

Metody numeryczne - Układy równań liniowych

Krzysztof Nasuta, s193328

15-04-2024

Spis treści

1. Wstęp	1
2. Opis metod	1
2.1. Metoda Jacobiego	1
2.2. Metoda Gaussa-Seidela	2
2.3. Faktoryzacja LU	2
3. Pierwszy układ równań	2
3.1. Wyniki	3
4. Drugi układ równań	3
4.1. Wyniki	3
5. Porównanie czasu wykonania względem rozmiaru macierzy	4
5.1. Porównanie implementacji wysokopoziomowej i niskopoziomowej.	5
6. Podsumowanie	6

1. Wstęp

W niniejszym sprawozdaniu przedstawiono wyniki analizy trzech metod rozwiązywania macierzowych równań liniowych. Do testowanych metod należą dwie metody iteracyjne: Jacobiego oraz Gaussa-Seidela oraz jedna metoda bezpośrednia: faktoryzacja LU. Kod źródłowy został napisany w języku Python oraz C++, a do wizualizacji wyników wykorzystano bibliotekę Matplotlib. Nie wykorzystano żadnych zewnętrznych bibliotek do rozwiązywania układów równań liniowych. Do przechowywania macierzy napisano własną klasę `Matrix`, która ułatwia późniejsze operacje na macierzach.

Powyżej wymienione metody zostały przetestowane na kilku układach równań i rozmiarów macierzy. Wyniki zostały przedstawione w postaci wykresów, które pokazują zbieżność metod iteracyjnych oraz czas wykonania dla każdej z metod.

2. Opis metod

Wszystkie podane metody służą do rozwiązywania układów równań liniowych postaci $Ax = b$, gdzie A to macierz współczynników, x to wektor niewiadomych, a b to wektor wyrazów wolnych.

2.1. Metoda Jacobiego

Pierwszą z badanych metod iteracyjnych jest metoda Jacobiego. Zaimplementowana wersja wykorzystuje rekurencyjny wzór operujący na poszczególnych elementach wektora x . Wzór ten wygląda następująco:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} * x_j^{(k)} \right), i = 1, 2, \dots, n$$

Wektor x jest inicjalizowany zerami, a następnie iteracyjnie poprawiany w każdej iteracji. Algorytm kończy się, gdy spełniony jest warunek zbieżności, czyli gdy norma błędu jest mniejsza od zadanego epsilon.

Metoda ta nie zawsze jest zbieżna, więc nie gwarantuje znalezienia rozwiązania.

2.2. Metoda Gaussa-Seidela

Druga badana metoda iteracyjna to metoda Gaussa-Seidela. Jest ona podobna do metody Jacobiego. Różnicą jest, że operuje także na wyliczonych w tej samej iteracji elementach nowego wektora x . Również tutaj zaimplementowano wzór elementowy. Wygląda on następująco:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} * x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} * x_j^{(k)} \right), i = 1, 2, \dots, n$$

Także w metodzie Gaussa-Seidela zaczynamy od wektora zerowego i iteracyjnie poprawiamy jego elementy. Algorytm kończy się, gdy spełniony jest warunek zbieżności.

Metoda Gaussa-Seidela także nie gwarantuje zbieżności.

2.3. Faktoryzacja LU

Metoda LU należy do grupy metod bezpośrednich. Oznacza to, że zawsze znajduje rozwiązanie układu równań. Polega na faktoryzacji macierzy współczynników A na iloczyn dwóch macierzy trójkątnych: dolnej L i górnej U . Uzyskujemy wtedy równanie $LUx = b$. Jego rozwiązanie możemy sprowadzić, rozwiązując dwa układy równań trójkątnych: $Ly = b$ oraz $Ux = y$. Do rozwiązania tych układów wykorzystujemy algorytm odpowiednio podstawiania w przód oraz w tył.

W algorytmach podstawiania wykorzystano wzory operujące na poszczególnych elementach wektora x . W przypadku podstawiania w przód wzór wygląda następująco:

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j, i = 1, 2, \dots, n$$

Natomiast w przypadku podstawiania w tył wzór wygląda następująco:

$$x_i = \frac{y_i - \sum_{j=i+1}^n u_{ij} x_j}{u_{ii}}, i = n, n-1, \dots, 1$$

3. Pierwszy układ równań

Rozmiar macierzy A to 928x928. Na diagonalu macierzy A znajduje się 8. Dwie kolumny obok diagonalu mają wartość -1 . Pozostałe elementy macierzy są równe 0.

