Metody Numeryczne, projekt 2
Układy równań liniowych

Jan Bancerewicz, 198099

9 kwietnia 2025

1 Układy równań liniowych

Projekt skupia się na implementacji oraz szczegółowej analizie różnych metod numerycznych służących do rozwiązywania układów równań liniowych, które stanowią podstawowy problem w algebrze liniowej oraz szeroko pojętej matematyce obliczeniowej. Układ równań liniowych składa się z N równań o maksymalnie N niewiadomych.

Do zapisu układów równań stosuje się zazwyczaj uproszczoną postać macierzową – równanie przyjmuje wtedy następującą formę:

$$Ax = b, \tag{1}$$

gdzie:

- $A \in \mathbb{R}^{n \times n}$ – macierz współczynników (tzw. macierz systemowa),
- $x \in \mathbb{R}^n$ – wektor niewiadomych (rozwiązań),
- $b \in \mathbb{R}^n$ – wektor wyrazów wolnych (tzw. wektor pobudzenia).

To samo równanie w postaci liniowej można przedstawić w następujący sposób:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{cases} \tag{2}$$

Przemnożenie wektora niewiadomych x przez współczynniki zapisane w macierzy A sprawia, że otrzymujemy wartości zapisane w wektorze wyrazów wolnych b.

1.1 Równanie macierzowe

Nasza macierz A jest pasmową macierzą o rozmiarze 1299×1299 oraz o szerokości pasma obejmującej dwa elementy. Wartości na diagonali wynoszą 5, a wartości na pasach poniżej i powyżej wynoszą -1. Dla przyspieszenia obliczeń dane są typu `numpy.float64`. Macierz A jest jest **diagonalnie dominująca, dodatnio określona, dobrze uwarunkowana** oraz symetryczna.

$$A = \begin{bmatrix} 5 & -1 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ -1 & 5 & -1 & -1 & \cdots & 0 & 0 & 0 & 0 \\ -1 & -1 & 5 & -1 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 5 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 5 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & -1 & 5 & -1 & -1 \\ 0 & 0 & 0 & 0 & \cdots & -1 & -1 & 5 & -1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & -1 & 5 \end{bmatrix} \quad x = \begin{bmatrix} 0.09197090 \\ -0.13146996 \\ 0.17920602 \\ -0.17753952 \\ \vdots \\ 0.10947654 \\ -0.18180257 \\ 0.14792459 \\ -0.18797556 \end{bmatrix} \quad b = \begin{bmatrix} 0.41211849 \\ -0.75098725 \\ 0.95637593 \\ 0.85090352 \\ \vdots \\ 0.66949057 \\ -0.91612320 \\ 0.99992457 \\ -0.90599987 \end{bmatrix}$$

Wektor b to wartości zwracane przez funkcję sinusoidalną o argumencie będącym wielokrotnością liczby 9.
Wartości x zapisane powyżej zostały obliczone za pomocą różnych metod opisanych w kolejnej sekcji. Dla każdej z badanych metod udało się osiągnąć wynik przybliżony do dokładnego rozwiązania.

Drugą macierzą na której operujemy jest C, która podobnie jak A jest macierzą pasmową o rozmiarze 1299×1299 oraz o szerokości pasma obejmującej dwa elementy. Różni się tym, że wartości na diagonali wynoszą 3, co przekłada się na to, że macierz ta **nie jest diagonalnie dominująca** (wartość bezwzględna sumy pozostałych elementów jest równa 4 i jest większa niż element na diagonali). Oprócz tego, po wywołaniu funkcji `np.linalg.eigvals(C)` otrzymaliśmy niektóre wartości ujemne, co mówi nam, że macierz **nie jest dodatnio określona**. Dodatkowo funkcja `np.linalg.cond(C)` zwróciła wartość 2896.59, co świadczy o **braku dobrego uwarunkowania funkcji**. Oznacza to, że niektóre algorytmy wyznaczania rozwiązania mogą mieć problem z wyznaczeniem dokładnego rozwiązania.

$$C = \begin{bmatrix} 3 & -1 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & \cdots & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 3 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 3 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & 0 & \cdots & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & -1 & 3 \end{bmatrix} \quad x = \begin{bmatrix} 0.07222211 \\ -0.32640396 \\ 0.13095183 \\ -0.43139860 \\ \vdots \\ 0.13872172 \\ -0.36473574 \\ 0.13137450 \\ -0.37978703 \end{bmatrix} \quad b = \begin{bmatrix} 0.41211849 \\ -0.75098725 \\ 0.95637593 \\ 0.85090352 \\ \vdots \\ 0.66949057 \\ -0.91612320 \\ 0.99992457 \\ -0.90599987 \end{bmatrix}$$

Wektor rozwiązań x w niektórych metodach nie zbiegał się do dokładnego rozwiązania, lecz dla pozostałych udało się tę zbieżność osiągnąć. Powyżej zapisano poprawną wartość x, wyliczoną za pomocą metody LU.

1.2 Norma residuum

Norma residuum opisuje, jak bardzo obliczone rozwiązanie x układu równań $Ax = b$ odbiega od rozwiązania dokładnego. Definiowana jest jako norma wektora różnicy pomiędzy lewą i prawą stroną równania. Dla metod bezpośrednich, norma residuum służy głównie do weryfikacji poprawności obliczeń. W przypadku metod iteracyjnych mamy do czynienia z sekwencją przybliżeń $x^{(k)}$ i odpowiadających im wektorów residuum $r^{(k)}$:

$$r^{(k)} = Ax^{(k)} - b \tag{3}$$

Na podstawie normy $r^{(k)}$ ocenia się, czy uzyskane przybliżenie jest wystarczająco bliskie rozwiązaniu dokładnemu. Iteracje kończą się, gdy spełnione zostanie warunek:

$$r^{(k)} < \varepsilon \tag{4}$$

gdzie ε to zdefiniowana przez nas dokładność (np. 10^{-9}).

