Piotr Noga, 188839
Układ równań liniowych – Projekt z przedmiotu Metody Numeryczne

Wprowadzenie:

Układ równań liniowych składa się z wielu niewiadomych, które aby obliczyć, wymagają często wiele czasu oraz sporej ilości kalkulacji. Sprawa się tym bardziej komplikuje im więcej niewiadomych znajduje się w danym układzie. Aby zapisać dany układ równań liniowych, często wykorzystuje się w tym celu macierze. Każda kolumna macierzy reprezentuje wtedy daną zmienną w każdym z równań, natomiast każdy wiersz oznacza dane równanie liniowe z układu. Taki zapis często ułatwia obliczanie wielu zmiennych, gdyż istnieją metody, które obliczają przybliżone wartości niewiadomych w stosunkowo krótkim czasie. Jedne z takich metod to metody iteracyjne.

Metody iteracyjne

Polegają na wyliczaniu coraz dokładniejszych wartości wraz z kolejną iteracją algorytmu liczącego dany układ równań liniowych. W celu zademonstrowania tych metod, w kodzie dołączonym do sprawozdania zostały zaimplementowane metody Jacobiego oraz Gaussa-Seidla. By porównać działanie tych metod, została zaimplementowana również metoda bezpośredniego rozwiązania układu równań liniowych poprzez faktoryzację LU.

Metody bezpośrednie

Jedną z metod bezpośrednich jest wspomniana wcześniej fakturyzacja LU. Polega na utworzeniu z danej macierzy dwóch macierzy trójkątnych, jedna górna (U) i jedna dolna z jedynkami na przekątnej (L), by następnie podstawianiem w przód (Ly = b) i podstawianiem w tył (Ux = y) wyliczyć wektor szukanych wartości (x). Wspomniane podstawianie w przód polega na iteracyjnym obliczaniu wartości kolejnych niewiadomych i podstawianiu już obliczonych wartości z poprzednich równań. Zasada działania podstawiania w tył wygląda podobnie, z tą różnicą, że wyliczone wartością podstawiane są w odwrotnej kolejności.

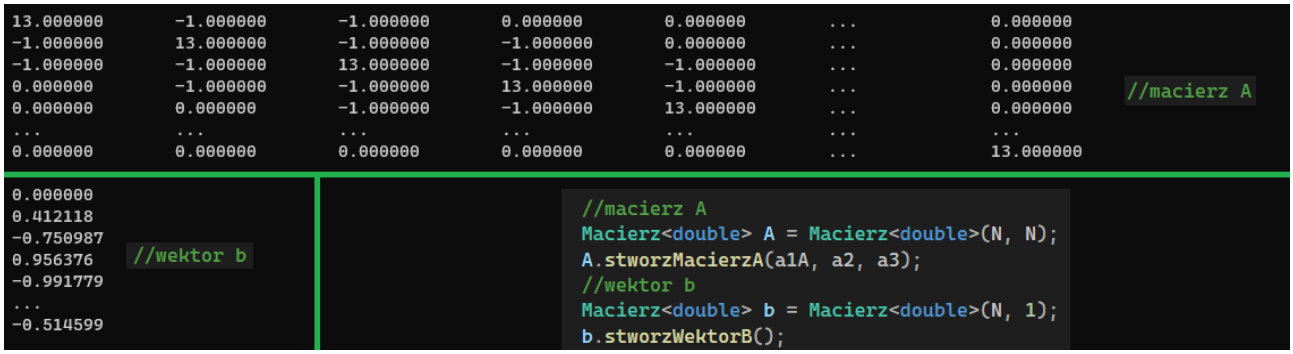
Implementacja wzorów

Wszelkie wzory dotyczące implementacji poszczególnych metod, które zostały wspomniane wyżej, jak i również pozostałych algorytmów wykorzystanych w kodzie programu dołączonym do sprawozdania w oddzielnych plikach, zaczerpnięte zostały z następujących źródeł:

- *https://enauczanie.pg.edu.pl/moodle/pluginfile.php/2310658/mod_resource/content/1/MN_projekt_2.pdf* – Instrukcja dotycząca tematu projektu. Zostały wykorzystane stąd forma macierzy oraz wektor residuum
- *https://enauczanie.pg.edu.pl/moodle/pluginfile.php/2310614/mod_resource/content/2/Wyklad2.pdf* – Wykład numer 2 z przedmiotu Metody Numeryczne, w którym został zawarty algorytm faktoryzacji LU
- *https://enauczanie.pg.edu.pl/moodle/pluginfile.php/2310636/mod_resource/content/3/Laboratorium_3.pdf* – Instrukcja dotycząca laboratorium numer 3 z przedmiotu Metody Numeryczne, w którym zawarte są wszelkie informacje na temat macierzy trójkątnych, diagonalnych, wzory do implementacji metody Jacobiego oraz metody Gaussa-Seidla
- *https://www.geeksforgeeks.org* – Różne artykuły związane z implementacją operacji związanych z macierzami – dodawanie i mnożenie macierzy, odwracanie macierzy diagonalnej, Forward Substitution oraz Backward Substitution, wyświetlanie macierzy

Zadanie A

W tym zadaniu należy utworzyć układ równań liniowych w postaci **Ax=b**, gdzie **A** to rozpatrywana macierz, **x** to wektor rozwiązań, a **b** to wektor pobudzeń (Rysunek 1). Macierz *A*, której rozmiar wynosi *N* x *N*, gdzie *N* jest równe 939, składa się z głównej diagonali, gdzie każda jej składowa (*a1*) wynosi 13. Posiada również cztery diagonale, które są sąsiednie do głównej, po dwie na każdą stronę. Ich składowe (*a2* oraz *a3*) są równe -1. Wektor *b* o długości *N* ma takie elementy, że *n*-ty element jest równy *sin(n(f+1))*, gdzie *f* jest równy 8. Aby rozwiązać ten układ równań, należy wyliczyć wektor *x*, co nastąpi w zadaniu B.

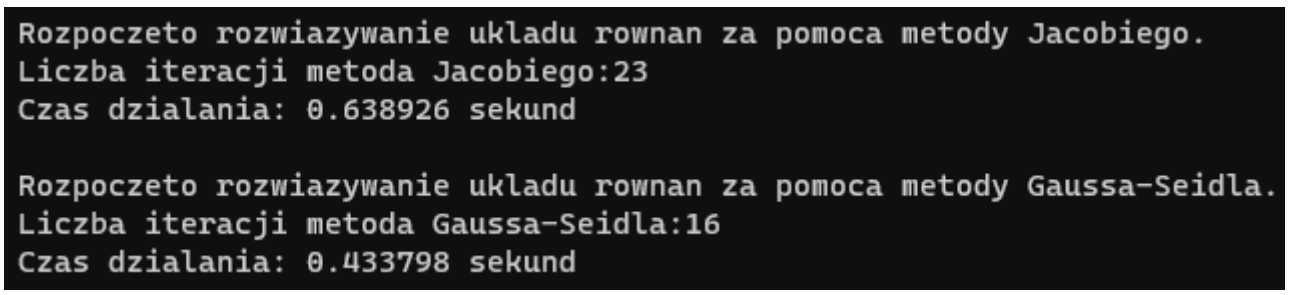


Rysunek 1: Utworzenie macierzy A i wektora b

Zadanie B

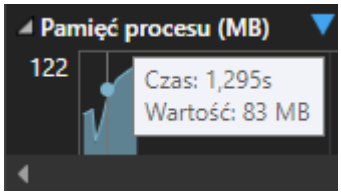
Porównywane są tutaj metody Jacobiego i Gaussa-Seidla. Na testowanym zestawie macierzy i wektorów wyniki wyglądają następująco (Rysunek 2):

Metoda Jacobiego wyliczyła wektor x w czasie około 0,64 sekundy i potrzebowała na to 23 iteracje. Metoda Gaussa-Seidla prezentuje się lepiej od poprzedniej, gdyż ten sam wektor x wyliczyła w krótszym czasie - 0,43 sekundy i potrzebowała przy tym mniejszej liczby iteracji, bo tylko 16. Na wykresie (Rysunek 5), można zobaczyć, jak stopniowo normy zbliżały się do zadowalającego poziomu.



Rysunek 2: Porównanie metod Jacobiego i Gaussa-Seidla pod względem czasu działania oraz liczby iteracji

W trakcie wykonywania tego zadania powoli można dostrzec narastający problem związany z zarządzaniem pamięcią w takim programie, co widać na poniższym rysunku (Rysunek 3).

