencyjne odwrócenie S_{22}:**

$$S_{22}^{-1} = \text{inverse}(S_{22})$$

5. **Obliczenia odwrotnych bloków:**

$$\begin{aligned} B_{11} &= A_{11}^{-1} \cdot (I + A_{12} \cdot S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1}) \\ B_{12} &= -A_{11}^{-1} \cdot A_{12} \cdot S_{22}^{-1} \\ B_{21} &= -S_{22}^{-1} \cdot A_{21} \cdot A_{11}^{-1} \\ B_{22} &= S_{22}^{-1} \end{aligned}$$

3.2 Algorytm rekurenyjnej LU faktoryzacji

Implementacja algorytmu w Pythonie:

```
def LU_recursive(A):
n = A.shape[0]
if n == 1:
    return np.array([[1]]), A

A11 = A[:n // 2, :n // 2]
A12 = A[:n // 2, n // 2:]
A21 = A[n // 2:, :n // 2]
A22 = A[n // 2:, n // 2:]

L11, U11 = LU_recursive(A11)
U11_inv = np.linalg.inv(U11)
L21 = np.dot(A21, U11_inv)
L11_inv = np.linalg.inv(L11)
U12 = np.dot(L11_inv, A12)
S = A22 - np.dot(np.dot(A21, U11_inv), L11_inv.dot(A12))

LS, US = LU_recursive(S)

L = np.block([[L11, np.zeros_like(L11)],
              [L21, LS]])
U = np.block([[U11, U12],
              [np.zeros_like(U12), US]])

return L, U
```

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$LU = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$$

$$A_{11} = L_{11}U_{11}$$

Mnożymy pierwszy wiersz z lewej strony przez L_{11}^{-1}:

$$\begin{bmatrix} L_{11} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} U_{11} & L_{11}^{-1}A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Odejmujemy od drugiego wiersza pierwszy wiersz przemnożony przez $A_{21}U_{11}^{-1}$:

$$\begin{bmatrix} L_{11} & 0 \\ A_{21}U_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} U_{11} & L_{11}^{-1}A_{12} \\ 0 & A_{22} - A_{21}U_{11}^{-1}L_{11}^{-1}A_{12} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ A_{21}U_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} U_{11} & L_{11}^{-1}A_{12} \\ 0 & S \end{bmatrix}$$

Obliczamy $S = L_S U_S$

$$LU = \begin{bmatrix} L_{11} & 0 \\ A_{21}U_{11}^{-1} & L_S \end{bmatrix} \begin{bmatrix} U_{11} & L_{11}^{-1}A_{12} \\ 0 & U_S \end{bmatrix}$$

Rekurencyjna faktoryzacja LU:

$$LU = \begin{bmatrix} L_{11} & 0 \\ A_{21}U_{11}^{-1} & L_S \end{bmatrix} \begin{bmatrix} U_{11} & L_{11}^{-1}A_{12} \\ 0 & U_S \end{bmatrix}$$

$$[L, U] = LU(A) = LU\left(\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}\right)$$

Obliczenia:

1. Oblicz rekurencyjnie $[L_{11}, U_{11}] = LU(A_{11})$
2. Oblicz rekurencyjnie $U_{11}^{-1} = \text{inverse}(U_{11})$
3. Oblicz $L_{21} = A_{21}U_{11}^{-1}$
4. Oblicz rekurencyjnie $L_{11}^{-1} = \text{inverse}(L_{11})$
5. Oblicz $U_{12} = L_{11}^{-1}A_{12}$
6. Oblicz $L_{22} = S = A_{22} - A_{21}U_{11}^{-1}L_{11}^{-1}A_{12}$
7. Oblicz rekurencyjnie $[L_S, U_S] = LU(S)$
8. $U_{22} = U_S$
9. $L_{22} = L_S$

3.3 Algorytm rekurencyjnego obliczania wyzancznika

Implementacja algorytmu w Pythonie:

```
def determinant_recursive(A):
L, U, count_ops = LU_recursive(A)
det_U = np.prod(np.diag(U))
return det_U, count_ops
```