Niska wartość normy residuum jest podstawą do uznania rozwiązania za poprawne, natomiast jej wzrost, zwłaszcza w trakcie iteracji, może świadczyć o rozbieżności algorytmu lub innych problemach.

2 Badane metody

2.1 Wyznaczanie rozwiązania

Aby rozwiązać układ równań liniowych musimy znaleźć takie wartości wektora x, aby po wykonaniu mnożenia, zachodziła równość po obu stronach.[1]

Metody rozwiązywania układów równań liniowych można podzielić na dwie kategorie:

- **Metody bezpośrednie** – polegają na przekształceniu układu w taki sposób, aby uzyskać dokładne (w sensie algebraicznym) rozwiązanie. Przykładami są eliminacja Gaussa, faktoryzacja LU czy wyznaczanie macierzy odwrotnej, takiej że doprowadzamy równanie do postaci $x = A^{-1}b$.[2]
- **Metody iteracyjne** – opierają się na przybliżaniu rozwiązania krok po kroku, zaczynając od zgadywanego wektora początkowego i poprawiając go w kolejnych iteracjach. Przykładami są m.in. metoda Jacobiego oraz metoda Gaussa-Seidla. Przerywamy iterowanie po otrzymaniu zadowalającej nas dokładności.

[1] W metodach numerycznych nie jesteśmy w stanie osiągnąć idealnej dokładności ze względu na skończoną precyzję reprezentacji liczb w komputerze. Dlatego celem jest minimalizacja błędu numerycznego, czyli znalezienie rozwiązania jak najbliższego dokładnemu. W implementacji metod iteracyjnych traktujemy błąd rzędu $\varepsilon = 10^{-9}$ za akceptowalne rozwiązanie.

[2] W praktyce jednak nie stosuje się metody wyznaczania macierzy odwrotnej, ponieważ jest ona bardzo złożona numerycznie i wrażliwa na błędy zaokrągleń. Znacznie bardziej stabilne i wydajne są metody, które unikają bezpośredniego obliczania A^{-1}

W projekcie wszystkie metody operują na macierzach gęstych. Porównywane są ich czasy działania, liczba iteracji (dla iteracyjnych) oraz zwracana norma residuum, czyli wartość określająca jak blisko dokładnego rozwiązania jesteśmy. Jeśli norma residuum jest wartością bliską zeru, to rozwiązanie, które otrzymujemy jest poprawne.

Spośród metod iteracyjnych zaimplementowano oraz poddano analizie:

- metodę Jacobiego,
- Gaussa-Seidla.

Z metod bezpośrednich zaimplementowano oraz zbadano:

- faktoryzację LU
- LU_optimized, czyli wersję faktoryzacji LU zoptymalizowaną pod przetwarzanie macierzy pasmowych.

W celu porównania wyników oraz zweryfikowania ich poprawności zbadano również dwie wbudowane funkcje dostępne w Pythonie:

- `scipy.linalg.lu()`
- `numpy.linalg.solve()`

2.2 Metoda Jacobiego

Metody Jacobiego i Gaussa-Seidla polegają na wyodrębnieniu z macierzy A trzech innych macierzy, a następnie przekształceniu równania, do postaci umożliwiającej iterację.
Rozbicie macierzy A na D, L, U:

$$A = D + L + U \tag{5}$$

Podstawnienie w miejsce A we wzorze ogólnym równania macierzowego nowo otrzymanych macierzy:

$$(D + L + U)x = b$$

Następnie dla metody Jacobiego przenosimy macierz $L + U$ na prawą stronę równania:

$$Dx = -(L + U)x + b \tag{6}$$

Wymnożenie obu stron przez odwrotność diagonali (czyli po prostu macierz diagonalną zawierającą odwrotne elementy, ta operacja nie jest złożona obliczeniowo) $L + U$ na prawą stronę równania:

$$x = -D^{-1}(L + U)x + D^{-1}b \tag{7}$$

gdzie:

- D jest macierzą zawierającą jedynie elementy z diagonali
- L - macierz trójkątną dolną (bez diagonali)
- U - macierz trójkątną górną (bez diagonali)

Przekształcamy równanie (7) poprzez przemnożenie stałych wartości i zapisanie ich jako osobna macierz i wektor:
$M = -D^{-1}(L + U)$
$w = D^{-1}b$
i zapisanie ich w następującej formie:

$$x^{(k+1)} = Mx^{(k)} + w \tag{8}$$

Iteracja polega na ciągłym obliczaniu coraz dokładniejszych wartości $x^{(k+1)}$, korzystając z poprzednich przybliżeń. W każdej iteracji korzystamy wyłącznie z wartości $x^{(k)}$ uzyskanych w poprzednim kroku.