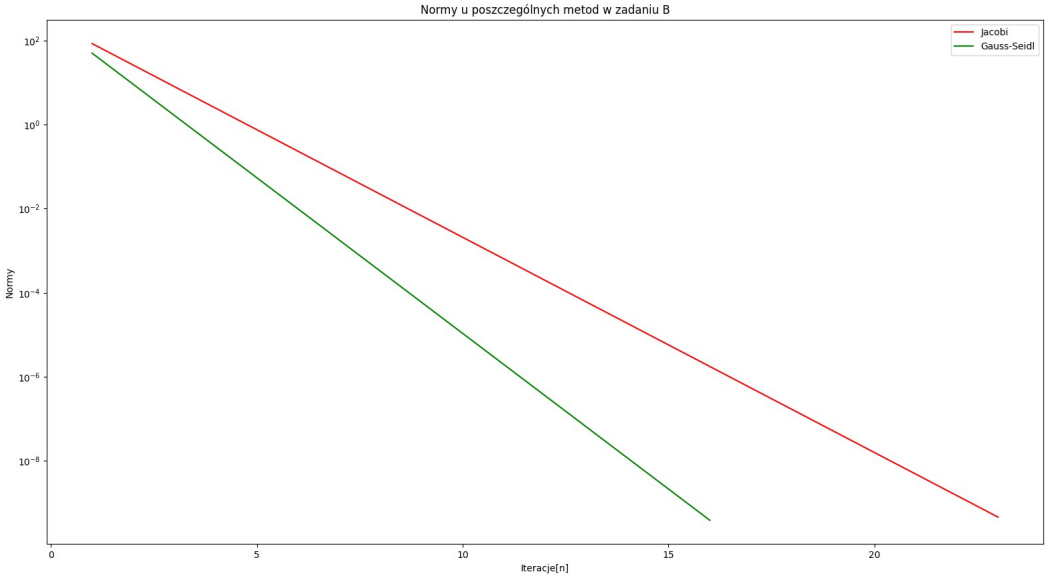


Rysunek 3: Pojemność zużywanej pamięci wzrasta wraz z kolejnymi iteracjami algorytmów

W wyniku działania programu w taki sposób, powinno się zwalniać z pamięci utworzone macierze, dlatego też zaraz po zmierzeniu czasu rozwiązywania układu równań liniowych oraz liczby iteracji, podstawowa macierz jest usuwana z pamięci (Rysunek 4).

```
//zad B
jacobi(A, b);
gauss(A, b);
A.usunMacierz();
```

Rysunek 4: Wywołanie metod Jacobiego oraz Gaussa-Seidla na macierzy A i wektorze b, a następnie usuwanie z pamięci macierzy A



Rysunek 5: Normy dla poszczególnych metod iteracyjnych w zależności od liczby iteracji

Zadanie C

Tworzenie zestawu macierzy jest niemal identyczne, co w zadaniu A (Rysunek 6 oraz Rysunek 7), z tą różnicą, że główna diagonalna macierzy A nie składa się z elementów a1 równymi 13, tylko 3.

3.000000	-1.000000	-1.000000	0.000000	0.000000	...	0.000000
-1.000000	3.000000	-1.000000	-1.000000	0.000000	...	0.000000
-1.000000	-1.000000	3.000000	-1.000000	-1.000000	...	0.000000
0.000000	-1.000000	-1.000000	3.000000	-1.000000	...	0.000000
0.000000	0.000000	-1.000000	-1.000000	3.000000	...	0.000000
...	...	...	...	...	...	...
0.000000	0.000000	0.000000	0.000000	0.000000	...	3.000000

Rysunek 6: Macierz A z zadania C

```
Macierz<double> Ac = Macierz<double>(N, N);
Ac.stworzMacierzA(a1C, a2, a3);
jacobi(Ac, b);
gauss(Ac, b);
```