Definiujemy obliczenia wyznacznika macierzy A jako:

$$det(A) = l_{11} * ...l_{nn} * u_{11} * ... * u_{nn} = (l_{ii} == 1) = u_{11} * ... * u_{nn}$$

gdzie:

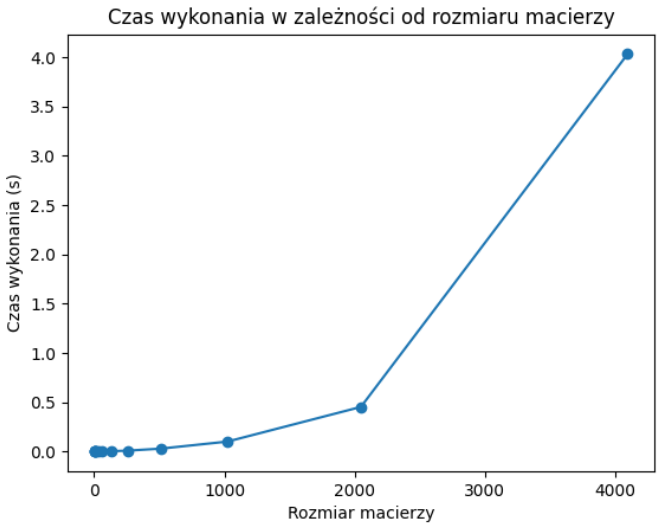
- l_{ii} to element w $i - tym$ wierszu i $i - tej$ kolumnie na przekątnej L,
- u_{ii} to element w itym wierszu i $i - tej$ kolumnie na przekątnej U.

Zaczynamy od rekurencyjen faktoryzacji LU macierzy A, a następnie mnożymy elementy na przekątnych wyznaczonych macierzy L i U.

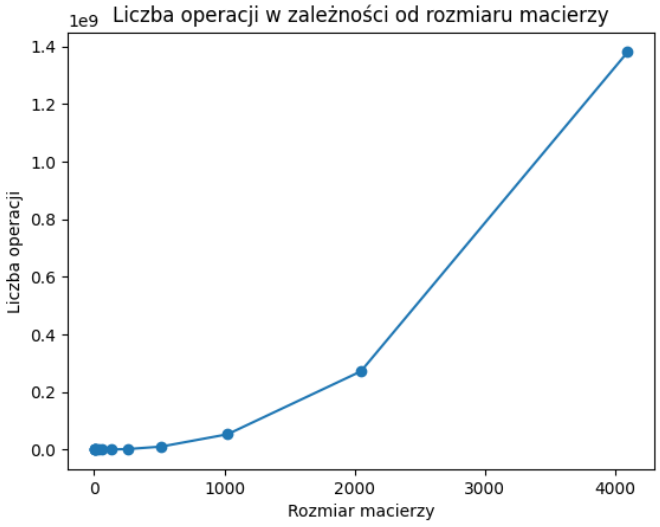
4 Testy wydajnościowe i czas działania

PRZYGOTOWANO testy wydajnościowe omawainych algorytmów w celu prezentacji czasu obliczeń oraz liczby wykonanych operacji zmiennoprzecinkowych. Czynność ta pozwoliła na praktyczne przetestowanie algorytmów oraz konformantacji ich teoretycznych złozoności obliczeniowych wraz z danymi empirycznymi.