Wektor b ma długość 928. N -ty element wektora b jest równy $\sin(4N)$.

$$A = \begin{pmatrix} 8 & -1 & -1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 8 & -1 & -1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 8 & -1 & -1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 8 & -1 & -1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 8 & -1 & -1 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 8 & -1 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 8 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 8 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & -1 & 8 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & -1 & -1 & 8 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & -1 & -1 & 8 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & -1 & -1 & 8 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & -1 & -1 & 8 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & -1 & -1 & 8 \end{pmatrix} b = \begin{pmatrix} -0.76 \\ 0.99 \\ -0.54 \\ -0.29 \\ 0.91 \\ -0.91 \\ 0.27 \\ \dots \\ -0.23 \\ -0.59 \\ 1.0 \\ -0.71 \\ -0.06 \\ 0.8 \\ -0.98 \end{pmatrix}$$

3.1. Wyniki

Oczekiwana norma błędu residuum wynosi 10^{-9} . Limit błędów został ustawiony na 10^9 .

Metoda	Iteracje	Norma błędu	Zbieżne	Czas wykonania
Jacobi	24	6.45982020781103e-10	✓	3.9153319s
Gauss-Seidel	17	4.647756224628755e-10	✓	2.6040142s
Faktoryzacja LU	-	2.1487378237870888e-15	-	33.2784297s

Możemy zauważyć, że obie metody iteracyjne zbiegają do rozwiązania. Metoda Jacobi potrzebuje 24 iteracje, natomiast Gaussa-Seidela 17. Pierwsza z tych metod zwraca wynik po 3.9s, druga natomiast po 2.6s. Obie metody iteracyjne zwracają wynik z błędem residuum mniejszym niż oczekiwany.

Faktoryzacja LU zwróciła dokładniejsze rozwiązanie, jednak czas wykonania był znacznie dłuższy. Wynik został uzyskany po ponad 33 sekundach, ale był obciążony bardzo małym błędem residuum. Jest to metoda bezpośrednia, więc zawsze zwraca dokładne rozwiązanie.

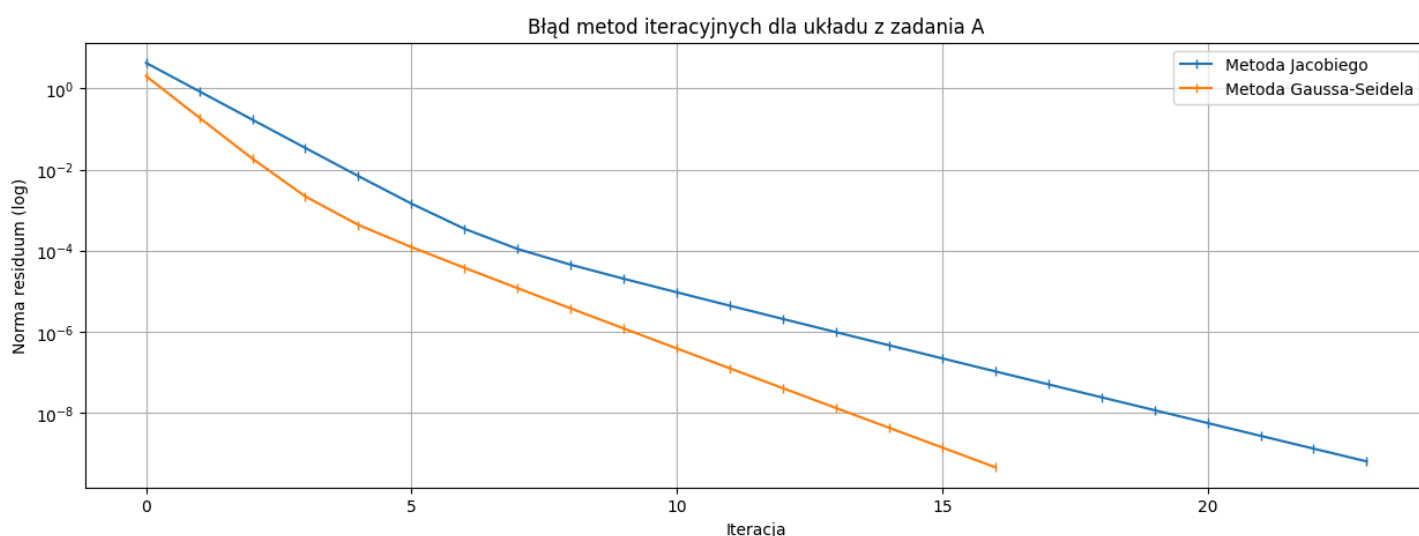


Figure 1: Zbieżność metod iteracyjnych w pierwszym układzie równań

Powyższy wykres przedstawia normę błędu residuum w skali logarytmicznej w zależności od numeru iteracji. Możemy zauważyć, że metoda Gaussa-Seidela zbiega szybciej niż metoda Jacobi.