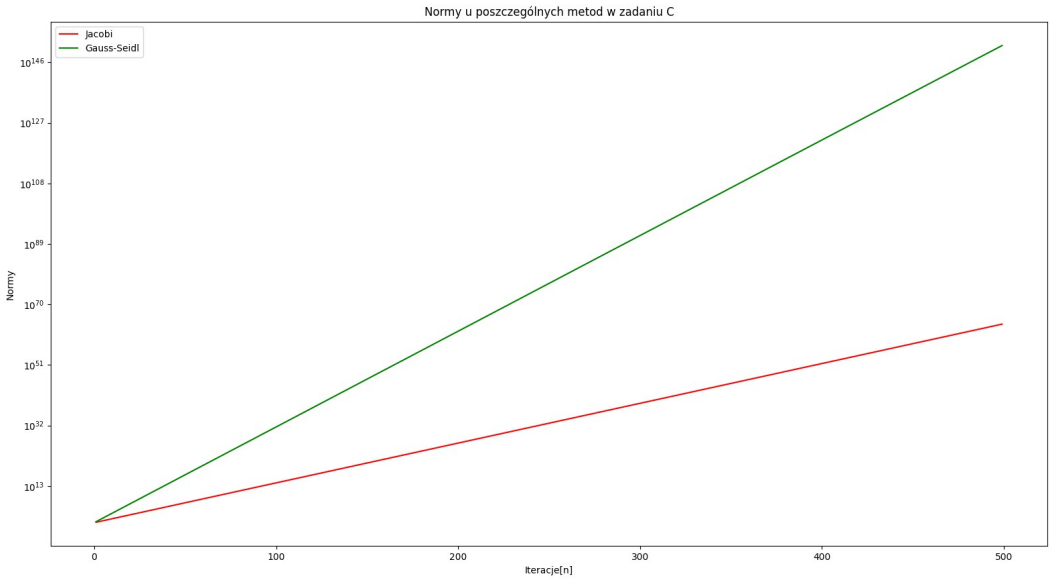
Rysunek 7: Tworzenie macierzy A do zadania C

Mimo tak małej zmiany, wpływa to diametralnie na rozwiązanie tego układu równań, gdyż obie metody iteracyjne mają problem z jej prawidłowym rozwiązaniem (Rysunek 8).

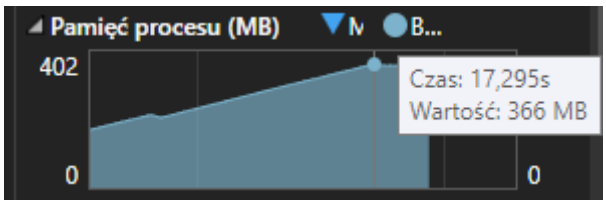
```
Rozpoczeto rozwiazywanie ukladu rownan za pomoca metody Jacobiego.
Przekroczono limit iteracji wynoszacy 500.
Czas dzialania: 7.703556 sekund

Rozpoczeto rozwiazywanie ukladu rownan za pomoca metody Gaussa-Seidla.
Przekroczono limit iteracji wynoszacy 500.
Czas dzialania: 8.947174 sekund
```

Rysunek 8: Przekroczenie limitu iteracji w obu metodach iteracyjnych

Wynika to z faktu, że wartości dla tych metod nie zbiegają się. Norma rezydualna zamiast zbiegać do zera, zbiega do nieskończoności (Rysunek 9).



Rysunek 9: Normy dla poszczególnych metod iteracyjnych w zależności od liczby iteracji



Rysunek 10: Zajęcie pamięci po zakończeniu działania metody Gaussa-Seidla

Warto zwrócić uwagę na pamięć, jaką do tej pory zajmuje program (Rysunek 10). Mimo próby usuwania z pamięci tymczasowych macierzy, które są tworzone w trakcie wykonywania danych algorytmów na przykład macierze trójkątne czy też macierze diagonalne, nie wszystkie z nich mogą zostać usunięte z pamięci. Próba napisania własnego destruktora klasy Macierz skutkowała tym, że był on wywoływany w złych momentach, przez co algorytmy działały nieprawidłowo. Stąd też decyzja, aby usuwać tyle obiektów klasy Macierz, na ile pozwala prawidłowe działanie algorytmów.

