Metody Numeryczne
Projekt 2 - Układy Równań Liniowych

Mateusz Blach 193174

1. Wstęp

Projekt polega na implementacji oraz analizie trzech metod rozwiązywania układów liniowych. Układy równań liniowych na ogół mają postać: **Ax = b**. Aby wyłuskać z nich wektor szukany **x**, musielibyśmy odwrócić macierz systemową **A**, co jest bardzo kosztowne zarówno czasowo, jak i pamięciowo. Z tego powodu takie układy rozwiązuje się na różne sposoby, które w zależności od problemu i parametrów, mogą radzić sobie gorzej, bądź lepiej. Metody te możemy podzielić na iteracyjne oraz bezpośrednie. Metody iteracyjne polegają na wykonaniu serii iteracji, które przy każdej kolejnej iteracji powinny przybliżać nas coraz bardziej do poprawnego rozwiązania układu. Na ogół są one mniej dokładne niż metody bezpośrednie, jednak z natury są one szybsze i wydajniejsze, w szczególności dla ogromnych układów równań. Najlepiej stosować je, gdy zależy nam na szybkości. Sporym minusem owych metod jest wymóg określonych warunków zbieżności, bez których rozwiązanie zamiast zbiegać do poprawnego, będzie dążyć do nieskończoności. Algorytmy te są bardzo czułe na warunki początkowe. Spośród metod iteracyjnych możemy wyróżnić dwie - **Metoda Jacobiego** oraz **Metoda Gaussa-Seidla**. To na nich skupimy naszą uwagę.

Drugim rodzajem metod rozwiązujących skomplikowane układy równań są metody bezpośrednie. Metody te polegają na rozwiązywaniu układów równań poprzez wykonywanie określonych kroków obliczeniowych, prowadzących bezpośrednio do uzyskania poprawnego rozwiązania. W przeciwieństwie do metod iteracyjnych nie są one aż tak wrażliwe na warunki początkowe, praktycznie zawsze zwracają poprawne wyniki kosztem wysokiego czasu obliczeń, widocznego szczególnie dla bardzo dużych układów równań. Jedną z metod bezpośrednich rozwiązywania układów równań liniowych jest **faktoryzacja LU**.

2. Metody Iteracyjne

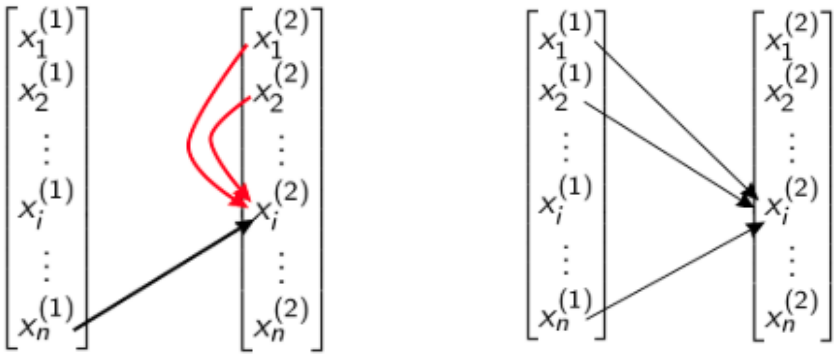
W pierwszej kolejności zajmiemy się metodami iteracyjnymi. W **metodzie Jacobiego** na początku inicjalizujemy wektor początkowy $\mathbf{x}^{(0)}$. W tym przypadku dla uproszczenia wszystkie wyrazy wektora wynosić będą zero, a następnie w kolejnych **k**-iteracjach nasz wektor wynikowy obliczać będziemy ze wzoru:

$$x_i^{(k+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)})/a_{ii}$$

Dla każdego **i**-tego równania rozważany jedno równanie liniowe w układzie. $\mathbf{b_i}$ to wartość oczekiwana po lewej stronie równania. Od niej odejmujemy sumę współczynników poza diagonalą przemnożonych przez wartość wektora wynikowego z poprzedniej iteracji, a następnie dzielimy różnicę przez wartość współczynnika leżącego na diagonali. W ten sposób krok po kroku otrzymujemy wartość **i**-tej zmiennej wektora **x**, tak aby lewa strona równania była zbliżona do prawej. Jeżeli zamiast używać danych z wektora poprzedniego, bralibyśmy również dane z aktualnego przybliżenia, będziemy stosować **metodę Gaussa-Seidla**.