4. Drugi układ równań

Drugi układ równań jest bardzo podobny do pierwszego. Różnica polega na tym, że na diagonalu macierzy A znajduje się 3 zamiast 8.

$$A = \begin{pmatrix} 3 & -1 & -1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & \dots & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & -1 & \dots & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 3 & -1 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 3 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 3 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & \dots & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & -1 & -1 & 3 \end{pmatrix} b = \begin{pmatrix} -0.76 \\ 0.99 \\ -0.54 \\ -0.29 \\ 0.91 \\ \dots \\ -0.71 \\ -0.06 \\ 0.8 \\ -0.98 \end{pmatrix}$$

4.1. Wyniki

Oczekawana norma oraz limit nie uległy zmianie.

Metoda	Iteracje	Norma błędu	Zbieżne	Czas wykonania
Jacobi	92	1234119999.2230158	×	15.3062337s
Gauss-Seidel	37	1183623809.2900004	×	5.6345844s
Faktoryzacja LU	-	1.211472788285935e-12	-	32.1004932s

Żadna z metod iteracyjnych nie zbiegła do rozwiązania. Aby przekroczyć górny limit błędu, wynoszący 10^9 , potrzeba było odpowiednio 92 iteracji dla metody Jacobiego oraz 37 dla metody Gaussa-Seidela. Z tego powodu czas wykonania był znacznie dłuższy niż w przypadku pierwszego układu równań. Metoda Jacobiego potrzebowała ponad 15 sekund, nie zwracając przy tym rozwiązania. Metoda Gaussa-Seidela również zakończyła się poprzez przekroczenie limitu błędu po 5.6 sekundach.

Metoda Jacobiego osiągnęła najniższą normę błędu przy iteracji ósmej, a Gaussa-Seidela przy iteracji trzeciej. Od tego momentu norma błędu residuum zaczęła rosnąć.

Metoda faktoryzacji LU zwróciła dokładne rozwiązanie, jednak czas wykonania był znacznie dłuższy. Wynik został uzyskany po ponad 32 sekundach, ale był obciążony bardzo małym błędem residuum.