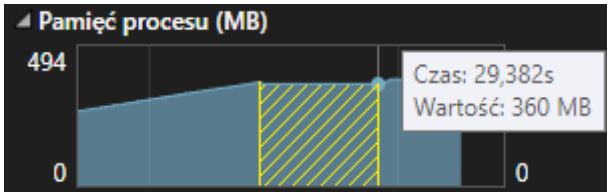
Zadanie D

Jeśli wartości w metodach iteracyjnych nie zbiegają się, to można spróbować rozwiązać układ równań liniowych w sposób bezpośredni. Jedną z takich metod rozwiązujących taki układ w sposób bezpośrednich jest faktoryzacja LU (Rysunek 11). W tym przypadku taka metoda faktycznie prawidłowo wyliczyła wektor x. Norma z residuum wyniosła 1,68 * 10^-12. Warto odnotować, że wyliczenie rozwiązania zajęło lekko ponad 4 sekundy, co jak się okaże w kolejnym zadaniu, jest to stosunkowo długo w porównaniu do metod iteracyjnych.

```
Rozpoczeto faktoryzacje LU.
Czas dzialania: 4.279753 sekund

Norma faktoryzacji LU wynosi: 0.00000000000167964254
9.147091
12.703754
14.737518
13.814534
9.298163
...
-9.367802
```

Rysunek 11: Faktoryzacja LU z sukcesem wyliczył normę faktoryzacji oraz wektor x, widoczny pod normą

W trakcie pracy programu można zauważyć, że pamięć wykorzystana przez niego w trakcie faktoryzacji LU jest stała (Rysunek 12), gdyż nie są tworzone tymczasowe obiekty klasy Macierz.



Rysunek 12: Pamięć procesu w trakcie faktoryzacji LU zaznaczona na żółto

Zadanie E

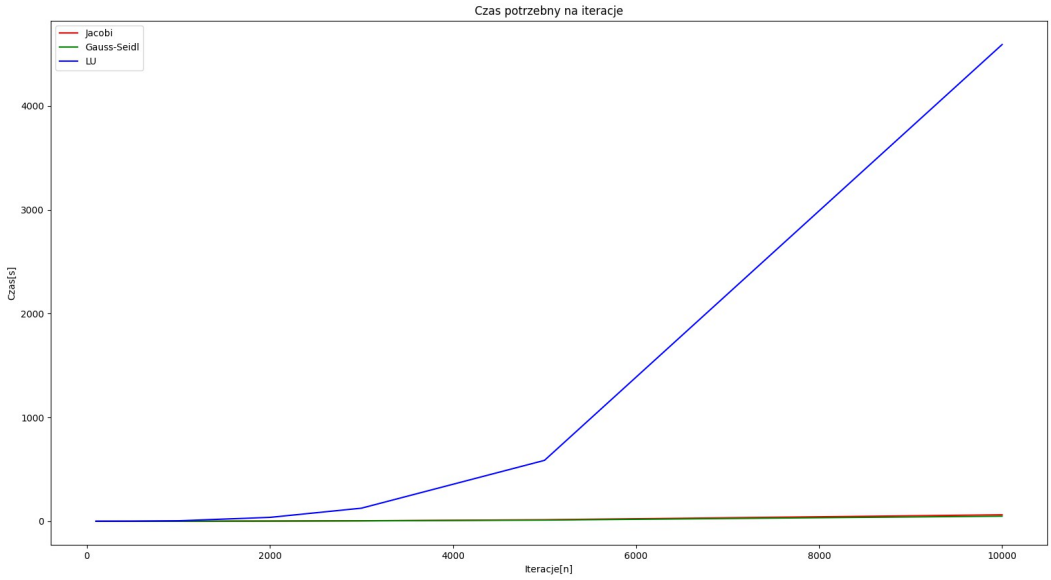
Rozpatrywane są macierze rozmiarów N={100, 500, 1000, 2000, 3000, 5000, 10000}.

Metody iteracyjne w stosunkowo krótkim czasie liczą wektor x, gdy faktoryzacja LU potrzebuje zdecydowanie więcej czasu, aby podać rozwiązanie (Rysunek 13 i Rysunek 19).