- **Algorytm rekurencyjnego odwracania macierzy**

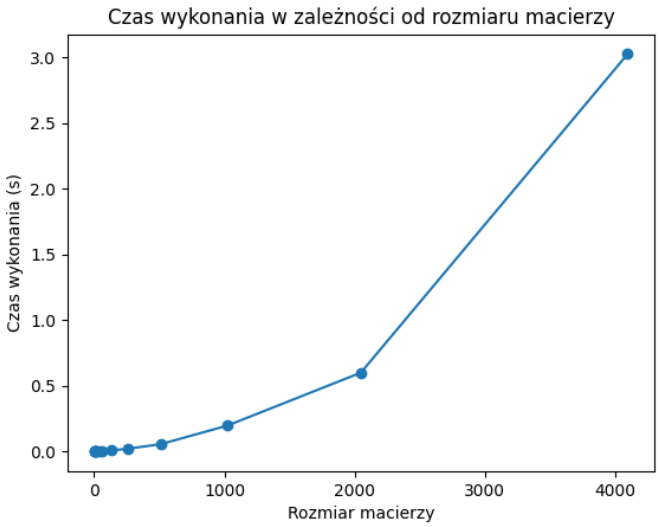


Rysunek 2: Wykres czasu obliczeń algorytmu rekurencyjnego odwracania macierzy

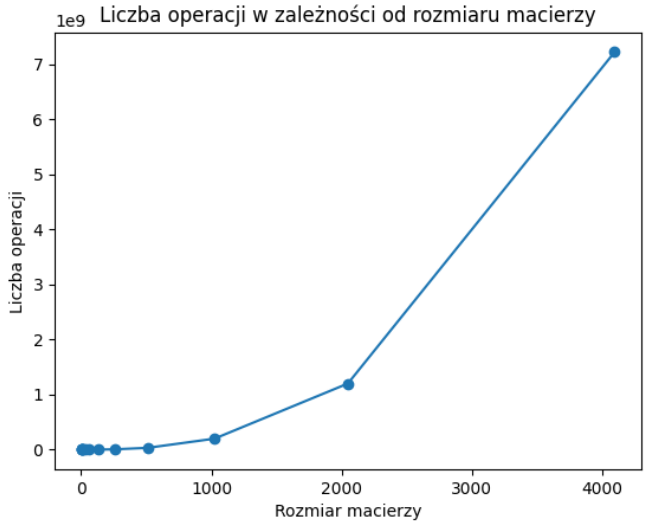


Rysunek 3: Wykres liczby operacji zmiennoprzecinkowych rekurencyjnego odwracania macierzy

- **Algorytm rekurencyjnej faktoryzacji LU macierzy**

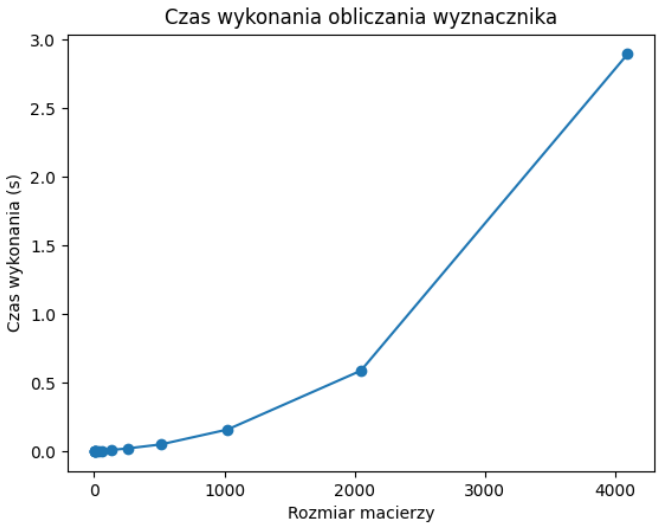


Rysunek 4: Wykres czasu obliczeń algorytmu rekurencyjnej faktoryzacji LU macierzy

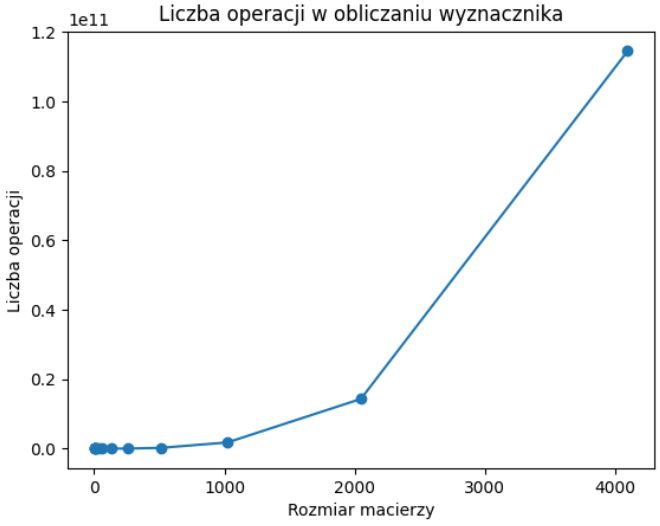


Rysunek 5: Wykres liczby operacji zmiennoprzecinkowych rekurencyjnej faktoryzacji LU macierzy