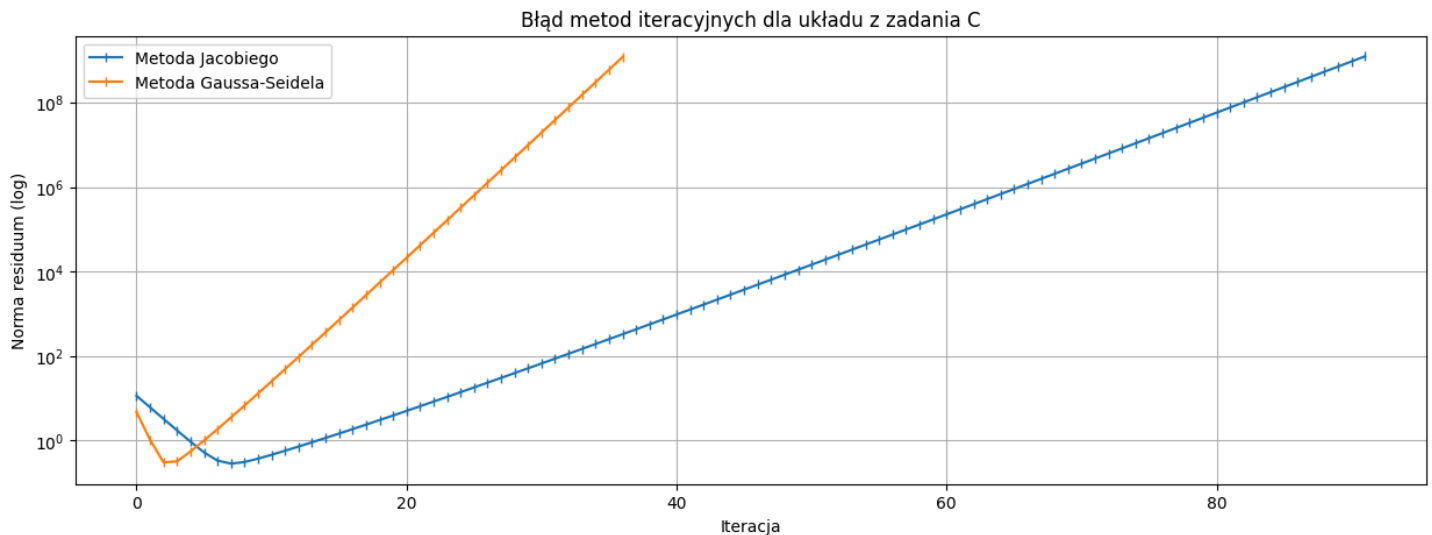


Figure 2: Zbieżność metod iteracyjnych w drugim układzie równań

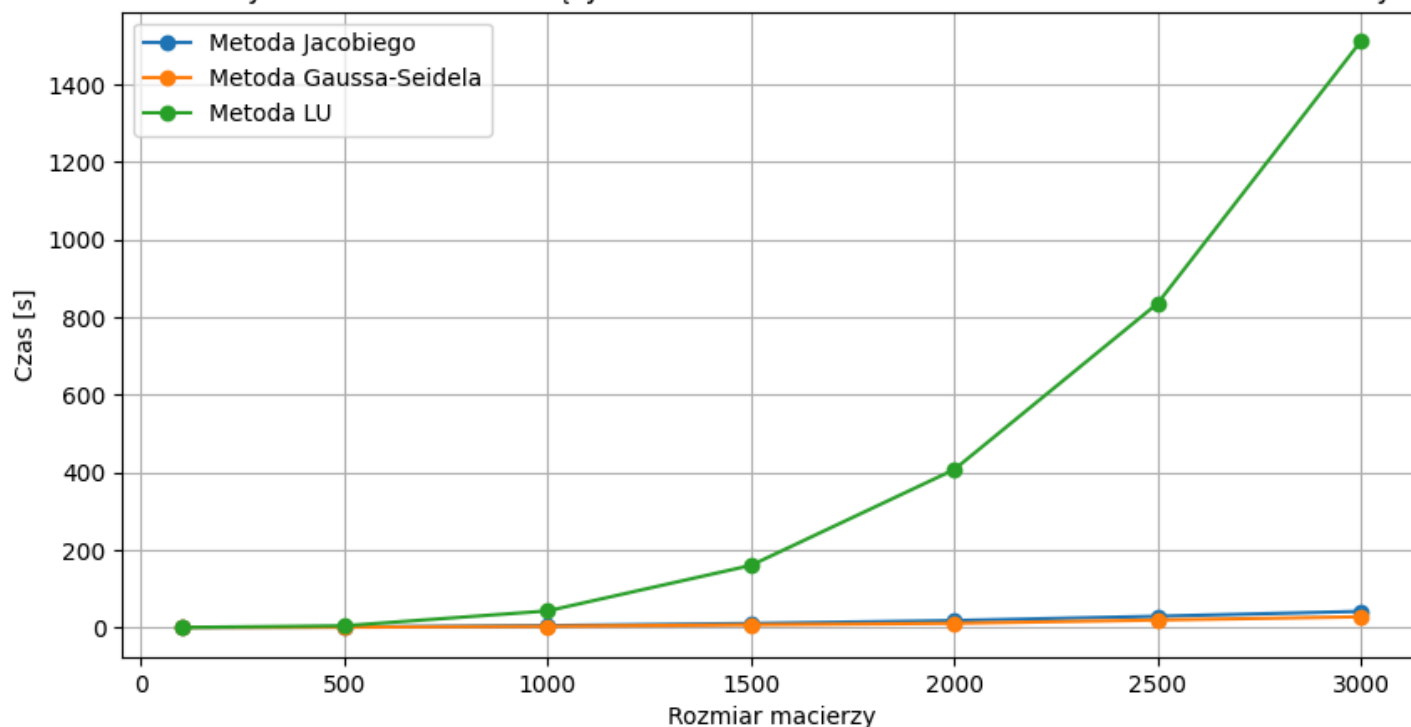
Powyższy wykres przedstawia normę błędu metod iteracyjnych dla równania drugiego.

5. Porównanie czasu wykonania względem rozmiaru macierzy

W celu zbadania zależności czasu wykonania od rozmiaru macierzy, przetestowano metody dla macierzy o rozmiarach należących do zbioru $\{100, 500, 1000, 1500, 2000, 2500, 3000\}$. W tabeli przedstawiono czasy wykonania dla każdej z metod. W nawiasach podano liczbę iteracji dla metod iteracyjnych.