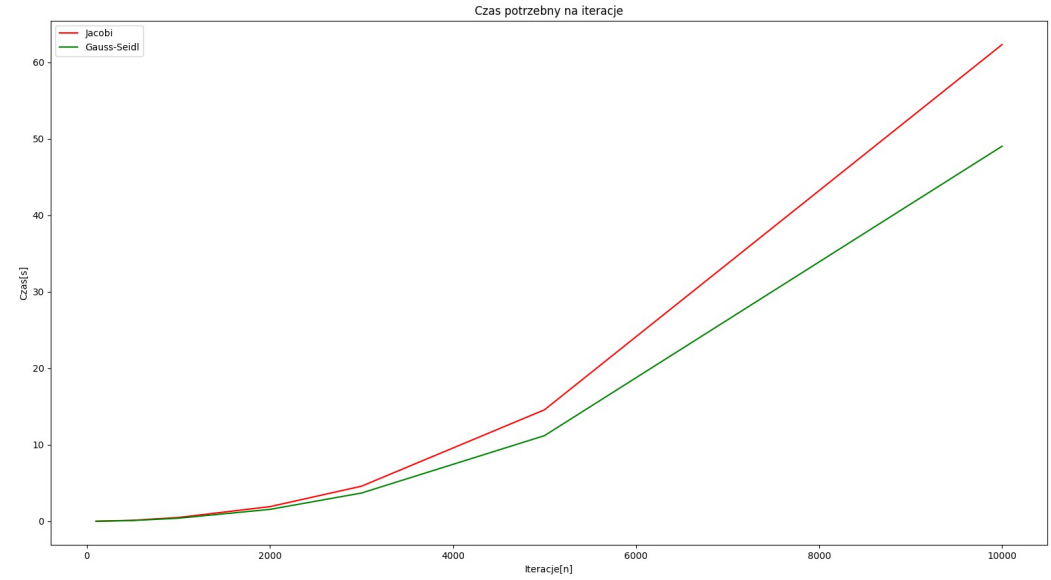
```
Obecnie rozpatrywany rozmiar macierzy N: 2000
Rozpoczeto rozwiazywanie ukladu rownan za pomoca metody Jacobiego.
Liczba iteracji metoda Jacobiego:23
Czas dzialania: 2.366021 sekund

Rozpoczeto rozwiazywanie ukladu rownan za pomoca metody Gaussa-Seidla.
Liczba iteracji metoda Gaussa-Seidla:16
Czas dzialania: 2.048963 sekund

Rozpoczeto faktoryzacje LU.
Czas dzialania: 43.402000 sekund
```

Rysunek 13: Przykładowy wynik działania poszczególnych metod, aby uzykać wektor x

W celu zobrazowania różnic w czasie między poszczególnymi metodami, program zapisał do pliku wyniki.csv (dołączonym do sprawozdania) wyniki działania poszczególnych metod w różnych sytuacjach tj. różny rozmiar macierzy oraz metoda obliczająca rozwiązanie. Wyniki przedstawiają się następująco (Rysunki 14, 15, 16, 17 i 18):



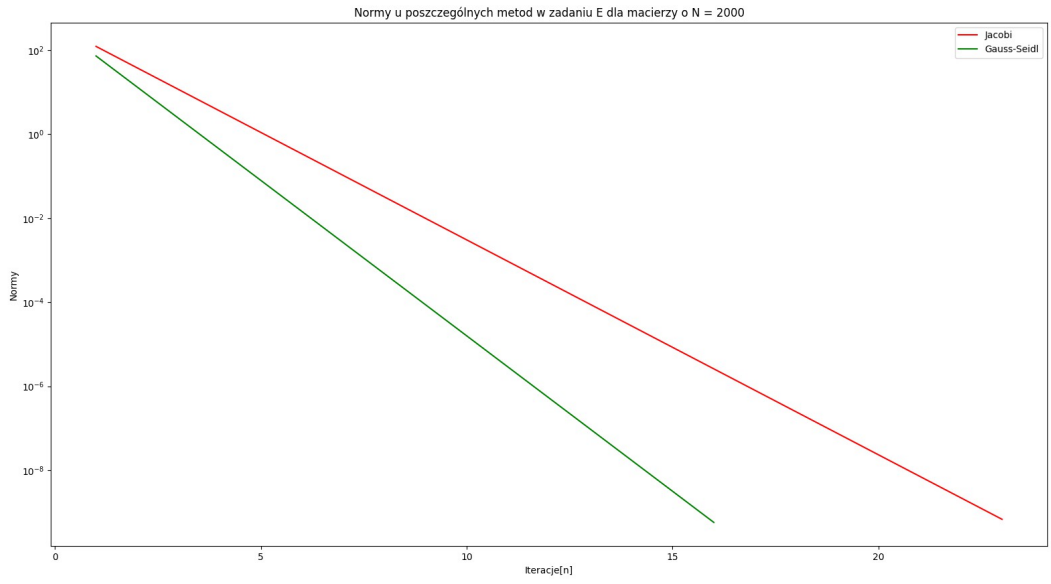
Rysunek 14: Wykres przedstawiający czas potrzebny na iteracje w zależności od liczby iteracji. Przedstawione zostały zastosowane metody