2.3 Metoda Gaussa-Seidla

Z kolei metoda Gaussa-Seidla wprowadza modyfikację – nowe wartości $x_i^{(k+1)}$ są wykorzystywane natychmiast w dalszych obliczeniach tej samej iteracji. Dzięki temu metoda szybciej zbiega do rozwiązania, ale wykonuje bardziej złożone obliczenia. Dla metody Gaussa-Seidla przekształcenia wyglądają następująco:

$$(L + D)x = b - Ux \tag{9}$$

Po przemnożeniu przez odwrotność macierzy dolnotrójkątnej nasz wzór przyjąłby następującą postać, dalsze obliczenia jednak wymagałyby obliczenia macierzy odwrotnej:

$$x = (L + D)^{-1}b - (L + D)^{-1}Ux$$

W praktyce nie oblicza się odwrotności macierzy $(L + D)$, ponieważ ta operacja jest bardzo kosztowna. Mając taki układ, rozwiązujemy go przez podstawienie:

$$(L + D)x^{(k+1)} = b - Ux^{(k)} \tag{10}$$

Z prawej strony znajduje się wektor $b - Ux^{(k)}$, który obliczany jest na początku każdej iteracji. Następnie rozwiązujemy powyższy układ metodą podstawienia w przód.

Oznacza to, że nowe wartości $x_i^{(k+1)}$ obliczane są jedna po drugiej, z wykorzystaniem już zaktualizowanych wartości $x_j^{(k+1)}$ dla $j < i$:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \tag{11}$$

Każda kolejna wartość wektora x obliczana w danej iteracji korzysta już z wartości „zaktualizowanych" podczas tej samej iteracji (co odróżnia tę metodę od metody Jacobiego, w której wszystkie wartości $x_j^{(k)}$ pochodzą z poprzedniego kroku iteracyjnego).

Zarówno w metodzie Jacobiego oraz Gaussa-Seidla, po każdej iteracji obliczane jest *residuum*, czyli wektor różnicy $Ax^{(k)} - b$, którego norma kontroluje zbieżność metody. Iteracja kończy się, gdy norma residuum spadnie poniżej zadanej tolerancji ε, w naszym przypadku $\varepsilon = 10^{-9}$.

2.4 Faktoryzacja LU

Faktoryzacja LU pozwala na obliczenie rozkładu macierzy A, a następnie wykorzystanie go do rozwiązywania wielu różnych układów równań, zmieniając tylko wektor b. Operacja faktoryzacji jest kosztowna obliczeniowo ($\mathcal{O}(N^3)$ dla macierzy $N \times N$).

Polega na rozkładzie macierzy kwadratowej A na iloczyn dwóch macierzy: dolną trójkątną L oraz górną trójkątną U, a następnie podstawienie ich w równaniu $Ax = b$. Oznacza to wykonanie następujących przekształceń:

$$A = LU \tag{12}$$

Macierze L i U wyznaczamy w następujący sposób zaczynając od wartości początkowych:

$$L = I, \quad U = A$$

gdzie I to macierz jednostkowa.

Następnie dla każdej kolumny i (od $i = 1$ do $N - 1$) aktualizujemy wartości L i U:

$$L_{ij} = \frac{U_{ij}}{U_{ii}}, \quad \text{dla} \quad i > j$$

Wyznaczamy kolumnowo elementy L poniżej głównej przekątnej. Są to wartości mnożnika, które następnie wykorzystamy do wyznaczania wartości kolejnych wierszy U:

$$U_{ij} \leftarrow U_{ij} - L_{ij} U_{jj}, \quad \text{dla} \quad i \leq j$$

Odejmowanie przemnożonych wierszy przeprowadzamy za pomocą eliminacji Gaussa, otrzymując macierz U.

Podstawiając LU zamiast A w oryginalnym układzie równań, otrzymujemy nowy układ:

$$LUx = b \tag{13}$$

Układ równań $LUx = b$ może być rozwiązany poprzez dwukrotne podstawienie, w następujący sposób:

- Podstawienie w przód (rozwiązanie układu) $Ly = b$, gdzie y jest wektorem pomocniczym,
- Podstawienie wstecz (rozwiązanie układu) $Ux = y$, gdzie x to rozwiązanie końcowe.

2.5 Optymalizacja faktoryzacji LU

Macierz zdefiniowana w projekcie ma pewne właściwości pozwalające usprawnić obliczenia – jest macierzą pasmową o rozmiarze $N \times N$ składającą się w znacznym stopniu z elementów zerowych. Wartości niezerowe występują tylko na paśmie zawierającym diagonalę oraz pas o szerokości 2 komórek, powyżej i poniżej diagonali.

Oznacza to, że w zamiast iterować po N wierszach oraz kolumnach w celu obliczenia wartości L i U, przeprowadzamy stałą liczbę operacji równą szerokości pasa, sprowadzając złożoność obliczeniową do $\mathcal{O}(N)$.

Zamiast klasycznego wzoru:

$$L_{ij} = \frac{U_{ij}}{U_{jj}}, \quad \text{dla} \quad i > j$$

$$U_{ik} \leftarrow U_{ik} - L_{ij}U_{jk}, \quad \text{dla} \quad k = j, j+1, \ldots, N-1$$

Ograniczamy się tylko do elementów w paśmie:

$$L_{ij} = \frac{U_{ij}}{U_{jj}}, \quad \text{dla} \quad j < i \leq j+2$$

$$U_{ik} \leftarrow U_{ik} - L_{ij}U_{jk}, \quad \text{dla} \quad j \leq k \leq j+2$$

Dzięki temu, zamiast przetwarzać cały wiersz i kolumnę, aktualizujemy tylko maksymalnie 2 elementy powyżej i poniżej diagonali, co znacząco zmniejsza koszt obliczeń. Pozwala to na zastosowanie faktoryzacji LU nawet dla bardzo dużych rozmiarów macierzy, bez ponoszenia dużych kosztów pamięciowych i czasowych.