Rozmiar macierzy	Jacobi	Gauss-Seidel	Faktoryzacja LU
100x100	0.0466804s (24)	0.0299448s (17)	0.0329174s
500x500	1.116892s (23)	0.7027835s (16)	4.2767321s
1000x1000	4.6454314s (24)	3.0135801s (17)	42.6112665s
1500x1500	10.3469868s (24)	6.8342752s (17)	159.927388s
2000x2000	17.7180553s (23)	10.7253274s (15)	406.9540574s
2500x2500	28.8284492s (24)	19.1137278s (17)	835.0686424s
3000x3000	41.4464903s (24)	27.4157338s (17)	1511.144846s

Czas wykonania metod rozwiązywania układu równań w zależności od rozmiaru macierzy

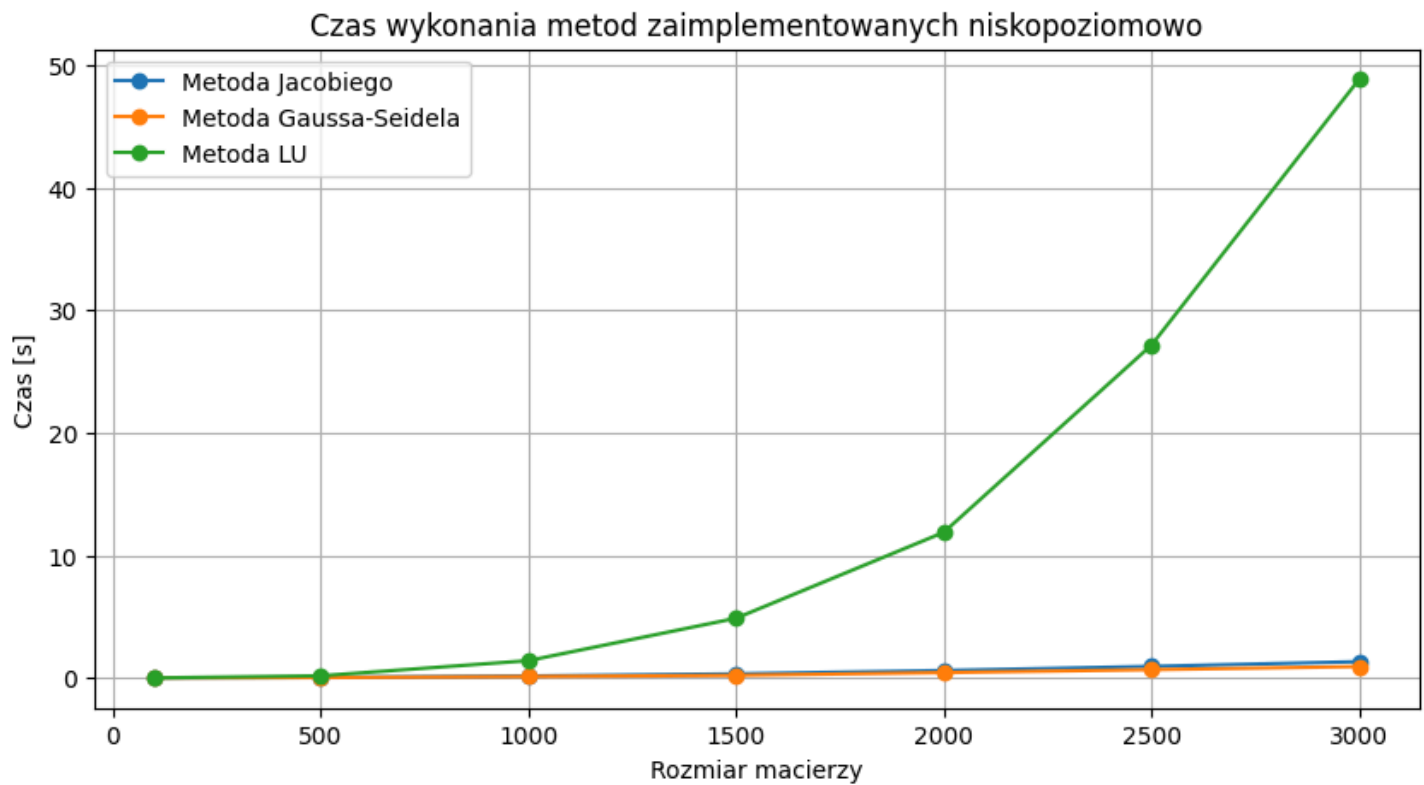


Czas wykonania każdej z metod rośnie wraz z rozmiarem macierzy. Dla metod iteracyjnych wzrost ten jest powolny, natomiast dla bezpośredniej metody faktoryzacji LU jest dużo szybszy. W przypadku macierzy o rozmiarze 1000x1000, metoda Jacobiego potrzebuje 4.6s, Gaussa-Seidela 3.0s, a faktoryzacja LU aż 42.6s. W przypadku macierzy o rozmiarze 2000x2000, czasy wynoszą odpowiednio 17.7s, 10.7s oraz 406.9s. Dla metod iteracyjnych tempo wzrostu można oszacować jako kwadratowe względem rozmiaru macierzy. Dla metody faktoryzacji LU wzrost czasu był znacznie szybszy. Szacować go można jako co najmniej sześcienny względem rozmiaru macierzy.

5.1. Porównanie implementacji wysokopoziomowej i niskopoziomowej.

Należy pamiętać, że obliczenia zostały wykonane przy pomocy programu napisanego w języku Python, który jest interpretowanym językiem wysokiego poziomu. Jego wydajność jest znacznie niższa niż języków kompilowanych, co wpływa na czas wykonania programu. W testach nie wykorzystano także bibliotek, które zawierają szybsze implementacje macierzy, takich jak NumPy. Uzyskane czasy wykonania programu nie są satysfakcjonujące, dlatego w projekcie zaimplementowano także wersję programu w języku C++, która jest znacznie szybsza. Poniższa tabela przedstawia porównanie czasów faktoryzacji LU oraz metody Gaussa-Seidela dla obu wersji programu.

Rozmiar macierzy	Python (Gauss-Seidel)	C++ (Gauss-Seidel)	Python (LU)	C++ (LU)
100x100	0.03s	0.003s	0.033s	0.001s
500x500	0.703s	0.031s	4.277s	0.174s