- **Algorytm rekurencyjnego obliczania wyznacznika macierzy**



Rysunek 6: Wykres czasu obliczeń algorytmu rekurencyjnego obliczania wyznacznika macierzy



Rysunek 7: Wykres liczby operacji zmiennoprzecinkowych rekurencyjnego obliczania wyznacznika macierzy

5 Złożoność obliczeniowa

DOKONANO próby oszacowania złożoności obliczeniowej omawiancyh algorytmów podejściem eksperymentalnym i teoretycznym. Pozwoliło to na empiryczne oszacowanie poprawności teoretycznie wyzanczonej złożoności.

5.1 Oszacowanie eksperymentalne

Przeprowadzone testy wydajnościowe wykazały, że oszacowane teoretycznie złożoności mają odzwierciedlenie w praktyce, możliwe jest zastosowanie dodatkowych narzędzi, które dokładniej wyzaczą dokładną wartość empirycznie otrzymanych złożoności, ale zasadnym jest twierdzenie, że algorytmy zostały zaimplementowane poprawnie i prawidłowo oszacowano ich złożoności obliczeniowe (w szczególności złożoności czasowe).

5.2 Oszacowanie teoretyczne

1. **Algorytm rekurencyjnego odwracania macierzy**
   Aby określić złożoność obliczeniową algorytmu rekurencyjnego odwracania macierzy, rozważmy następujące równanie rekurencyjne:

   $$T(n) = 2 \cdot T(n/2) + 10 \cdot M(n)$$

   gdzie: $T(n)$ - złożoność obliczeniowa algorytmu rekurencyjnego odwracania macierzy dla rozmiaru n, $M(n)$ - złożoność obliczeniowa algorytmu mnożenia macierzy dla rozmiaru n.

   Załóżmy, że używamy algorytmu Strassena o złożoności $O(n^{\log_2 7})$.

   Korzystając z twierdzenia o rekurencji uniwersalnej (znanej również jako Master Theorem), otrzymujemy:

   $$T(n) = 2 \cdot T(n/2) + 10 \cdot O(n^{\log_2 7})$$

   Zgodnie z twierdzeniem Mastera, dla tego przypadku rekurencyjnego otrzymujemy: $a = 2$, $b = 2$, $f(n) = 10 \cdot n^{\log_2 7}$.

   Ponieważ $f(n)$ jest wielomianowo większe niż $n^{\log_b a}$ dla $a = 2$ i $b = 2$, zatem złożoność obliczeniowa algorytmu rekurencyjnego odwracania macierzy wynosi $O(n^{\log_2 7})$.

   Zatem złożoność rekurencyjnego odwracania macierzy zależy od złożoności algorytmu mnożenia macierzy.

2. **Algorytm rekurencyjnej faktoryzacji LU macierzy**

Równanie rekurencyjne:

$$T(n) = 4T\left(\frac{n}{2}\right) + 5M(n) + d$$

Gdzie: - $T(n)$ jest złożonością czasową algorytmu dla danych o rozmiarze n. - n jest rozmiarem danych (macierzy). - $M(n)$ reprezentuje złożoność obliczeniową algorytmu mnożenia macierzy dla danych o rozmiarze n. - d to liczba pozostałych operacji o stałej złożoności, które pomijamy w analizie.

Teraz, zgodnie z twierdzeniem o rekurencji uniwersalnej, jeśli:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

gdzie $a \geqslant 1$, $b > 1$, a $f(n)$ jest wielomianowo większe niż $n^{\log_b a}$, to złożoność czasowa $T(n)$ to $\Theta(f(n))$.