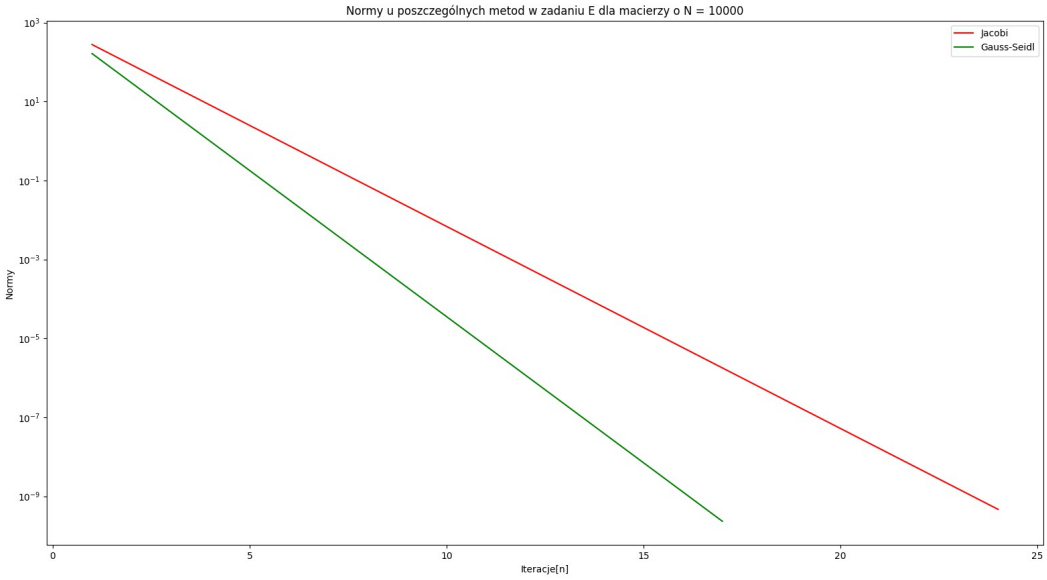
Rysunek 15: Wykres przedstawiający czas potrzebny na iteracje w zależności od liczby iteracji, z pominięciem faktoryzacji LU, w celu lepszego podglądu na różnicę czasową między metodami iteracyjnymi

0	method	size	iterations	time
1	Ja	100	22	0.006653
2	GS	100	15	0.005124
3	LU	100	1	0.005428
4	Ja	500	23	0.128625
5	GS	500	16	0.100947
6	LU	500	1	0.588634
7	Ja	1000	23	0.496992
8	GS	1000	16	0.403071
9	LU	1000	1	4.668764
10	Ja	2000	23	1.915702
11	GS	2000	16	1.558081
12	LU	2000	1	37.362798
13	Ja	3000	23	4.585888
14	GS	3000	16	3.693905
15	LU	3000	1	126.024851
16	Ja	5000	24	14.579891
17	GS	5000	16	11.197917
18	LU	5000	1	586.694629
19	Ja	10000	24	62.332603
20	GS	10000	17	49.043876
21	LU	10000	1	4592.443786

Rysunek 16: Tabela przedstawiająca jaka metoda została zastosowana (Ja – Jacobi, GS – Gauss-Seidl, LU – faktoryzacja LU), jakiego rozmiaru była rozpatrywana macierz, ile iteracji było potrzebnych do uzyskania wyniku oraz ile czasu w sekundach potrzebował dany algorytm.