$$x_i^{(k+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)})/a_{ii}$$

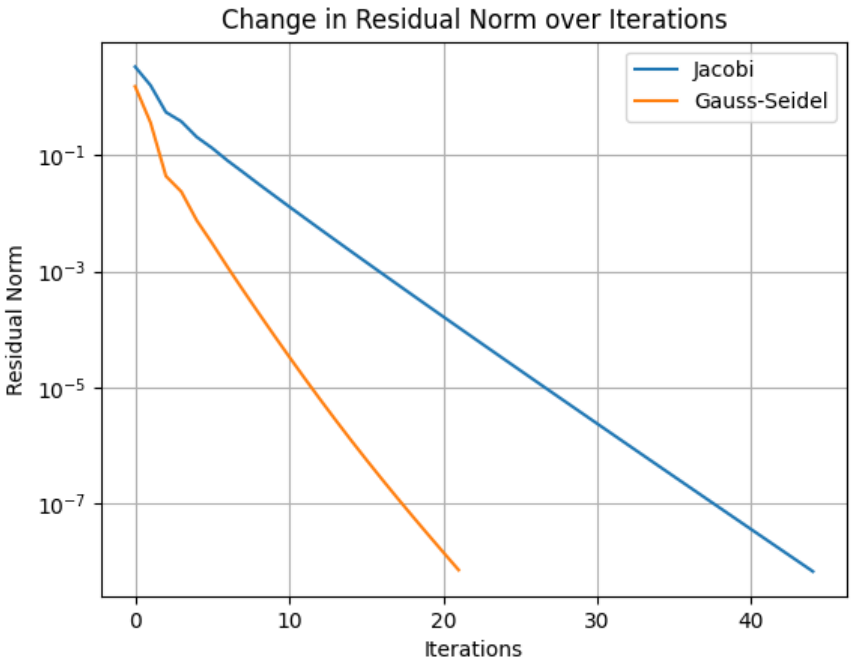
.



Po lewej stronie widzimy graficzną reprezentację iteracji w metodzie Gaussa-Seidla, po prawej – Jacobiego.

W celu określenia jakości przybliżenia wektora szukanego, tworzymy wektor **residuum,** który powstaje poprzez przeniesienie wszystkich składników równania na jedną stronę, stąd: $\mathbf{res^{(k)} = Ax^{(k)} - b}$. Badając normę euklidesową residuum ($\mathbf{norm(res^{(k)})}$), która jest niczym innym jak pierwiastkiem z sumy kwadratów wszystkich elementów szukanego wektora, możemy określić błąd wektora $\mathbf{x^{(k)}}$ dla każdej iteracji. Residuum jest tym mniejsze, im dokładniejsze jest nasze rozwiązanie. Dla dokładnego rozwiązania, zbiegnie się ono do zera. W metodach iteracyjnych osiągnięcie takiego rozwiązania jest raczej niemożliwe, więc aby zapobiec nieskończonemu wykonywaniu się algorytmu, stosujemy **kryterium stopu**. Może ono przyjąć dwie formy – limit iteracji oraz osiągnięcie **normy residuum** mniejszej

niż np. $\mathbf{10^{-9}}$. To pierwsze chroni nas także w przypadku nietrafionych warunków początkowych, a drugie zapewni nam satysfakcjonujące przybliżenie. Warto zauważyć, że dzięki wystąpieniu kwadratów elementów wektora wynikowego nie mamy problemu z wartościami ujemnymi tzn. $\mathbf{-10^{-3} < 10^{-9}}$, ale kryterium określa nam dokładność ilości cyfr po przecinku, więc porównanie z liczbami ujemnymi byłoby błędne.