Obliczamy $f(n)$ na podstawie równania:

$$f(n) = 5 \cdot \left(\frac{n}{2}\right)^{2.81}$$

Teraz sprawdzamy, czy $f(n)$ jest wielomianowo większe niż $n^{\log_b a} = n^{\log_2 4} = n^2$:

$$f(n) = 5 \cdot \left(\frac{n}{2}\right)^{2.81} = 5 \cdot n^{2.81-2} = 5n^{0.81}$$

Widzimy, że $f(n)$ jest wielomianowo większe niż n^2, ponieważ $f(n) = 5n^{0.81}$ nie jest ograniczone przez wielomian, a n^2 jest wielomianem stopnia 2.

Zatem na podstawie twierdzenia o rekurencji uniwersalnej, złożoność czasowa algorytmu rekurencyjnego odwracania macierzy wynosi $\Theta(f(n)) = \Theta(n^{2.81})$. Ostatecznie, złożoność rekurencyjnego odwracania macierzy zależy od złożoności algorytmu mnożenia macierzy, reprezentowanego przez $M(n)$.

3. **Algorytm rekurencyjnego obliczania wyznacznika macierzy kwadratowej**

Złożoność całego algorytmu rekurencyjnego obliczania wyznacznika macierzy wynosi:

$$O(n^{\log_2 7} + n)$$

, jednakże biorąc pod uwagę, że $n^{\log_2 7}$ jest dominującym czynnikiem dla rosnącego n, możemy uprościć wyrażenie do $O(n^{\log_2 7})$. W analizie asymptotycznej złożoności, skupiamy się na dominującym czynniku, który ma największy wpływ na złożoność, zwłaszcza gdy n rośnie, zatem ostatecznie całkowita złożonośc algorytmu wynosi:

$$O(n^{\log_2 7}) \approx O(n^{2,81}).$$

6 Porównanie wyników

WYKORZYSTANO narzędzie Octave do przeliczenia przykładu dla wybranej małej macierzy i porównano wynik z tym, otrzymanym własną implementacją.

6.1 Octave

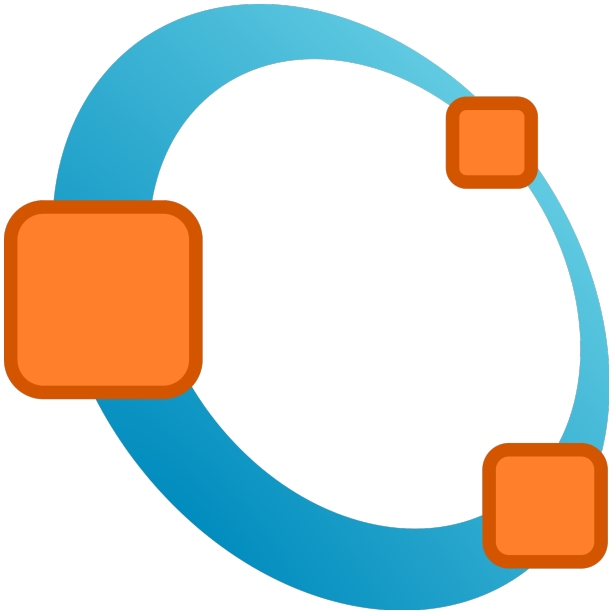
GNU Octave to darmowy otwartoźródłowy język programowania oraz środowisko do obliczeń numerycznych, które zostało zaprojektowane przede wszystkim do wykonywania zadań związanych z matematyką, inżynierią i naukami ścisłymi. Octave jest podobny do komercyjnego oprogramowania MATLAB, co oznacza, że ma wiele funkcji do manipulacji macierzami, obliczeń matematycznych, tworzenia wykresów i analizy danych.

6.2 Wykorzystanie Octave

Program ten umożliwia wykonywanie zaawansowanych obliczeń matematycznych i numerycznych, obsługuje operacje na wektorach i macierzach, rozwiązywanie równań różniczkowych, algorytmy optymalizacji, analizę sygnałów, przetwarzanie obrazów i wiele innych zastosowań.

Jego interfejs użytkownika opiera się na wierszu poleceń, ale dostępne są również różne interfejsy graficzne, które ułatwiają pracę z programem. Octave ma duże wsparcie społeczności, co oznacza, że istnieje wiele dodatkowych pakietów i rozszerzeń dostępnych do pobrania, rozszerzających jego funkcjonalność.