Rysunek 17: Normy w trakcie obliczania wyniku metodami iteracyjnymi dla N = 2000



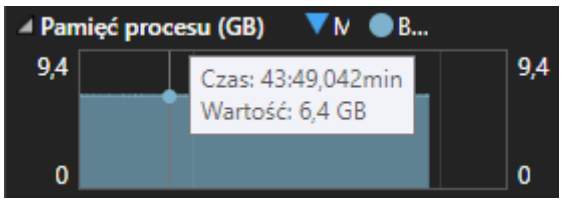
Rysunek 18: Normy w trakcie obliczania wyniku metodami iteracyjnymi dla N = 10000

Dla macierzy mających rozmiar N powyżej 2000 widać znaczny wzrost czasu potrzebnego na wyliczenie układu równań poprzez faktoryzację LU w porównaniu do metod iteracyjnych. W przypadku N = 10000 czas potrzebny na uzyskanie wyniku metodą faktoryzacji LU jest ponad 93 razy większy (!) od czasu zużytego przez metodę Gaussa-Seidla. Wynosi on w przeliczeniu na minuty ponad 76 minut, gdy metoda Gaussa-Seidla uzyskuje podobny wynik w przeciągu 49 sekund. Poniższy rysunek (Rysunek 19) przedstawia podobną sytuację, wykonaną podczas innej próby:

```
Obecnie rozpatrywany rozmiar macierzy N: 10000
Rozpoczeto rozwiazywanie ukladu rownan za pomoca metody Jacobiego.
Liczba iteracji metoda Jacobiego:24
Czas dzialania: 60.404086 sekund

Rozpoczeto rozwiazywanie ukladu rownan za pomoca metody Gaussa-Seidla.
Liczba iteracji metoda Gaussa-Seidla:17
Czas dzialania: 48.489064 sekund

Rozpoczeto faktoryzacje LU.
Czas dzialania: 4801.333844 sekund
```

Rysunek 19: Próba rozwiązania układu równań dla macierzy A o rozmiarze N = 10000 za pomocą trzech metod

Warto dodać, że w przypadku macierzy o rozmiarze N = 10000 program zajmuje ponad 6GB pamięci (Rysunek 20).



Rysunek 20: Pamięć w trakcie obliczania wyniku dla macierzy o rozmiarze N = 10000

Zadanie F/Podsumowanie:

Po obserwacji poprzednich zadań nasuwają się dwa wnioski. Jednym z nich jest wybór między metodami iteracyjnymi, a metodą bezpośrednią. Jeśli użytkownikowi nie zależy w istotnym stopniu na dokładności obliczeń i pragnie otrzymać wyniki jak najszybciej, to z pewnością będzie wolał wybrać metody iteracyjne, a spośród nich metodę Gaussa-Seidla ze względu na szybsze działanie algorytmu od metody Jacobiego. Jeśli dokładność obliczeń dla użytkownika jest dość istotna, to może spróbować wpierw wykorzystać metody iteracyjne, w których zaostrzy wymagania co do normy euklidesowej wektora residuum, co umożliwi uzyskanie jeszcze bardziej dokładnych wyników, niż w standardowej implementacji tych metod. Jeśli uzyskanie wyników tymi metodami nie będzie skuteczne na przykład w wyniku braku zbieżności wyników tymi metodami, użytkownik będzie musiał użyć metody bezpośredniej. Użycie faktoryzacji LU zajmuje zdecydowanie więcej czasu, by otrzymać wynik, jednakże użytkownik z pewnością otrzyma szukany wynik oraz norma residuum będzie na zadowalającym poziomie.

Drugim wnioskiem, który przychodzi na myśl, jest wybór między własną implementacją metod, bądź wykorzystaniu zewnętrznych rozwiązań. Rozpatrując powyższe wyniki można dojść do konkluzji, iż warto jest korzystać z rozwiązań przygotowanych przez zewnętrzne firmy, gdyż prowadzą one do niemal tych samych wyników, przy często krótszym czasie pracy i przy mniejszym wykorzystaniu pamięci komputera. Dodatkowo są one sprawdzane od wielu lat przez wielu użytkowników, przez co są one bardziej odporne na wszelkiego rodzaju błędy, które mogłyby wystąpić przy własnoręcznej implementacji tychże rozwiązań. Własną implementację powinno się stosować jedynie w skrajnych przypadkach, gdy potrzebne są niestandardowe typy danych, które mogą przechowywać również inne informacje, niż tylko samą macierz.

Wykorzystane języki programowania

Główna część programu tj. stosowne obliczenia, implementacja macierzy, metod iteracyjnych oraz metody bezpośredniej, została napisana w języku C++, natomiast wizualizacja wyników w formie wykresu (rysunki 14, 15, 16, 17 i 18), została napisana w języku Python.