1000x1000	3.014s	0.112s	42.611s	1.41s
1500x1500	6.834s	0.212s	159.927s	4.87s
2000x2000	10.725s	0.438s	406.954s	11.881s
2500x2500	19.113s	0.674s	835.069s	27.152s
3000x3000	27.416s	0.928s	1511.145s	48.847s



Jak można zauważyć, wykres jest podobny do poprzedniego. Ten sam algorytm został zaimplementowany, dlatego jego złożoność obliczeniowa jest taka sama. Warto jednak zwrócić uwagę, że bezwzględne czasy wykonania programów są znacznie krótsze. Dla metody Gaussa-Seidela dla macierzy o rozmiarze 1000x1000, czas wykonania w języku Python wynosi 3.0s, natomiast w języku C++ 0.11s. Dla metody faktoryzacji LU dla tej samej macierzy, czas wykonania w języku Python wynosi 42.6s, natomiast w języku C++ 1.41s. W przypadku faktoryzacji LU macierzy o rozmiarze 3000x3000, czas wykonania w języku Python wynosi ponad 25 minut, natomiast w języku C++ - 48.9s.

6. Podsumowanie

Z przeprowadzonych testów wynika, że bezpośrednia metoda faktoryzacji LU jest znacząco wolniejsza niż metody iteracyjne. Jej główną zaletą jest wysoka dokładność oraz gwarancja uzyskania wyniku. Metody iteracyjne są dużo szybsze, jednak nie zawsze zwracają rozwiązanie. Dla pewnych macierzy metody iteracyjne mogą się rozbiegać. W przypadku zbieżności, metoda Gaussa-Seidela w każdym z badanych przypadków zwróciła wyniki po mniejszej ilości iteracji niż metoda Jacobiego. Z tego powodu czas wykonywania tej metody był krótszy. W takim razie, aby rozwiązać układ równań, możemy zastosować metodę Gaussa-Seidela, która jest szybsza i zwraca wynik w krótszym czasie. Następnie, w przypadku braku zbieżności (czyli kiedy norma błędu rośnie zamiast maleć), możemy zastosować metodę faktoryzacji LU. Zastosowanie metody iteracyjnej oraz bezpośredniej pozwala nam w pełni wykorzystać ich zalety.

Ponadto, warto zwrócić uwagę na różnice w czasie wykonania programów napisanych w języku Python oraz C++. Kompilowany do kodu maszynowego język C++ jest znacznie szybszy niż interpretowany język Python. Często sama zmiana języka programowania pozwala na przyspieszenie programu kilkanaście do kilkudziesięciu razy.