Jest to potężne narzędzie do naukowców, inżynierów, studentów i każdego, kto zajmuje się analizą danych, matematyką czy innymi dziedzinami, które wymagają zaawansowanych obliczeń numerycznych. Ponadto, jako projekt



Rysunek 8: Logo GNU Octave (3)

o otwartym kodzie źródłowym, Octave jest stale rozwijany i udoskonalany przez społeczność użytkowników i programistów.

6.3 Odwracanie macierzy w Octave

Kod testujący:

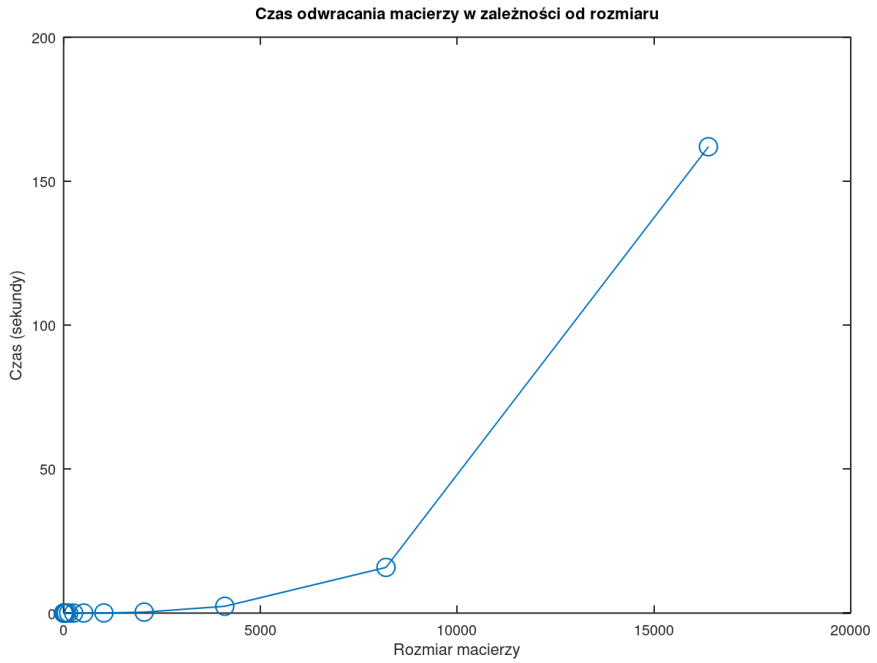
```
sizes = [];
times_inv = [];

for i = 1:13
    matrix_size = 2^i;
    A = rand(matrix_size);
    tic;
    inverted_A = inv(A);
    time_taken = toc;

    sizes = [sizes, matrix_size];
    times_inv = [times_inv, time_taken];
end

plot(sizes, times_inv, '-o');
```

```
title('Czas odwracania macierzy w zależności od rozmiaru');
xlabel('Rozmiar macierzy');
ylabel('Czas (sekundy)');
```



Rysunek 9: Wyniki odwracania macierzy w Octave

6.4 LU faktoryzacja macierzy w Octave

Kod testujący:

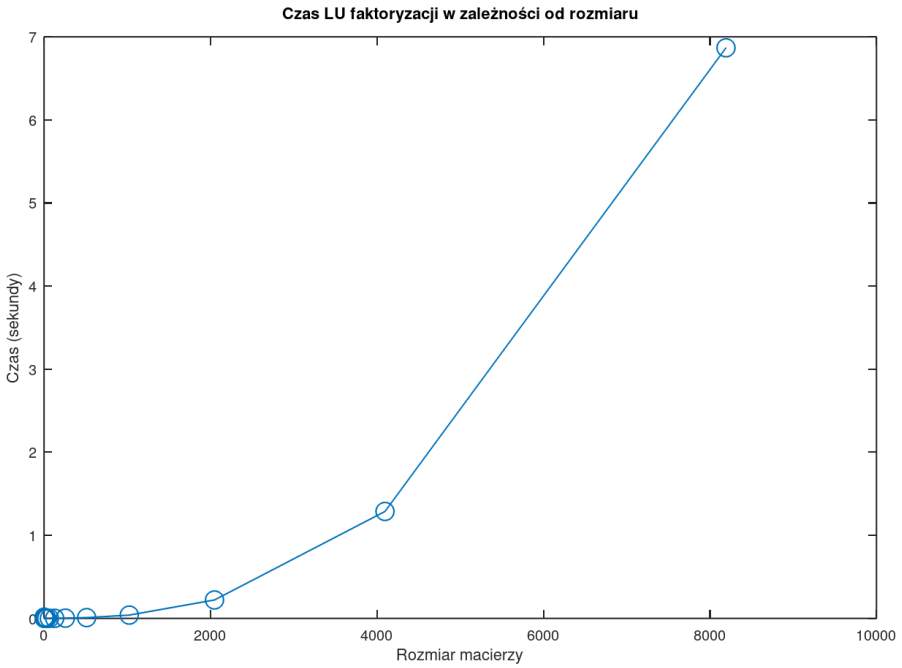
```
sizes = [];
times_lu = [];

for i = 1:13
    matrix_size = 2^i;
    A = rand(matrix_size);
    tic;
    [L, U] = lu(A);
    time_taken = toc;

    sizes = [sizes, matrix_size];
    times_lu = [times_lu, time_taken];
```

```
end

plot(sizes, times_lu, '-o');
title('Czas LU faktoryzacji w zależności od rozmiaru');
xlabel('Rozmiar macierzy');
ylabel('Czas (sekundy)');
```



Rysunek 10: Wyniki LU faktoryzacji macierzy w Octave

6.5 Obliczanie wyznacznika macierzy w Octave

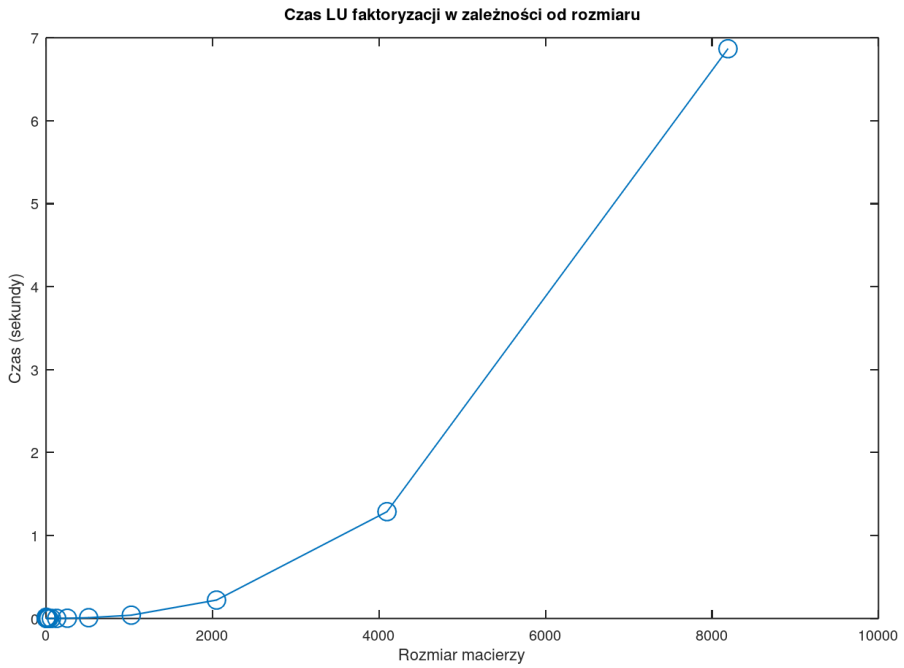
Kod testujący:

```
sizes = [];
times_det = [];

for i = 1:13
    matrix_size = 2^i;
    A = rand(matrix_size);
    tic;
    determinant_A = det(A);
    time_taken = toc;
```

```
    sizes = [sizes, matrix_size];
    times_det = [times_det, time_taken];
end

plot(sizes, times_det, '-o');
title('Czas obliczania wyznacznika w zależności od rozmiaru');
xlabel('Rozmiar macierzy');
ylabel('Czas (sekundy)');
```



Rysunek 11: Wyniki obliczania wyznacznika macierzy w Octave

Udało się wykonać zadania dla $i = 1, 2, 3, ..., 13$.