2.6 Wbudowane funkcje

Wbudowana funkcja `scipy.linalg.lu()` wyznacza trzy macierze: P, L oraz U. Macierz P jest macierzą permutacji która powstała w skutek stosowania pivotowania oraz zamiany wierszy miejscami, tak aby uzyskać diagonalę bez zerowych elementów. Funkcja wykorzystuje złożoną optymalizację w celu osiągnięcia szybkości dla dowolnej macierzy.

Wbudowana funkcja `numpy.linalg.solve()` służy do bezpośredniego rozwiązywania układów równań liniowych. Wewnątrz biblioteki `numpy` wykorzystywane są zoptymalizowane algorytmy numeryczne pozwalające na szybkie i dokładne rozwiązanie układu równań. W zależności od macierzy, funkcja dynamicznie wybiera optymalną metodę.

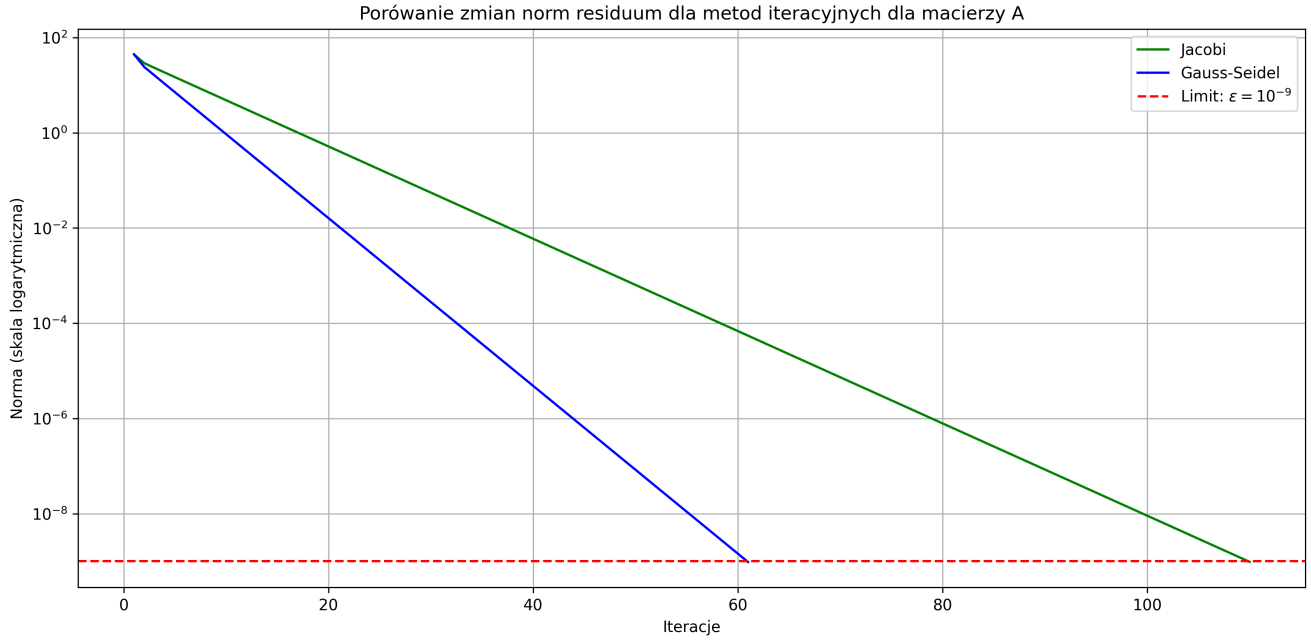
3 Analiza algorytmów

Analiza algorytmów polegała na zbadaniu każdego z sześciu algorytmów, wyznaczając rozwiązanie dla macierzy A oraz dla macierzy C, (która nie jest diagonalnie dominująca). Oprócz tego przeprowadzono symulację, generując rozwiązania kolejno dla macierzy o strukturze odpowiadającej A, lecz z rozmiarami rosnącymi od $N = 100$ do $N = 3500$, lecz została ona opisana w kolejnej sekcji.

3.1 Algorytmy iteracyjne

Algorytmy iteracyjne są metodami przybliżonymi, nie gwarantują dokładnego rozwiązania po skończonej liczbie kroków, lecz dążą do rozwiązania z określoną dokładnością.

Na wykresie przedstawiono zmiany w wartości normy residuum dla metody Jacobiego i Gaussa-Seidla. W przypadku macierzy A, metody te cechują się zbieżnością.



Rysunek 1: Porównanie metody Jacobiego i Gaussa-Seidla w wyznaczaniu rozwiązania dla macierzy A

Wynika z tego, że metoda Gaussa-Seidla osiąga pożądaną dokładność w mniejszej ilości iteracji, a sama norma residuum, w logarytmicznym tempie zbiega się do zera. Metoda Jacobiego wymaga nieco więcej iteracji, lecz nie oznacza to, że jest wolniejsza. Program zmierzył następujące czasy oraz właściwości dla obu metod:

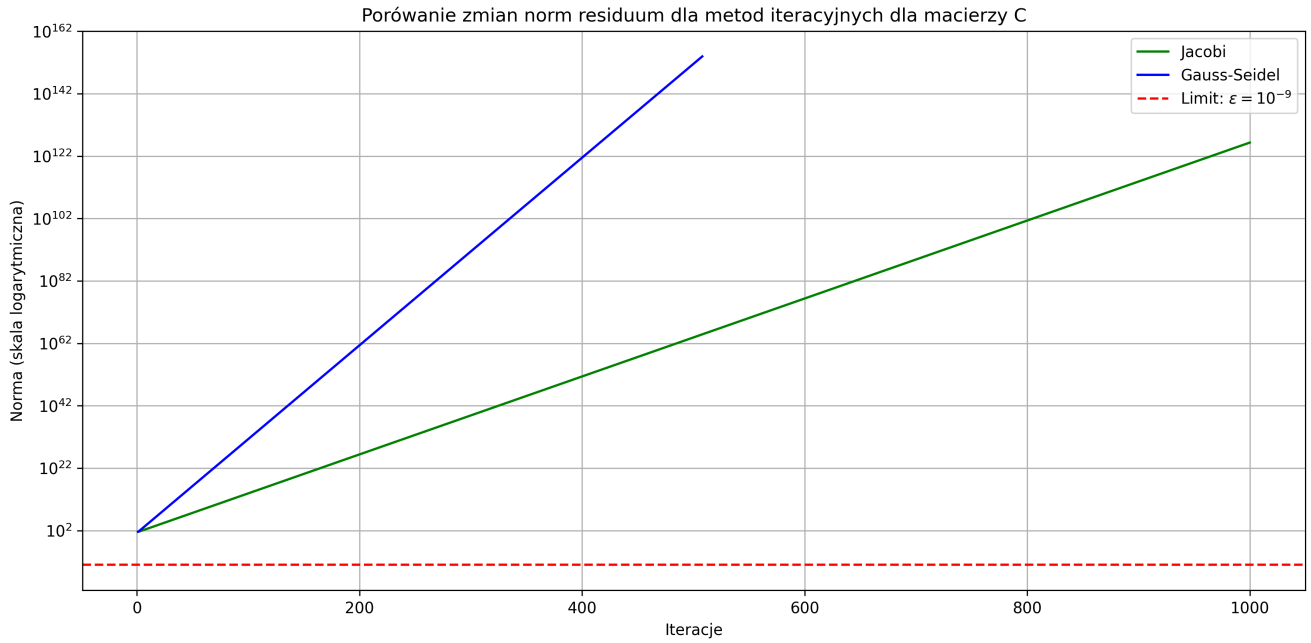
Tabela 1: Porównanie wyników metod iteracyjnych dla macierzy A, przy $N = 1299$

Metoda	**Norma residuum**	**Liczba iteracji**	**Czas wykonania [s]**
Jacobi	$9{,}7202 \times 10^{-10}$	110	0,159
Gauss-Seidel	$9{,}6398 \times 10^{-10}$	61	0,222

- Metoda Jacobiego okazała się szybsza dla badanej macierzy, obliczenia wykonywane przez tę metodę nie są skomplikowane, co przekłada się dobrze na czas, natomiast minusem jest większa liczba iteracji potrzebna do osiągnięcia rozwiązania.
- Metoda Gaussa-Seidla przeprowadza mniejszą liczbę operacji, lecz są one bardziej złożone. Czas działania jest nieco dłuższy, lecz dalej jest on bardzo szybki.

Zaletą metod iteracyjnych jest szybki czas wykonania oraz logarytmiczny wzrost liczby iteracji potrzebnej do osiągnięcia zadowalającego nas wyniku, co przekłada się na fakt, że dużą zaletą metod iteracyjnych jest ich skalowalność.

W przypadku macierzy C, algorytmy iteracyjne nie zwracają poprawnego wyniku. Macierz C nie jest diagonalnie dominująca, co może wpływać na to, że wyniki się rozbiegają. Na poniższym wykresie widać wzrost normy residuum, która powinna maleć, lecz osiąga duże wartości:



Rysunek 2: Rozbieżność przy wyznaczaniu rozwiązania dla macierzy C używając metod Jacobiego i Gaussa-Seidla

Funkcje zwracają w rozwiązaniu liczby różniące się drastycznie od oczekiwanego wyniku. Limit normy residuum wynoszącej 10^{-9} nie jest osiągany, a sama funkcja kończy się poprzez osiągnięcie ograniczenia 1000 iteracji. Na wykresie nie widać sposobu, w jaki metoda Gaussa-Seidlera osiąga 1000 iteracji, lecz jest to spowodowane jej szybszym wzrostem, który przekłada się na operowaniu na liczbach tak dużych, że są one interpretowane jako nieskończoność, a wartość (inf) nie jest definiowana jako element, który powinien się znaleźć na wykresie.

Tabela 2: Porównanie wyników metod iteracyjnych dla macierzy C, przy $N = 1299$

Metoda	**Norma residuum**	**Liczba iteracji**	**Czas wykonania [s]**
Jacobi	$2{,}2444 \times 10^{126}$	1000	1,136
Gauss-Seidel	inf	1000	3,4782

3.1.1 Wnioski

Metody iteracyjne, takie jak Jacobiego i Gaussa-Seidla, dobrze sprawdzają się dla macierzy spełniających warunki zbieżności — takich jak macierz A, która jest diagonalnie dominująca.

Mimo że metoda Jacobiego potrzebuje większej liczby iteracji, jej prostsza struktura przekłada się na krótszy czas obliczeń w analizowanym przypadku. Gauss-Seidel z kolei osiąga zbieżność w mniejszej liczbie iteracji, ale pojedyncza iteracja jest bardziej kosztowna obliczeniowo. Metody osiągają zbieżność w czasie logarytmicznym, co jest bardzo dobrym objawem.

Dla macierzy C, która nie spełnia warunku dominacji diagonalnej, obie metody iteracyjne zawodzą — obserwujemy rozbieżność, a wyniki są błędne. To pokazuje, jak silnie skuteczność metod iteracyjnych zależy od własności strukturalnych macierzy.