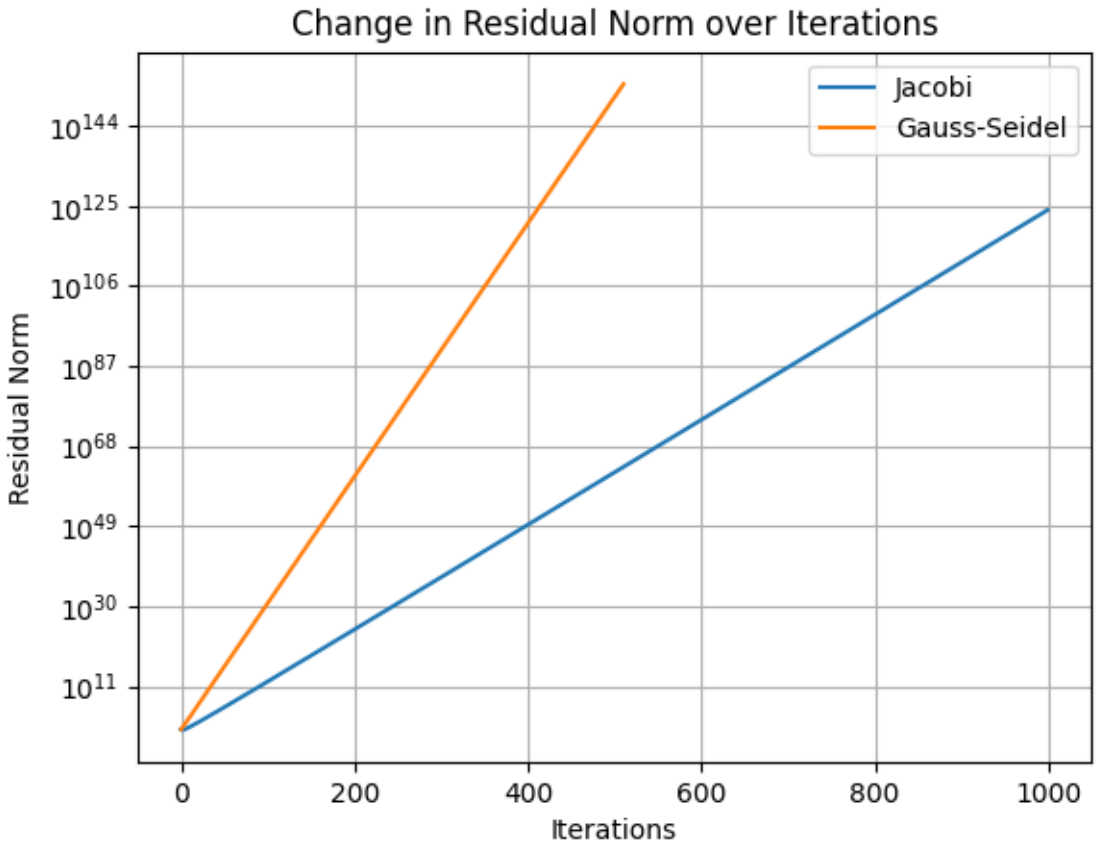


Wykres zależności normy residuum od ilości iteracji w metodzie Jacobiego i Gaussa-Seidla

Na wykresie widzimy, że wykorzystanie najnowszych danych wektora szukanego zmniejsza ilość iteracji nawet dwukrotnie. **Norma residuum** zmniejsza się **logarytmicznie** z każdą iteracją. Czasy wykonywania algorytmów:

Metoda Jacobiego	Metoda Gaussa-Siedla
0.131701s	0.099317s

Czasy obu algorytmów są porównywalne, z lekką przewagą **metody Gaussa-Siedla,** wynikającą prawdopodobnie z mniejszej liczby potrzebnych iteracji do uzyskania satysfakcjonującego wyniku.



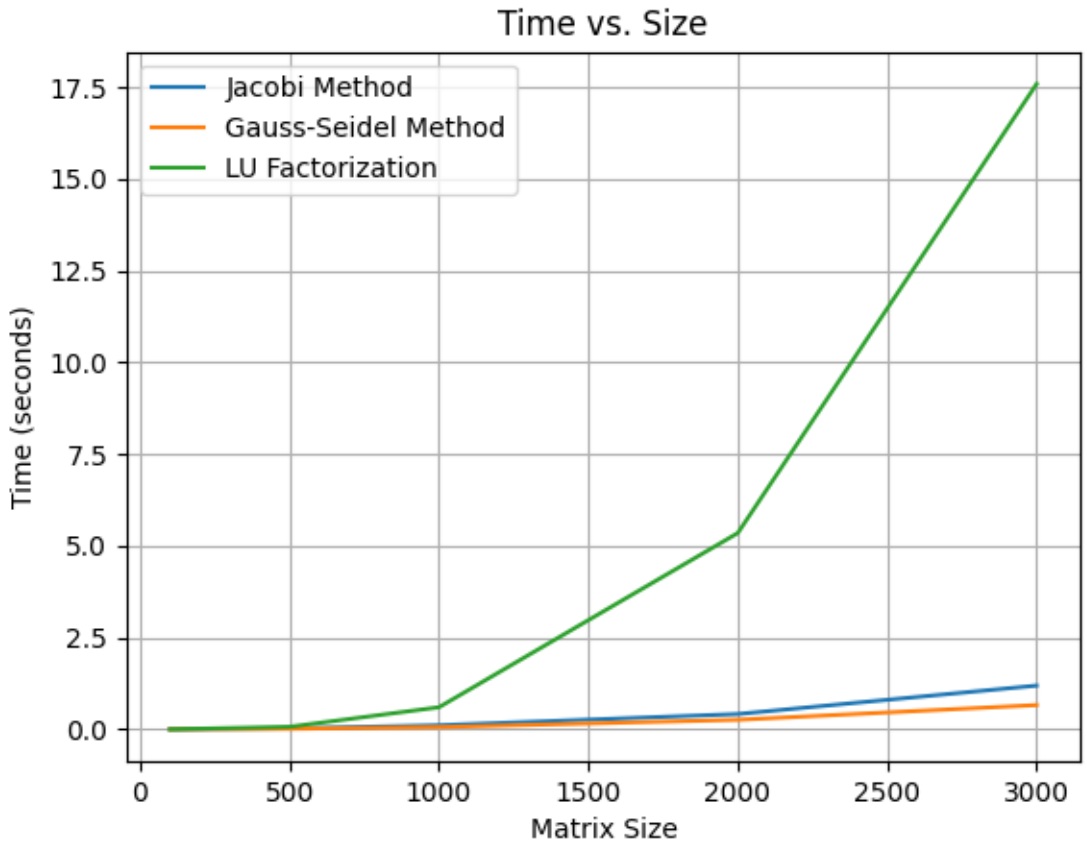
Wykres zależności normy residuum od ilości iteracji w metodzie Jacobiego i Gaussa-Seidla w przypadku braku zbieżności