7 Wnioski

Laboratorium wykazało istotność zagadnienia operacji na macierzach takie jak odwracanie macierzy, faktoryzacja LU i obliczanie wyznacznika macierzy.

Refleksje płynące z laboratirum:

- **Złożoność obliczeniowa**
  Wykorzystanie tych "sprytnych" algorytmów pozwala na obniżenie złożoności obliczeniowej, a szczególnie czasowej, co ma ogromne znaczenie dla dużych danych.

- **Podział na mniejsze problemy**
  Podział na mniejsze podproblemy, często wykorzystywany w rekurencyjnych algorytmach macierzowych, ułatwia zarządzanie i rozwiązywanie bardziej złożonych problemów. Dzięki temu możliwe jest skuteczne rozwiązanie większych zbiorów danych.

- **Koszt pamięciowy**
  Analiza zużycia pamięci jest równie istotna co złożoność obliczeniowa. Algorytmy rekurencyjne, często korzystające z wielu operacji i przechowywania danych, mogą generować większy koszt pamięciowy.

- **Efektywność w praktyce**
  Otrzymane wyniki w laboratorium mogą różnić się od efektywności algorytmów w rzeczywistych zastosowaniach. Czynniki takie jak specyfika danych czy sposób ich optymalizacji mogą mieć duży wpływ na efektywność algorytmów.

- **Liczba operacji na liczbach zmiennoprzecinkowych**
  Kluczowym elementem analizy jest liczba operacji wykonywanych na liczbach zmiennoprzecinkowych podczas działania algorytmu. Precyzja obliczeń oraz wydajność czasowa i pamięciowa mogą być silnie uzależnione od wykonywanych operacji na liczbach zmiennoprzecinkowych.

- **Wpływ platformy sprzętowej, narzędzi i implementacji algorytmów**
  Analiza rezultatów laboratorium uwzględnia także wpływ specyfiki platformy sprzętowej na wydajność rekurencyjnych algorytmów macierzowych. Różne architektury sprzętowe mogą mieć istotny wpływ na efektywność algorytmów, a także wykorzystane narzędzia i techniki implementacji.

Algorytm rekurencyjnego odwracania macierzy:

- Złożoność obliczeniowa tego algorytmu jest zazwyczaj efektywniejsza niż metody prostego odwracania macierzy.

- Dzięki podziałowi macierzy na mniejsze bloki i zastosowaniu rekurencji możliwe jest efektywne obliczanie odwrotności macierzy dla większych zbiorów danych.

- Koszt pamięciowy może być wyższy z powodu rekurencyjnego stosowania algorytmu, ale jego złożoność obliczeniowa jest często niższa.

Algorytm rekurencyjnej faktoryzacji LU:

- Rekurencyjna faktoryzacja LU opiera się na podziale macierzy na mniejsze bloki, co pozwala na efektywne rozwiązanie całego problemu.
- Podział na mniejsze podproblemy ułatwia proces faktoryzacji, co może przyspieszyć działanie algorytmu, zwłaszcza dla dużych macierzy.
- Jednak koszt pamięciowy może być wyższy z powodu rekurencyjnego podejścia.

Algorytm rekurencyjnego obliczania wyznacznika macierzy:

- Rekurencyjne obliczanie wyznacznika opiera się na podziale macierzy na podmacierze, co może usprawnić proces obliczeniowy dla większych macierzy.
- Złożoność obliczeniowa i koszt pamięciowy są istotnymi aspektami, które należy uwzględnić podczas implementacji tego algorytmu.

Temat ten jest szczególnie istotny ze względu na tak szerokie obecnie wykorzystanie mnożenia macierzy w informatyce.

Literatura

[1] Wykłady i laboratoria prowadzone przez Pana prof. dr hab. Macieja Paszyńskiego

[2] Wikipedia - macierz kwadratowa

[3] Wikipedia - GNU Octave

***