3.2 Metody bezpośrednie

Metody bezpośrednie dążą do wyznaczenia dokładnego rozwiązania układu równań liniowych w skończonej liczbie operacji. W odróżnieniu od metod iteracyjnych, nie polegają one na stopniowym przybliżaniu rozwiązania, lecz dokonują jednoznacznego przekształcenia macierzy wejściowej. W projekcie przetestowano następujące podejścia:

- klasyczną faktoryzację LU bez optymalizacji (o złożoności $\mathcal{O}(N^3)$),
- zoptymalizowaną faktoryzację LU (o złożonośći $\mathcal{O}(N)$),

- metody biblioteczne: `scipy.linalg.lu()` oraz `numpy.linalg.solve()`.

W tabeli poniżej przedstawiono czasy wykonania tych metod oraz osiąganą przez nie normę residuum w przypadku macierzy A:

Tabela 3: Porównanie wyników metod bezpośrednich dla macierzy A, przy $N = 1299$

Metoda	**Norma residuum**	**Czas wykonania [s]**
LU (klasyczne)	$2{,}595 \times 10^{-15}$	2,6974
LU (zoptymalizowane)	$2{,}595 \times 10^{-15}$	0.0230
`scipy.linalg.lu()`	$2{,}582 \times 10^{-15}$	0,0501
`numpy.linalg.solve()`	$3{,}342 \times 10^{-15}$	0,0401

Wszystkie metody bezpośrednie zwracają bardzo dokładne wyniki — norma residuum jest rzędu 10^{-15}, co świadczy o wysokiej precyzji rozwiązania. Klasyczna faktoryzacja LU okazała się najwolniejszą z badanych metod, co jest zgodne z oczekiwaniami ze względu na brak optymalizacji i wykonywanie niepotrzebnych operacji dla elementów zerowych. Najszybszą metodą okazała się funkcja `numpy.linalg.solve()`, która jest silnie zoptymalizowana i wykorzystuje niskopoziomowe algorytmy.

Zaimplementowane przez nas metody (z pominięciem funkcji bibliotecznych), są jednak wolniejsze od metod iteracyjnych, co potwierdza się z naszymi założeniami. Zwracany przez nie wynik jest natomiast dokładniejszy.

Tabela 4: Porównanie wyników metod bezpośrednich dla macierzy C, przy $N = 1299$

Metoda	**Norma residuum**	**Czas wykonania [s]**
LU (klasyczne)	$7{,}315 \times 10^{-14}$	2,985
LU (zoptymalizowane)	$7{,}315 \times 10^{-14}$	0,0230
`scipy.linalg.lu()`	$18{,}7481 \leftarrow$ BŁĄD	0,0543
`numpy.linalg.solve()`	$4{,}777 \times 10^{-15}$	0,0419

Ciekawym przypadkiem, który udało nam się zbadać jest fakt, że funkcja biblioteczna `scipy.linalg.lu()` **myli się** w wyznaczaniu rozwiązania dla macierzy C, osiągając przy tym normę daleko odbiegającą od pozostałych przypadków. Może to być spowodowane tym, że macierz C ma własności takie jak: brak dominacji diagonalnej, złe uwarunkowanie, brak dodatniej określoności, które sprawiają nawarstwianie się błędów numerycznych. Istniały podejrzenia, że niektóre błędy są spowodowane zastosowaniem pivotowania, lecz macierz C już zawiera maksymalne elementy na diagonali, więc realistycznie pivotowanie nie jest stosowane w żadnym z badanych algorytmów.

3.2.1 Wnioski

Metody bezpośrednie są skuteczne w uzyskiwaniu dokładnych wyników, co potwierdzają bardzo małe wartości normy residuum (rzędu 10^{-15}). W przypadku macierzy A wszystkie metody bezpośrednie dają bardzo podobne wyniki zarówno pod względem normy residuum, jak i czasu wykonania, z tym że klasyczna faktoryzacja LU, ze względu na swoją złożoność obliczeniową jest znacznie wolniejsza od wersji zoptymalizowanej oraz funkcji bibliotecznych. Zoptymalizowana wersja LU jest najszybsza dla macierzy o rozmiarze $N = 1299$, dzięki temu, że została przygotowana pod operacje na macierzy pasmowej.

W przypadku macierzy C, której własności obejmują brak dominacji diagonalnej, złe uwarunkowanie i brak dodatniej określoności, zaobserwowano, że funkcja `scipy.linalg.lu()` generuje niepoprawne wyniki. Może to być efektem nawarstwiających się błędów numerycznych, które są szczególnie widoczne przy macierzach źle uwarunkowanych jak i przystosowania algorytmu bardziej pod przypadek ogólny, niż dla takiej wyszukanej macierzy jaką jest C. Problem ten nie występuje w przypadku pozostałych metod, które przeprowadzają faktoryzację LU w prosty sposób, bez niskopoziomowych optymalizacji.

Z powyższych wyników wynika, że metody bezpośrednie są zalecane, gdy zależy nam na bardzo dokładnym rozwiązaniu, jednak w przypadku dużych układów, w szczególności z rzadkimi macierzami, lepiej sprawdzają się metody iteracyjne, które, mimo że mogą nie być tak dokładne, oferują znacznie mniejsze wymagania obliczeniowe.