Nasza macierz systemowa **A** w pierwszym przypadku jest diagonalnie dominująca, czyli ($|a_{ii}| > P_{j \neq i} |a_{ij}|$), dzięki czemu nasze rozwiązanie zbiega do oczekiwanej wartości. W tym przypadku, kiedy $a_1 = 3$, $a_2 = a_3 = -1$, suma wartości bezwzględnych elementów poza diagonalą jest równa, bądź większa od współczynnika na diagonali, więc warunek zbieżności nie jest spełniony. Metody iteracyjne dla takich wartości macierzy nie zbiegają się. Aby temu zapobiec, warto na początku sprawdzić warunki zbieżności określone dla danej metody oraz zastosować iteracyjne kryterium stopu. W tym przypadku zastosowałem limit iteracji równy **1000**, co widać na wykresie dla **metody Jacobiego**. Dla **metody Gaussa-Siedla** natomiast, podejrzewam, że problemem był zakres wartości dla typu **double**.

3. Metody bezpośrednie

W takich przypadkach, kiedy metody iteracyjne zawodzą, zmuszeni jesteśmy do użycia metod bezpośrednich. Dadzą nam one dokładne wyniki dla dowolnej macierzy kosztem większej złożoności obliczeniowej. Metoda **faktoryzacji LU** polega na rozkładzie macierzy **A** na iloczyn dwóch macierzy trójkątnych: dolnej **L** (*lower*) i górnej U(*upper*). Ten rozkład pozwala na efektywne rozwiązanie układu równań **Ax=b** poprzez rozwiązanie dwóch prostszych układów równań: $\boldsymbol{Ly=b}$ i **Ux=y**, gdzie **y** to wektor pośredni. **Metoda LU** jest użyteczna, gdy trzeba rozwiązać ten sam układ równań dla wielu różnych wektorów **b**,

ponieważ rozkład macierzy $\boldsymbol{A}$ wykonuje się tylko raz, a jest to najbardziej złożony obliczeniowo etap tej metody. Metoda ta uzyskała wektor wynikowy, którego norma residuum wynosi ≈ **0.000000000014**, czyli około $\mathbf{10^{-10}}$.



Wykres zależności czasu od rozmiaru macierzy dla poszczególnych metod w przypadku spełnienia warunków zbieżności

Powyższy wykres świetnie obrazuje różnicę w złożoności obliczeniowej metod iteracyjnych oraz metod bezpośrednich. Metody iteracyjne dla coraz to większych macierzy, utrzymują nadal dobry czas. Można powiedzieć, że rośnie on **logarytmicznie** względem rozmiaru. W przypadku metod bezpośrednich sytuacja wygląda gorzej. Już dla macierzy powyżej rozmiaru rzędu **1000**, czas wykonania znacząco wzrasta. Rośnie on **wykładniczo** względem rozmiaru.

4. Wnioski

Na podstawie powyższych rozważań, dochodzę do wniosku, iż **nie ma** najlepszej ogólnej metody do rozwiązywania układów równań liniowych. Jeżeli zależy nam na szybkości oraz spełnione są warunki zbieżności dla określonych metod **iteracyjnych**, a macierz ma rozmiar rzędu **1000**, bądź większy, najrozsądniej jest użyć owych metod. Jeżeli zależy nam na dokładności rozwiązania, bądź macierz nie spełnia warunków zbieżności, zmuszeni jesteśmy do użycia metody **bezpośredniej**. Jednak musimy liczyć się z tym, że czas oczekiwania na odpowiedź będzie znacznie wyższy. W skrajnych przypadkach mogą być to godziny, dni, a może nawet **miesiące**.

Przed rozpoczęciem pracy nad jakimś układem równań liniowych warto przeanalizować posiadane dane w celu dobrania najlepiej pasującego algorytmu. Być może zadaną macierz da się przerobić na taką, aby spełniała ona warunki **zbieżności** metod iteracyjnych? Może dokładne rozwiązanie wcale nie jest nam potrzebne? A może ostatnio nieco za bardzo się przemęczamy i dwa tygodnie wakacji... to znaczy oczekiwania na przetrawienie przez komputer naszych układów jest nam na rękę. Dziękuję za uwagę!