4 Symulacja poszczególnych algorytmów

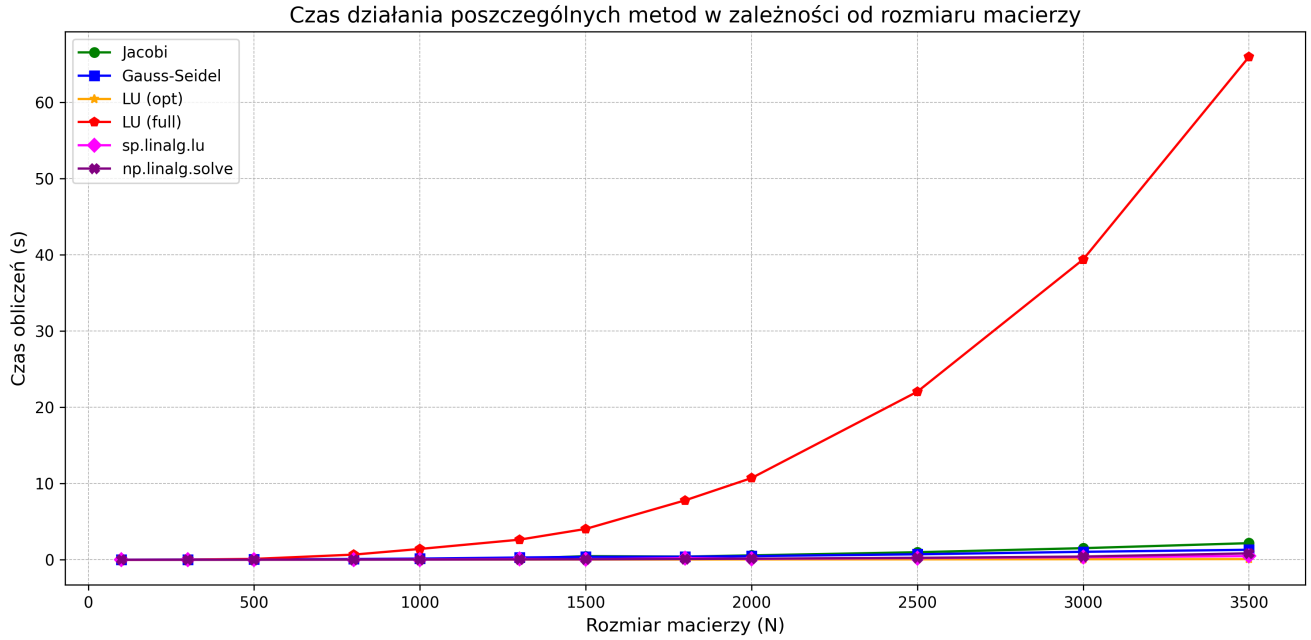
4.1 Generacja symulacji

Zbadaliśmy czasy wyznaczenia rozwiązania dla różnych metod, w zależności od rozmiaru macierzy. Następnie informacje zapisano do pliku i utworzono poniższe wykresy. Symulację przeprowadzono dla metod:

- Jacobiego,
- Gaussa-Seidla,
- LU — dla pełnej macierzy,
- LU, wersja zoptymalizowana,
- `scipy.linalg.lu()`,
- `numpy.linalg.solve()`.

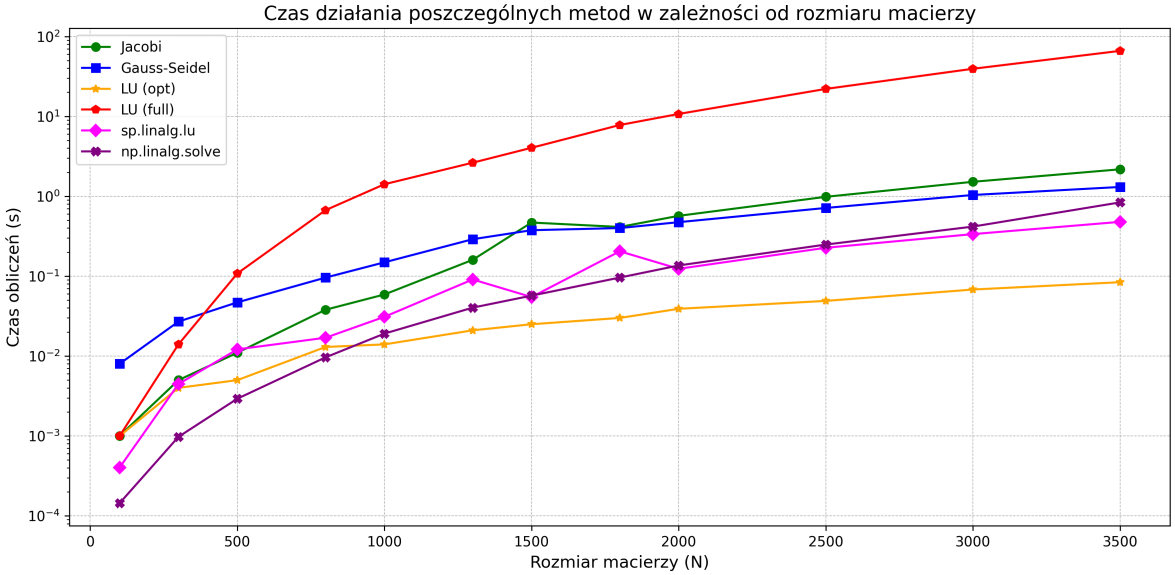
Testy wykonano dla macierzy o strukturze A, z rozmiarem N wynoszącym kolejno:

$$[100, 300, 500, 800, 1000, 1300, 1500, 1800, 2000, 2500, 3000, 3500]$$



Rysunek 3: Symulacja czasu działania algorytmów dla różnych wielkości macierzy A

Wykres na rysunku prezentuje bardzo szybki wzrost czasu obliczeń dla klasycznej faktoryzacji LU, co jest zgodne z oczekiwaną złożonością $\mathcal{O}(N^3)$. Czas dla pozostałych metod rośnie znacznie wolniej, rozmiar macierzy nie jest dla nich przeszkodą. Dla przeprowadzonej symulacji, czasy wszystkich metod oprócz pełnej faktoryzacji LU wynoszą maksymalnie do 2.17 sekund, w przypadku $N = 3500$. Natomiast zaimplementowana przez nas funkcja LU, dla największej macierzy wyznaczała rozwiązanie przez aż 65.96 sekund. Najszybsziej wypadła wersja LU z optymalizacją – wyznaczyła ona wynik w 0.084 sek.



Rysunek 4: Symulacja czasu działania algorytmów dla różnych wielkości macierzy A, skala logarytmiczna

Wykres logarytmiczny pozwala lepiej porównać metody ze sobą. Można zauważyć, że pełna wersja LU ma wyraźnie najgorszą skalowalność i staje się czasochłonna już dla macierzy o $N = 2000$. Metody wbudowane są zdecydowanie najszybsze dla macierzy o kompaktowych rozmiarach (do 1000 wierszy).

Metoda Gaussa-Seidla szybciej się zbiega i robi to w mniejszej liczbie iteracji od metody Jacobiego. Jednak dla małych macierzy metoda Jacobiego jest szybsza ze względu na prostotę obliczeń.

Metody iteracyjne są wolniejsze od funkcji bibliotecznych, ale zauważalnie szybsze od faktoryzacji LU i wykazują akceptowalną skalowalność. Najszybsza okzauje się jednak zoptymalizowana metoda LU, znając strukturę badanej przez nas macierzy jesteśmy w stanie o wiele bardziej usprawnić czas obliczeń. Dzieje się tak dlatego, że dla bardzo dużych macierzy uwzględnianie jedynie struktury pasmowej jest bardzo wydajne.

4.2 Wnioski

Dla dużych rozmiarów macierzy metoda pełnego LU jest bardzo nieefektywna. Wykorzystanie właściwości strukturalnych macierzy, w naszym przypadku fakt, że badana macierz jest macierzą pasmową pozwala znacząco zredukować czas działania — optymalizowana wersja LU osiąga złożoność liniową i jest znacznie szybsza od innych badanych metod dla macierzy sporych rozmiarów. Metody iteracyjne zachowują równowagę między szybkością a prostotą implementacji i dobrze się skalują, co czyni je praktycznym wyborem w wielu zastosowaniach.

5 Podsumowanie

Projekt zawiera implementację czterech metod rozwiązywania układów równań liniowych oraz porównanie ich z dwiema wbudowanymi funkcjami bibliotecznymi. Zwracane wyniki zależą również od właściwości macierzy takich jak diagonalna dominacja, dodatnia określoność, uwarunkowanie. Oto parę najważniejszych wniosków:

- Klasyczna metoda LU okazuje się najwolniejszym z badanych algorytmów. Ma ona złożoność $\mathcal{O}(N^3)$ i przetwarza macierz o rozmiarze $N = 1299$ w 3.36 s, dla macierzy o $N = 3500$, czas dochodzi do 65.96 s,
- Warunkiem zbieżności metody Jacobiego oraz Gaussa-Seidla jest diagonalna dominacja badanej macierzy. Nie każda macierz jest możliwa do przetworzenia tymi metodami, natomiast plusem metod iteracyjnych jest ich wolne tempo wzrostu czasu obliczeń dla sporych macierzy. Metoda Gaussa-Seidla zbiega się szybciej i przeprowadza mniej iteracji od metody Jacobiego, choć dla małych macierzy metoda Jacobiego jest szybsza.
- Optymalizacja algorytmu pod kątem informacji o danych wejściowych, pozwala osiągnąć o wiele lepszy czas i wyeliminować niepotrzebne operacje. W przypadku wersji LU zoptymalizowanej pod przetwarzanie macierzy pasmowych osiągnęliśmy czas 0.084 s. dla macierzy o rozmiarze $N = 3500$.

- Udało nam się znaleźć dosyć rażący błąd – funkcja biblioteczna `scipy.linalg.lu()` zwracała błędny wynik dla macierzy C, która nie jest dodatnio określona, nie jest dobrze uwarunkowana i nie jest diagonalnie dominująca, co czyni ją trudną w przetwarzaniu. Natomiast wszystkie pozostałe metody bezpośrednie, zwróciły dla tej macierzy dokładne rozwiązanie z wysoką precyzją wyniku.
- Wykorzystanie biblioteki `numpy` do reprezentacji danych zapisanych macierzy jako typ `numpy.float64` oraz do szybkich operacji na macierzach, znacznie przyspieszyło obliczenia oraz pozwoliło zachować dużą precyzję. Wpływ na szybkość obliczeń miały również zastąpienie pętli `for` przez operacje na wektorach.

Ostatecznie, dobór odpowiedniej metody numerycznej w zależności od typu i właściwości macierzy ma kluczowe znaczenie dla efektywności i dokładności obliczeń. Metody iteracyjne mogą być niezwykle wydajne przy dużych, dobrze uwarunkowanych macierzach, natomiast metody bezpośrednie zapewniają wysoką precyzję, nawet dla bardziej wymagających przypadków. Optymalną metodą wyznaczenia rozwiązania układu równań liniowych jest wpierw zbadanie macierzy, a następnie dobranie algorytmu dobrze spisującego się dla macierzy o danych właściwościach.